Fast Discrete Distribution Clustering Using Wasserstein Barycenter with Sparse Support

Jianbo Ye, Panruo Wu, James Z. Wang and Jia Li *Senior Member, IEEE*

Abstract—In a variety of research areas, the weighted bag of vectors and the histogram are widely used descriptors for complex objects. Both can be expressed as discrete distributions. D2-clustering pursues the minimum total within-cluster variation for a set of discrete distributions subject to the Kantorovich-Wasserstein metric. D2-clustering has a severe scalability issue, the bottleneck being the computation of a centroid distribution, called Wasserstein barycenter, that minimizes its sum of squared distances to the cluster members. In this paper, we develop a modified Bregman ADMM approach for computing the approximate discrete Wasserstein barycenter of large clusters. In the case when the support points of the barycenters are unknown and have low cardinality, our method achieves high accuracy empirically at a much reduced computational cost. The strengths and weaknesses of our method and its alternatives are examined through experiments, and we recommend scenarios for their respective usage. Moreover, we develop both serial and parallelized versions of the algorithm. By experimenting with large-scale data, we demonstrate the computational efficiency of the new methods and investigate their convergence properties and numerical stability. The clustering results obtained on several datasets in different domains are highly competitive in comparison with some widely used methods in the corresponding areas.

Index Terms—Discrete distribution, K-means, Clustering, Large-scale learning, Parallel computing, ADMM.

I. INTRODUCTION

The discrete distribution, or discrete probability measure, is a well-adopted and succinct way to summarize a batch of data. It often serves as a descriptor for complex instances encountered in machine learning, *e.g.*, images, sequences, and documents, where each instance itself is converted to a data collection instead of a vector. In a variety of research areas including multimedia retrieval and document analysis, the celebrated bag of "words" data model is intrinsically a discrete distribution. The widely used normalized histogram is a special case of discrete distributions with a fixed set of support points across the instances. In many applications involving moderate or high dimensions, the number of bins in the histograms is enormous because of the diversity across instances, while the histogram for any individual instance is highly sparse. This distribution pattern is frequently observed for collections of images and text documents. The discrete distribution can function as a sparse representation for the histogram, where

J. Ye and J. Z. Wang are with College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802, USA (emails: {jxy198, jwang}@ist.psu.edu).

J. Li is with Department of Statistics, The Pennsylvania State University, University Park, PA, 16802, USA (email: jiali@stat.psu.edu).

P. Wu is with Department of Computer Science and Engineering, University of California, Riverside, CA 92521, USA (email: pwu011@cs.ucr.edu).

support points and their probabilities are specified in pairs, eliminating the restriction of a fixed quantization codebook across instances.

The goal of this paper is to develop computationally efficient algorithms for clustering discrete distributions under the Wasserstein distance. The Wasserstein distance is a true metric for measures [1] and can be traced back to the mass transport problem proposed by Monge in the 1780s and the relaxed formulation by Kantorovich in the 1940s [2]. Mallows [3] used this distance to prove some theoretical results in statistics, and thus the name Mallows distance has been used by statisticians. In computer science, this formulation is better known as the Earth Mover's Distance [4]. In signal processing, it is closely related to the Optimal Sub-Pattern Assignment (OSPA) distance [5] used recently for multi-target tracking [6]. A more complete account on the history of this distance can be found in [7]. The Wasserstein distance is computationally expensive because it has no closed form solution and can only be solved with a worst case time complexity $O(m^3 \log m)$ subject to the number of support points in the distribution [8].

We adopt the well-accepted criterion of minimizing the total within-cluster variation under the Wasserstein distance, similarly as Lloyd's K-means for vectors under the Euclidean distance. This clustering problem was originally explored by Li and Wang [9], who coined the phrase *D2-clustering*, referring to both the optimization problem and their particular algorithm. Motivations for using the Wasserstein distance in practice are strongly argued by researchers (*e.g.* [4], [9]–[12]), and its theoretical significance is well supported in the optimal transport (OT) literature (see the seminal books of Villani [2], [7]). D2-clustering computes a principled centroid to summarize each cluster of data. The centroids computed from those clusters—also represented by discrete distributions—are highly valuable for subsequent learning or data mining tasks. Although D2-clustering holds much promise because of the advantages of the Wasserstein distance and its kinship with K-means, the high computational complexity has limited its applications in large-scale learning.

A. Our Contributions

We have developed an efficient and robust method for optimizing the centroids in D2-clustering based on a modified version of the Bregman ADMM (B-ADMM). The data setting in D2-clustering has subtle differences from those taken by the previous OT literature on computing barycenters. As a result, the modified B-ADMM approach is currently the most suitable (yet approximate) method for D2-clustering, surpassing the other Wasserstein barycenter approaches adopted into

the clustering process, namely: subgradient based method, ADMM, and entropy regularization. The properties of the modified B-ADMM approach are thoroughly examined via experiments, and the new method called *AD2-clustering* is capable of handling a large number of instances, provided the support size of each instance is limited to several hundred. We have also developed a parallel algorithm for the modified B-ADMM method in a multi-core environment with adequate scaling efficiency subject to hundreds of CPUs. Finally, we tested our new algorithm on several large-scale real-world datasets, revealing the high competitiveness of AD2-clustering.

B. Related Work

Clustering distributions: Distribution clustering can be subjected to different affinity definitions. For example, Bregman clustering pursues the minimal distortion between the cluster prototype, called the Bregman representative, and cluster members according to a certain Bregman divergence [13]. In comparison, D2-clustering is an extension of K-means to discrete distributions under the Wasserstein distance [9], and the cluster prototype is an approximate Wasserstein barycenter with a sparse finite support set. In the D2-clustering framework, solving the cluster prototype or the centroid for discrete distributions under the Wasserstein distance is computationally challenging [12], [14], [15]. In order to scale up the computation of D2-clustering, a divide-and-conquer approach has been proposed [15], but the method is ad-hoc from an optimization perspective. A standard ADMM approach has also been explored [14], but its efficiency is still inadequate for large datasets. Although fast computation of Wasserstein distances has been much explored [10], [11], [16], how to perform top-down clustering efficiently based on the distance has not. We present below the related work and discuss their relationships with our current work.

The true discrete Wasserstein barycenter and two approximation strategies: The centroid of a collection of distributions minimizing the average pth-order power of the L_p Wasserstein distance is called Wasserstein barycenter [17]. In the D2-clustering algorithm [9], the 2nd-order Wasserstein barycenter is simply referred to as a prototype or centroid, and is solved for the case of an unknown support with a pre-given cardinality. The existence, uniqueness, regularity and other properties of the 2nd-order Wasserstein barycenter have been established mathematically for continuous measures in the Euclidean space [17]. The situation for discrete distributions, however, is more intricate, as will be explained later.

Given N arbitrary discrete distributions each with $\bar{m}$ support points, their true Wasserstein barycenter in theory can be solved via linear programming [17], [18]. This approach is logical because the support points of the Wasserstein barycenter can only locate at a finite (yet huge) number of possible positions. Yet, solving the true discrete barycenter quickly becomes intractable even for a rather small number of distributions containing only 10 support points each. Anderes *et al.* [18] made important theoretical progress on this

particular challenge by proving that the actual support of a true barycenter of N such distributions is extremely sparse, with cardinality m no greater than $\bar{m}N$. Unfortunately, the complexity of the problem is not reduced practically because so far there is no theoretically ensured way to sift out the optimal sparse locations. Anderes *et al.*'s approach, though, does backup the practice of assuming a pre-selected number of support points in a barycenter as an approximation to the true solution.

To achieve good approximation, two computational strategies are useful in an optimization framework.

(i) Carefully select beforehand a large and representative set of support points as an approximation to the support of the true barycenter (*e.g.*, K-means).

(ii) Allow the support points in a barycenter to adjust positions at every τ iterations.

The first strategy of fixing the support of a barycenter can yield adequate approximation quality in low dimensions (*e.g.* 1D/2D histogram data) [12], [19], but can face the challenge of an exponentially growing support size when the dimension increases. The second strategy allows one to use a possibly much smaller number of support points in a barycenter to achieve the same level of accuracy [9], [12], [14], [15]. Because the time complexity per iteration of existing iterative methods is $O(\bar{m}mN)$, a smaller m can also save much computation, and the extra amount of time $O(\bar{m}mdN/\tau)$ can be used to recalculate the distance matrices. In the extreme case when the barycenter support size is set to one ($m = 1$), D2-clustering reduces to K-means on the distribution means, which is a meaningful way of data reduction in its own right. Our experiments indicate that in practice a large m in D2-clustering is usually unnecessary (see Section VII for related discussions).

In applications on high-dimensional data, optimizing the support points is preferred to fixing them from the beginning. This option, however, leads to a non-convex optimization problem. Our work aims at developing practical numerical methods. In particular, our method optimizes jointly the locations and weights of the support points in a single loop without resorting to a bi-level optimization reformulation, as was done in earlier work [9], [12].

Solving discrete Wasserstein barycenter in different data settings: Recently, a series of works have been devoted to solving the Wasserstein barycenter given a set of distributions (*e.g.* [12], [14], [19]–[21]). How our method compares with the existing ones depends strongly on the specific data setting. We discuss the comparisons in details below and promote the use of our new method, AD2-clustering.

In [11], [12], [19], novel algorithms have been developed for solving the Wasserstein barycenter by adding an entropy regularization term on the optimal transport matching weights. The regularization is imposed on the transport weights, but not the barycenter distribution. In particular, iterative Bregman projection (IBP) [19] can solve an approximation to the Wasserstein barycenter. IBP is highly memory efficient for distributions with a shared support set (*e.g.* histogram data), with a memory

complexity $O((m + \bar{m})N)$. In comparison, our modified B-ADMM approach is of the same time complexity, but requires $O(m\bar{m}N)$ memory. If N is large, memory constraints can limit our approach to problems with relatively small m or $\bar{m}$. While the first strategy may not meet the memory constraint, the second approximation strategy is crucial for reaching high accuracy with our approach. Conventional OT literature emphasizes computing the Wasserstein barycenter for a small number of instances with dense representations (*e.g.* [22], [23]); and IBP is more suitable. Yet in many machine learning and signal processing applications, each instance is represented by a discrete distribution with a sparse finite support set (*i.e.*, $\bar{m}$ is small). The memory limitation of B-ADMM can be avoided via parallelization until the time allocation is spent. Our focus is thus to achieve scalability in N.

As demonstrated by experiments, B-ADMM has advantages over IBP that motivate its usage in our algorithm. If the distributions do not share the support set, IBP has the same memory complexity $O(m\bar{m}N)$ (for caching the distance matrix per instance) as our approach has. In addition, B-ADMM [16], based on which our approach is developed, has the following advantages: (1) It yields the exact OT and distance in the limit of iterations. Note that the ADMM parameter does not trade off the convergence rate. (2) It requires little tuning of hyper-parameters and easily accommodates warm starts (to be illustrated later), which are valuable traits for the task of D2-clustering. (3) It works well with single-precision floats, and thus it is not restricted by the machine precision constraint. In contrast, IBP requires more tuning and may encounter precision overflow which is hard to address. Our experiments show that when its coupling solutions are used, the resulting discrete Wasserstein barycenters with sparse finite support sets are not as accurate as those by B-ADMM (see [19] and our experiments).[1]

Optimization methods revisited: Our main algorithm is inspired by the B-ADMM algorithm of Wang and Banerjee [16] for solving OT fast. They developed the two-block variant of ADMM [24] along with Bregman divergence to solve OT when the number of support points is extremely large. Its algorithmic relation to IBP [19] is discussed in Section IV. The OT problem at a moderate scale can in fact be efficiently handled by state-of-the-art LP solvers [25]. As demonstrated by the line of work on solving the barycenter, optimizing the Wasserstein barycenter is rather different from computing the distance. Whereas naïvely adapting the B-ADMM to Wasserstein barycenter does not result in a proper algorithm, our modified B-ADMM algorithm effectively addresses the Wasserstein barycenter problem. The modification we made on B-ADMM is necessary. Although the modified B-ADMM approach is not guaranteed to converge to a local optimum, it often yields a solution very close to the local optimum. The new method is shown empirically to achieve higher accuracy than IBP or its derivatives when distributions are supported on finite sets.

---

[1] Here "accurate" means close to a local minimizer of sum of the (squared) Wasserstein distances.

Finally, we note that although solving a single barycenter for a fixed set is a key component in D2-clustering, the task of clustering per se bears some extra technical subtleties. In a clustering setup, the partition of samples varies over the iterations, and a sequence of Wasserstein barycenters are solved. We found that the robustness with respect to the hyper-parameters in the optimization algorithms is as important as the speed of solving one centroid because tuning these parameters over many iterations of different partitions is impractical.

Outline: The rest of the paper is organized as follows. In Section II, we define notations and provide preliminaries on D2-clustering. In Section III, we develop two scalable optimization approaches for the Wasserstein barycenter problem and explain how they are embedded in the overall algorithm for D2-clustering. In Section IV, our main algorithm is compared with the IBP approach. In Section V, several implementation issues, including initialization, additional techniques for speed-up, and parallelization, are addressed. Comparisons of the algorithms in complexity/performance and guidelines for usage in practice are provided in Section VI. Section VII reports experimental results. The numerical properties and computing performance of the algorithms are investigated; and clustering results on multiple datasets from different domains are compared with those obtained by widely used methods in the respective areas. Finally, we conclude in Section VIII and discuss the limitations of the current work.

II. DISCRETE DISTRIBUTION CLUSTERING

Consider discrete distributions with sparse finite support specified by a set of support points and their associated probabilities, a.k.a. weights:

$$\{(w_1, x_1), \ldots, (w_m, x_m)\},$$

where $\sum_{i=1}^{m} w_i = 1$ with $w_i \geqslant 0$, and $x_i \in \mathbb{M}$ for $i = 1, \ldots, m$. Usually, $\mathbb{M} = \mathbb{R}^d$ is the d-dimensional Euclidean space with the L_p norm, and x_i's are also called support vectors. $\mathbb{M}$ can also be a symbolic set provided with symbol-to-symbol dissimilarity. The Wasserstein distance between distributions $P^{(a)} = \{(w_i^{(a)}, x_i^{(a)}), i = 1, ..., m_a\}$ and $P^{(b)} = \{(w_i^{(b)}, x_i^{(b)}), i = 1, ..., m_b\}$ is solved by the following linear programming (LP). For notation brevity, let $c(x_i^{(a)}, x_j^{(b)}) = \|x_i^{(a)} - x_j^{(b)}\|_p^p$. Define index set $\mathcal{I}_a = \{1, ..., m_a\}$ and $\mathcal{I}_b$ likewise. We define $\left(W_p(P^{(a)}, P^{(b)})\right)^p :=$

$$\begin{aligned}
\min_{\{\pi_{i,j} \geqslant 0\}} \quad & \sum_{i \in \mathcal{I}_a, j \in \mathcal{I}_b} \pi_{i,j} c(x_i^{(a)}, x_j^{(b)}) , \\
\text{s.t.} \quad & \sum_{i=1}^{m_a} \pi_{i,j} = w_j^{(b)} , \ \forall j \in \mathcal{I}_b , \\
& \sum_{j=1}^{m_b} \pi_{i,j} = w_i^{(a)} , \ \forall i \in \mathcal{I}_a .
\end{aligned} \tag{1}$$

We call $\{\pi_{i,j}\}$ the *matching weights* between support points $x_i^{(a)}$ and $x_j^{(b)}$ or the *optimal coupling* for $P^{(a)}$ and $P^{(b)}$. In D2-clustering, we use the L_2 Wasserstein distance. From now on, we will denote W_2 simply by W.

Consider a set of discrete distributions $\{P^{(k)}, k = 1, ..., \bar{N}\}$, where $P^{(k)} = \{(w_i^{(k)}, x_i^{(k)}), i = 1, ..., m_k\}$. The goal of D2-clustering is to find a set of centroid distributions $\{Q^{(i)}, i =$

Algorithm 1 D2 Clustering [9]

1: **procedure** D2CLUSTERING($\{P^{(k)}\}_{k=1}^M$, K)
2:     Denote the label of each objects by $l^{(k)}$.
3:     Initialize K random centroid $\{Q^{(i)}\}_{i=1}^K$.
4:     **repeat**
5:         **for** $k = 1, \ldots, M$ **do**   ▷ Assignment Step
6:             $l^{(k)} := \operatorname{argmin}_i W(Q^{(i)}, P^{(k)})$;
7:         **for** $i = 1, \ldots, K$ **do**   ▷ Update Step
8:             $Q^{(i)} := \operatorname{argmin}_Q \sum_{l^{(k)}=i} W(Q, P^{(k)})$ (*)
9:     **until** the number of changes of $\{l^{(k)}\}$ meets some stopping criterion
10:    **return** $\{l^{(k)}\}_{k=1}^M$ and $\{Q^{(i)}\}_{i=1}^K$.

$1, \ldots, K\}$ such that the total within-cluster variation is minimized:

$$\min_{\{Q^{(i)}\}} \sum_{k=1}^{\bar{N}} \min_{i=1,\ldots,K} W^2(Q^{(i)}, P^{(k)}) .$$

Similarly as in K-means, D2-clustering alternates the optimization of the centroids $\{Q^{(i)}\}$ and the assignment of each instance to the nearest centroid, the iteration referred to as the *outer loop* (Algorithm 1). The major computational challenge in the algorithm is to compute the optimal centroid for each cluster at each iteration. This computation also marks the main difference between D2-clustering and K-means in which the optimal centroid is in a simple closed form. The new scalable algorithms we develop here aim primarily to expedite this optimization step. For clarity of presentation, we now focus on this optimization problem and describe the notation below. Suppose we have a set of discrete distributions $\{P^{(1)}, \ldots, P^{(N)}\}$. N is the sample size for computing one Wasserstein barycenter. We want to find a centroid P : $\{(w_1, x_1), \ldots, (w_m, x_m)\}$, such that

$$\min_P \frac{1}{N} \sum_{k=1}^N W^2(P, P^{(k)}) \tag{2}$$

with respect to the weights and support points of P. This is the **main question** we tackle in this paper. There is an implicit layer of optimization in (2)—the computation of $W^2(P, P^{(k)})$. The variables in optimization (2) thus include the weights in the centroid $\{w_i \in \mathbb{R}^+\}$, the support points $\{x_i \in \mathbb{R}^d\}$, and the optimal coupling between P and $P^{(k)}$ for each k, denoted by $\{\pi_{i,j}^{(k)}\}$ (see Eq. (1)).

To solve (2), D2-clustering alternates the optimization of $\{w_i\}$ and $\{\pi_{i,j}^{(k)}\}$, $k = 1, \ldots, N$, versus $\{x_i\}$.

1) Δ_k denotes a probability simplex of k dimensions.
2) $\mathbf{1}$ denotes a vector with all elements equal to one.
3) $\boldsymbol{x} = (x_1, \ldots, x_m) \in \mathbb{R}_{d \times m}$, $\boldsymbol{w} = (w_1, \ldots, w_m) \in \Delta_m$.
4) $\boldsymbol{x}^{(k)} = (x_1^{(k)}, \ldots, x_{m_k}^{(k)}) \in \mathbb{R}_{d \times m_k}$, $k = 1, \ldots, N$.
5) $\boldsymbol{w}^{(k)} = (w_1^{(k)}, \ldots, w_{m_k}^{(k)}) \in \Delta_{m_k}$.
6) $C(\boldsymbol{x}, \boldsymbol{x}^{(k)}) = (\|x_i - x_j^{(k)}\|^2)_{i,j} \in \mathbb{R}^{m \times m_k}$.
7) $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}) \in \mathbb{R}_{d \times n}$, where $n = \sum_{k=1}^N m_k$.
8) $\Pi^{(k)} = (\pi_{i,j}^{(k)}) \in \mathbb{R}_{m \times m_k}^+$, $k = 1, \ldots, N$.
9) $\Pi = (\Pi^{(1)}, \ldots, \Pi^{(N)}) \in \mathbb{R}_{m \times n}^+$.

10) Index set $\mathcal{I}^c = \{1, \ldots, N\}$, $\mathcal{I}_k = \{1, \ldots, m_k\}$, for $k \in \mathcal{I}^c$, and $\mathcal{I}' = \{1, \ldots, m\}$.

With $\boldsymbol{w}$ and Π fixed, the cost function (2) is quadratic in terms of $\boldsymbol{x}$, and the optimal $\boldsymbol{x}$ is solved by:

$$x_i := \frac{1}{N w_i} \sum_{k=1}^N \sum_{j=1}^{m_k} \pi_{i,j}^{(k)} x_j^{(k)}, \quad i \in \mathcal{I}', \tag{3}$$

Or, we can write it in matrix form: $\boldsymbol{x} := \frac{1}{N} X \Pi^T \operatorname{diag}(1./\boldsymbol{w})$. However, with fixed $\boldsymbol{x}$, updating $\boldsymbol{w}$ and Π is challenging. D2-clustering solves a large LP as follows:

$$\min_{\Pi \in \mathbb{R}_{m \times n}^+, \boldsymbol{w} \in \Delta_m} \sum_{k=1}^N \langle C(\boldsymbol{x}, \boldsymbol{x}^{(k)}), \Pi^{(k)} \rangle, \tag{4}$$

$$\text{s.t.} \quad \mathbf{1} \cdot (\Pi^{(k)})^T = \boldsymbol{w}, \quad \mathbf{1} \cdot \Pi^{(k)} = \boldsymbol{w}^{(k)}, \forall k \in \mathcal{I}^c,$$

where the inner product $\langle A, B \rangle := \operatorname{tr}(AB^t)$.

Algorithm 2 Centroid Update with Full-batch LP [9], [12]

1: **procedure** CENTROID($\{P^{(k)}\}_{k=1}^N$)
2:     **repeat**
3:         Updates $\{x_i\}$ from Eq. (3);
4:         Updates $\{w_i\}$ from solving full-batch LP (4);
5:     **until** P converges
6:     **return** P

By iteratively solving (3) and (4), referred to as the *inner loop*, the step of updating the cluster centroid in Algorithm 1 is fulfilled [9], [12]. We present the centroid update step in Algorithm 2. In summary, D2-clustering is given by Algorithm 1 with Algorithm 2 embedded as one key step.

The major difficulty in solving (4) is that a standard LP solver typically has a polynomial complexity in terms of the number of variables $m + \sum_{k=1}^N m_k m$, prohibiting its scalability to a large number of discrete distributions in one cluster. When the cluster size is small or moderate, say dozens, it is shown that the standard LP solver can be faster than a scalable algorithm [14]. However, when the cluster size grows, the standard solver slows down quickly. This issue has been demonstrated by multiple empirical studies [9], [14], [15].

Our key observation is that in the update of a centroid distribution, the variables in $\boldsymbol{w}$ are much more important than are the matching weights in Π needed for computing the Wasserstein distances. The parameter $\boldsymbol{w}$ is actually part of the output centroid, while Π is not, albeit accounting for the vast majority of the variables in (4). We also note that the solution (4) is not the end result, but rather it is one round of centroid update in the outer loop. It is thus adequate to have a sufficiently accurate solution to (4), motivating us to pursue scalable methods such as ADMM, known to be fast for reaching the vicinity of the optimal solution.

III. SCALABLE CENTROID COMPUTATION

We propose algorithms scalable with large-scale datasets, and compare their performance in terms of speed and memory. They are (a) subgradient descent with N mini-LP following similar ideas of [12] (included in Appendix A), (b) standard ADMM with N mini-QP, and (c) modified B-ADMM with

closed forms in each iteration of the inner loop. The bottleneck in the computation of D2-clustering is the inner loop, detailed in Algorithm 2. The approaches we develop here all aim for fast solutions for the inner loop, that is, to improve Algorithm 2. These new methods can reduce the computation for centroid update to a comparable (or even lower) level as the label assignment step, usually negligible in the original D2-clustering. As a result, we also take measures to expedite the labeling step, with details provided in Section V.

A. Alternating Direction Method of Multipliers

ADMM typically solves a problem with two sets of variables (in our case, they are Π and $\boldsymbol{w}$), which are only coupled in constraints, while the objective function is separable across this splitting of the two sets (in our case, $\boldsymbol{w}$ is not present in the objective function) [24]. Because problem (4) has multiple sets of constraints including both equalities and inequalities, it is not a typical scenario to apply ADMM. We propose to relax all equality constraints $\sum_{l=1}^{m_k} \pi_{i,l}^{(k)} = w_i$, $\forall k \in \mathcal{I}^c$, $i \in \mathcal{I}'$ in (4) to their corresponding augmented Lagrangians and use the other constraints to determine a convex set for the parameters being optimized. Let $\Lambda = (\lambda_{i,k})$, $i \in \mathcal{I}'$, $k \in \mathcal{I}^c$. Let ρ be a parameter to balance the objective function and the augmented Lagrangians. Define $\Delta_\Pi = \left\{ (\pi_{i,j}^{(k)}) : \sum_{i=1}^{m} \pi_{i,j}^{(k)} = w_j^{(k)}, \pi_{i,j}^{(k)} \geqslant 0, k \in \mathcal{I}^c, i \in \mathcal{I}', j \in \mathcal{I}_k \right\}$. Recall that $\Delta_m = \{ (w_1, \ldots, w_m) | \sum_{i=1}^{m} w_i = 1, w_i \geqslant 0 \}$. As in the method of multipliers, we form the scaled augmented Lagrangian $L\rho(\Pi, \boldsymbol{w}, \Lambda)$ as follows

$$L_\rho(\Pi, \boldsymbol{w}, \Lambda) = \sum_{k=1}^{N} \langle C(\boldsymbol{x}, \boldsymbol{x}^{(k)}), \Pi^{(k)} \rangle +$$
$$\rho \sum_{\substack{i \in \mathcal{I}' \\ k \in \mathcal{I}^c}} \lambda_{i,k} \left(\sum_{j=1}^{m_k} \pi_{i,j}^{(k)} - w_i \right) + \frac{\rho}{2} \sum_{\substack{i \in \mathcal{I}' \\ k \in \mathcal{I}^c}} \left(\sum_{j=1}^{m_k} \pi_{i,j}^{(k)} - w_i \right)^2 . \tag{5}$$

Problem (4) can be solved using ADMM iteratively as follows.

$$\Pi^{n+1} := \operatorname*{argmin}_{\Pi \in \Delta_\Pi} L_\rho(\Pi, \boldsymbol{w}^n, \Lambda^n) , \tag{6}$$

$$\boldsymbol{w}^{n+1} := \operatorname*{argmin}_{\boldsymbol{w} \in \Delta_m} L_\rho(\Pi^{n+1}, \boldsymbol{w}, \Lambda^n) , \tag{7}$$

$$\lambda_{i,k}^{n+1} := \lambda_{i,k}^n + \sum_{j=1}^{m_k} \pi_{i,j}^{(k),n+1} - w_i^{n+1} , i \in \mathcal{I}', k \in \mathcal{I}^c . \tag{8}$$

Based on (6), Π can be updated by updating $\Pi^{(k)}$, $k = 1, ..., N$ separately. Comparing with the full batch LP in (4) which solves all $\Pi^{(k)}$, $k = 1, ..., N$, together, ADMM solves instead N disjoint constrained quadratic programming (QP). This step is the key for achieving computational complexity linear in N, the main motivation for employing ADMM. Specifically, we solve (4) by solving (9) below for each $k = 1, ..., N$:

$$\min_{\pi_{i,j}^{(k)} \geqslant 0} \quad \langle C(\boldsymbol{x}, \boldsymbol{x}^{(k)}), \Pi^{(k)} \rangle$$
$$+ \frac{\rho}{2} \sum_{i=1}^{m} \left(\sum_{j=1}^{m_k} \pi_{i,j}^{(k)} - w_i^n + \lambda_{i,k}^n \right)^2 \tag{9}$$
$$\text{s.t.} \quad \mathbf{1} \cdot \Pi^{(k)} = \boldsymbol{w}^{(k)}, k \in \mathcal{I}^c .$$

Since we need to solve small-size problem (9) in multiple rounds, we prefer active set method with warm start. Define $\tilde{w}_i^{(k),n+1} := \sum_{j=1}^{m_k} \pi_{i,j}^{(k),n+1} + \lambda_{i,k}^n$ for $i = 1, ..., m$, $k = 1, ..., N$. We can rewrite step (7) as

$$\min_{\boldsymbol{w} \in \Delta_m} \sum_{i=1}^{m} \sum_{k=1}^{N} (\tilde{w}_i^{(k),n+1} - w_i)^2 . \tag{10}$$

Algorithm 3 Centroid Update with ADMM [14]

1: **procedure** CENTROID($\{P^{(k)}\}_{k=1}^{N}$, P, Π)
2:      Initialize $\Lambda^0 = 0$ and $\Pi^0 := \Pi$.
3:      **repeat**
4:          Updates $\{x_i\}$ from Eq.(3);
5:          Reset dual coordinates Λ to zero;
6:          **for** $iter = 1, \ldots, T_{admm}$ **do**
7:              **for** $k = 1, \ldots, N$ **do**
8:                  Update $\{\pi_{i,j}\}^{(k)}$ based on QP Eq.(9);
9:              Update $\{w_i\}$ based on QP Eq.(10);
10:              Update Λ based on Eq. (8);
11:      **until** P converges
12:      **return** P

We summarize the computation of the centroid distribution P for distributions $P^{(k)}$, $k = 1, ..., N$, in Algorithm 3. There are two hyper-parameters to choose: ρ and the number of iterations T_{admm}. We empirically select ρ proportional to the averaged transportation costs:

$$\rho = \frac{\rho_0}{Nnm} \sum_{k=1}^{N} \sum_{i \in \mathcal{I}'} \sum_{j \in \mathcal{I}_k} c(x_i, x_j^{(k)}) . \tag{11}$$

Let us compare the computational efficiency of ADMM and the subgradient descent method. In gradient descent based approaches, it is costly to choose an effective step-size along the descending direction because at each search point, we need to solve N LP — an issue also discussed in [19]. ADMM solves N QP sub-problems instead of LP. The amount of computation in each sub-problem of ADMM is thus usually higher and grows faster with the number of support points in $P^{(k)}$'s. Whether the increased complexity at each iteration of ADMM is paid off by a better convergence rate (*i.e.*, a smaller number of iterations) is unclear. The computational limitation of ADMM caused by QP motivates us to explore B-ADMM that avoids QP in each iteration.

B. Bregman ADMM

Bregman ADMM (B-ADMM) replaces the quadratic augmented Lagrangians by the Bregman divergence when updating the split variables [26]. Similar ideas trace back at least to the early 1990s [27], [28]. We adapt the design in [11], [16] for solving the OT problem with a large set of support points. Consider two sets of variables $\Pi_{(k,1)} = (\pi_{i,j}^{(k,1)})$, $i \in \mathcal{I}'$,

$j \in \mathcal{I}_k$, and $\Pi_{(k,2)} = (\pi_{i,j}^{(k,2)})$, $i \in \mathcal{I}'$, $j \in \mathcal{I}_k$, for $k = 1, ..., N$ under the following constraints. Let

$$\Delta_{k,1} := \left\{ \pi_{i,j}^{(k,1)} \geqslant 0 : \sum_{i=1}^{m} \pi_{i,j}^{(k,1)} = w_j^{(k)}, j \in \mathcal{I}_k \right\}, \quad (12)$$

$$\Delta_{k,2}(\boldsymbol{w}) := \left\{ \pi_{i,j}^{(k,2)} \geqslant 0 : \sum_{j=1}^{m_k} \pi_{i,j}^{(k,2)} = w_i, i \in \mathcal{I}' \right\}, \quad (13)$$

then $\Pi^{(k,1)} \in \Delta_{k,1}$ and $\Pi^{(k,2)} \in \Delta_{k,2}(\boldsymbol{w})$. We introduce some extra notations:

1) $\bar{\Pi}^{(1)} = \{\Pi^{(1,1)}, \Pi^{(2,1)}, \ldots, \Pi^{(N,1)}\}$,
2) $\bar{\Pi}^{(2)} = \{\Pi^{(1,2)}, \Pi^{(2,2)}, \ldots, \Pi^{(N,2)}\}$,
3) $\bar{\Pi} = \{\bar{\Pi}^{(1)}, \bar{\Pi}^{(2)}\}$,
4) $\Lambda = \{\Lambda^{(1)}, \ldots, \Lambda^{(N)}\}$, where $\Lambda^{(k)} = (\lambda_{i,j}^{(k)})$, $i \in \mathcal{I}'$, $j \in \mathcal{I}_k$, is a $m \times m_k$ matrix.

B-ADMM solves (4) by treating the augmented Lagrangians conceptually as a designed divergence between $\Pi^{(k,1)}$ and $\Pi^{(k,2)}$, adapting to the updated variables. It restructures the original problem (4) as

$$\min_{\bar{\Pi}, \boldsymbol{w}} \quad \sum_{k=1}^{N} \langle C(\boldsymbol{x}, \boldsymbol{x}^{(k)}), \Pi^{(k,1)} \rangle, \quad (14)$$

$$\text{s.t.} \quad \boldsymbol{w} \in \Delta_m,$$
$$\Pi^{(k,1)} \in \Delta_{k,1}, \quad \Pi^{(k,2)} \in \Delta_{k,2}(\boldsymbol{w}), \quad k = 1, \ldots, N,$$
$$\Pi^{(k,1)} = \Pi^{(k,2)}, \quad k = 1, \ldots, N.$$

Denote the dual variables $\Lambda^{(k)} = (\lambda_{i,j}^{(k)})$, $i \in \mathcal{I}'$, $j \in \mathcal{I}_k$, for $k = 1, ..., N$. Use $\text{KL}(\cdot, \cdot)$ to denote the Kullback–Leibler divergence between two distributions. The B-ADMM algorithm adds the augmented Lagrangians for the last set of constraints in its updates, yielding the following equations.

$$\bar{\Pi}^{(1),n+1} := \operatorname*{argmin}_{\{\Pi^{(k,1)} \in \Delta_{k,1}\}} \sum_{k=1}^{N} \left(\langle C(\boldsymbol{x}, \boldsymbol{x}^{(k)}), \Pi^{(k,1)} \rangle \right.$$
$$\left. + \langle \Lambda^{(k),n}, \Pi^{(k,1)} \rangle + \rho \, \text{KL}(\Pi^{(k,1)}, \Pi^{(k,2),n}) \right), \quad (15)$$

$$\bar{\Pi}^{(2),n+1}, \boldsymbol{w}^{n+1} := \operatorname*{argmin}_{\substack{\{\Pi^{(k,2)} \in \Delta_{k,1}(\boldsymbol{w})\} \\ \boldsymbol{w} \in \Delta_m}} \sum_{k=1}^{N} \left(-\langle \Lambda^{(k),n}, \Pi^{(k,2)} \rangle \right.$$
$$\left. + \rho \, \text{KL}(\Pi^{(k,2)}, \Pi^{(k,1),n+1}) \right), \quad (16)$$

$$\Lambda^{n+1} := \Lambda^n + \rho(\bar{\Pi}^{(1),n+1} - \bar{\Pi}^{(2),n+1}). \quad (17)$$

We note that if $\boldsymbol{w}$ is fixed, (15) and (16) can be split by index $k = 1, ..., N$, and have closed form solutions for each k. Let eps be the floating-point tolerance (*e.g.* 10^{-16}). For any $i \in \mathcal{I}'$, $j \in \mathcal{I}_k$,

$$\tilde{\pi}_{i,j}^{(k,2),n} := \pi_{i,j}^{(k,2),n} \exp\left[\frac{c\left(x_i, x_j^{(k)}\right) + \lambda_{i,j}^{(k),n}}{-\rho}\right] + eps, \quad (18)$$

$$\pi_{i,j}^{(k,1),n+1} := \frac{\tilde{\pi}_{i,j}^{(k,2),n}}{\sum_{l=1}^{m} \tilde{\pi}_{l,j}^{(k,2),n}} \cdot w_j^{(k)}, \quad (19)$$

$$\tilde{\pi}_{i,j}^{(k,1),n+1} := \pi_{i,j}^{(k,1),n+1} \exp\left[\frac{1}{\rho} \lambda_{i,j}^{(k),n}\right] + eps, \quad (20)$$

$$\pi_{i,j}^{(k,2),n+1} := \frac{\tilde{\pi}_{i,j}^{(k,1),n+1}}{\sum_{l=1}^{m_k} \tilde{\pi}_{i,l}^{(k,1),n+1}} \cdot w_i. \quad (21)$$

Because we need to update $\boldsymbol{w}$ in each iteration, it is not easy to solve (16). We consider decomposing (16) into two stages. Observe that the minimum value of (16) under a given $\boldsymbol{w}$ is

$$\min_{\boldsymbol{w} \in \Delta_m} \sum_{k=1}^{N} \sum_{i=1}^{m} w_i \left[\log(w_i) - \log\left(\sum_{j=1}^{m_k} \tilde{\pi}_{i,j}^{(k,1),n+1}\right) \right]. \quad (22)$$

The above term (a.k.a. the consensus operator) is minimized by

$$w_i^{n+1} \propto \left[\prod_{k=1}^{N} \left(\sum_{j=1}^{m_k} \tilde{\pi}_{i,j}^{(k,1),n+1} \right) \right]^{1/N}, \quad \sum_{i=1}^{m} w_i^{n+1} = 1. \quad (23)$$

However, the above equation is a geometric mean, which is numerically unstable when $\sum_{j=1}^{m_k} \tilde{\pi}_{i,j}^{(k,1),n+1} \to 0^+$ for some combination of i and k. Here, we employ a different technique. Let

$$\tilde{w}_i^{(k,1),n+1} \propto \sum_{j=1}^{m_k} \tilde{\pi}_{i,j}^{(k,1),n+1}, \quad \text{s.t.} \sum_{i=1}^{m} \tilde{w}_i^{(k,1),n+1} = 1.$$

Let the distribution $\tilde{\boldsymbol{w}}^{(k),n+1} = (\tilde{w}_i^{(k,1),n+1})_{i=1,\ldots,m}$. Then Eq. (22) is equivalent to $\min_{\boldsymbol{w} \in \Delta_m} \sum_{k=1}^{N} \text{KL}(\boldsymbol{w}, \tilde{\boldsymbol{w}}^{(k),n+1})$. Essentially, a consensus $\boldsymbol{w}$ is sought to minimize the sum of KL divergence. In the same spirit, we propose to find a consensus by changing the order of $\boldsymbol{w}$ and $\tilde{\boldsymbol{w}}^{(k),n+1}$ in the KL divergence: $\min_{\boldsymbol{w} \in \Delta_m} \sum_{k=1}^{N} \text{KL}(\tilde{\boldsymbol{w}}^{(k),n+1}, \boldsymbol{w}) =$

$$\min_{\boldsymbol{w} \in \Delta_m} \sum_{k=1}^{N} \sum_{i=1}^{m} \tilde{w}_i^{(k,1),n+1} (\log(\tilde{w}_i^{(k,1),n+1}) - \log(w_i)), \quad (24)$$

which again has a closed form solution:

$$(\text{R1}): w_i^{n+1} \propto \frac{1}{N} \sum_{k=1}^{N} \tilde{w}_i^{(k,1),n+1}, \quad \sum_{i=1}^{m} w_i^{n+1} = 1. \quad (25)$$

The solution of Eq. (24) overcomes the numerical instability. We will call this heuristic update rule as (R1), which has been employed in the Bregman clustering method [13]. In addition, a slightly different version of update rule can be

$$(\text{R2}): \left(w_i^{n+1}\right)^{1/2} \propto \frac{1}{N} \sum_{k=1}^{N} \left(\tilde{w}_i^{(k,1),n+1}\right)^{1/2}, \quad \sum_{i=1}^{m} w_i^{n+1} = 1. \quad (26)$$

In Section IV, we conduct experiments for testing both (R1) and (R2). We have tried other update rules, such as Fisher-Rao Riemannian center [29], and found that the experimental results do not differ much in terms of the converged objective function. It is worth mentioning that neither (R1) nor (R2) ensures the convergence to a (local) minimum.

We summarize the B-ADMM approach in Algorithm 4. The implementation involves one hyper-parameters ρ (by default, $\tau = 10$). In our implementation, we choose ρ relatively according to Eq. (11). To the best of our knowledge, the convergence of B-ADMM has not been proved for our formulation (even under fixed support points x) although this topic has been pursued in recent literature [16]. In the general case of solving Eq. (2), the optimization of the cluster centroid is non-convex because the support points are updated after B-ADMM is applied to optimize the weights. In Section VII-A, we empirically test the convergence of the centroid optimization algorithm based on B-ADMM. We found that B-ADMM usually converges quickly to a moderate accuracy, making it preferable for D2-clustering. In our implementation, we use a fixed number of B-ADMM iterations (by default, 100) across multiple assignment-update rounds in D2-clustering.

Algorithm 4 Centroid Update with B-ADMM

1: **procedure** CENTROID($\{P^{(k)}\}_{k=1}^N$, P, Π).
2:      $\Lambda := 0$; $\bar{\Pi}^{(2),0} := \Pi$.
3:      **repeat**
4:          Update x from Eq.(3) per τ loops;
5:          **for** $k = 1, \ldots, N$ **do**
6:              Update $\Pi^{(k,1)}$ based on Eq.(18) (19);
7:              Update $\{\tilde{\pi}_{i,j}^{(k,1)}\}$ based on Eq.(20);
8:          Update w based on Eq.(25) or Eq.(26);
9:          **for** $k = 1, \ldots, N$ **do**
10:             Update $\Pi^{(k,2)}$ based on Eq.(21);
11:             $\Lambda^{(k)} := \Lambda^{(k)} + \rho(\Pi^{(k,1)} - \Pi^{(k,2)})$;
12:      **until** P converges
13:      **return** P

IV. COMPARISON BETWEEN B-ADMM AND ITERATIVE BREGMAN PROJECTION

Both the B-ADMM and IBP can be rephrased into two-step iterative algorithms via mirror maps (in a similar way of mirror prox [30] or mirror descent [31]). One step is the free-space move in the dual space, and the other is the Bregman projection (as used in IPFP) in the primal space. Let $\Phi(\cdot)$ be the entropy function, the mirror map used by IBP is $\Phi(\Pi)$, while the mirror map of B-ADMM is

$$\Phi(\Pi^{(1)}, \Pi^{(2)}, \Lambda) = \Phi(\Pi^{(1)}) + \Phi(\Pi^{(2)}) + \frac{\|\Lambda\|^2}{\rho^2},$$

where Λ is the dual coordinate derived from relaxing constraints $\Pi^{(1)} = \Pi^{(2)}$ to a saddle point reformulation. IBP alternates the move $-\begin{bmatrix} \dfrac{C}{\varepsilon} \end{bmatrix}$ in the dual space and projection

in Δ_1 or Δ_2 of the primal space. In comparison, B-ADMM alternates the move

$$-\begin{bmatrix} \dfrac{C + \Lambda}{\rho} + \nabla\Phi(\Pi^{(1)}) - \nabla\Phi(\Pi^{(2)}) \\ -\dfrac{\Lambda}{\rho} + \nabla\Phi(\Pi^{(2)}) - \nabla\Phi(\Pi^{(1)}) \\ \rho(\Pi^{(1)} - \Pi^{(2)}) \end{bmatrix},$$

and the projection in $\Delta_1 \times \Delta_2 \times \mathbb{R}_{m_1 \times m_2}$.[2] The convergence of B-ADMM is not evident from the conventional optimization literature [32] because the move is not monotonic (It is still monotonic for standard ADMM). We conduct two pilot studies to compare B-ADMM and IBP in terms of convergence behavior and the quality of the Wasserstein barycenters with a sparse finite support set, where quality is measured by the objective function achieved.

Although the second pilot study shows certain advantages of B-ADMM, we clarify that the study is not intended to demonstrate which algorithm is better in the general context of the OT problem. In fact, for the OT problem alone, IBP is more solid in theory than B-ADMM in two aspects. First, IBP has linear convergence, while B-ADMM only has a sub-linear rate [16]. Second, IBP yields an OT solution more accurately satisfying the coupling constraints than B-ADMM can offer with the same computational time. In the Wasserstein barycenter problem we tackle here, either algorithm must be embedded into an outer loop, which calls for practical considerations other than solving a stand-alone OT problem. We will elaborate on these points below.

Benamou *et al.* [19], in their Figure 1, show how their algorithm progressively shifts mass away from the diagonal over the iterations. We adopt the same study here and visualize in Fig. 1 how the mass transport between two 1D distributions evolves over iterations. As a qualitative study, we observe that the entropy regularization term in IBP introduces clearly smoothing effect on the final solutions. It has been pointed out in the literature that the smoothing effect of entropy regularization diminishes as parameter ε decreases. In our study, we found the smoothing effect is also affected by the support size of a distribution. A smooth distribution with large support tends to have higher entropy, thus a relatively smaller ε is needed to achieve similar results. Fig. 1 shows that at $\varepsilon = 0.1/N$, the mass transport of IBP at 5,000 iterations is close to the unregularized solution albeit with noticeable difference. Setting ε even smaller (say $0.04/N$) introduces double-precision overflow. As suggested by one of the reviewers, this numerical difficulty can be addressed by thresholding entries that are too small. We applied the reviewer's suggested IBP code with this technique implemented and obtained an incorrect coupling result with artifacts unexplainable by smoothing, *e.g.*, a non-zero region separated from the correct non-zero region. Yet more recently, we learned from the reviewer that active research on the thresholding technique is currently underway with new manuscripts emerging while we approached the final revision of this paper. In particular, log domain scaling or more sophisticated schemes

---

[2]The update is done in the Gauss-Seidel type, not in the usual Jacobi type.
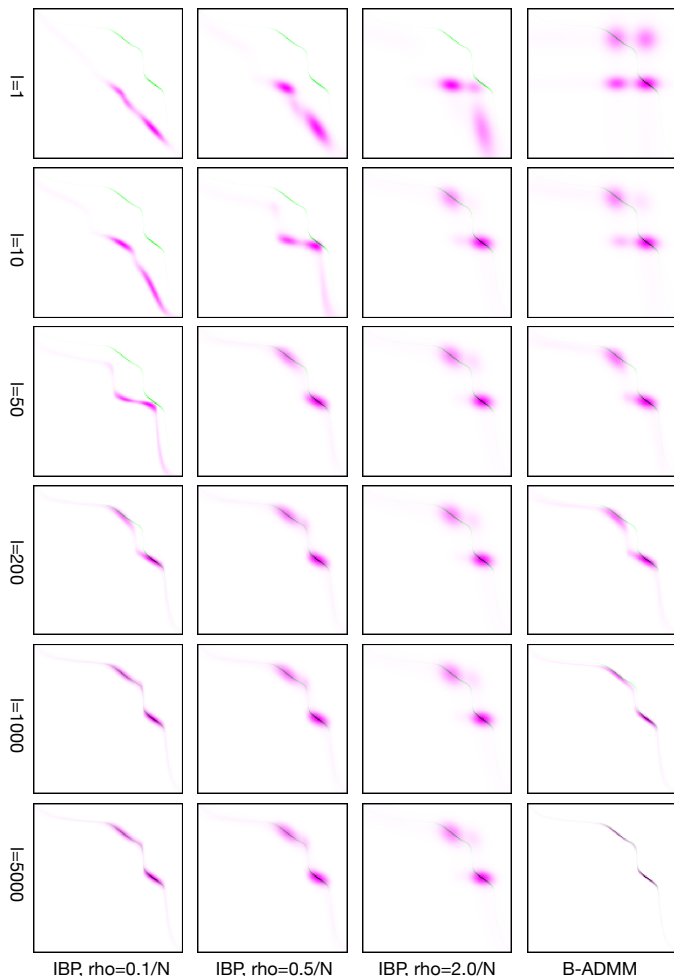
Figure 1. Convergence behavior of IBP and B-ADMM. The solutions by IBP and B-ADMM are shown in pink, while the exact solution by linear programming is shown in green. Images in the rows from top to bottom present results at different iterations $\{1, 10, 50, 200, 1000, 5000\}$; The left three columns are by IBP with $\varepsilon = \{0.1/N, 0.5/N, 2/N\}$, where $[0, 1]$ is discretized with $N = 128$ uniformly spaced points. The last column is by B-ADMM. This comparison is not intended for speed. IBP is about 10 to 15 times faster per iteration in this experiment.

have been proposed to stabilize the low ε case for Sinkhorn algorithm [33], [34]. At extra computational costs, these new methods produce sharper coupling results than the standard IBP does. Investigating the effectiveness of those algorithms for the Wasserstein barycenter problem is an interesting future work. In contrast, the output of mass transport by B-ADMM ($\rho_0 = 2$, the default setting) at 5,000 iteration is nearly indiscernible from the unregularized solution by LP. This example suggests that if the smoothing effect on the final coupling is to be avoided, B-ADMM may be preferred over IBP with $\epsilon \to 0$, albeit at a cost of increased computation time. With our implementation of B-ADMM and the IBP code provided by the reviewer, we found that IBP is 10 to 15 times faster per iteration.

Because B-ADMM does not require $\rho_0 \to 0$, no numerical difficulty has been encountered in practice. In fact, the convergence speed of B-ADMM is proven to be independent with ρ_0 [16]. As for the Wasserstein barycenter problem, the

minimal required tuning makes embedding B-ADMM in an outer loop easy. Putting the IBP in the outer loop reduces its speed advantage. In existing machine learning practice as well as with our algorithm here, pre-converged solutions of IBP or B-ADMM are used, making the use of a very small ϵ in IBP unnecessary. In our algorithm, however, B-ADMM per iteration slows the process as does updating barycenter support points in high dimensions. Thus, the computational gain from replacing B-ADMM by IBP is clipped by the notable proportion of time required to update the support points. In addition, although the coupling weights solved by B-ADMM do not satisfy the constraints as well as those by IBP do, they are auxiliary variables, while the barycenters are primary. In summary, the edge of either IBP or B-ADMM over the other subsides when they are embedded in our algorithm. In order to quantitatively compare the methods by measures most pertinent to the users, we examine the objective function and computation time in the second pilot study below.

For the second pilot study, we generated a set of 1,000 discrete distributions, each with a sparse finite support set obtained by clustering pixel colors of images [9] ($d = 3$). The average number of support points is around 6. Starting from the same initial estimate, we calculate the approximate Wasserstein barycenter by different methods. We obtained results for two cases: barycenters with support size $m = 6$ and barycenters with support size $m = 60$. We use relatively small values of m here in comparison with the existing applications of IBP in imaging science because our focus is on large data size but low support size (sparse support). The obtained barycenters are then evaluated by comparing the objective function (Eq. (2)) with that solved directly by the full batch LP (Algorithm 2)—the yardstick in the comparison.

The bare form of IBP treats the case of pre-fixed support points while B-ADMM does not constrain the locations. In order to compare the two methods on a common ground, we used two tracks of experiments. In the first track the locations of support points in the barycenters are fixed and shared by all the algorithms, while in the second track both locations and weights are optimized. To adopt IBP in the second track, we experimented with a version of IBP that can automatically restart and update its support points. The restart criterion is that the constraint satisfies certain conditions described in [12], [19]. Generally speaking, the larger ε_0 is the fewer iterations are needed before a restart is evoked. This variant of IBP does not have a descending objective.[3] Therefore, we chose to terminate the iterations when $\langle C, \Pi \rangle$ is detected to increase. The actual implementation can be found in our supplement. In MATLAB, we found IBP to be approximately 10 times faster than B-ADMM per iteration. On the other hand, being faster in one iteration does not necessarily mean faster speed overall. We report the exact numbers of iterations and seconds before reaching a stopping criterion for both methods.

The results of the two tracks of experiments for support size $m = 6, 60$ by LP, the modified B-ADMM (R1 and R2 as explained in Section III-B) and IBP, are presented in Table I.

---

[3]It is because the entropic regularization term is not considered in the update of support points.

| Solved Wasserstein barycenter with a pre-fixed support | | | |
| $(m = 6)/(m = 60)$ | | | |
method	iterations	seconds	obj.
full LP	NA	-	834.1 / 709.6
Our approach (R1)	500 / 800	0.78 / 13.0	838.6 / 713.4
Our approach (R2)	400 / 700	0.58 / 11.8	835.5 / 712.3
IBP[19] $\varepsilon_0 = 0.2$	150 / 40	0.06 / 0.11	978.3 / 1073.5
IBP $\varepsilon_0 = 0.1$	620 / 80	0.16 / 0.23	965.9 / 1051.6
IBP $\varepsilon_0 = 0.02$	6,640 / 760	1.50 / 1.49	957.1 / 1039.4
IBP $\varepsilon_0 = 0.01$	17,420 / 9,460	4.17 / 17.0	960.2 / 2343.8*
IBP $\varepsilon_0 = 0.005$	36,270 / 5,110	8.88 / 9.83	1345.1* / 7112.2*

| Solved Wasserstein barycenter with an optimized support | | | |
| $(m = 6)/(m = 60)$ | | | |
method	iterations	seconds	obj.
full LP	20	-	717.8 / 692.3
Our approach (R1)	2,000	2.91 / 31.1	723.3 / 692.6
Our approach (R2)	2,000	3.02 / 32.2	722.7 / 692.5
IBP $\varepsilon_0 = 0.2$	190 / 60	0.06 / 0.19	733.7 / 703.5
IBP $\varepsilon_0 = 0.1$	730 / 130	0.22 / 0.31	734.4 / 699.5
IBP $\varepsilon_0 = 0.02$	11,860 / 1,590	2.67 / 2.73	734.9 / 705.5
IBP $\varepsilon_0 = 0.01$	33,940 / 5,130	7.71 / 9.01	734.9 / 708.3
IBP $\varepsilon_0 = 0.005$	69,860 / 16,910	16.5 / 30.87	736.1 / 708.6

The performance is measured by the value achieved for the objective (a.k.a. distortion) function. The results show that in both tracks, when dealing with sparse support distribution data, B-ADMM achieves lower distortion than IBP does, and furthermore, B-ADMM is quite close to LP. The gap between B-ADMM and LP is smaller than the gap between IBP and B-ADMM. On the other hand, when ϵ_0 is relatively large, IBP can be much faster. There is no tuning of hyper-parameters in B-ADMM. For IBP, ε_0 influences the result, but not by too much as long as it is not too small. Considering the fast speed at large ε_0, we may favor relatively large ε_0 when applying IBP. For the first track experiments, the objective function obtained by IBP is considerably larger than that B-ADMM obtains. We could not push the objective function by IBP to the same level of B-ADMM by letting $\varepsilon_0 \to 0$. At $\varepsilon_0 = 0.01, 0.005$, because double-precision overflow occurs, triggering the thresholding trick. The IBP results with thresholding triggered are marked by star (*) in Table I. These results are actually much worse than the others, an observation consistent with the incorrect coupling weights obtained when this trick is applied in the first pilot study. For the second track experiments, B-ADMM still achieves lower values of the objective function, but the difference from IBP is not as remarkable.

In comparison to the full LP approach, the modified B-ADMM does not yield the exact local minimum. This result reminds us that the modified B-ADMM is still an approximation method, and it cannot fully replace subgradient descent based methods that minimize the objective to a true local minimum.

V. ALGORITHM INITIALIZATION AND IMPLEMENTATION

In this section, we explain some specifics in the implementation of the algorithms, such as initialization, warm-start in optimization, measures for further speed-up, and the method for parallelization.

The number of support vectors in the centroid distribution, denoted by m, is set to the average number of support vectors in the distributions in the corresponding cluster. To initialize a centroid, we select randomly a distribution with at least m support vectors from the cluster. If the number of support vectors in the distribution is larger than m, we will merge recursively a pair of support vectors according to an optimal criterion until the support size reaches m, similar to the process used in linkage clustering. Consider a chosen distribution $P = \{(w_1, x_1), ..., (w_m, x_m)\}$. We merge x_i and x_j to $\bar{x} = (w_i x_i + w_j x_j)/\bar{w}$, where $\bar{w} = w_i + w_j$ is the new weight for $\bar{x}$, if (i, j) solves

$$\min_{i,j} w_i w_j \|x_i - x_j\|^2 / (w_i + w_j) . \tag{27}$$

Let the new distribution after one merge be P'. It is sensible to minimize the Wasserstein distance between P and P' to decide which support vectors to merge. We note that

$$W^2(P, P') \leq w_i \|x_i - \bar{x}\|^2 + w_i \|x_j - \bar{x}\|^2 .$$

This upper bound is obtained by the transport mapping x_i and x_j exclusively to $\bar{x}$ and the other support vectors to themselves. To simplify computation, we instead minimize the upper bound, which is achieved by the $\bar{x}$ given above and by the pair (i, j) specified in Eq. (27).

The B-ADMM method requires an initialization for $\Pi^{(k,2)}$, where k is the index for every cluster member, before starting the inner loops (see Algorithm 4). We use a warm-start for $\Pi^{(k,2)}$. Specifically, for the members whose cluster labels are unchanged after the most recent label assignment, $\Pi^{(k,2)}$ is initialized by its value solved (and cached) in the previous round (with respect to the outer loop) . Otherwise, we initialize $\Pi^{(k,2)} = (\pi_{i,j}^{(k,2)})$, $i = 1, ..., m$, $j = 1, ..., m_k$ by $\pi_{i,j}^{(k,2),0} := w_i w_j^{(k)}$. This scheme of initialization is also applied in the first round of iteration when class labels are assigned for the first time and there exists no previous solution for this parameter.

At the relabeling step (*i.e.*, to assign data points to centroids after centroids are updated), we need to compute $\bar{N}K$ Wasserstein distances, where $\bar{N}$ is the data size and K is the number of centroids. This part of the computation, usually negligible in the original D2-clustering, is a sizable cost in our new algorithms. To further boost the scalability, we employ the technique of [35] to skip unnecessary distance calculation by exploiting the triangle inequality of a metric.

In our implementation, we use a fixed number of iterations ϵ_i for all inner loops for simplicity. Obtaining highly accurate result for the inner loop is not crucial because the partition will be changed by the outer loop. For B-ADMM, we found that setting ϵ_i to tens or a hundred suffices. For subgradient descent and ADMM, an even smaller ϵ_i is sufficient, *e.g.*, around or below ten. The number of iterations of the outer loop ϵ_o is not fixed, but rather is adaptively determined when a certain termination criterion is met.

With an efficient serial implementation, our algorithms can be deployed to handle moderate scale data on a single PC. We also implemented their parallel versions which are scalable to a large data size and a large number of clusters. We use the commercial solver provided by Mosek,[4] which is among the fastest LP/QP solvers available. In particular, Mosek provides optimized simplex solver for transportation problems that fits our needs well.

The algorithms we have developed here are all readily parallelizable by adopting the Allreduce framework in MPI. In our implementation, we divide data evenly into trunks and process each trunk at one processor. Each trunk of data stay at the same processor during the whole program. We can parallelize the algorithms simply by dividing the data because in the centroid update step, the computation comprises mainly separate per data point optimization problems. The main communication cost is on synchronizing the update for centroids by the inner loop. The synchronization time with equally partitioned data is negligible.

We experimented with discrete distributions over a vector space endowed with the Euclidean distance as well as over a symbolic set. In the second case, a symbol-to-symbol distance matrix is provided. When applying D2-clustering to such data, the step of updating the support vectors can be skipped since the set of symbols is fixed. In some datasets, the support vectors in the distributions locate only on a pre-given grid. We can save memory in the implementation by storing the indices of the grid points rather than the direct vector values.

Although we assume each instance is a single distribution in all the previous discussion, it is straightforward to generalize to the case when an instance is an array of distributions (indeed that is the original setup of D2-clustering in [9]). For instance, a protein sequence can be characterized by three histograms over respectively amino acids, dipeptides, and tripeptides. This extension causes little extra work in the algorithms. When updating the cluster centroids, the distributions of different modalities can be processed separately, while in the update of cluster labels, the sum of squared Wasserstein distances for all the distributions is used as the combined distance.

VI. COMPLEXITY AND PERFORMANCE COMPARISONS

Recall some notations: $\bar{N}$ is the data size (total number of distributions to be clustered); d is the dimension of the support vectors; K is the number of clusters; and ϵ_i or ϵ_o is the number of iterations in the inner or outer loop. Let $\bar{m}$ be the average number of support vectors in each distribution in the training set and m be the number of support vectors in each centroid distribution ($\bar{m} = m$ in our setup).

In our implementation, to reduce the time of dynamic memory allocation, we retain the memory for the matching weights between the support vectors of each distribution and its corresponding centroid. Hence, the memory allocation is of order $O(\bar{N}\bar{m}m) + O(d\bar{N}\bar{m} + dKm)$.

For computational complexity, first consider the time for assigning cluster labels in the outer loop. Without the acceleration yielded from the triangle inequality, the com-

plexity is $O(\epsilon_o \bar{N}Kl(\bar{m}m, d))$, where $l(\bar{m}m, d)$ is the average time to solve the Wasserstein distance between distributions on a d dimensional metric space. Empirically, we found that by omitting unnecessary distance computation via the triangle inequality, the complexity is reduced roughly to $O(\epsilon_o(\bar{N} + K^2)l(\bar{m}m, d))$. For the centroid update step, the time complexity of the serial version of the ADMM method is $O(\epsilon_o\epsilon_i\bar{N}md) + O(T_{admm} \cdot \epsilon_o\epsilon_i\bar{N}q(\bar{m}m, d))$, where $q(m'm, d)$ is the average time to solve QPs (Eq. (9)). The complexity of the serial B-ADMM is $O(\epsilon_o\epsilon_i\bar{N}md/\tau) + O(\epsilon_o\epsilon_i\bar{N}\bar{m}m)$. Note that in the serial algorithms, the complexity for updating centroids does not depend on K, but only on data size $\bar{N}$. For the parallel versions of the algorithms, the communication load per iteration in the inner loop is $O(T_{admm}Kmd)$ for ADMM and $O(Km(1 + d/\tau))$ for the B-ADMM.

Both analytical and empirical studies (Section VII-B) show that the ADMM algorithm is significantly slower than the other two when the data size is large due to the many constrained QP sub-problems required. Although the theoretical properties of convergence are better understood for ADMM, our experiments show that B-ADMM performs well consistently in terms of both convergence and the quality of the clustering results.

Although the preference for B-ADMM is experimentally validated, given the lack of strong theoretical results on its convergence, it is not clear-cut that B-ADMM can always replace the alternatives. We were thus motivated to develop the subgradient descent (in our supplement) and standard ADMM algorithms to serve at least as yardsticks for comparison. We provide the following guidelines on the usage of the algorithms.

- We recommend the modified B-ADMM as the default data processing pipeline for its scalability, stability, and fast performance. Large memory is assumed to be available under the default setting.
- It is known that ADMM type methods can approach the optimal solution quickly at the beginning when the current solution is far from the optimum while the convergence slows down substantially when the solution is in the proximity of the optimum. Because we always reset the Lagrangian multipliers in B-ADMM at the beginning of every round of the inner loop and a fixed number of iterations are performed within the loop, our scheme does not pursue aggressively high accuracy for the resulting centroids at every round. However, if the need arises for highly accurate centroids, we recommend the subgradient descent method that takes as initialization the centroids first obtained by B-ADMM.

VII. EXPERIMENTS

We have conducted experiments to examine the convergence of the algorithms, stability, computational/memory efficiency and scalability of the algorithms, and quality of the clustering results on large data from several domains.

Table II lists the basic information about the datasets used in our experiments. For the synthetic data, the support vectors are generated by sampling from a multivariate normal distribution

---

[4]https://www.mosek.com

Table II
DATASETS IN THE EXPERIMENTS. $\bar{N}$: DATA SIZE, d: DIMENSION OF THE
SUPPORT VECTORS ("SYMB" FOR SYMBOLIC DATA), m: NUMBER OF
SUPPORT VECTORS IN A CENTROID, K: MAXIMUM NUMBER OF CLUSTERS
TESTED. AN ENTRY WITH THE SAME VALUE AS IN THE PREVIOUS ROW IS
INDICATED BY "-".

Data	$\bar{N}$	d	m	K
synthetic	2,560,000	≥ 16	≥ 32	256
image color	5,000	3	8	10
image texture	-	-	-	-
USPS digits	11,000	2	80	360
BBC news abstract	2,225	300	16	15
Wiki events abstract	1,983	400	16	100
20newsgroups GV	18,774	300	64	40
20newsgroups WV	-	400	100	-

and then adding a heavy-tailed noise from the student's t-distribution. The probabilities on the support vectors are perturbed and normalized samples from Dirichlet distribution with symmetric prior. We omit details due to lack of space. The synthetic data are only used to study the scalability of the algorithms. The image color or texture data are created from crawled general-purpose photographs. Local color or texture features around each pixel in an image are clustered (*i.e.* quantized) to yield color or texture distributions. The USPS digit images are treated as normalized histograms over the pixel locations covered by the digits, where the support vector is the 2D coordinate of a pixel and the weight corresponds to pixel intensity. For the *20newsgroups* data, we use the recommended "bydate" MATLAB version which includes 18,774 documents and 61,188 unique words. The two datasets, "20 newsgroup GV" and "20newsgroup WV" are created by characterizing the documents in different ways. The "BBC news abstract" and "Wiki events abstract" datasets are truncated versions of two document collections [36], [37]. These two sets of short documents retain only the title and the first sentence of each original post. The purpose of using these severely cut documents is to investigate a more challenging setting for existing document or sentence analysis methods, where semantically related sentences are less likely to share the exact same words. For example, "NASA revealed its ambitions that humans can set foot on Mars" and "US is planning to send American astronauts to Red Planet" describe the same event. More details on the data are referred to Section VII-C.

A. Convergence Analysis

We empirically test the convergence and stability of the three approaches: modified B-ADMM, ADMM, and subgradient descent method, based on their sequential versions implemented in the C programming language. Four datasets are used in the test: protein sequence 1-gram, 3-gram data, and the image color and texture data. In summary, the experiments show that the modified B-ADMM method has achieved the best numerical stability with respect to hyper-parameters while keeping a comparable convergence rate as the subgradient descent method in terms of CPU time. To conserve space, detailed results on the study of stability are provided in Appendix B. Despite of its popularity in large-scale machine

learning problems, by lifting $\bar{N}$ LPs to $\bar{N}$ QPs, the ADMM approach is much slower on large datasets than the other two approaches are.

We examine the convergence property of the B-ADMM approach for computing the centroid of a single cluster (the inner loop). In this experiment, a subset of image color or texture data with size $2,000$ is used. For the two protein sequence datasets, the whole sets are used. Fig. 2 shows the convergence analysis results on the four datasets. The vertical axis in the plots in the first row of Fig. 2 is the objective function of B-ADMM, given in Eq. (15), but not the original objective function of clustering in Eq. (2). The runtime is based on a single thread with 2.6 GHz Intel Core i7. The plots reveal two characteristics about the B-ADMM approach: 1) The algorithm achieves consistent and comparable convergence rate under a wide range of values for the hyper-parameter $\rho_0 \in \{0.5, 1.0, 2.0, 4.0, 8.0, 16.0\}$ and is numerically stable; 2) The effect of the hyper-parameter on the decreasing ratio of the dual and primal residuals follows similar patterns across the datasets.

B. Efficiency and Scalability

We now study the computational/memory efficiency and scalability of AD2-clustering with the B-ADMM algorithm embedded for computing cluster centroids. We use the synthetic data that allow easy control over data size and other parameters in order to test their effects on the computational and memory load (*i.e.*, workload) of the algorithm. We study the *scalability* of our parallel implementation on a cluster computer with distributed memory. Scalability here refers to the ability of a parallel system to utilize an increasing number of processors.

AD2-clustering can be both CPU-bound and memory-bound. Based on the observations from the above serial experiments, we conducted three sets of experiments to test the scalability of AD2-clustering in a multi-core environment, specifically, strong scaling efficiency, weak scaling efficiency with respect to $\bar{N}$ or m. The configuration ensures that each iteration finishes within one hour and the memory of a typical computer cluster is sufficient.

Table III
SCALING EFFICIENCY OF AD2-CLUSTERING IN PARALLEL
IMPLEMENTATION.

# processors	32	64	128	256	512
SSE (%)	93.9	93.4	92.9	84.8	84.1
WSE on $\bar{N}$ (%)	99	94.8	95.7	93.3	93.2
WSE on m (%)	96.6	89.4	83.5	79.0	-

Strong scaling efficiency (SSE) is about the speed-up gained from using more and more processors when the problem is fixed in size. Ideally, the runtime on parallel CPUs is the time on a single thread divided by the number of CPUs. In practice, such a reduction in time cannot be fully achieved due to communication between CPUs and time for synchronization. We thus measure SSE by the ratio between the ideal and the actual amount of time. We chose a moderate size problem

Figure 2. Convergence analysis of the B-ADMM method for computing a single centroid based on four datasets. First row: objective function of B-ADMM based centroid computation with respect to CPU time; Second row: the trajectory of dual residual vs. primal residual (in the negative log scale).

that can fit in the memory of a single machine (50GB): $\bar{N} = 250000$, $d = 16$, $m = 64$, $k = 16$. Table III shows the SSE values with the number of processors ranging from 32 to 512. The results show that AD2-clustering scales well in SSE when the number of processors is up to hundreds.

Weak scaling efficiency (WSE) measures how stable the real computation time can be when proportionally more processors are used as the size of the problem grows. We compute WSE with respect to both $\bar{N}$ and m. Let np be the number of processors. For WSE on $\bar{N}$, we set $\bar{N} = 5000 \cdot np$, $d = 64$, $m = 64$, and $K = 64$ on each processor. The per-node memory is roughly $1GB$. For WSE on m, we set $\bar{N} = 10000$, $K = 64$, $d = 64$, and $m = 32 \cdot \sqrt{np}$. Table III shows the values of WSE on $\bar{N}$ and m. We can see that AD2-clustering also has good weak scalability, making it suitable for handling large scale data. In summary, our proposed method can be effectively accelerated with an increasing number of CPUs.

C. Quality of Clustering Results

Handwritten Digits: We conducted two experiments to evaluate the results of AD2-clustering on USPS data, which contain 1100×10 instances ($1,100$ per class). First, we cluster the images at $K = 30, 60, 120, 240$ and report in Figure 3a the homogeneity versus completeness [38] of the obtained clustering results. We set K to large values because clustering performed on such image data is often for the purpose of quantization where the number of clusters is much larger than the number of classes. In this case, homogeneity and completeness are more meaningful measures than the others used in the literature (several of which will be used later for the next two datasets). Roughly speaking, completeness measures how likely members of the same true class fall into the same cluster, while homogeneity measures how likely members of the same cluster belong to the same true class. By construction, the two

measures have to be traded off. We compared our method with Kmeans++ [39]. For this dataset, we found that Kmeans++, with more careful initialization, yields better results than the standard K-means. Their difference on the other datasets is negligible. Figure 3a shows that AD2-clustering obviously outperforms Kmeans++ cross K's.

Secondly, we tested AD2-clustering for quantization with the existence of noise. In this experiment, we corrupted each sample by "blankout"—randomly deleting a percentage of pixels occupied by the digit (setting to zero the weights of the corresponding bins), as is done in [40]. Then each class is randomly split into 800/300 training and test samples. Clustering is performed on the 8000 training samples; and a class label is assigned to each cluster by majority vote. In the testing phase, classifying an instance entails locating its nearest centroid and then assigning the class label of the corresponding cluster. The test classification error rates with respect to K and the blankout rate are plotted in Figure 3b. The comparison with Kmeans++ demonstrates that AD2-clustering performs consistently better, and the margin is remarkable when the number of clusters is large and the blankout rate is high.

Documents as Bags of Word-vectors: The idea of treating each document as a bag of vectors has been explored in previous work where a nearest neighbor classifier is constructed using Wasserstein distance [41], [42]. One advantage of the Wasserstein distance is to account for the many-to-many mapping between two sets of words. However, clustering based on Wasserstein distance, especially the use of Wasserstein barycenter, has not been explored in the literature of document analysis. We have designed two kinds of experiments using different document data to assess the power of AD2-clustering.

To demonstrate the robustness of D2-clustering across different word embedding spaces, we use 20newsgroups processed based on two pre-trained word embedding models.

(a) Homogeneity vs. completeness   (b) Test error rate vs. blankout rate
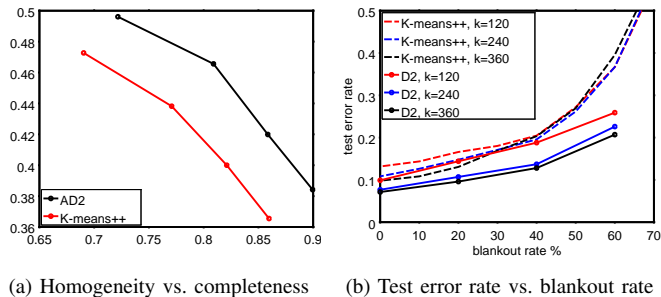
Figure 3. Comparisons between Kmeans++ and AD2-clustering on USPS dataset. We empirically set the number of support vectors in the centroids $m = 80(1 - blankout\_rate)$.

Table IV
COMPARE CLUSTERING RESULTS OF AD2-CLUSTERING AND SEVERAL BASELINE METHODS USING TWO VERSIONS OF BAG-OF-WORDS REPRESENTATION FOR THE 20NEWSGROUPS DATA. TOP PANEL: THE DATA ARE EXTRACTED USING THE GV VOCABULARY; BOTTOM PANEL: WV VOCABULARY. AD2-CLUSTERING IS PERFORMED ONCE ON 16 CORES WITH LESS THAN 5GB MEMORY. RUN-TIMES OF AD2-CLUSTERING ARE REPORTED (ALONG WITH THE TOTAL NUMBER OF ITERATIONS).

GV Vocab.	tf-idf	LDA	LDA naïve	Avg. vector	AD2	AD2	AD2
K	40	20	20	30	20	30	40
AMI	0.447	0.326	0.329	0.360	0.418	**0.461**	0.446
ARI	0.151	0.160	0.187	0.198	0.260	0.281	**0.284**
hours					5.8	7.5	10.4
# iter.					44	45	61

WV Vocab.	tf-idf	LDA	LDA naïve	Avg. vector	AD2	AD2	AD2
K	20	25	20	20	20	30	40
AMI	0.432	0.336	0.345	0.398	0.476	**0.477**	0.455
ARI	0.146	0.164	0.183	0.212	**0.289**	0.274	0.278
hours					10.0	11.3	17.1
# iter.					28	29	36

We pre-processed the dataset by two steps: remove stop words and remove other words that do not belong to a pre-selected background vocabulary. In particular, two background vocabularies are tested: English Gigaword-5 (denoted by GV) [43] and a Wikipedia dump with minimum word count of 10 (denoted by WV) [44]. Omitting details due to lack of space, we validated that under the GV or WV vocabulary information relevant to the class identities of the documents is almost intact. The words in a document are then mapped to a vector space. For GV vocabulary, the Glove mapping to a vector space of dimension 300 is used [43], while for WV, the Skip-gram model is used to train a mapping space of dimension 400 [44]. The frequencies on the words are adjusted by the popular scheme of tf-idf. The number of different words in a document is bounded by m (its value in Table II). If a document has more than m different words, some words are merged into hyper-words recursively until reaching m, in the same manner as the greedy merging scheme used in centroid initialization described in Section V.

We evaluate the clustering performance by two widely used metrics: AMI [45] and ARI [46], [47]. The baseline methods for comparison include K-means on the raw tf-idf word frequencies, K-means on the LDA topic proportional vectors [48] (the number of LDA topics is chosen from $\{40, 60, 80, 100\}$), K-means on the average word vectors, and the naïve way of treating the 20 LDA topics as clusters. For each baseline method, we tested the number of clusters $K \in \{10, 15, 20, 25, 30, 40\}$ and report only the best performance for the baseline methods in Table IV, while for AD2-clustering, $K = 20, 30, 40$ are reported. Under any given setup of a baseline method, multiple runs were conducted with different initialization and the median value of the results was taken. The experimental results show that AD2-clustering achieves the best performance on the two datasets according to both AMI and ARI. Comparing with most baseline methods, the boost in performance by AD2-clustering is substantial. Furthermore, we also vary m in the experimental setup of AD2-clustering. At $m = 1$, our method is exactly equivalent to K-means of the distribution means. We increased m empirically to see how the results improve with a larger m. We did not observe any further performance improvement for $m \geq 64$.

We note that the high competitiveness of AD2-clustering can be credited to (1) a reasonable word embedding model and (2) the bag-of-words model. When the occurrence of words is sparse across documents, the semantic relatedness between different words and their compositions in a document plays a critical role in measuring the document similarity.

In our next experiment, we study AD2-clustering for short documents, a challenging setting for almost all existing methods based on the bag-of-words representation. The results show that the performance boost of AD2-clustering is also substantial. We use two datasets, one is called "BBC news abstract" and the other "Wiki events abstract". Each document is represented by only the title and the first sentence from a news article or an event description. Their word embedding models are same as the one used by the "WV" version in our previous experiment. The "BBC news" dataset contains five news categories, and "Wiki events" dataset contains 54 events. In the supplement materials, the raw data we used are provided. Clustering such short documents is more challenging due to the sparse nature of word occurrences. As shown by Table V, in terms of generating clusters coherent with the labeled categories or events, methods which leverage either the bag-of-words model or the word embedding model (but not both) are outperformed by AD2-clustering which exploits both. In addition, AD2-clustering is fast for those sparse support discrete distribution data. It takes only several minutes to finish the clustering in an 8-core machine.

To quantify the gain from employing an effective word embedding model, we also applied AD2-clustering to a random word embedding model, where a vector sampled from a multivariate Gaussian with 300 dimensions is used to represent a word in vocabulary. We found that the results are much worse than those reported in Table V for AD2-clustering. The best AMI for "BBC news abstract" is 0.187 and the best AMI for "Wiki events abstract" is 0.369, comparing respectively with 0.759 and 0.545 obtained from a carefully trained word embedding model.

Table V
BEST AMIs ACHIEVED BY DIFFERENT METHODS ON THE TWO SHORT
DOCUMENT DATASETS. NMF DENOTES FOR THE NON-NEGATIVE MATRIX
FACTORIZATION METHOD.

	Tf-idf	LDA	NMF	Avg. vector	AD2
BBC news abstract	0.376	0.151	0.537	0.753	0.759
Wiki events abstract	0.448	0.280	0.395	0.312	0.545

VIII. CONCLUSIONS AND FUTURE WORK

Two first-order methods for clustering discrete distributions under the Wasserstein distance have been developed and empirically compared in terms of speed, stability, and convergence. The experiments identified the modified B-ADMM method as most preferable for D2-clustering in an overall sense under common scenarios. The resulting clustering tool is easy to use, requiring no tuning of optimization parameters. We applied the tool to several real-world datasets and evaluated the quality of the clustering results by comparing with the ground truth class labels. Experiments show that the accelerated D2-clustering often clearly outperforms other widely used methods in the respective domain of the data.

One limitation of our current work is that the theoretical convergence property of the modified B-ADMM algorithm is not well understood. In addition, to expedite D2-clustering, we have focused on scalability with respect to data size rather than the number of support points per distribution. Thus there is room for further improvement on the efficiency of D2-clustering. Given that the algorithms explored in this paper have different strengths, it is of interest to investigate in the future whether they can be integrated to yield even stronger algorithms.
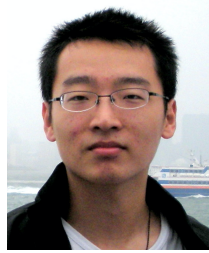
ACKNOWLEDGMENT

REFERENCES

[1] S. T. Rachev, "The Monge-Kantorovich mass transference problem and its stochastic applications," *Theory of Probability and Its Applications*, vol. 29, pp. 647–676, 1984.
[2] C. Villani, *Topics in Optimal Transportation*. American Mathematical Soc., 2003, no. 58.
[3] C. L. Mallows, "A note on asymptotic joint normality," *Annals of Mathematical Statistics*, vol. 43, no. 2, pp. 508–515, 1972.
[4] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *Int. J. Comp. Vision (IJCV)*, vol. 40, no. 2, pp. 99–121, 2000.
[5] D. Schuhmacher, B.-T. Vo, and B.-N. Vo, "A consistent metric for performance evaluation of multi-object filters," *IEEE Trans. Signal Processing*, vol. 56, no. 8, pp. 3447–3457, 2008.
[6] M. Baum, P. Willett, and U. D. Hanebeck, "On Wasserstein barycenters and MMOSPA estimation," *IEEE Signal Processing Letters*, vol. 22, no. 10, pp. 1511–1515, 2015.
[7] C. Villani, *Optimal Transport: Old and New*. Springer Science & Business Media, 2008, vol. 338.
[8] J. B. Orlin, "A faster strongly polynomial minimum cost flow algorithm," *Operations research*, vol. 41, no. 2, pp. 338–350, 1993.
[9] J. Li and J. Z. Wang, "Real-time computerized annotation of pictures," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 30, no. 6, pp. 985–1002, 2008.
[10] O. Pele and M. Werman, "Fast and robust earth mover's distances," in *Proc. Int. Conf. Comp. Vision (ICCV)*. IEEE, 2009, pp. 460–467.
[11] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 2292–2300.
[12] M. Cuturi and A. Doucet, "Fast computation of Wasserstein barycenters," in *Proc. Int. Conf. Machine Learning (ICML)*, 2014, pp. 685–693.
[13] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *The Journal of Machine Learning Research*, vol. 6, pp. 1705–1749, 2005.
[14] J. Ye and J. Li, "Scaling up discrete distribution clustering using admm," in *Proc. Int. Conf. Image Processing (ICIP)*. IEEE, 2014, pp. 5267–5271.
[15] Y. Zhang, J. Z. Wang, and J. Li, "Parallel massive clustering of discrete distributions," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, no. 4, pp. 49:1–49:24, Jun 2015.
[16] H. Wang and A. Banerjee, "Bregman alternating direction method of multipliers," in *Advances in Neural Information Processing Systems (NIPS)*, 2014, pp. 2816–2824.
[17] M. Agueh and G. Carlier, "Barycenters in the Wasserstein space," *SIAM J. Math. Analysis*, vol. 43, no. 2, pp. 904–924, 2011.
[18] E. Anderes, S. Borgwardt, and J. Miller, "Discrete wasserstein barycenters: Optimal transport for discrete data," *arXiv preprint arXiv:1507.07218*, 2015.
[19] J.-D. Benamou, G. Carlier, M. Cuturi, L. Nenna, and G. Peyré, "Iterative Bregman projections for regularized transportation problems," *SIAM J. Sci. Comp. (SJSC)*, vol. 37, no. 2, pp. A1111–A1138, 2015.
[20] G. Carlier, A. Oberman, and E. Oudet, "Numerical methods for matching for teams and wasserstein barycenters," *arXiv preprint arXiv:1411.3602*, 2014.
[21] M. Cuturi, G. Peyré, and A. Rolet, "A smoothed dual approach for variational wasserstein problems," *arXiv preprint arXiv:1503.02533*, 2015.
[22] J. Solomon, F. de Goes, G. Peyré, M. Cuturi, A. Butscher, A. Nguyen, T. Du, and L. Guibas, "Convolutional wasserstein distances: Efficient optimal transportation on geometric domains," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 66:1–66:11, Jul. 2015.
[23] J. Rabin, G. Peyré, J. Delon, and M. Bernot, "Wasserstein barycenter and its application to texture mixing," in *Scale Space and Variational Methods in Computer Vision*. Springer, 2011, pp. 435–446.
[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.
[25] Y. Tang, L. H. U, Y. Cai, N. Mamoulis, and R. Cheng, "Earth mover's distance based similarity search at scale," *Proc. VLDB Endow.*, vol. 7, no. 4, pp. 313–324, Dec. 2013.
[26] L. M. Bregman, "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 3, pp. 200–217, 1967.
[27] Y. Censor and S. A. Zenios, "Proximal minimization algorithm with d-functions," *Journal of Optimization Theory and Applications*, vol. 73, no. 3, pp. 451–464, 1992.
[28] J. Eckstein, "Nonlinear proximal point algorithms using bregman functions, with applications to convex programming," *Mathematics of Operations Research*, vol. 18, no. 1, pp. 202–226, 1993.
[29] A. Srivastava, I. Jermyn, and S. Joshi, "Riemannian analysis of probability density functions with applications in vision," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.
[30] A. Nemirovski, "Prox-method with rate of convergence o (1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems," *SIAM Journal on Optimization*, vol. 15, no. 1, pp. 229–251, 2004.
[31] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 31, no. 3, pp. 167–175, 2003.
[32] S. Bubeck *et al.*, "Convex optimization: Algorithms and complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.

[33] L. Chizat, G. Peyré, B. Schmitzer, and F.-X. Vialard, "Scaling algorithms for unbalanced transport problems," *arXiv preprint arXiv:1607.05816*, Jul. 2016.

[34] B. Schmitzer, "Stabilized sparse scaling algorithms for entropy regularized transport problems," *arXiv preprint arXiv:1610.06519*, Oct. 2016.

[35] C. Elkan, "Using the triangle inequality to accelerate k-means," in *Proc. Int. Conf. Machine Learning (ICML)*, vol. 3, 2003, pp. 147–153.

[36] D. Greene and P. Cunningham, "Practical solutions to the problem of diagonal dominance in kernel document clustering," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 377–384.

[37] Z. Wu, C. Liang, and C. L. Giles, "Storybase: Towards building a knowledge base for news events," *ACL-IJCNLP 2015*, p. 133, 2015.

[38] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure." in *Proc. Joint Conf. EMNLP-CoNLL*, vol. 7, 2007, pp. 410–420.

[39] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[40] A. Globerson and S. Roweis, "Nightmare at test time: robust learning by feature deletion," in *Proc. Int. conf. Machine learning (ICML)*. ACM, 2006, pp. 353–360.

[41] X. Wan, "A novel document similarity measure based on earth movers distance," *Information Sciences*, vol. 177, no. 18, pp. 3718–3730, 2007.

[42] M. J. Kusner, Y. Sun, N. I. Kolkin, Nicholas, and K. Q. Weinberger, "From word embeddings to document distances," in *Proc. Int. Conf. Machine Learning (ICML)*, 2015, pp. 957–966.

[43] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 2014, pp. 1532–1543.

[44] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 3111–3119.

[45] N. X. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance," *The Journal of Machine Learning Research (JMLR)*, vol. 11, pp. 2837–2854, 2010.

[46] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. 846–850, 1971.

[47] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, no. 1, pp. 193–218, 1985.

[48] M. Hoffman, F. R. Bach, and D. M. Blei, "Online learning for latent dirichlet allocation," in *Advances in Neural Information Processing Systems (NIPS)*, 2010, pp. 856–864.

Panruo Wu received his BS degree in Mathematics from the University of Science and Technology of China in 2011. He is now a PhD candidate in the Supercomputing Laboratory (SuperLab) at the University of California, Riverside. He is interested in high performance computing, algorithm-based fault tolerance, and power and energy efficient computing. Wu received a student travel grant from IPDPS'14 and he was a student volunteer for ICS'15 and SC'16. He did an internship at Oak Ridge National Laboratory in 2013, an internship at Amazon in 2014, and an internship at Los Alamos National Laboratory in 2015-2016.

James Z. Wang is a Professor of Information Sciences and Technology at The Pennsylvania State University. He received the bachelor's degree in mathematics and computer science *summa cum laude* from the University of Minnesota, and the MS degree in mathematics, the MS degree in computer science and the PhD degree in medical information sciences, all from Stanford University. His research interests include computational aesthetics and emotions, automatic image tagging, image retrieval, and computerized analysis of paintings. He was a visiting professor at the Robotics Institute at Carnegie Mellon University (2007-2008), a lead special section guest editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence (2008), and a program manager at the Office of the Director of the National Science Foundation (2011-2012). He was a recipient of a National Science Foundation Career award (2004).
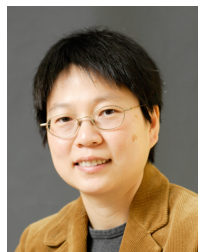
Jianbo Ye received his BS degree in Mathematics from the University of Science and Technology of China in 2011. He worked as a research postgraduate at The University of Hong Kong, from 2011 to 2012, and as a research intern at Intel Labs, China in 2013. He is currently a PhD candidate and Research Assistant at the College of Information Sciences and Technology, The Pennsylvania State University. His research interests include statistical modeling and learning, numerical optimization and method, and human computation for interdisciplinary studies.

Jia Li is a Professor of Statistics at The Pennsylvania State University. She received the MS degree in Electrical Engineering, the MS degree in Statistics, and the PhD degree in Electrical Engineering, all from Stanford University. She worked as a Program Director in the Division of Mathematical Sciences at the National Science Foundation from 2011 to 2013, a Visiting Scientist at Google Labs in Pittsburgh from 2007 to 2008, a researcher at the Xerox Palo Alto Research Center from 1999 to 2000, and a Research Associate in the Computer Science Department at Stanford University in 1999. Her research interests include statistical modeling and learning, data mining, computational biology, image processing, and image annotation and retrieval.

APPENDIX

A. Subgradient Descent Method

We describe a subgradient descent approach in this section for the purpose of experimental comparison.

Eq. (4) can be casted as multi-level optimization by treating $\boldsymbol{w}$ as policies/parameters and Π as variables. Express $W^2(P, P^{(k)})$, the squared distance between P and $P^{(k)}$, as a function of $\boldsymbol{w}$ denoted by $\tilde{W}(\boldsymbol{w})^{(k)}$. $\tilde{W}(\boldsymbol{w})^{(k)}$ is the solution to a designed optimization, but has no closed form. Let $\tilde{W}(\boldsymbol{w}) = \frac{1}{N} \sum_{k=1}^{N} \tilde{W}(\boldsymbol{w})^{(k)}$, where N is the number of instances in the cluster. Note that Eq. (4) minimizes $\tilde{W}(\boldsymbol{w})$ up to a constant multiplier. The minimization of $\tilde{W}$ with respect to $\boldsymbol{w}$ is thus a bi-level optimization problem. In the special case when the designed problem is LP and the parameters only appear on the right hand side (RHS) of the constraints or are linear in the objective, the subgradient, specifically $\nabla \tilde{W}(\boldsymbol{w})^{(k)}$ in our problem, can be solved via the same (dual) LP.

Again, we consider the routine that alternates the updates of $\{x_i\}$ and $\{\pi_{i,j}\}^{(k)}$ iteratively. With fixed $\{x_i\}$, updating $\{\pi_{i,j}\}^{(k)}$ involves solving N LP (1). With LP (1) solved, we can write $\nabla \tilde{W}(\boldsymbol{w})^{(k)}$ in closed form, which is given by the set of dual variables $\{\lambda_i^{(k)}\}_{i=1}^m$ corresponding to $\left\{ \sum_{j=1}^{m_k} \pi_{i,j} = w_i, i = 1, ..., m \right\}$. Because $\{w_i\}$ must reside in the facets defined by Δ_m, the projected subgradient $\nabla \tilde{W}(\boldsymbol{w})^{(k)}$ is given by

$$\nabla \tilde{W}(\boldsymbol{w})^{(k)} = (\lambda_1^{(k)}, \ldots, \lambda_m^{(k)}) - \left(\sum_{i=1}^{m} \lambda_i^{(k)} \right) (1, \ldots, 1) . \tag{28}$$

In the standard method of gradient descent, a line search is conducted in each iteration to determine the step-size for a strictly descending update. Line search however is computationally intensive for our problem because Eq. (28) requires solving a LP and we need Eq. (28) sweeping over $k = 1, ..., N$. In machine learning algorithms, one practice to avoid expensive line search is by using a pre-determined step-size, which is allowed to vary across iterations. We adopt this approach here.

One issue resulting from a pre-determined step-size is that the updated weight vector $\boldsymbol{w}$ may have negative components. We overcome this numerical instability by the technique of re-parametrization. Let

$$w_i(\boldsymbol{s}) := \frac{\exp(s_i)}{\sum \exp(s_i)} , \ i = 1, ..., m . \tag{29}$$

We then compute the partial subgradient with respect to s_i instead of w_i, and update w_i by updating s_i. Furthermore $\exp(s_i)$ are re-scaled in each iteration such that $\sum_{i=1}^m s_i = 0$.

The step-size $\alpha(\boldsymbol{w})$ is chosen by

$$\sigma(\boldsymbol{w}) := \min \left(\frac{\alpha}{\|\nabla_{\boldsymbol{s}} \tilde{W}(\boldsymbol{w}(\boldsymbol{s}))\|}, \zeta \right) . \tag{30}$$

The two hyper-parameters α and ζ trade off the convergence speed and the guaranteed decrease of the objective. Another hyper-parameter is τ which indicates the ratio between the update frequency of weights $\{w_i\}$ and that of support points $\{x_i\}$. In our experiments, we alternate one round of update for both $\{w_i\}$ and $\{x_i\}$. We summarize the subgradient descent approach in Algorithm 5.

Algorithm 5 Centroid Update with Subgradient Descent

1: **procedure** CENTROID($\{P^{(k)}\}_{k=1}^N$, P)▷ with initial guess
2:     **repeat**
3:         Updates $\{x_i\}$ from Eq.(3) Every τ iterations;
4:         **for** $k = 1, \ldots, N$ **do**
5:             Obtain $\Pi^{(k)}$ and $\Lambda^{(k)}$ from LP: $W(P, P^{(k)})$
6:         $\nabla \tilde{W}(\boldsymbol{w}) := \frac{1}{N} \sum_{k=1}^N \nabla \tilde{W}(\boldsymbol{w})^{(k)}$; ▷ See Eq.(28)
7:         $s_i := s_i - \sigma(\boldsymbol{w}) \nabla \tilde{W}(\boldsymbol{w}) \cdot \dfrac{\partial \boldsymbol{w}}{\partial s_i}$ ▷ See Eq. (30)
8:         $s_i := s_i - \sum_{j=1}^m s_j, i = 1, \ldots m$; ▷ rescaling step
9:         $w_i := \dfrac{\exp(s_i)}{\sum \exp(s_i)}, i = 1, \ldots m$;   ▷ sum-to-one
10:    **until** P converges
11:    **return** P

If the support points $\{x_i\}$ are fixed, the centroid optimization is a linear programming in terms of $\{w_i\}$, thus convex. The subgradient descent method converges under mild conditions on the smoothness of the solution and small or adaptive step-sizes. In Algorithm 5, the support points are also updated, and the problem becomes non-convex.

It is technically subtle to compare the convergence and stability of the overall AD2-clustering embedded with different algorithms for computing the centroid. Because of the many iterations in the outer loop, the centroid computation algorithm (solving the inner loop) may behave quite differently over the outer-loop rounds. For instance, if an algorithm is highly sensitive to a hyper-parameter in optimization, the hyper-parameter chosen based on earlier rounds may yield slow convergence later or even cause the failure of convergence. Moreover, achieving high accuracy for centroids in earlier rounds, usually demanding more inner-loop iterations, may not necessarily result in faster decrease in the clustering objective function because the cluster assignment step also matters.

In light of these issues, we employ a protocol described in Algorithm 6 to decide the number of iterations in the inner loop. The protocol specifies that in each iteration of the outer loop, the inner loop for updating centroids should complete within $\eta T_a/K$ amount of time, where T_a is the time used by the assignment step and K is the number of clusters. As we have pointed out, the LP/QP solver in the subgradient descent method or standard ADMM suffers from rapidly increasing complexity when the number of support points per distribution increases. In contrast, the effect on B-ADMM is much lower. In the experiment below, the datasets contain distributions with relatively small support sizes (a setup favoring the former two methods). A relatively tight time-allocation $\eta = 2.0$ is set. The subgradient descent method finishes at most 2 iterations in the inner loop, while B-ADMM on average finishes more than 60 iterations on the color and texture data, and more than 20 iterations on the protein sequence 1-gram and 3-gram data. The results by the ADMM method are omitted because this method cannot finish a single iteration under this time allocation.

Algorithm 6 Time allocation based algorithmic profile protocol

1: **procedure** PROFILE($\{P^{(k)}\}_{k=1}^{M}$, Q, K, η).
2:     Start profiling;
3:     $T = 0$;
4:     **repeat**
5:         $T_a = 0$;
6:         Assignment Step;
7:         GetElaspedCPUTime(T_a, T);
8:         GetAndDisplayPerplexity(T);
9:         Update Step within CPU time allocation $\eta T_a / K$;
10:     **until** $T < T_{total}$
11:     **return**

(a) image color

(b) image texture

(c) prot. seq. 1-gram
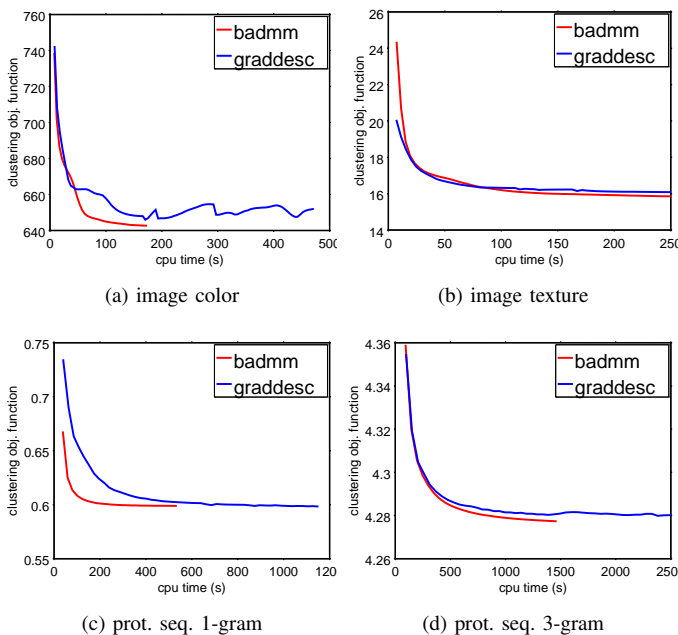
(d) prot. seq. 3-gram

Figure 4. Convergence performance of B-ADMM and the subgradient descent method for D2-clustering based on four datasets. The clustering objective function versus CPU time is shown. Here, $K = 10$, and the time-allocation ratio $\eta = 2.0$.

B. Stability analysis of AD2-clustering

In Fig. 4, we compare the convergence performance of the overall clustering process employing B-ADMM at $\rho_0 = 2.0$ and the subgradient descent method with fine-tuned values for the step-size parameter $\alpha \in \{0.05, 0.1, 0.25, 0.5, 1.0, 2.0, 5.0, 10.0\}$. The step-size is chosen as the value yielding the lowest clustering objective function in the first round. In this experiment, the whole image color and texture data are used. In the plots, the clustering objective function (Eq. (2)) is shown with respect to the CPU time. We observe a couple of advantages of the B-ADMM method. First, with a fixed parameter ρ_0, B-ADMM yields good convergence on all the four datasets, while the subgradient descent method requires manually tuning the step-size α in order to achieve comparable convergence speed. Second, B-ADMM achieves consistently lower values for the objective function across time. On the protein sequence 1-gram data, B-ADMM converges substantially faster than

the subgradient descent method with a fine-tuned step-size. Moreover, the subgradient descent method is numerically less stable. Although the step-size is fine-tuned based on the performance at the beginning, on the image color data, the objective function fluctuates noticeably in later rounds. Striking a balance (assuming it exists) between stability and speed for the subgradient descent method is a difficult dilemma.