A Data-dependent Approach for High-dimensional (Robust) Wasserstein Alignment

HU DING and WENJIE LIU, University of Science and Technology of China, China
MINGQUAN YE, University of Illinois Chicago, USA

Many real-world problems can be formulated as the alignment between two geometric patterns. Previously, a great amount of research focus on the alignment of two-dimensional (2D) or 3D patterns in the field of computer vision. Recently, the alignment problem in high dimensions finds several novel applications in practice. However, the research is still rather limited in the algorithmic aspect. To the best of our knowledge, most existing approaches are just simple extensions of their counterparts for 2D and 3D cases and often suffer from the issues such as high computational complexities. In this article, we propose an effective framework to compress the high-dimensional geometric patterns. Any existing alignment method can be applied to the compressed geometric patterns and the time complexity can be significantly reduced. Our idea is inspired by the observation that high-dimensional data often has a low intrinsic dimension. Our framework is a “data-dependent” approach that has the complexity depending on the intrinsic dimension of the input data. Our experimental results reveal that running the alignment algorithm on compressed patterns can achieve similar qualities, comparing with the results on the original patterns, but the runtimes (including the times cost for compression) are substantially lower.

CCS Concepts: • Theory of computation → Theory and algorithms for application domains;

Additional Key Words and Phrases: Wasserstein distance, Procrustes analysis, doubling dimension, network alignment, unsupervised cross-lingual learning, domain adaptation

ACM Reference format:
Hu Ding, Wenjie Liu, and Mingquan Ye. 2023. A Data-dependent Approach for High-dimensional (Robust) Wasserstein Alignment. ACM J. Exp. Algor. 28, 1, Article 1.8 (August 2023), 32 pages. https://doi.org/10.1145/3604910

1 INTRODUCTION

Given two geometric patterns, the problem of alignment is to find their appropriate spatial positions so as to minimize the difference between them. In general, a geometric pattern is represented by a set of (weighted) points in the space, and their difference is often measured by some objective function. Geometric alignment finds many applications in the field of computer vision, such as image retrieval, pattern recognition, fingerprint and facial shape alignment, and so on [17, 21, 57]. For different applications, we may have different constraints for the alignment, e.g., we can allow...
rigid transformations for fingerprint alignment. In addition, the Wasserstein distance (which is also called “earth mover’s distance” in some applications) [67] has been widely adopted for measuring the difference of patterns in computer vision, where its major advantage over other measures is the robustness in practice [80]. Besides the computer vision applications in two-dimensional (2D)/3D, recent research shows that a number of high-dimensional problems can be solved through geometric alignments. We briefly introduce several interesting examples below.

- The research on natural language processing has revealed that different languages often share similar structure at the word level [84]; in particular, the recent study on word semantics embedding has also shown the existence of structural isomorphism across languages [1, 27, 58] and finds that the Wasserstein distance can serve as a good distance for languages and documents [49, 85]. Zhang et al. [85] proposed to learn the transformation between different languages without any cross-lingual supervision, and the problem is reduced to minimizing the Wasserstein distance via finding the optimal geometric alignment in high dimensions. A number of improved algorithms that use the Wasserstein distance to compute the alignment between languages have also been proposed in recent years [6, 36].

- A Protein-Protein Interaction (PPI) network is a graph representing the interactions among proteins. Given two PPI networks, finding their alignment is a fundamental bioinformatics problem for understanding the correspondences between different species [56]. However, most existing approaches require to solve the NP-hard subgraph isomorphism problem and often suffer from high computational complexities. To resolve this issue, Liu et al. [54] recently applied the geometric embedding techniques to develop a new framework based on geometric alignment in Euclidean space.

- In supervised learning, our task usually is to learn the knowledge from a given labeled training dataset. However, labeled data could be very limited in practice. We can generate the labels for an unlabeled dataset by exploiting an existing annotated dataset, that is, transfer the knowledge from a source domain to a target domain. The problem is called “domain adaptation” in the field of transfer learning [65]. The problem has received a great amount of attention in the past years [12, 15]. Recently, Courty et al. [24] modeled the domain adaptation problem as a transportation problem of minimizing the Wasserstein distance between the source and target domains in high dimensions.

Despite of the above studies in the practical areas, the research on the algorithmic aspect of high-dimensional alignment is still rather limited. We need to take into account of the high dimensionality and large number of points of the geometric patterns simultaneously. In particular, as the developing of data acquisition techniques, the data sizes also increase very fast. For example, due to the rapid progress of high-throughput sequencing technologies, biological data are growing exponentially in recent years [83]. So it is quite important to develop efficient algorithmic techniques for dealing with the high-dimensional geometric alignment problem. Unfortunately, it has been shown that any constant factor approximation for geometric alignment under rigid transformation with Wasserstein distance takes at least $\Omega(n^d)$ time in the worst case, where $n$ is the maximum size of the two input patterns and $d$ is the dimensionality, unless $\text{SNP} \subset \text{DTIME}(2^{o(n)})$ [29]. The reason why the alignment problem is so hard is that it needs to compute the “transformation” and “matching” jointly for optimizing the objective. Namely, the transformation and matching interact and influence each other during the optimization procedure. Moreover, it is complicated to determine a rigid transformation in high-dimensional space. For instance, we can imagine that determining an appropriate rigid transformation in $\mathbb{R}^d$ needs to fix at least $d$ points in the space (so intuitively the complexity can be as large as $\binom{n}{d}$). Therefore, it is natural to ask the following question:
Is it possible to design low-complexity algorithm for high-dimensional geometric alignment under some reasonable assumption?

To the best of our knowledge, we are the first to consider this problem from theoretical perspective. Our idea is inspired by the observation that many real-world datasets often manifest low intrinsic dimensions [11]. For example, human handwriting images can be well embedded to some low-dimensional manifold though the Euclidean dimension can be very high. Following this observation, we consider to exploit the widely used notion, “doubling dimension” [26, 39, 44, 48, 74], to deal with the geometric alignment problem. The doubling dimension is particularly suitable to depict the data having low intrinsic dimension. Our framework is a “data-dependent” approach where the complexity depends on the doubling dimension of the input data. We prove that the given geometric patterns with low doubling dimensions can be substantially compressed so as to save a large amount of running time when computing the alignment. More importantly, our compression approach is an independent step and hence can serve as the preprocessing for different alignment methods. A preliminary version of this work has appeared in Reference [30].

Note: Independent of our work [30] published in 2019, Beugnot et al. [14] also considered the speedup for computing Wasserstein distance and proposed a k-means+-based compression method (which is somewhat similar to our k-center-based method in Section 2). Nevertheless, there are several significant differences between our work and Reference [14]. First, we consider the alignment problem under rigid transformation, where the work in Reference [14] only considers the static version. Also, we extend our method to fractional Wasserstein distance but it is unclear whether the theoretical analysis of Reference [14] is also available for the case with outliers.

The rest of the article is organized as follows. We introduce several important definitions in Section 1.1 and discuss some related works and our main idea in Sections 1.2 and 1.3, respectively. Then, we present our algorithm, analysis, and the time complexity in Sections 2 and 3. Finally, we study the practical performance of our proposed algorithm in Section 4.

### 1.1 Preliminaries

Before introducing the formal definition of geometric alignment, we need to define “Wasserstein distance” and “rigid transformation” first. In general, the Wasserstein distance is used to measure the difference between two distributions. In this article, we consider the case that the distributions are discrete point sets. Given two points \( p \) and \( q \) \( \in \mathbb{R}^d \), we use \( ||p - q|| \) to denote their Euclidean distance.

**Definition 1 (Wasserstein Distance \( W_2^2 \) [80]).** Let \( A = \{a_1, a_2, \ldots, a_{n_1}\} \) and \( B = \{b_1, b_2, \ldots, b_{n_2}\} \) be two sets of weighted points in \( \mathbb{R}^d \) with nonnegative weights \( a_i \) and \( \beta_j \) for each \( a_i \in A \) and \( b_j \in B \), respectively, and \( W_A \) and \( W_B \) be their respective total weights. The Wasserstein distance between \( A \) and \( B \) is

\[
W_2^2(A, B) = \frac{1}{\min\{W_A, W_B\}} \min_F \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} ||a_i - b_j||^2,
\]

where \( F = \{ f_{ij} \} \) is a feasible flow from \( A \) to \( B \), i.e., each \( f_{ij} \geq 0 \), \( \sum_{i=1}^{n_1} f_{ij} \leq \beta_j \), \( \sum_{j=1}^{n_2} f_{ij} \leq a_i \), and \( \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} = \min\{W_A, W_B\} \) (see Figure 1(a)).

**Remark 1.** (1) Usually we assume \( W_A = W_B = 1 \) for convenience, but in this article we also consider the “partial matching” (\( W_A \) can be not equal to \( W_B \)). (2) Intuitively, the Wasserstein distance can be viewed as the minimum transportation cost between \( A \) and \( B \), where the weights of \( A \) and \( B \) are the “supplies” and “demands,” respectively, and the cost of each edge connecting a pair of
1. The flows of the Wasserstein distance from $A = \{a_i \mid 1 \leq i \leq n_1\}$ to $B = \{b_j \mid 1 \leq j \leq n_2\}$, where each $f_{ij}$ is the flow from $a_i$ to $b_j$. (b) To reduce the Wasserstein distance between (the fixed) $A$ and $B$, we find a rigid transformation for the pattern $B$ that transforms it from the light blue point set to the dark blue point set. (c) The left image has some missing parts (masked by the white patches) and the right image has some noise. So it is more appropriate to compute the fractional Wasserstein distance to match them partially.

2. Points from $A$ to $B$ is their “ground distance.” In general, the “ground distance” can be defined in various forms, and here we use the squared Euclidean distance due to its simplicity in practice.

In some scenarios, we may only want a “partial matching” between $A$ and $B$ (e.g., $A$ or $B$ may contain some outliers, or a small pattern $A$ only corresponds to a part of a large pattern $B$) [59]. See Figure 1(c) for an example. For the applications in high dimensions, it is also natural to consider the partial matching. For example, one language may contain some words that cannot find their perfectly matched counterparts in the other language in the problem of cross-lingual alignment [85] (this phenomenon is very common between Chinese and English). Therefore we introduce the fractional Wasserstein distance below.

**Definition 2 (Fractional Wasserstein Distance).** Let $\lambda \in [0, 1]$. In Definition 1, if we replace the constraint “$\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} = \min\{W_A, W_B\}$” by “$\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} = \lambda \cdot \min\{W_A, W_B\}$,” then we call the distance as the fractional Wasserstein distance and denote it as $W_2^2(A, B, \lambda)$.

We consider the rigid transformation for alignment, because it is very natural to interpret in real world and has already been widely used in the aforementioned applications.

**Definition 3 (Rigid Transformation).** Let $P$ be a set of points in $\mathbb{R}^d$. A rigid transformation $T$ on $P$ is a transformation (i.e., rotation, translation, reflection, or their combination) that preserves the pairwise distances of the points in $P$.

**Definition 4 (Wasserstein Alignment).** Given two weighted point sets $A$ and $B$ as described in Definition 1, the problem of Wasserstein alignment between $A$ and $B$ under rigid transformation is to determine a rigid transformation $T$ for $B$ so as to minimize the Wasserstein distance $W_2^2(A, T(B))$, or $W_2^2(A, T(B), \lambda)$ if we consider the fractional Wasserstein distance (see Figure 1(b)).

As previously mentioned, we use the doubling dimension to describe high-dimensional data with low intrinsic dimension. We denote a metric space by $(X, d_X)$ where $d_X$ is the distance function of the set $X$. For instance, we can imagine that $X$ is a set of points in a low-dimensional manifold and $d_X$ is simply the Euclidean distance. For any $x \in X$ and $r \geq 0$, we use $\text{Ball}(x, r) = \{p \in X \mid d_X(x, p) \leq r\}$ to denote the ball of radius $r$ around $x$.

**Definition 5 (Doubling Dimension [48, 74]).** The doubling dimension of a metric space $(X, d_X)$ is the smallest number $\rho$, such that for any $x \in X$ and $r \geq 0$, $\text{Ball}(x, 2r)$ is always covered by the union of at most $2^\rho$ balls with radius $r$. 
Remark 2. The doubling dimension describes the expansion rate of \((X, d_X)\); intuitively, we can imagine a special case: A set of points are uniformly distributed inside a \(\rho\)-dimensional hypercube, where its doubling dimension is \(O(\rho)\) but the Euclidean dimension can be very high. For a more general case, a manifold in high-dimensional Euclidean space may have a very low doubling dimension, as many examples studied in machine learning [11]. Unfortunately, as shown in Reference [50], such low doubling dimensional metrics cannot always be embedded to low-dimensional Euclidean spaces with low distortion of Euclidean distance.

1.2 Related Works

Wasserstein distance. If building a bipartite graph, where the two columns of vertices correspond to the points of \(A\) and \(B\), respectively, and each edge connecting \((a_i, b_j)\) has the weight \(\|a_i - b_j\|^2\), then we can see that computing the Wasserstein distance actually is a min-cost flow problem from \(A\) to \(B\). A number of minimum cost flow algorithms have been developed in the past decades [4, 33, 63, 64, 75]. Suppose \(n\) and \(m\) are the numbers of vertices and edges in the bipartite graph, respectively, and \(U\) is the maximum weight. Orlin [62] developed a strongly polynomial algorithm with the time complexity \(O(n \log n (m + n \log n))\). Lee and Sidford [52] proposed an algorithm that can solve the minimum cost flow problem in \(O(n^{2.3} \text{poly}(\log U))\) time. Very recently, Chen et al. [19] improved the time complexity to be \(O(m^{1+o(1)})\). Sherman [72] provided a \((1 + \epsilon)\)-approximation algorithm based on preconditioning, and the running time is \(O(n^{2+o(1)} \epsilon^{-2})\). Andoni et al. [9] further proposed a parallel \((1 + \epsilon)\)-approximation algorithm that runs in \(\frac{1}{\epsilon} \text{poly}(\log n)\) time using \(O(\frac{1}{\epsilon^2} m)\) expected work.

In the community of machine learning, Cuturi [25] proposed a much faster “Sinkhorn Distance” algorithm that yields an approximation for the Wasserstein distance. Following Cuturi’s work, Altschuler et al. [5] provided an additive approximation\(^1\) algorithm for computing Wasserstein distance with the running time \(O(n^2 L^3 \log n \epsilon^{-3})\), where \(L\) is the maximum edge weight in the bipartite graph. Recently, the problems of robust Wasserstein distance and unbalanced Wasserstein distance (e.g., allow a small number of outliers) have attracted a lot of attention in the machine learning applications [18, 59].

When we focus on the Wasserstein distance problem in Euclidean space \(\mathbb{R}^d\), a number of faster algorithms have been proposed in computational geometry [3, 8, 16, 40, 70, 71, 77, 78]. Recently, Agarwal et al. [2] improved the running time to be \(O(n^{3/2} \epsilon^{-d} \text{poly}(\log U) \text{poly}(\log n))\). Using the idea of preconditioning [72], Khesin et al. [45] developed two randomized \((1 + \epsilon)\)-approximation algorithms with the running times \(O(n \epsilon^{-O(d)} \log(M) \log(M) \log n)\) and \(O(n \epsilon^{-O(d)} \log(U) \log(U) \log n)\), respectively (\(M\) is the aspect ratio of the given point sets). Fox and Lu [32] also presented a near-linear time algorithm, but their time bound is independent of \(M\) and \(U\). Most of these algorithms rely on the low-dimensional geometric techniques, and therefore their complexities are exponential in the dimensionality \(d\). Li [53] studied the problem of estimating the Wasserstein distance; his idea is a generalization of the method of [40] that yields an \(O(\rho)\)-approximate estimate of the Wasserstein distance, where \(\rho\) is the doubling dimension of the given data; but the approximation factor is relatively large and the algorithm needs at least a quadratic preprocessing time. Recently Chen et al. [20] applied the Quadtree to design a streaming Wasserstein distance algorithm in high dimensions, where the approximation factor is \(O(\log n)\). Ding et al. [28] considered the Wasserstein distance query problem for quickly answering the question that whether the distance is lower or higher than a query number; their algorithm is based on a hierarchical \(k\)-center clustering method in doubling metric.

\(^1\)The “additive approximation” means that the computed Wasserstein distance is at most \(OPT + \epsilon\) if \(OPT\) is the optimal Wasserstein distance.
Geometric alignment. Computing the geometric alignment of A and B is more challenging, since we need to determine the rigid transformation and the Wasserstein flow simultaneously. Moreover, due to the flexibility of rigid transformations, we cannot apply the Wasserstein distance embedding techniques [7, 41] to resolve the challenges. For example, the embedding can only preserve the Wasserstein distance between A and B; however, since there are infinite number of possible rigid transformations T for B (note that we do not know T in advance), it is difficult to also preserve the Wasserstein distance between A and T(B). In theory, Cabello et al. [16] presented a \(2 + \varepsilon\)-approximation solution for the 2D case, and later Klein and Veltkamp [47] achieved an \(O(d^{d-1})\)-approximation in \(\mathbb{R}^d\); Ding and Xu [29] proposed a PTAS for constant dimensional space. However, these theoretical algorithms cannot be efficiently implemented when the dimensionality is not constant. It was also mentioned in Reference [29] that any constant factor approximation needs a time complexity at least \(n^{\Omega(d)}\) based on some reasonable assumption in the theory of computational complexity, where \(n = \max(|A|, |B|)\). That is, it is unlikely to obtain a \((1 + \varepsilon)\)-approximation within a practical running time, especially when \(n\) is very large. The recent works [43, 61] proposed a core-set-based compression approach to speed up the computation of alignment. However, their compression methods only work for the case that \(d\) is small. We also refer the reader to the recent surveys [31, 60] for the detailed introductions on coresets.

In practice, Cohen and Guibas [21] proposed an alternating minimization approach for computing the geometric alignment of A and B. Several other approaches [22, 76] based on graph matching are inappropriate to be extended for high-dimensional alignment. In machine learning, a related topic is called "manifold alignment" [38, 82]; however, it usually has different settings and applications, and thus is out of the scope of this article.

1.3 Overview of the Alignment Procedure

Because the approach in Reference [21] is closely related to our proposed algorithm, we introduce it with more details for the sake of completeness. Roughly speaking, their approach is similar to the Iterative Closest Point method [13], where in each iteration it alternatively updates the Wasserstein distance flow and the rigid transformation. Thus it converges to some local optimum eventually. To update the rigid transformation, we can apply the Orthogonal Procrustes (OP) analysis [69]. The original OP analysis is only for unweighted point sets, and the weighted OP analysis was studied in the Wahba’s problem [81]. But our problem with the Wasserstein distance flow is a special case of the weighted OP analysis, and thus the complexity can be further reduced via a more careful analysis below.

Let A and B be the two sets of weighted points for alignment (as Definition 4). Suppose that the Wasserstein distance flow \(F = \{f_{ij}\}\) is fixed and the rigid transformation is waiting to update in the current stage. We imagine that there exist two new sets of weighted points,

\[
\hat{A} = \bigcup_{i=1}^{n_1} \{a_i^1, a_i^2, \ldots, a_i^{n_2}\},
\]

\[
\hat{B} = \bigcup_{j=1}^{n_2} \{b_j^1, b_j^2, \ldots, b_j^{n_1}\},
\]

where each \(a_i^j\) (respectively, \(b_j^i\)) has the weight \(f_{ij}\) and the same spatial position of \(a_i\) (respectively, \(b_j\)). That is, each \(a_i\) (respectively, \(b_j\)) is decomposed into \(n_2\) (respectively, \(n_1\)) copies. First, we take a translation vector \(\vec{v}\) such that the weighted mean points of \(\hat{A}\) and \(\hat{B} + \vec{v}\) coincide with each other (this can be easily derived, due to the fact that the objective function uses squared distance [21]). Second, by using the weighted OP analysis, we compute an orthogonal matrix \(\mathcal{R}\) for \(\hat{B} + \vec{v}\) to minimize its weighted \(L_2^2\) difference to \(\hat{A}\). For this purpose, we generate two \(d \times (n_1n_2)\) matrices \(\mathcal{M}_A\) and \(\mathcal{M}_B\) as follows.
We use $a_i$ (respectively, $b_j$) to denote the corresponding $d$-dimensional column vector of $a_i$ (respectively, $b$). Each point of $\hat{A}$ (respectively, $\hat{B} + \hat{\mathcal{D}}$) corresponds to an individual column of $\hat{M}_A$ (respectively, $\hat{M}_B$); for example, a point $a_i \in \hat{A}$ (respectively, $b_j + \mathcal{D} \in \hat{B} + \mathcal{D}$) corresponds to a column $\sqrt{f_{ij} a_i}$ (respectively, $\sqrt{f_{ij} (b_j + \mathcal{D})}$) in $\hat{M}_A$ (respectively, $\hat{M}_B$). Formally,

$$
\hat{M}_A = \begin{bmatrix}
\sqrt{f_{11} a_1}, & \sqrt{f_{12} a_1}, & \ldots, & \sqrt{f_{1n_1} a_1}, & \sqrt{f_{21} a_2}, & \ldots, & \sqrt{f_{2n_2} a_2}, \\
\ldots, & \sqrt{f_{n_11} a_{n_1}}, & \sqrt{f_{n_12} a_{n_1}}, & \ldots, & \sqrt{f_{n_1n_2} a_{n_1}}
\end{bmatrix},
$$

$$
\hat{M}_B = \begin{bmatrix}
\sqrt{f_{11} (b_1 + \mathcal{D})}, & \sqrt{f_{12} (b_2 + \mathcal{D})}, & \ldots, & \sqrt{f_{1n_2} (b_1 + \mathcal{D})}, & \sqrt{f_{21} (b_1 + \mathcal{D})}, & \ldots, & \sqrt{f_{2n_2} (b_2 + \mathcal{D})}, \\
\ldots, & \sqrt{f_{n_11} (b_n_1 + \mathcal{D})}, & \sqrt{f_{n_12} (b_n_2 + \mathcal{D})}, & \ldots, & \sqrt{f_{n_1n_2} (b_n_2 + \mathcal{D})}
\end{bmatrix}.
$$

Let the SVD of $\hat{M}_A \times \hat{M}_B^T$ be $U \Sigma V^T$, and the optimal orthogonal matrix $R$ should be $UV^T$ through the OP analysis.

The above idea has a drawback that the matrices $\hat{M}_A$ and $\hat{M}_B$ are too large and thus the complexity for computing the multiplication $\hat{M}_A \times \hat{M}_B^T$ can be very high (which is $O(n_1n_2d^2)$). Actually we do not need to really construct the large matrices $\hat{M}_A$ and $\hat{M}_B$, since many of the columns are identical (except for the scalar $\sqrt{f_{ij}}$). Instead, we can compute the multiplication $\hat{M}_A \times \hat{M}_B^T$ in $O(n_1n_2d + \min(n_1, n_2) \cdot d^2)$ time. Note that both the complexities “$O(n_1n_2d^2)$” and “$O(n_1n_2d + \min(n_1, n_2) \cdot d^2)$” can be slightly improved by using the faster rectangle matrix multiplication algorithms [51], but it needs a somewhat complicated discussion on the relation of the values $n_1, n_2$, and $d$, and so it is out of the scope of this article.

**Claim 1.** The multiplication $\hat{M}_A \times \hat{M}_B^T$ can be computed in $O(n_1n_2d + \min(n_1, n_2) \cdot d^2)$ time.

**Proof.** With a slight abuse of notations, we also use $F$ to denote the $n_1 \times n_2$ matrix of the Wasserstein distance flow where each entry is $f_{ij}$; also, “$F_{ik}$” represents the $i$th row of the matrix $F$. Given a vector $t$, we use $\sqrt{t}$ to denote the new vector with each entry being the square root of the corresponding one in $t$. Also, we use $\text{diag}(t)$ to denote the diagonal matrix where the $i$th diagonal entry is the $i$th entry of $t$. For example, if $t = [t_1, t_2, \ldots, t_n]$, then $\sqrt{t} = [\sqrt{t_1}, \sqrt{t_2}, \ldots, \sqrt{t_n}]$ and

$$
\text{diag}(\sqrt{t}) = \begin{bmatrix}
\sqrt{t_1}, & 0, & 0, & \ldots, & 0 \\
0, & \sqrt{t_2}, & 0, & \ldots, & 0 \\
0, & 0, & \sqrt{t_3}, & \ldots, & 0 \\
\cdots, & \cdots, & \cdots, & \cdots, & \cdots \\
0, & 0, & 0, & \ldots, & \sqrt{t_n}
\end{bmatrix}.
$$

Following the constructions of $\hat{M}_A$ and $\hat{M}_B$ with some simple calculation, we have

$$
\hat{M}_A = \begin{bmatrix}
\sqrt{f_{11} a_1}, & \sqrt{f_{12} a_1}, & \ldots, & \sqrt{f_{1n_1} a_1}, & \sqrt{f_{21} a_2}, & \ldots, & \sqrt{f_{2n_2} a_2}, \\
\ldots, & \sqrt{f_{n_11} a_{n_1}}, & \sqrt{f_{n_12} a_{n_1}}, & \ldots, & \sqrt{f_{n_1n_2} a_{n_1}}
\end{bmatrix} = [a_1 \sqrt{F_{1,..}} a_2 \sqrt{F_{2,..}}, \ldots, a_{n_1} \sqrt{F_{n_1,..}}];
$$

$$
\hat{M}_B = \begin{bmatrix}
\sqrt{f_{11} (b_1 + \mathcal{D})}, & \sqrt{f_{12} (b_2 + \mathcal{D})}, & \ldots, & \sqrt{f_{1n_2} (b_1 + \mathcal{D})}, & \sqrt{f_{21} (b_1 + \mathcal{D})}, & \ldots, & \sqrt{f_{2n_2} (b_2 + \mathcal{D})}, \\
\ldots, & \sqrt{f_{n_11} (b_n_1 + \mathcal{D})}, & \sqrt{f_{n_12} (b_n_2 + \mathcal{D})}, & \ldots, & \sqrt{f_{n_1n_2} (b_n_2 + \mathcal{D})}
\end{bmatrix} = [b_1 + \mathcal{D}, \ldots, b_{n_2} + \mathcal{D}] \times [\text{diag}(\sqrt{F_{1,..}}), \text{diag}(\sqrt{F_{2,..}}), \ldots, \text{diag}(\sqrt{F_{n_1,..}})].
$$
For simplicity, we let $A = [a_1, \ldots, a_{n_1}]$ and $B = [b_1 + \overrightarrow{v}, \ldots, b_{n_2} + \overrightarrow{v}]$. Then,

$$M_A \times M_B = \sum_{i=1}^{n_1} (a_i \sqrt{F_{i,1}}) \times \left( \text{diag}(\sqrt{F_{i,1}})B^T \right)$$

$$= \sum_{i=1}^{n_1} a_i F_{i,1} B^T = AFB^T.$$  

The sizes of $A$, $F$, and $B$ are $d \times n_1$, $n_1 \times n_2$, and $d \times n_2$, respectively. So it is easy to see that computing $\left( AFB^T \right)^n$ takes $O(n_1 n_2 d + \min\{n_1, n_2\} \cdot d^2)$ time (e.g., if $n_1 \geq n_2$, then we compute $AF$ first and then compute $(AF) \times B^T$).

Therefore, the time complexity for obtaining the optimal $R$ is $O(n_1 n_2 d + \min\{n_1, n_2\} \cdot d^2 + d^3)$, where the term “$d^3$” comes from the computation of the SVD.

**Proposition 1.** Each iteration of the approach in Reference [21] takes $\Gamma(n_1, n_2, d) + O(n_1 n_2 d + \min\{n_1, n_2\} \cdot d^2 + d^3)$ time, where $\Gamma(n_1, n_2, d)$ denotes the time complexity of the used Wasserstein distance algorithm. In practice, we usually assume $n_1, n_2 = O(n)$ with some $n \geq d$, and then the complexity can be simply written as $\Gamma(n, d) + O(n^3 d)$.

The bottleneck is that the algorithm needs to repeatedly compute the Wasserstein distance and transformation, especially when $n$ and $d$ are large (usually $\Gamma(n, d) = \Omega(n^2 d)$). Based on the property of low doubling dimension, we construct a pair of compressed point sets to replace the original $A$ and $B$, and run the same algorithm on the compressed data instead. As a consequence, the running time is reduced significantly. Note that our compression step is independent of the approach [21]; actually, any alignment method with the same objective function in Definition 4 can benefit from our compression idea.

## 2 THE ALGORITHM AND ANALYSIS

Our idea starts from the widely studied $k$-center clustering problem. Given an integer $k \geq 1$ and a point set $P$ in some metric space, the $k$-center clustering is to partition $P$ into $k$ clusters and cover each cluster by an individual ball, such that the maximum radius of the balls is minimized. Gonzalez [35] presented an elegant $2$-approximation algorithm, where the radius of each resulting ball (i.e., cluster) is at most two times the optimum. Initially, it sets $S = \{c_1\}$ where $c_1$ is an arbitrary point selected from $P$; then in each of the following $k - 1$ iterations, a new point who has the largest distance to $S$ among the points of $P$ is added to $S$ (we define the distance between a point $q$ and $S$ to be $\min\{|q - p| \mid p \in S\}$). Let $S = \{c_1, \ldots, c_k\}$, and then $P$ is covered by the $k$ balls $\text{Ball}(c_1, r), \ldots, \text{Ball}(c_k, r)$ with

$$r \leq \min\{|c_i - c_j| \mid 1 \leq i \neq j \leq k\}. \quad (6)$$

It is easy to prove that $r$ is at most two times the optimal radius of the given instance.

Let $P$ be a point set in $\mathbb{R}^d$ with the doubling dimension $\rho$. The diameter of $P$ is denoted by $\Delta$, i.e., $\Delta = \max\{|p - q| \mid p, q \in P\}$. Then we have the following lemma.

**Lemma 1.** Given an integer $k \geq 1$, if one runs the Gonzalez’s $k$-center clustering algorithm, then the obtained radii of the clusters are at most $\frac{2}{k^{1/\rho}} \Delta$.

**Proof.** Let $S$ be the set of $k$ points obtained by the Gonzalez’s algorithm and the obtained radius be $r$. We also define the aspect ratio of $S$ as the ratio of the maximum to the minimum pairwise distance in $S$. Then, it is easy to see that the aspect ratio of $S$ is at most $\Delta/r$ from (6). Now, we need the following Claim 2 from Reference [48, 74]. Actually, the claim can be obtained by recursively applying the definition of doubling dimension.
Let \( (X, d_X) \) be a metric space with the doubling dimension \( \rho \), and \( Y \subset X \). If the aspect ratio of \( Y \) is upper bounded by some positive value \( \alpha \), then \( |Y| \leq 2^\rho \left[ \log_2 \alpha \right] \).

Replacing \( X \) and \( Y \) by \( P \) and \( S \), respectively, in the above claim, we have
\[
|S| \leq 2^\rho \left[ \log_2 \Delta / r \right] \leq 2^\rho (1 + \log_2 \Delta) r.
\]
(7)
Since \( |S| = k \), (7) implies \( r \leq \frac{2}{k} \log_2 \Delta \). \( \square \)

**Corollary 1.** If we let \( k = (\frac{\Delta}{2})^\rho \) with some small \( \epsilon > 0 \), then the radius in Lemma 1 will be \( \epsilon \Delta \).

**Our compression algorithm.** Let \( A \) and \( B \) be the two given point sets in Definition 4, and we assume their diameters are \( \Delta_A \) and \( \Delta_B \), respectively. Let \( \Delta = \max\{\Delta_A, \Delta_B\} \). We also assume that they both have the doubling dimension at most \( \rho \). Our idea for compressing \( A \) and \( B \) is as follows. As described in Lemma 1, we run the Gonzalez’s algorithm on \( A \) and \( B \), respectively. We denote by \( S_A = \{c_{1}^A, \ldots, c_{k}^A\} \) and \( S_B = \{c_{1}^B, \ldots, c_{k}^B\} \) the obtained sets of \( k \)-cluster centers with \( k = (\frac{\Delta}{2})^\rho \). For each cluster center \( c_{j}^A \) (respectively, \( c_{j}^B \)), we assign a weight that is equal to the total weight of the points in the corresponding cluster. As a consequence, we obtain a new instance \((S_A, S_B)\) for geometric alignment. It is easy to know that the total weight of \( S_A \) (respectively, \( S_B \)) is equal to \( W_A \) (respectively, \( W_B \)). For the sake of convenience, we name this compression method as KCcenter. The following theorem shows that one can achieve an approximate solution for the instance \((A, B)\) by solving the alignment of \((S_A, S_B)\).

**Theorem 2.** Suppose \( \epsilon > 0 \) is a small parameter in Corollary 1. Given any \( c \geq 1 \), let \( \tilde{T} \) be a rigid transformation yielding \( c \)-approximation for minimizing \( \mathcal{W}^2_2(S_A, \tilde{T}(B)) \) in Definition 4. Then,
\[
\mathcal{W}^2_2(A, \tilde{T}(B)) \leq c(1 + 2\epsilon^2) \cdot \min_{\tilde{T}} \mathcal{W}^2_2(A, \tilde{T}(B)) + 2c(1 + 2\epsilon)(1 + \epsilon)\Delta^2
\]
\[
= c(1 + O(\epsilon)) \cdot \min_{\tilde{T}} \mathcal{W}^2_2(A, \tilde{T}(B)) + 2c(1 + O(\epsilon))(1 + \epsilon)\Delta^2.
\]
(8)

**Proof.** First, we denote by \( T_{\text{opt}} \) the optimal rigid transformation achieving \( \min_{T} \mathcal{W}^2_2(A, T(B)) \). Since \( \tilde{T} \) yields \( c \)-approximation for minimizing \( \mathcal{W}^2_2(S_A, \tilde{T}(S_B)) \), we have
\[
\mathcal{W}^2_2(S_A, \tilde{T}(S_B)) \leq c \cdot \min_{\tilde{T}} \mathcal{W}^2_2(S_A, \tilde{T}(S_B))
\]
\[
\leq c \cdot \mathcal{W}^2_2(S_A, T_{\text{opt}}(S_B)).
\]
(9)
Recall that each point \( c_{j}^A \) (respectively, \( c_{j}^B \)) has the weight equal to the total weights of the points in the corresponding cluster. For instance, if the cluster contains \( \{a_{j(1)}, a_{j(2)}, \ldots, a_{j(h)}\} \), then the weight of \( c_{j}^A \) should be \( \sum_{l=1}^{h} a_{j(l)} \); actually, we can view \( c_{j}^A \) as \( h \) overlapping points \( \{a'_{j(1)}, a'_{j(2)}, \ldots, a'_{j(h)}\} \) with each \( a'_{j(l)} \) having the weight \( a_{j(l)} \). Therefore, for the sake of convenience, we use another representation for \( S_A \) and \( S_B \) in our proof below:
\[
S_A = \{a'_{1}, \ldots, a'_{n_1}\} \quad \text{and} \quad S_B = \{b'_{1}, \ldots, b'_{n_2}\},
\]
(10)
where each \( a'_{j} \) (respectively, \( b'_{j} \)) has the weight \( \alpha_{j} \) (respectively, \( \beta_{j} \)). Note that \( S_A \) and \( S_B \) only have \( k \) distinct positions respectively in the space. Moreover, due to Corollary 1, we know that \( ||a'_{i} - a_{i}||, ||b'_{j} - b_{j}|| \leq \epsilon \Delta \) for any \( 1 \leq i \leq n_1 \) and \( 1 \leq j \leq n_2 \), and these bounds are invariant under any rigid transformation in the space. Consequently, for any pair \((i, j)\) and any rigid transformation \( T \), we have
\[
\|a_i - T(b_j)\|^2 \\
\leq \left(\|a_i - a'_i\| + \|a'_i - T(b'_j)\| + \|T(b'_j) - T(b_j)\|\right)^2 \\
\leq \left(\|a'_i - T(b'_j)\| + 2\epsilon\Delta\right)^2 \\
= \|a'_i - T(b'_j)\|^2 + 4\epsilon\Delta\|a'_i - T(b'_j)\| + 4\epsilon^2\Delta^2 \\
\leq \|a'_i - T(b'_j)\|^2 + 2\epsilon\left(\Delta^2 + \|a'_i - T(b'_j)\|^2\right) + 4\epsilon^2\Delta^2 \\
= (1 + 2\epsilon)\|a'_i - T(b'_j)\|^2 + (2\epsilon + 4\epsilon^2)\Delta^2 \quad (11)
\]

through the triangle inequality. Using exactly the same idea, we also have

\[
\|a'_i - T(b'_j)\|^2 \leq (1 + 2\epsilon)\|a_i - T(b_j)\|^2 + (2\epsilon + 4\epsilon^2)\Delta^2. \quad (12)
\]

Based on Definition 1, we denote by \(\tilde{F} = (\tilde{f}_{ij})\) the induced flow of \(W_2^2(S_A, \tilde{T}(S_B))\) (using the Equation (10) for \(S_A\) and \(S_B\)). Then formula (11) directly implies that

\[
W_2^2(A, \tilde{T}(B)) \leq \frac{1}{\min\{W_A, W_B\}} \sum_{i=1}^{n_A} \sum_{j=1}^{n_B} \tilde{f}_{ij}\|a_i - \tilde{T}(b_j)\|^2 \\
\leq \frac{1 + 2\epsilon}{\min\{W_A, W_B\}} \sum_{i=1}^{n_A} \sum_{j=1}^{n_B} \tilde{f}_{ij}\|a'_i - \tilde{T}(b'_j)\|^2 + (2\epsilon + 4\epsilon^2)\Delta^2 \\
= (1 + 2\epsilon)W_2^2(S_A, \tilde{T}(S_B)) + (2\epsilon + 4\epsilon^2)\Delta^2. \quad (13)
\]

By using the similar idea (replacing \(\tilde{T}\) by \(T_{\text{opt}}\), and exchanging the roles of \((A, B)\) and \((S_A, S_B)\)), inequality (12) directly implies that

\[
W_2^2(S_A, T_{\text{opt}}(S_B)) \leq (1 + 2\epsilon)W_2^2(A, T_{\text{opt}}(B)) + (2\epsilon + 4\epsilon^2)\Delta^2. \quad (14)
\]

Combining inequalities (9), (13), and (14), we have

\[
W_2^2(A, \tilde{T}(B)) \leq (1 + 2\epsilon)W_2^2(S_A, \tilde{T}(S_B)) + (2\epsilon + 4\epsilon^2)\Delta^2 \\
\leq (1 + 2\epsilon) \cdot c \cdot W_2^2(S_A, T_{\text{opt}}(S_B)) + (2\epsilon + 4\epsilon^2)\Delta^2 \\
\leq c(1 + 2\epsilon)^2 \cdot W_2^2(A, T_{\text{opt}}(B)) + 2\epsilon(c + 1 + 2\epsilon)(1 + 2\epsilon)\Delta^2. \quad (15)
\]

and the proof is completed. \(\Box\)

When \(\epsilon\) is small enough, Theorem 2 shows that \(W_2^2(A, \tilde{T}(B)) \approx c \cdot W_2^2(A, T_{\text{opt}}(B))\). That is, \(\tilde{T}\), the solution of \((S_A, S_B)\), achieves roughly the same performance on \((A, B)\). Consequently, we propose the approximation algorithm for geometric alignment (see Algorithm 1). We emphasize that though we use the algorithm from Reference [21] in Step 3, Theorem 2 is an independent result; that is, any alignment method with the same objective function in Definition 4 can benefit from Theorem 2.

### 2.1 Extension I: When \(k\) Is Not Given

As discussed in Corollary 1, the value \(k = (2/\epsilon)^{\rho}\) depends on the doubling dimension \(\rho\) if we require the compression error (i.e., the radius) to be no larger than \(\epsilon\Delta\). However, the exact doubling dimension \(\rho\) usually is not easy to obtain [39]. So we consider another scenario. Let \(\epsilon\Delta\) be the pre-specified compression error bound, and we run the Gonzalez’s algorithm iteratively until the radius is reduced to be no larger than the bound. The question is that

**can our algorithm be aware when the stopping condition is reached?**
We answer this question in the affirmative, where the only change is that the value of \( k \) is required to be larger than \((2/\varepsilon)^{\rho}\). In the worst case, \( k \) can be as large as \((4/\varepsilon)^{\rho}\) (so the value \( k \) is enlarged by a factor \( 2^{\rho} \)).

First, we need to estimate the diameter \( \Delta \). In practice we often avoid to compute the exact value of \( \Delta \), since it takes at least quadratic complexity. Instead, we can simply take a point \( p \) and its farthest point \( q \) from \( P \); let \( \tilde{\Delta} = ||p - q|| \) and then we know

\[
\frac{1}{2} \Delta \leq \tilde{\Delta} \leq \Delta.
\]  

(16)

Then we have the following lemma (which is a counterpart of Lemma 1).

**Lemma 2.** Given a small parameter \( \varepsilon > 0 \), if one runs the Gonzalez’s \( k \)-center clustering algorithm iteratively until the obtained radius is no larger than \( \varepsilon \tilde{\Delta} \), then the number of obtained clusters is at most \( (\frac{4}{\varepsilon})^{\rho}\).

**Proof.** Let \( S \) be the set of \( k \) points by Gonzalez’s algorithm, and the obtained radius be \( r \leq \varepsilon \tilde{\Delta} \). It is easy to see that the aspect ratio of \( S \) is at most \( \Delta/(\varepsilon \tilde{\Delta}) \leq \frac{2}{\varepsilon} \) (by inequality of (16)). From Claim 2, we have

\[
|S| \leq 2^{\rho \lceil \log_2 \frac{2}{\varepsilon} \rceil} \leq 2^{\rho \left(1 + \log_2 \frac{2}{\varepsilon}\right)} = (4/\varepsilon)^{\rho}.
\]  

(17)

From Lemma 2, we just need to modify Step 1 of Algorithm 1 for the case that the doubling dimension \( \rho \) is not given.

### 2.2 Extension II: \( k \)-means + \( k \)-center

In Theorem 2, we show that the Gonzalez’s \( k \)-center clustering algorithm can be used to compress input data. So a natural question is that whether other clustering method, such as the widely used \( k \)-means clustering, can be also applied to achieve this purpose. To answer this question, we need to revisit the proof of Theorem 2. In \( k \)-center clustering, each obtained cluster has a sufficiently small radius, and so we can obtain the inequalities (11) and (12). But if we run \( k \)-means instead, then the obtained clusters may have large radii and thus inequalities (11) and (12) can be violated. This observation inspires the following compression method that combines \( k \)-center and \( k \)-means.

**KCenter+.** In Step 1 of Algorithm 1, the obtained sets \( S_A \) and \( S_B \) are the corresponding ball centers (each cluster is covered by a ball with radius \( \leq \varepsilon \tilde{\Delta} \) if \( k \geq (\frac{2}{\varepsilon})^{\rho} \)). Actually, we can further improve the result locally. For each ball center, we can replace it by the mean point of the cluster. Since our Wasserstein distance \( \mathcal{W}_2^2(\cdot, \cdot) \) is the sum of a set of weighted squared Euclidean distances, the additive error bound \( ^*2\varepsilon c(1 + 2\varepsilon)(1 + 2\varepsilon)\Delta^{2n} \) in inequality of (8) can be reduced. For example, in the second inequality of (11), the items \(^*||a_i - a'_i||\) and \( ^*||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||\) are just simply bounded.
Fig. 2. We add two dummy points $a_0$ and $b_0$, where their weights are both $(1 - \lambda) \cdot \min\{W_A, W_B\}$. The point $a_0$ connects all the $b_j$s with the edge cost 0, and so does $b_0$; the cost of the edge connecting $a_0$ and $b_0$ is $+\infty$. The total flow from the left to the right is $\min\{W_A, W_B\} + (1 - \lambda) \cdot \min\{W_A, W_B\}$. Intuitively, the dummy point $a_0$ absorbs the $(1 - \lambda) \cdot \min\{W_A, W_B\}$ outlier flows from $B$, and the dummy point $b_0$ absorbs the $(1 - \lambda) \cdot \min\{W_A, W_B\}$ outlier flows from $A$.

by $\epsilon \Delta$; but if we take a more careful analysis, then the formula (11) can be rewritten as follows:

$$||a_i - \mathcal{T}(b_j)||^2$$
$$\leq (||a_i - a'_i|| + ||a'_i - \mathcal{T}(b_j)|| + ||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||)^2$$
$$= ||a'_i - \mathcal{T}(b'_j)||^2 + 2(||a_i - a'_i|| + ||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||) \times ||a'_i - \mathcal{T}(b'_j)||$$
$$+ (||a_i - a'_i|| + ||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||)^2$$
$$\leq ||a'_i - \mathcal{T}(b'_j)||^2 + 4\epsilon \Delta \times ||a'_i - \mathcal{T}(b'_j)|| + 2||a_i - a'_i||^2 + 2||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||^2$$
$$\leq ||a'_i - \mathcal{T}(b'_j)||^2 + 2(\epsilon \Delta^2 + ||a'_i - \mathcal{T}(b'_j)||^2) + 2||a_i - a'_i||^2 + 2||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||^2$$
$$= (1 + 2\epsilon)||a'_i - \mathcal{T}(b'_j)||^2 + 2\epsilon \Delta^2 + 2||a_i - a'_i||^2 + 2||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||^2$$

Comparing formula (18) with formula (11), we can see that the item “$4\epsilon^2 \Delta^2$” in the right-hand side bound is replaced by “$2||a_i - a'_i||^2 + 2||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||^2$.” Note that $a'_i$ and $\mathcal{T}(b'_j)$ are actually the centers of the balls that respectively cover $a_i$ and $\mathcal{T}(b_j)$. So if we replace the ball centers by the means, then the accumulated additive errors in inequalities (13) and (14) can be further reduced (though this bound remains the same in the worst case, e.g., all the points of the cluster locate uniformly on the sphere of the ball, and then $||a_i - a'_i||$ and $||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||$ are always equal to $\epsilon \Delta$).

Intuitively, we use $k$-center clustering first to bound the radius of each cluster, and then compute the mean of each cluster to refine the result. We name this method as KCenter+. In Section 4, our experiments also verified the fact that KCenter+ usually can achieve better performance than KCenter.

2.3 Extension III: Robust Alignment with Fractional Wasserstein Distance

We further consider the alignment problem with the fractional Wasserstein distance as Definition 2. The first question is how to compute this new distance. Recall that the vanilla Wasserstein distance is equivalent to computing the minimum cost maximum flow on the bipartite graph of $A$ and $B$. When we consider the fractional Wasserstein distance that allows $1 - \lambda$ outliers, we can modify the bipartite graph by adding two “dummy” points (see Figure 2 for an illustration). Then, we can directly apply any off-the-shelf Wasserstein distance algorithms (e.g., the network simplex algorithm [4] or the Sinkhorn distance algorithm [25]) to compute the fractional Wasserstein distance $W^2_2(A, B, \lambda)$.  

ACM Journal of Experimental Algorithmics, Vol. 28, No. 1, Article 1.8. Publication date: August 2023.
**Algorithm 2:** Fractional Wasserstein distance

**Input:** An instance \((A, B)\) of the Wasserstein distance problem in Definition 2 with the fraction \(\lambda\) (0 < \(\lambda\) ≤ 1).

**Output:** The fractional Wasserstein distance \(W^2(A, B, \lambda)\) and the corresponding flows from \(A\) to \(B\).

1. Set \(w_0 = (1 - \lambda) \min \{W_A, W_B\}\).
2. Add the dummy points \(a_0\) and \(b_0\) to \(A\) and \(B\), respectively; their weights are both \(w_0\). Denote the new point sets by \(\tilde{A}\) and \(\tilde{B}\), respectively.
3. Build the bipartite graph of \(\tilde{A}\) and \(\tilde{B}\), and set the ground distance (edge cost) as Figure 2.
4. Apply the existing Wasserstein distance algorithm e.g., Reference [25], to compute the optimal transport flow matrix and the Wasserstein distance.

**Lemma 3.** The Algorithm 2 returns the optimal fractional Wasserstein distance flow of \((A, B)\) with the total flow being equal to \(\lambda\) \(\min\{W_A, W_B\}\). The time complexity is as same as the vanilla Wasserstein distance algorithm in Reference [4] or Reference [25].

**Proof.** We denote the obtained flows as \(F = \{f_{ij} \mid 0 \leq i \leq n_1, 0 \leq j \leq n_2\}\). First, we need to show that \(f_{00} = 0\). Otherwise, we can arbitrarily pick a non-zero flow, say, \(f_{i_0j_0}\), from \(F \setminus \{f_{00}\}\) and perform the following modification: Let \(\delta = \min\{f_{00}, f_{i_0j_0}\}\), and update

\[
\begin{align*}
    f_{00} &\rightarrow f_{00} - \delta; \\
    f_{i_0j_0} &\rightarrow f_{i_0j_0} - \delta; \\
    f_{0j_0} &\rightarrow f_{0j_0} + \delta; \\
    f_{i_00} &\rightarrow f_{i_00} + \delta.
\end{align*}
\]

From the edge costs in Figure 2 (the edge cost connecting \(a_0\) and \(b_0\) is \(\infty\)), we know that the total transportation cost is reduced after the above modification. This is in contradiction with the optimality of \(F\). Also, since \(\alpha_0 = \beta_0 = (1 - \lambda) \cdot \min\{W_A, W_B\}\), together with \(f_{00} = 0\), we have the total flows from \(A\) to \(B\), that is,

\[
\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} = \lambda \cdot \min\{W_A, W_B\}. \tag{20}
\]

Now, we prove the optimality of the flows with respect to Definition 2. Suppose the flows \(F' = \{f'_{ij} \mid 1 \leq i \leq n_1, 1 \leq j \leq n_2\}\) yield the optimal fractional Wasserstein distance, i.e.,

\[
\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f'_{ij} ||a_i - b_j||^2 < \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} ||a_i - b_j||^2. \tag{21}
\]

We can easily augment \(F'\) to be a solution for \((\tilde{A}, \tilde{B})\) with the dummy points \(a_0\) and \(b_0\): \(F' \rightarrow F' \cup \{f'_{00}, f'_{0j}, f'_{i0} \mid 1 \leq i \leq n_1, 1 \leq j \leq n_2\}\), where

\[
\begin{align*}
    f'_{00} &= 0; \\
    f'_{0j} &= \frac{(1 - \lambda) \cdot \min\{W_A, W_B\}}{W_B - \lambda \cdot \min\{W_A, W_B\}} \left( \beta_j - \sum_{i=1}^{n_1} f'_{ij} \right); \\
    f'_{i0} &= \frac{(1 - \lambda) \cdot \min\{W_A, W_B\}}{W_A - \lambda \cdot \min\{W_A, W_B\}} \left( \alpha_i - \sum_{j=1}^{n_2} f'_{ij} \right). \tag{22}
\end{align*}
\]
It is easy to verify that the augmented $F'$ is a feasible flow set from $\hat{A}$ to $\hat{B}$ with the total flows being equal to $\min\{W_A, W_B\} + (1 - \lambda) \cdot \min\{W_A, W_B\}$. From the edge costs in Figure 2, together with inequality of (21), we know that the augmented $F'$ yields a lower transportation cost between $\hat{A}$ and $\hat{B}$, which is in contradiction with the optimality of $F$. Therefore, Algorithm 2 returns the optimal fractional Wasserstein distance flow of $(A, B)$. The time complexity is as same as the vanilla Wasserstein distance algorithm in Reference [4] or Reference [25], since we only add two more points to the input. □

Using Algorithm 2, we can compute the partial alignment between $A$ and $B$. We still use our proposed Algorithm 1, where the only difference is that we need to apply Algorithm 2 to compute the fractional Wasserstein distance between $S_A$ and $\mathcal{T}(S_B)$ (respectively, $A$ and $\mathcal{T}(B)$). Similarly with Theorem 2, we also have the following Theorem 3 for the partial alignment with any given parameter $\lambda \in [0, 1]$. The proof (which is very similar with that of Theorem 2) is placed in the appendix.

**Theorem 3.** Suppose $\epsilon > 0$ is a small parameter in Corollary 1. Let $\lambda \in [0, 1]$. Given any $c \geq 1$, let $\tilde{T}$ be a rigid transformation yielding $c$-approximation for minimizing $\mathcal{W}^2_2(S_A, \mathcal{T}(S_B), \lambda)$ in Definition 4. Then,

$$\mathcal{W}^2_2(A, \tilde{T}(B), \lambda) \leq c(1 + 2\epsilon)^2 \cdot \min_{\mathcal{T}} \mathcal{W}^2_2(A, \mathcal{T}(B), \lambda) + 2\epsilon(1 + 1 + 2\epsilon)(1 + 2\epsilon)A^2$$

$$= c(1 + O(\epsilon)) \cdot \min_{\mathcal{T}} \mathcal{W}^2_2(A, \mathcal{T}(B), \lambda) + 2\epsilon(1 + O(\epsilon))(c + 1)A^2. \quad (23)$$

## 3 THE TIME COMPLEXITY

We analyze the time complexity of Algorithm 1 and consider Step 1–3 separately. To simplify our description, we use $n$ to denote $\max\{n_1, n_2\}$. In Step 2, we suppose that the iterative approach [21] takes $h \geq 1$ rounds.

**Step 1.** A straightforward implementation of the Gonzalez’s algorithm is selecting the $k$ cluster centers iteratively with the running time $O(knd)$. Several faster implementations for the high-dimensional case with low doubling dimension have been studied before; their idea is to maintain some data structures to reduce the amortized complexity of each iteration. We refer the reader to Reference [39] for more details. Also note that if we use KCenter+, then it only yields an extra $O(nd)$ time, since we just need to scan the whole data in one-pass for computing the means.

**Step 2.** Since we run the algorithm [21] on the smaller instance $(S_A, S_B)$ instead of $(A, B)$, we know that the complexity of Step 2 is $O(h(\Gamma(k, d) + k^2d + kd^2 + d^3))$ by Proposition 1.

**Step 3.** We need to compute the transformed $\mathcal{T}(B)$ first and then the Wasserstein distance $\mathcal{W}^2_2(A, \mathcal{T}(B))$. Note that the transformation $\mathcal{T}$ is not off-the-shelf, because it is the combination of a sequence of rigid transformations from the iterative approach [21] in Step 2. Since it takes $h$ rounds, $\mathcal{T}$ should be the multiplication of $h$ rigid transformations. We use $(\mathcal{R}_l, \mathcal{V}_l)$ to denote the orthogonal matrix and translation vector obtained in the $l$th round for $1 \leq l \leq h$. We can update $B$ round by round: Starting from $l = 1$, update $B$ to be $\mathcal{R}_l B + \mathcal{V}_l$ in each round; the whole time complexity will be $O(hnd^2)$. In fact, we have a more efficient way by computing $\mathcal{T}$ first before transforming $B$.

**Lemma 4.** Let $(\mathcal{R}, \mathcal{V})$ be the orthogonal matrix and translation vector of $\mathcal{T}$. Then

$$\mathcal{R} = \Pi_{l=1}^{\lambda} \mathcal{R}_l,$$

$$\mathcal{V} = (\Pi_{l=2}^{\lambda} \mathcal{R}_l) \mathcal{V}_1 + (\Pi_{l=3}^{\lambda} \mathcal{R}_l) \mathcal{V}_2 + \cdots + \mathcal{R}_\lambda \mathcal{V}_{\lambda-1} + \mathcal{V}_\lambda, \quad (24)$$

and $\mathcal{T}(B)$ can be obtained in $O(hd^3 + nd^2)$ time.
Suppose \( \Pi_{i=1}^h \mathcal{R}_i \) for \( i = h, h-1, \ldots, 1 \). Consequently, the orthogonal matrix \( \mathcal{R} \) and translation vector \( \mathbf{T} \) can be obtained in \( O(hd^2) \) time. In addition, the complexity for computing \( T(B) = \mathcal{R}B + \mathbf{T} \) is \( O(nd^2) \).

Lemma 4 provides a complexity significantly lower than the previous \( O(hnd^2) \) (usually \( n \) is much larger than \( d \) in practice). After obtaining \( T(B) \), we can compute \( \mathcal{W}_2(A, T(B)) \) in \( \Gamma(n, d) \) time. Note that the complexity \( \Gamma(n, d) \) usually is \( \Omega(n^2d) \), which dominates the complexity of Step 1 and the second term \( nd^2 \) in the complexity of Lemma 4. Overall, we have the following theorem for the total runtime.

**Theorem 4.** Suppose \( n = \max\{n_1, n_2\} \geq d \) and the algorithm of [21] takes \( h \geq 1 \) rounds. The running time of Algorithm 1 is \( O(h(\Gamma(k, d) + k^2d + kd^2 + d^3)) + \Gamma(n, d) \), where \( k = (\frac{2}{\epsilon})^\rho \).

If we run the same number of rounds on the original instance \((A, B)\) by the approach in Reference [21], then the total running time will be \( O(h(\Gamma(n, d) + n^2d)) \) by Proposition 1. When \( k \ll n \), Algorithm 1 achieves a significant reduction on the running time.

### 4 EXPERIMENTS

We consider three applications in our experiments: PPI network alignment, unsupervised bilingual lexicon induction, and domain adaption. All the experimental results were obtained on a server equipped with 2.4-GHz Intel CPUs and 256-GB main memory; the algorithms were implemented in Matlab. For each instance, we run 20 trials and report the average results. Our code is publicly available at [https://github.com/lwjie595/RobustGeometricAlignment](https://github.com/lwjie595/RobustGeometricAlignment).

#### 4.1 Datasets

1. For PPI network alignment, we use the popular benchmark dataset NAPAbench [68] of PPI networks. It consists of several different families of PPI networks generated from the real proteins. The duplication mutation complementation (DMC) [79] and duplication with random mutation (DMR) model [73] are two different node-duplication network growth models, while the crystal growth (CG) model [46] generates the networks by simulating the physics of growing protein crystals. We use three pairs of PPI networks from these three models, where each network is a graph containing 3,000 to 10,000 nodes. In the preprocessing step, we apply the popular node2vec technique [37] to represent each network by a group of vectors in \( \mathbb{R}^{50} \); following the approach of Reference [54], we assign a unit weight to each vector.

2. For unsupervised bilingual lexicon induction, we have five pairs of languages: Chinese–English (zh-en), Spanish–English (es-en), Italian–English (it-en), Japanese–Chinese (ja-zh), and Turkish–English (tr-en). Given the datasets from Reference [85], each language has a vocabulary list containing 3,000 to 13,000 words; we also follow their preprocessing method that represents all the words by the vectors in \( \mathbb{R}^{50} \) through the embedding technique [58]. Each vocabulary list is represented by a distribution in the space where each vector has the weight equal to the corresponding frequency in the language.

3. For domain adaption, we consider the Caltech-Office dataset from Reference [34] that was widely studied before. The Caltech-Office dataset contains 10 categories of images across four different domains: Amazon (A), Caltech10 (C), DSLR (D), and Webcam (W). Each dataset contains 3,000 to 7,000 data items in \( \mathbb{R}^{800} \). We use “→” to denote the transform between two domains, e.g., “C→A” represents the transform from Caltech10 to Amazon. Similarly with the previous research on domain adaptation [23, 34, 42, 66], we use the labeled data in the source domain as the training data and use the k-Nearest Neighbor method as the classifier to predict the labels in the target domain.
4.2 Algorithms for Testing
We use the alignment algorithm [21] (due to its simplicity and practicality) and consider the following data compression methods for comparison:

- **Original**: directly run the alignment algorithm on the original datasets without compression.
- **KCenter**: our proposed method Algorithm 1, i.e., run the Gonzalez’s k-center algorithm to compress the input data.
- **KCenter+**: replace each ball center by the mean point of the cluster (see Section 2.2).
- **KMeans**: replace the Gonzalez’s k-center algorithm by the k-means clustering algorithm [55] for compression (the initialization is implemented by the k-means++ seeding [10]).
- **Random**: sample k points uniformly at random from A and B separately, and each sampled point of A (respectively, B) has the weight \( \frac{W_A}{k} \) (respectively, \( \frac{W_B}{k} \)).
- **Random+**: sample k points uniformly at random from A and B separately; we assign each point of A (respectively, B) to its nearest sampled point; each sampled point of A (respectively, B) has the weight equal to the total weights of the points that assigned to it.
- **StochasticOpt**: the stochastic gradient descent algorithm that uses the batches of subsamples from the two input patterns to compute the optimal transportation and the orthogonal matrices for rigid transformations [36].

4.3 Results
For each instance, we vary the compression rate \( \gamma = \frac{k}{(n_1 + n_2) / 2} \) (recall that \( n_1 \) and \( n_2 \) are respectively the numbers of points of A and B). We also consider the robust alignment with fractional Wasserstein distance. We set the value \( \lambda \in [0.9, 1] \). For PPI network alignment and unsupervised bilingual lexicon induction, we evaluate the performance based on the obtained Wasserstein distance of two patterns and the normalized running time over the time of Original (e.g., if the normalize running time is 0.1, then it means the algorithm saves 90% runtime compared with Original). The runtime of each method includes the time for data compression, the time for alignment on the compressed data, and the time for computing the final Wasserstein flow.

**PPI network alignment.** To see the trends of the algorithms, we first show the results of PPI network alignment on the CG dataset in Figure 3. The compression rate \( \gamma \) ranges from 0.02 to 0.1. Our KCenter+ compression method can achieve the performance close to Original, but it takes significant lower runtimes comparing with other baselines (Random is always the fastest one, since it is just simple uniform sampling; but it always obtained the largest Wasserstein distance). The similar results of the other two PPI datasets are shown in Figures 4 and 5, respectively. The algorithms also achieve the similar performances for the experiments on fractional Wasserstein distance, where the results on the three PPI datasets are shown in Figures 6–8, respectively. We vary the fraction parameter \( \lambda \) from 0.9 to 1, and fix the compression rate \( \gamma \) to be 0.1. The experimental results suggest that our proposed methods KCenter and KCenter+ also work well for fractional Wasserstein distance. To see the influence from the dimension, we also conduct the following experiment on the CG dataset. Previously, we set the dimension to be 50 through node2vec [37]; now we vary the dimension from 50 to 300 by tuning the node2vec algorithm. We show the results of Wasserstein alignment and fraction Wasserstein alignment in Figures 9 and 10, respectively. The compression rate \( \gamma \) is fixed to be 0.1, and the fraction value \( \lambda \) is fixed to be 0.9 for the experiment on fractional Wasserstein distance. We can see that for different dimensions, KCenter+ can achieve the performance close to Original but with much lower running time.
Fig. 3. The Wasserstein distance and normalized running time on CG for PPI network alignment.

Fig. 4. The Wasserstein distance and normalized running time on DMC for PPI network alignment.

Fig. 5. The Wasserstein distance and normalized running time on DMR for PPI network alignment.
Fig. 6. The Wasserstein distance and normalized running time on CG for PPI network alignment with different fraction $\lambda$.

Fig. 7. The Wasserstein distance and normalized running time on DMC for PPI network alignment with different fraction $\lambda$.

Fig. 8. The Wasserstein distance and normalized running time on DMR for PPI network alignment with different fraction $\lambda$. 

ACM Journal of Experimental Algorithmics, Vol. 28, No. 1, Article 1.8. Publication date: August 2023.
Unsupervised bilingual lexicon induction. The parameters $\gamma$ and $\lambda$ for unsupervised bilingual lexicon induction are set to be as same as the experiments for PPI network alignment. We show the results of Wasserstein alignment and fractional Wasserstein alignment on es-en in Figures 11 and 12, respectively. Similarly with the experiments on PPI networks, our KCen- ter+ compression method can achieve the performance close to Original. For the other three unsupervised bilingual lexicon induction pairs, the similar experimental results are shown in Figures 13–16, respectively; the experimental results on fractional Wasserstein distance are shown in Figures 17–20, respectively.

Domain adaptation. The results of Wasserstein distance, normalized time and accuracy for different compression rates are shown in Tables 1–3, respectively. We illustrate the best results with bold in the tables. We can see that KCenter+ performs better than the other five baselines for most cases. We also illustrate the results on fractional Wasserstein distance for domain adaption in Tables 4–6, where their performances are similar to the results in Tables 1–3. We vary the value of $\lambda$ from 0.9 to 1.0, and the compression rate $\gamma$ is fixed to be 0.1.

Experimental conclusion. Overall, our proposed KCenter+ usually outperforms the other baselines and achieves the results close to Original. Although KMeans can also archive similar
Table 1. Wasserstein Distance for DA with Different Compression Level

| Compression | Original | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
|-------------|----------|---------|----------|--------|---------------|--------|---------|
| D→A         |          |         |          |        |               |        |         |
| 0.02        | 17.675   | 20.736  | **20.421** | 20.662 | 22.260        | 21.387 | 21.546  |
| 0.04        | 17.675   | 19.267  | **18.998** | 19.077 | 21.686        | 20.414 | 20.581  |
| 0.06        | 17.675   | 18.851  | **18.098** | 18.293 | 21.008        | 19.726 | 19.973  |
| 0.08        | 17.675   | 18.568  | **17.959** | 18.050 | 20.385        | 19.4   | 19.527  |
| 0.10        | 17.675   | 18.417  | **17.810** | 17.946 | 19.817        | 19.103 | 19.185  |
| D→C         |          |         |          |        |               |        |         |
| 0.02        | 18.915   | 21.114  | **20.976** | 21.313 | 22.588        | 21.845 | 22.102  |
| 0.04        | 18.915   | 19.598  | **19.564** | 20.075 | 22.106        | 21.028 | 21.378  |
| 0.06        | 18.915   | 19.559  | **19.048** | 19.411 | 21.605        | 20.549 | 20.895  |
| 0.08        | 18.915   | 19.464  | **18.987** | 19.224 | 21.125        | 20.181 | 20.483  |
| 0.10        | 18.915   | 19.367  | **18.935** | 19.149 | 20.704        | 19.951 | 20.179  |
| D→W         |          |         |          |        |               |        |         |
| 0.02        | 13.549   | **21.351** | 21.491  | 23.112 | 27.227        | 24.06  | 24.229  |
| 0.04        | 13.549   | 15.914  | **15.616** | 17.458 | 25.318        | 20.721 | 20.866  |
| 0.06        | 13.549   | 14.108  | **13.640** | 14.128 | 23.174        | 18.471 | 18.623  |
| 0.08        | 13.549   | 13.727  | **13.565** | 13.600 | 21.009        | 16.976 | 17.034  |
| 0.10        | 13.549   | 13.566  | **13.530** | 13.532 | 19.171        | 15.923 | 15.829  |
| W→A         |          |         |          |        |               |        |         |
| 0.02        | 12.962   | 17.119  | **17.001** | 17.335 | 18.547        | 17.708 | 17.878  |
| 0.04        | 12.962   | 15.804  | **15.648** | 16.093 | 18.117        | 16.749 | 16.957  |
| 0.06        | 12.962   | 14.726  | **14.442** | 14.930 | 17.661        | 16.036 | 16.27   |
| 0.08        | 12.962   | 13.824  | **13.477** | 14.036 | 17.185        | 15.485 | 15.725  |
| 0.10        | 12.962   | 13.506  | **13.060** | 13.417 | 16.655        | 15.049 | 15.275  |
| W→C         |          |         |          |        |               |        |         |
| 0.02        | 14.080   | **17.483** | 17.552  | 17.867 | 18.830        | 18.126 | 18.337  |
| 0.04        | 14.080   | 16.328  | **16.265** | 16.823 | 18.473        | 17.303 | 17.651  |
| 0.06        | 14.080   | 15.457  | **15.271** | 15.907 | 18.105        | 16.694 | 17.054  |
| 0.08        | 14.080   | 14.782  | **14.459** | 15.071 | 17.729        | 16.237 | 16.560  |
| 0.10        | 14.080   | 14.511  | **14.129** | 14.547 | 17.290        | 15.844 | 16.178  |
| W→D         |          |         |          |        |               |        |         |
| 0.02        | 13.548   | **21.342** | 21.756  | 23.110 | 27.263        | 24.059 | 24.18   |
| 0.04        | 13.548   | 15.939  | **15.633** | 17.378 | 25.315        | 20.74  | 20.869  |
| 0.06        | 13.548   | 14.091  | **13.654** | 14.152 | 23.092        | 18.266 | 18.65   |
| 0.08        | 13.548   | 13.740  | **13.563** | 13.603 | 21.091        | 16.936 | 17.128  |
| 0.10        | 13.548   | 13.564  | **13.529** | 13.547 | 19.301        | 16.018 | 15.962  |

Fig. 11. The Wasserstein distance and normalized running time on es-en for bilingual lexicon induction.
Fig. 12. The Wasserstein distance and normalized running time on es-en for bilingual lexicon induction with different fraction $\lambda$.

Table 2. The Normalized Time for DA with Different Compression Level

| Compression | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
|-------------|---------|----------|--------|---------------|--------|---------|
| $D \rightarrow A$ | 0.02 | 0.142 | 0.145 | 0.241 | 0.134 | 0.13 | 0.133 |
|              | 0.04 | 0.151 | 0.151 | 0.261 | 0.138 | 0.142 | 0.14 |
|              | 0.06 | 0.159 | 0.156 | 0.294 | 0.141 | 0.138 | 0.145 |
|              | 0.08 | 0.162 | 0.169 | 0.293 | 0.150 | 0.147 | 0.15 |
|              | 0.10 | 0.171 | 0.173 | 0.309 | 0.156 | 0.15 | 0.155 |
| $D \rightarrow C$ | 0.02 | 0.140 | 0.140 | 0.242 | 0.126 | 0.127 | 0.126 |
|              | 0.04 | 0.146 | 0.147 | 0.283 | 0.133 | 0.137 | 0.128 |
|              | 0.06 | 0.154 | 0.153 | 0.328 | 0.140 | 0.133 | 0.14 |
|              | 0.08 | 0.160 | 0.161 | 0.320 | 0.144 | 0.138 | 0.145 |
|              | 0.10 | 0.167 | 0.169 | 0.337 | 0.149 | 0.144 | 0.148 |
| $D \rightarrow W$ | 0.02 | 0.184 | 0.177 | 0.235 | 0.189 | 0.183 | 0.186 |
|              | 0.04 | 0.206 | 0.184 | 0.255 | 0.191 | 0.193 | 0.196 |
|              | 0.06 | 0.218 | 0.214 | 0.269 | 0.199 | 0.201 | 0.197 |
|              | 0.08 | 0.221 | 0.224 | 0.309 | 0.212 | 0.203 | 0.207 |
|              | 0.10 | 0.232 | 0.230 | 0.317 | 0.218 | 0.214 | 0.216 |
| $W \rightarrow A$ | 0.02 | 0.144 | 0.146 | 0.251 | 0.134 | 0.133 | 0.131 |
|              | 0.04 | 0.151 | 0.152 | 0.268 | 0.139 | 0.137 | 0.142 |
|              | 0.06 | 0.159 | 0.157 | 0.295 | 0.145 | 0.143 | 0.146 |
|              | 0.08 | 0.164 | 0.169 | 0.312 | 0.153 | 0.147 | 0.153 |
|              | 0.10 | 0.172 | 0.174 | 0.318 | 0.159 | 0.154 | 0.158 |
| $W \rightarrow C$ | 0.02 | 0.145 | 0.146 | 0.262 | 0.129 | 0.13 | 0.13 |
|              | 0.04 | 0.148 | 0.150 | 0.299 | 0.135 | 0.133 | 0.134 |
|              | 0.06 | 0.158 | 0.155 | 0.318 | 0.138 | 0.137 | 0.144 |
|              | 0.08 | 0.168 | 0.165 | 0.336 | 0.148 | 0.142 | 0.145 |
|              | 0.10 | 0.171 | 0.172 | 0.342 | 0.152 | 0.145 | 0.155 |
| $W \rightarrow D$ | 0.02 | 0.195 | 0.191 | 0.248 | 0.195 | 0.194 | 0.197 |
|              | 0.04 | 0.211 | 0.196 | 0.266 | 0.204 | 0.211 | 0.209 |
|              | 0.06 | 0.220 | 0.220 | 0.293 | 0.210 | 0.211 | 0.209 |
|              | 0.08 | 0.231 | 0.232 | 0.317 | 0.222 | 0.215 | 0.224 |
|              | 0.10 | 0.243 | 0.242 | 0.344 | 0.233 | 0.226 | 0.23 |
Fig. 13. The Wasserstein distance and normalized running time on it-en for bilingual lexicon induction.

Table 3. Accuracy for DA with Different Compression Level

| Compression | Original | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
|-------------|----------|---------|---------|--------|---------------|--------|--------|
| D→A         |          |         |         |        |               |        |        |
| 0.02        | 0.779    | 0.803   | 0.792   | 0.785  | 0.785         | 0.79   | 0.783  |
| 0.04        | 0.779    | 0.788   | 0.776   | 0.779  | 0.780         | 0.778  | 0.775  |
| 0.06        | 0.779    | 0.782   | 0.800   | 0.775  | 0.770         | 0.789  | 0.766  |
| 0.08        | 0.779    | 0.771   | 0.787   | 0.774  | 0.767         | 0.778  | 0.765  |
| 0.10        | 0.779    | 0.771   | 0.795   | 0.775  | 0.765         | 0.778  | 0.763  |
| D→C         |          |         |         |        |               |        |        |
| 0.02        | 0.748    | 0.728   | 0.736   | 0.776  | 0.753         | 0.754  | 0.728  |
| 0.04        | 0.748    | 0.699   | 0.736   | 0.771  | 0.769         | 0.757  | 0.722  |
| 0.06        | 0.748    | 0.723   | 0.752   | 0.760  | 0.762         | 0.747  | 0.734  |
| 0.08        | 0.748    | 0.711   | 0.756   | 0.763  | 0.756         | 0.742  | 0.732  |
| 0.10        | 0.748    | 0.727   | 0.741   | 0.756  | 0.750         | 0.746  | 0.731  |
| D→W         |          |         |         |        |               |        |        |
| 0.02        | 0.927    | 0.768   | 0.788   | 0.786  | 0.735         | 0.79   |
| 0.04        | 0.927    | 0.859   | 0.857   | 0.807  | 0.750         | 0.81   |
| 0.06        | 0.927    | 0.890   | 0.892   | 0.864  | 0.779         | 0.853  |
| 0.08        | 0.927    | 0.925   | 0.922   | 0.896  | 0.775         | 0.87   |
| 0.10        | 0.927    | 0.922   | 0.928   | 0.914  | 0.785         | 0.86   |
| W→A         |          |         |         |        |               |        |        |
| 0.02        | 0.678    | 0.612   | 0.659   | 0.649  | 0.609         | 0.59   |
| 0.04        | 0.678    | 0.622   | 0.660   | 0.653  | 0.643         | 0.64   |
| 0.06        | 0.678    | 0.627   | 0.660   | 0.660  | 0.649         | 0.64   |
| 0.08        | 0.678    | 0.668   | 0.683   | 0.664  | 0.643         | 0.64   |
| 0.10        | 0.678    | 0.661   | 0.692   | 0.668  | 0.653         | 0.66   |
| W→C         |          |         |         |        |               |        |        |
| 0.02        | 0.600    | 0.530   | 0.541   | 0.588  | 0.538         | 0.563  |
| 0.04        | 0.600    | 0.544   | 0.566   | 0.598  | 0.551         | 0.57   |
| 0.06        | 0.600    | 0.563   | 0.600   | 0.607  | 0.569         | 0.582  |
| 0.08        | 0.600    | 0.563   | 0.593   | 0.600  | 0.568         | 0.579  |
| 0.10        | 0.600    | 0.572   | 0.599   | 0.602  | 0.579         | 0.587  |
| W→D         |          |         |         |        |               |        |        |
| 0.02        | 0.911    | 0.642   | 0.779   | 0.749  | 0.571         | 0.664  |
| 0.04        | 0.911    | 0.771   | 0.832   | 0.810  | 0.644         | 0.753  |
| 0.06        | 0.911    | 0.853   | 0.856   | 0.848  | 0.662         | 0.794  |
| 0.08        | 0.911    | 0.909   | 0.899   | 0.894  | 0.708         | 0.819  |
| 0.10        | 0.911    | 0.907   | 0.911   | 0.901  | 0.726         | 0.835  |
A Data-dependent Approach for High-dimensional (Robust) Wasserstein Alignment

Fig. 14. The Wasserstein distance and normalized running time on ja-zh for bilingual lexicon induction.

Table 4. Wasserstein Distance for DA with Different Fraction \( \lambda \)

| \( \lambda \) | Original | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
|---|---|---|---|---|---|---|---|
| \( \lambda = 1.0 \) | 17.671 | 18.416 | 17.817 | 17.943 | 19.824 | 19.111 | 19.191 |
| \( \lambda = 0.98 \) | 17.492 | 18.293 | 17.690 | 17.798 | 19.652 | 18.936 | 19.045 |
| \( \lambda = 0.96 \) | 17.394 | 18.189 | 17.546 | 17.633 | 19.534 | 18.749 | 18.845 |
| \( \lambda = 0.94 \) | 17.293 | 18.046 | 17.416 | 17.496 | 19.349 | 18.613 | 18.744 |
| \( \lambda = 0.92 \) | 17.139 | 17.921 | 17.308 | 17.340 | 19.15 | 18.435 | 18.588 |
| \( \lambda = 0.90 \) | 16.972 | 17.761 | 17.142 | 17.176 | 18.71 | 18.258 | 18.477 |
| \( \lambda = 0.98 \) | 18.911 | 19.353 | 18.943 | 19.146 | 20.711 | 19.954 | 20.187 |
| \( \lambda = 0.96 \) | 18.829 | 19.272 | 18.883 | 18.995 | 20.534 | 19.775 | 20.077 |
| \( \lambda = 0.94 \) | 18.726 | 19.182 | 18.746 | 18.713 | 20.22 | 19.422 | 19.806 |
| \( \lambda = 0.92 \) | 18.426 | 18.917 | 18.471 | 18.555 | 20.027 | 19.24 | 19.647 |
| \( \lambda = 0.90 \) | 18.276 | 18.766 | 18.323 | 18.416 | 18.886 | 18.111 | 18.516 |
| \( \lambda = 1.0 \) | 13.555 | 13.554 | 13.531 | 13.536 | 19.239 | 15.999 | 15.919 |
| \( \lambda = 0.98 \) | 13.209 | 13.313 | 13.198 | 13.254 | 18.831 | 15.731 | 15.799 |
| \( \lambda = 0.96 \) | 12.999 | 13.061 | 12.984 | 12.965 | 18.726 | 15.39 | 15.325 |
| \( \lambda = 0.94 \) | 12.670 | 12.807 | 12.678 | 12.650 | 18.304 | 14.955 | 14.907 |
| \( \lambda = 0.92 \) | 12.380 | 12.505 | 12.364 | 12.351 | 18.127 | 14.525 | 14.795 |
| \( \lambda = 0.90 \) | 12.037 | 12.175 | 12.045 | 12.016 | 17.91 | 14.243 | 14.463 |
| \( \lambda = 1.0 \) | 12.960 | 13.523 | 13.063 | 13.413 | 16.657 | 15 | 15.223 |
| \( \lambda = 0.98 \) | 12.815 | 13.379 | 12.903 | 12.352 | 16.51 | 14.846 | 15.074 |
| \( \lambda = 0.96 \) | 12.675 | 13.225 | 12.746 | 13.093 | 16.329 | 14.669 | 14.939 |
| \( \lambda = 0.94 \) | 12.502 | 13.059 | 12.600 | 12.922 | 16.267 | 14.518 | 14.759 |
| \( \lambda = 0.92 \) | 12.364 | 12.923 | 12.453 | 12.818 | 16.141 | 14.309 | 14.638 |
| \( \lambda = 0.90 \) | 12.227 | 12.776 | 12.310 | 12.663 | 15.962 | 14.22 | 14.51 |
| \( \lambda = 1.0 \) | 14.079 | 14.508 | 14.131 | 14.548 | 17.329 | 15.849 | 16.187 |
| \( \lambda = 0.98 \) | 13.923 | 14.372 | 13.985 | 14.400 | 17.123 | 15.695 | 16.028 |
| \( \lambda = 0.96 \) | 13.821 | 14.235 | 13.850 | 14.229 | 16.989 | 15.472 | 15.899 |
| \( \lambda = 0.94 \) | 13.682 | 14.087 | 13.720 | 14.116 | 16.889 | 15.312 | 15.756 |
| \( \lambda = 0.92 \) | 13.525 | 13.960 | 13.583 | 13.985 | 16.715 | 15.154 | 15.655 |
| \( \lambda = 0.90 \) | 13.375 | 13.822 | 13.447 | 13.861 | 16.593 | 15.023 | 15.504 |

ACM Journal of Experimental Algorithmics, Vol. 28, No. 1, Article 1.8. Publication date: August 2023.
Fig. 15. The Wasserstein distance and normalized running time on tr-en for bilingual lexicon induction.

Fig. 16. The Wasserstein distance and normalized running time on zh-en for bilingual lexicon induction.

Fig. 17. The Wasserstein distance and normalized running time on it-en for bilingual lexicon induction with different fraction $\lambda$. 
Table 5. The Normalized Time for DA with Different Fraction $\lambda$

|       | $\lambda = 1.0$ | $\lambda = 0.98$ | $\lambda = 0.96$ | $\lambda = 0.94$ | $\lambda = 0.92$ | $\lambda = 0.90$ |
|-------|-----------------|------------------|------------------|------------------|------------------|------------------|
| $D \rightarrow A$ | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
| $\lambda = 1.0$ | 0.171 | 0.175 | 0.299 | 0.156 | 0.154 | 0.156 |
| $\lambda = 0.98$ | 0.137 | 0.140 | 0.248 | 0.128 | 0.124 | 0.126 |
| $\lambda = 0.96$ | 0.138 | 0.140 | 0.238 | 0.128 | 0.124 | 0.126 |
| $\lambda = 0.94$ | 0.135 | 0.137 | 0.242 | 0.128 | 0.122 | 0.125 |
| $\lambda = 0.92$ | 0.143 | 0.145 | 0.248 | 0.131 | 0.129 | 0.13 |
| $\lambda = 0.90$ | 0.137 | 0.138 | 0.236 | 0.129 | 0.121 | 0.127 |

|       | $\lambda = 1.0$ | $\lambda = 0.98$ | $\lambda = 0.96$ | $\lambda = 0.94$ | $\lambda = 0.92$ | $\lambda = 0.90$ |
|-------|-----------------|------------------|------------------|------------------|------------------|------------------|
| $D \rightarrow C$ | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
| $\lambda = 1.0$ | 0.166 | 0.164 | 0.329 | 0.156 | 0.144 | 0.149 |
| $\lambda = 0.98$ | 0.137 | 0.137 | 0.255 | 0.131 | 0.12 | 0.123 |
| $\lambda = 0.96$ | 0.130 | 0.131 | 0.242 | 0.123 | 0.115 | 0.117 |
| $\lambda = 0.94$ | 0.130 | 0.130 | 0.239 | 0.123 | 0.113 | 0.116 |
| $\lambda = 0.92$ | 0.133 | 0.139 | 0.256 | 0.130 | 0.119 | 0.122 |
| $\lambda = 0.90$ | 0.140 | 0.139 | 0.262 | 0.133 | 0.124 | 0.128 |

|       | $\lambda = 1.0$ | $\lambda = 0.98$ | $\lambda = 0.96$ | $\lambda = 0.94$ | $\lambda = 0.92$ | $\lambda = 0.90$ |
|-------|-----------------|------------------|------------------|------------------|------------------|------------------|
| $D \rightarrow W$ | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
| $\lambda = 1.0$ | 0.234 | 0.245 | 0.337 | 0.225 | 0.219 | 0.225 |
| $\lambda = 0.98$ | 0.275 | 0.274 | 0.374 | 0.256 | 0.256 | 0.259 |
| $\lambda = 0.96$ | 0.192 | 0.197 | 0.263 | 0.181 | 0.178 | 0.179 |
| $\lambda = 0.94$ | 0.184 | 0.183 | 0.248 | 0.173 | 0.168 | 0.172 |
| $\lambda = 0.92$ | 0.184 | 0.181 | 0.255 | 0.173 | 0.167 | 0.174 |
| $\lambda = 0.90$ | 0.181 | 0.180 | 0.254 | 0.177 | 0.173 | 0.175 |

|       | $\lambda = 1.0$ | $\lambda = 0.98$ | $\lambda = 0.96$ | $\lambda = 0.94$ | $\lambda = 0.92$ | $\lambda = 0.90$ |
|-------|-----------------|------------------|------------------|------------------|------------------|------------------|
| $W \rightarrow A$ | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
| $\lambda = 1.0$ | 0.188 | 0.191 | 0.346 | 0.172 | 0.169 | 0.172 |
| $\lambda = 0.98$ | 0.144 | 0.148 | 0.265 | 0.136 | 0.132 | 0.135 |
| $\lambda = 0.96$ | 0.145 | 0.146 | 0.253 | 0.132 | 0.128 | 0.133 |
| $\lambda = 0.94$ | 0.151 | 0.156 | 0.260 | 0.140 | 0.137 | 0.141 |
| $\lambda = 0.92$ | 0.150 | 0.153 | 0.263 | 0.139 | 0.136 | 0.14 |
| $\lambda = 0.90$ | 0.145 | 0.148 | 0.263 | 0.136 | 0.131 | 0.136 |

|       | $\lambda = 1.0$ | $\lambda = 0.98$ | $\lambda = 0.96$ | $\lambda = 0.94$ | $\lambda = 0.92$ | $\lambda = 0.90$ |
|-------|-----------------|------------------|------------------|------------------|------------------|------------------|
| $W \rightarrow C$ | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
| $\lambda = 1.0$ | 0.168 | 0.169 | 0.334 | 0.147 | 0.144 | 0.148 |
| $\lambda = 0.98$ | 0.125 | 0.127 | 0.238 | 0.112 | 0.109 | 0.114 |
| $\lambda = 0.96$ | 0.125 | 0.127 | 0.238 | 0.112 | 0.11 | 0.112 |
| $\lambda = 0.94$ | 0.134 | 0.137 | 0.251 | 0.117 | 0.115 | 0.119 |
| $\lambda = 0.92$ | 0.128 | 0.131 | 0.237 | 0.114 | 0.11 | 0.115 |
| $\lambda = 0.90$ | 0.127 | 0.129 | 0.242 | 0.112 | 0.111 | 0.116 |

|       | $\lambda = 1.0$ | $\lambda = 0.98$ | $\lambda = 0.96$ | $\lambda = 0.94$ | $\lambda = 0.92$ | $\lambda = 0.90$ |
|-------|-----------------|------------------|------------------|------------------|------------------|------------------|
| $W \rightarrow D$ | KCenter | KCenter+ | KMeans | StochasticOpt | Random | Random+ |
| $\lambda = 1.0$ | 0.221 | 0.220 | 0.329 | 0.208 | 0.197 | 0.206 |
| $\lambda = 0.98$ | 0.165 | 0.162 | 0.236 | 0.154 | 0.147 | 0.153 |
| $\lambda = 0.96$ | 0.168 | 0.169 | 0.237 | 0.154 | 0.152 | 0.153 |
| $\lambda = 0.94$ | 0.166 | 0.165 | 0.232 | 0.153 | 0.152 | 0.153 |
| $\lambda = 0.92$ | 0.161 | 0.167 | 0.232 | 0.151 | 0.147 | 0.15 |
| $\lambda = 0.90$ | 0.169 | 0.167 | 0.244 | 0.160 | 0.154 | 0.159 |

good performance as KCenter+, it takes significant higher runtimes than KCenter+. Also, KCenter+ often has a higher classification accuracy for domain adaption than the other baselines (except for Original).

5 CONCLUSION

In this article, we propose a novel framework for compressing point sets in high dimensions, so as to approximately preserve the quality for the Wasserstein alignment. This work is motivated by several emerging applications in the fields of machine learning and bioinformatics. Our method utilizes the property of low doubling dimension, and yields a significant speedup for the computation of alignment. In the experiments, we show that the proposed compression approach can efficiently reduce the running time to a great extent. In the future, it is interesting to consider the alignment...
Fig. 18. The Wasserstein distance and normalized running time on ja-zh for bilingual lexicon induction with different fraction $\lambda$.

Fig. 19. The Wasserstein distance and normalized running time on tr-en for bilingual lexicon induction with different fraction $\lambda$.

Fig. 20. The Wasserstein distance and normalized running time on zh-en for bilingual lexicon induction with different fraction $\lambda$. 
problems for other distances rather than the Wasserstein distance. It is also important to study several other issues of the alignment problems, such as the robustness and parallel implementation.

APPENDIX

A PROOF OF THEOREM 3

Let $T_{opt}$ be the optimal rigid transformation with respect to $\min_{T} \mathcal{W}_2^2(A, T(B), \lambda)$. Since $\tilde{T}$ yields $c$-approximation for minimizing $\mathcal{W}_2^2(S_A, T(S_B), \lambda)$, we have

\[
\mathcal{W}_2^2(S_A, \tilde{T}(S_B), \lambda) \leq c \cdot \min_{T} \mathcal{W}_2^2(S_A, T(S_B), \lambda) \\
\leq c \cdot \mathcal{W}_2^2(S_A, T_{opt}(S_B), \lambda).
\]  

(25)
We denote the flow of $\mathcal{W}_2^2(S_A, \tilde{T}(S_B), \lambda)$ as $\tilde{F} \equiv \{\tilde{f}_{ij}\}$ and the flow of $\mathcal{W}_2^2(A, \tilde{T}(B), \lambda)$ as $\tilde{F} = \{\tilde{f}_{ij}\}$. Then we have

$$\mathcal{W}_2^2(A, \tilde{T}(B), \lambda)$$

$$= \frac{1}{\lambda \min\{W_A, W_B\}} \min_{F} \frac{1}{n_1 \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} \left| |a_i - \tilde{T}(b_j)| \right|^2}$$

$$\leq \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij} \left( 1 + 2\epsilon \right) \left| |a_i' - \tilde{T}(b_j')| \right|^2$$

$$\leq \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \frac{\tilde{f}_{ij} \left( 1 + 2\epsilon \right) \left| |a_i' - \tilde{T}(b_j')| \right|^2}{\lambda \min\{W_A, W_B\}}$$

$$= \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \frac{\tilde{f}_{ij} \left( 1 + 2\epsilon \right) \left| |a_i' - \tilde{T}(b_j')| \right|^2}{\lambda \min\{W_A, W_B\}}$$

By using the similar idea, we denote the flow of $\mathcal{W}_2^2(S_A, \mathcal{T}_{opt}(S_B), \lambda)$ as $F^s \equiv \{f_{ij}^s\}$ and the flow of $\mathcal{W}_2^2(A, \mathcal{T}_{opt}(B), \lambda)$ as $F = \{f_{ij}\}$. Then we have

$$\mathcal{W}_2^2(A, \mathcal{T}_{opt}(B), \lambda)$$

$$= \frac{1}{\lambda \min\{W_A, W_B\}} \min_{F^s} \frac{1}{n_1 \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij}^s \left| |a_i' - \mathcal{T}_{opt}(b_j')| \right|^2}$$

$$\leq \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} \left( 1 + 2\epsilon \right) \left| |a_i' - \mathcal{T}_{opt}(b_j')| \right|^2$$

$$\leq \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \frac{f_{ij} \left( 1 + 2\epsilon \right) \left| |a_i' - \mathcal{T}_{opt}(b_j')| \right|^2}{\lambda \min\{W_A, W_B\}}$$

Combining (25), (26), and (27), we have

$$\mathcal{W}_2^2(A, \tilde{T}(B), \lambda)$$

$$\leq (1 + 2\epsilon) \mathcal{W}_2^2(S_A, \tilde{T}(S_B), \lambda) + (2\epsilon + 4\epsilon^2) \Delta^2$$

$$\leq (1 + 2\epsilon) \cdot c \cdot \mathcal{W}_2^2(S_A, \mathcal{T}_{opt}(S_B), \lambda) + (2\epsilon + 4\epsilon^2) \Delta^2$$

$$\leq c(1 + 2\epsilon) \cdot ((1 + 2\epsilon) \mathcal{W}_2^2(A, \mathcal{T}_{opt}(B), \lambda) + (2\epsilon + 4\epsilon^2) \Delta^2) + (2\epsilon + 4\epsilon^2) \Delta^2$$

$$= c(1 + 2\epsilon)^2 \cdot \mathcal{W}_2^2(A, \mathcal{T}_{opt}(B), \lambda) + 2\epsilon(c + 1 + 2\epsilon)(1 + 2\epsilon) \Delta^2,$$

and the proof is completed.

**ACKNOWLEDGMENTS**

We thank the anonymous reviewers for their helpful comments.
A Data-dependent Approach for High-dimensional (Robust) Wasserstein Alignment

REFERENCES

[1] Prince Osei Aboagye, Yan Zheng, Chin-Chia Michael Yeh, Junpeng Wang, Wei Zhang, Liang Wang, Hao Yang, and Jeff M. Phillips. 2022. Normalization of language embeddings for cross-lingual alignment. In Proceedings of the 10th International Conference on Learning Representations (ICLR’22). OpenReview.net.

[2] Pankaj K. Agarwal, Kyle Fox, Debamalya Panigrahi, Kasturi R. Varadarajan, and Allen Xiao. 2017. Faster algorithms for the geometric transportation problem. In Proceedings of the 33rd International Symposium on Computational Geometry (SoCG’17). 7:1–7:16.

[3] Pankaj K. Agarwal and Kasturi R. Varadarajan. 2004. A near-linear constant-factor approximation for euclidean bipartite matching? In Proceedings of the 20th ACM Symposium on Computational Geometry. 247–252.

[4] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. 1993. Network Flows: Theory, Algorithms, and Applications. Prentice Hall.

[5] Jason Altschuler, Jonathan Weed, and Philippe Rigollet. 2017. Near-linear time approximation algorithms for optimal transport via Sinkhorn iteration. In Proceedings of the Annual Conference on Neural Information Processing Systems. 1964–1974.

[6] David Alvarez-Melis and Tommi S. Jaakkola. 2018. Gromov-wasserstein alignment of word embedding spaces. In Proceedings of the Conference on Empirical Methods in Natural Language Processing. Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsuji (Eds.). Association for Computational Linguistics. 1881–1890.

[7] Alexandr Andoni, Khanh Do Ba, Piotr Indyk, and David Woodruff. 2009. Efficient sketches for earth-mover distance, with applications. In Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS’09). IEEE, 324–330.

[8] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. 2014. Parallel algorithms for geometric graph problems. In Proceedings of the Symposium on Theory of Computing (STOC’14). 574–583.

[9] Alexandr Andoni, Clifford Stein, and Peilin Zhong. 2020. Parallel approximate undirected shortest paths via low hop emulators. In Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC’20). Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy (Eds.). ACM, 322–335.

[10] David Arthur and Sergei Vassilvitskii. 2007. K-means++ the advantages of careful seeding. In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms. 1027–1035.

[11] Mikhail Belkin. 2003. Problems of Learning on Manifolds. The University of Chicago.

[12] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. 2010. A theory of learning from different domains. Mach. Learn. 79, 1-2 (2010), 151–175.

[13] P. J. Beal and Neil D. McKay. 1992. A method for registration of 3-D shapes. IEEE Trans. Pattern Anal. Mach. Intell. 14, 2 (1992), 239–256.

[14] Gaspard Beugnot, Aude Genevay, Kristjan Greenewald, and Justin Solomon. 2021. Improving approximate optimal transport distances using quantization. In Uncertainty in Artificial Intelligence. PMLR, 290–300.

[15] John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman. 2007. Learning bounds for domain adaptation. In Proceedings of the 21st Annual Conference on Neural Information Processing Systems. 129–136.

[16] Sergio Cabello, Panos Giannopoulos, Christian Knauer, and Günter Rote. 2008. Matching point sets with respect to the Earth Mover’s Distance. Comput. Geom. 39, 2 (2008), 118–133.

[17] Xudong Cao, Yichen Wei, Fang Wen, and Jian Sun. 2014. Face alignment by explicit shape regression. Int. J. Comput. Vis. 107, 2 (2014), 177–190.

[18] Laetitia Chapel, Rémi Flamary, Haoran Wu, Cédric Fêvotte, and Gilles Gasso. 2021. Unbalanced optimal transport through non-negative penalized linear regression. In Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS’21). Marc’Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 23270–23282.

[19] Xi Chen, Rajesh Jayaram, Amit Levi, and Erik Waingarten. 2022. New streaming algorithms for high dimensional EMD and MST. In Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing. 222–233.

[20] Scott Cohen and Leonidas Guibas. 1999. The earth mover’s distance under transformation sets. In Proceedings of the 7th IEEE International Conference on Computer Vision. 1.

[21] Nicu D. Cornea, M. Fatih Demirci, Deborah Silver, S. J. Dickinson, and P. B. Kantor. 2005. 3D object retrieval using many-to-many matching of curve skeletons. In Proceedings of the International Conference on Shape Modeling and Applications. IEEE, 366–371.

[22] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. 2016. Optimal transport for domain adaptation. IEEE Trans. Pattern Anal. Mach. Intell. 39, 9 (2016), 1853–1865.
[24] Marco Cuturi. 2013. Sinkhorn distances: Lightspeed computation of optimal transport. In Proceedings of the 27th Annual Conference on Neural Information Processing Systems. 2292–2300.

[25] Sanjoy Dasgupta and Kaushik Sinha. 2013. Randomized partition trees for exact nearest neighbor search. In Proceedings of the Conference on Learning Theory. 317–337.

[26] Hu Ding, Tan Chen, Fan Yang, and Mingyue Wang. 2021. A data-dependent algorithm for querying earth mover’s distance with low doubling dimensions. In Proceedings of the SIAM International Conference on Data Mining (SDM’21), Carlotta Damiani and Ian Davidson (Eds.). SIAM, 630–638.

[27] Hu Ding and Jinhui Xu. 2017. FPTAS for minimizing the earth mover’s distance under rigid transformations and related problems. Algorithmica 78, 3 (2017), 741–770.

[28] Hu Ding and Mingquan Ye. 2019. On geometric alignment in low doubling dimension. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33. 1460–1467.

[29] Kyle Fox and Jiashuai Lu. 2022. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and speed. J. Comput. Geom. 13, 1 (2022), 204–225.

[30] Andrew V. Goldberg and Robert Endre Tarjan. 1989. Finding minimum-cost circulations by canceling negative cycles. J. ACM 36, 4 (1989), 873–886.

[31] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. 2012. Geodesic flow kernel for unsupervised domain adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2066–2073.

[32] Kyle Fox and Jiashuai Lu. 2022. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread. J. Comput. Geom. 13, 1 (2022), 204–225.

[33] Andrew V. Goldberg and Robert Endre Tarjan. 1989. Finding minimum-cost circulations by canceling negative cycles. J. ACM 36, 4 (1989), 873–886.

[34] Piotr Indyk. 2007. A near linear time constant factor approximation for Euclidean bichromatic matching (cost). In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms. Society for Industrial and Applied Mathematics, 39–42.

[35] Piotr Indyk and N. Thaper. 2003. Fast color image retrieval via embeddings. In Workshop on Statistical and Computational Theories of Vision (at ICCV’03).

[36] Kun Jin, Chaoyue Liu, and Cathy Xia. 2021. Two-sided wasserstein procrustes analysis. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI’21). 3515–3521.

[37] Sanjoy Dasgupta and Kaushik Sinha. 2013. Randomized partition trees for exact nearest neighbor search. In Proceedings of the Conference on Learning Theory. 317–337.

[38] Robert Krauthgamer and James R. Lee. 2004. Navigating nets: Simple algorithms for proximity search. In Proceedings of the Conference on Learning Theory. 317–337.

[39] David R. Karger and Matthias Ruhl. 2002. Finding nearest neighbors in growth-restricted metrics. In Proceedings of the 35th Annual Symposium on Theory of Computing. ACM, 741–750.

[40] Andrey Boris Khesin, Aleksandar Nikolov, and Dmitriy Paramonov. 2019. Preconditioning for the geometric transportation problem. In Proceedings of the 55th International Symposium on Computational Geometry. 15:1–15:14.

[41] Wan Kyu Kim and Edward M. Marcotte. 2008. Age-dependent evolution of the yeast protein interaction network suggests a limited role of gene duplication and divergence. PLoS Comput. Biol. 4, 11 (2008), e1000232.

[42] Teofilo F. Gonzalez. 1985. Clustering to minimize the maximum intercluster distance. Theor. Comput. Sci. 38 (1985), 293–306.

[43] Aditya Grover and Jure Leskovec. 2021. Provably approximated ICP. arXiv:2101.03588.

[44] Jihun Ham, Daniel D. Lee, and Lawrence K. Saul. 2005. Semisupervised alignment of manifolds. In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’05). 120–127.

[45] Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. 2012. Geodesic flow kernel for unsupervised domain adaptation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2066–2073.

[46] David R. Karger and Matthias Ruhl. 2002. Finding nearest neighbors in growth-restricted metrics. In Proceedings of the 35th Annual Symposium on Theory of Computing. ACM, 741–750.

[47] Sariel Har-Peled and Manor Mendel. 2006. Fast construction of nets in low-dimensional metrics and their applications. SIAM J. Comput. 35, 5 (2006), 1148–1184.

[48] Kyle Fox and Jiashuai Lu. 2022. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and speed. J. Comput. Geom. 13, 1 (2022), 204–225.

[49] Carlotta Damiani and Ian Davidson (Eds.). SIAM, 630–638.

[50] Thomas Laakso. 2002. Plane with $\mathbb{R}^n$. Bull. Lond. Math. Soc. 34, 6 (2002), 667–676.
A Data-dependent Approach for High-dimensional (Robust) Wasserstein Alignment

François Le Gall. 2012. Faster algorithms for rectangular matrix multiplication. In Proceedings of the IEEE 53rd Annual Symposium on Foundations of Computer Science. IEEE, 514–523.

Yin Tat Lee and Aaron Sidford. 2014. Path finding methods for linear programming: Solving linear programs in Õ(vrank) iterations and faster algorithms for maximum flow. In Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science. 424–433.

Shi Li. 2010. On constant factor approximation for earth mover distance over doubling metrics. CoRR abs/1002.4034.

Yangwei Liu, Hu Ding, Danyang Chen, and Jinhui Xu. 2017. Novel geometric approach for global alignment of PPI networks. In Proceedings of the 31st AAAI Conference on Artificial Intelligence. 31–37.

Stuart Lloyd. 1982. Least squares quantization in PCM. IEEE Trans. Inf. Theory 28, 2 (1982), 129–137.

Noël Malod-Dognin, Kristina Ban, and Nataša Pržulj. 2017. Unified alignment of protein-protein interaction networks. Sci. Rep. 7, 1 (2017), 953.

Davide Maltoni, Dario Maio, Anil K. Jain, and Sail Prabhakar. 2009. Handbook of Fingerprint Recognition. Springer Science & Business Media.

Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. 2013. Exploiting similarities among languages for machine translation. arXiv:1309.4168. Retrieved from https://arxiv.org/abs/1309.4168.

Debarghya Mukherjee, Aritra Guha, Justin M. Solomon, Yuekai Sun, and Mikhail Yurochkin. 2021. Outlier-robust optimal transport. In International Conference on Machine Learning. PMLR, 7850–7860.

Alexander Munteanu and Chris Schwiegelshohn. 2018. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. Künstl. Intell. 32, 1 (2018), 37–53.

Soliman Nasser, Ibrahim Jubran, and Dan Feldman. 2015. Low-cost and faster tracking systems using core-sets for pose-estimation. CoRR abs/1511.09120 (2015).

James B. Orlin. 1988. A faster strongly polynomial minimum cost flow algorithm. In Proceedings of the 20th Annual ACM Symposium on Theory of Computing. 377–387.

James B. Orlin. 1997. A polynomial time primal network simplex algorithm for minimum cost flows. Math. Program. 78, 2 (1997), 109–129.

James B. Orlin, Serge A. Plotkin, and Éva Tardos. 1993. Polynomial dual network simplex algorithms. Math. Program. 60, 1-3 (1993), 255–276.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning.

James B. Orlin, Serge A. Plotkin, and Éva Tardos. 1993. Polynomial dual network simplex algorithms. Math. Program. 60, 1-3 (1993), 255–276.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22, 10 (2010), 1345–1359.

Michael Perrot, Nicolas Courty, Rémi Flamary, and Amaury Habrard. 2016. Mapping estimation for discrete optimal transport. In Advances in Neural Information Processing Systems, 29.

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The earth mover’s distance as a metric for image retrieval. Int. J. Comput. Vis. 40, 2 (2000), 99–121.

Ebrahim Sahraeian Sayed Mohammad and Byung-Jun Yoon. 2012. A network synthesis model for generating protein interaction network families. PloS One 7 (August 2012).

Peter H. Schönenmann. 1966. A generalized solution of the orthogonal procrustes problem. Psychometrika 31, 1 (1966), 1–10.

R. Sharathkumar and Pankaj K. Agarwal. 2012. Algorithms for the transportation problem in geometric settings. In Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA’12). 306–317.

R. Sharathkumar and Pankaj K. Agarwal. 2012. A near-linear time ε-approximation algorithm for geometric bipartite matching. In Proceedings of the 44th Symposium on Theory of Computing Conference (STOC’12). 385–394.

Jonah Sherman. 2017. Generalized preconditioning and undirected minimum-cost flow. In Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms. 772–780.

Ricard V. Solé, Romualdo Pastor-Satorras, Eric Smith, and Thomas B. Kepler. 2002. A model of large-scale proteome evolution. Adv, Complex Syst, 5, 01 (2002), 43–54.

Kunal Talwar. 2004. Bypassing the embedding: Algorithms for low dimensional metrics. In Proceedings of the 36th Annual ACM Symposium on Theory of Computing. 281–290.

Éva Tardos. 1985. A strongly polynomial minimum cost circulation algorithm. Combinatorica 5, 3 (1985), 247–256.

Sinisa Todorovic and Narendra Ahuja. 2008. Region-based hierarchical image matching. Int. J. Comput. Vis. 78, 1 (2008), 47–66.

Pravin M. Vaidya. 1989. Geometry helps in matching. SIAM J. Comput. 18, 6 (1989), 1201–1225.

Kasturi R. Varadarajan and Pankaj K. Agarwal. 1999. Approximation algorithms for bipartite and non-bipartite matching in the plane. In Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms. 805–814.

Alexei Vázquez, Alessandro Flammini, Amos Maritan, and Alessandro Vespignani. 2003. Modeling of protein interaction networks. Complexus 1, 1 (2003), 38–44.

Cédric Villani. 2008. Topics in optimal transportation. Am. Math. Soc. 58 (2008).

Grace Wahba. 1965. A least squares estimate of satellite attitude. SIAM Rev. 7, 3 (1965), 409–409.
[82] C. Wang, P. Krafft, S. Mahadevan, Y. Ma, and Y. Fu. 2011. Manifold alignment. In Manifold Learning: Theory and Applications, CRC Press. 95–120.

[83] Zekun Yin, Haidong Lan, Guangming Tan, Mian Lu, Athanasios V. Vasilakos, and Weiguo Liu. 2017. Computing platforms for big biological data analytics: Perspectives and challenges. Comput. Struct. Biotechnol. J. 15 (2017), 403–411.

[84] Hyejin Youn, Logan Sutton, Eric Smith, Cristopher Moore, Jon F. Wilkins, Ian Maddieson, William Croft, and Tanmoy Bhattacharya. 2016. On the universal structure of human lexical semantics. Proc. Natl. Acad. Sci. U.S.A. 113, 7 (2016), 1766–1771.

[85] Meng Zhang, Yang Liu, Huanbo Luan, and Maosong Sun. 2017. Earth mover’s distance minimization for unsupervised bilingual lexicon induction. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2017). 1934–1945.

Received 23 September 2022; revised 29 March 2023; accepted 27 May 2023