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

Hu Ding[1], Wenjie Liu[1], and Mingquan Ye[2]

[1] University of Science and Technology of China
He Fei, Anhui, China
huding@ustc.edu.cn, lwj1217@mail.ustc.edu.cn
[2] University of Illinois Chicago
Chicago, Illinois, USA
mye9@uic.edu

Abstract. 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 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 paper, 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.

Keywords: Wasserstein distance, Procrustes analysis, doubling dimension, network alignment, unsupervised cross-lingual learning, domain adaptation

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, *etc* [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 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 $n^{\Omega(d)}$ time in the worst case**, where n is the maximum size of the two input patterns and d is the dimensionality, unless $\mathtt{SNP} \subset \mathtt{DTIME}(2^{o(n)})$ [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 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 [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 [14]. First, we consider the alignment problem under rigid transformation, where the work of [14] only considers the static version. Also, we extend our

method to fractional Wasserstein distance but it is unclear whether the theoretical analysis of [14] is also available for the case with outliers.

The rest of the paper is organized as follows. We introduce several important definitions in Section 1.1, and discuss some related works and our main idea in Section 1.2 and Section 1.3 respectively. Then, we present our algorithm, analysis, and the time complexity in Section 2 and Section 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 paper, 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 $\mathcal{W}_2^2$ [80]). *Let $A = \{a_1, a_2, \cdots, a_{n_1}\}$ and $B = \{b_1, b_2, \cdots, b_{n_2}\}$ be two sets of weighted points in $\mathbb{R}^d$ with nonnegative weights α_i and β_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*

$$\mathcal{W}_2^2(A, B) = \frac{1}{\min\{W_A, W_B\}} \min_F \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} ||a_i - b_j||^2, \tag{1}$$

where $F = \{f_{ij}\}$ is a feasible flow from A to B, i.e., each $f_{ij} \geq 0$, $\sum_{i=1}^{n_1} f_{ij} \leq \beta_j$, $\sum_{j=1}^{n_2} f_{ij} \leq \alpha_i$, 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 paper 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 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.

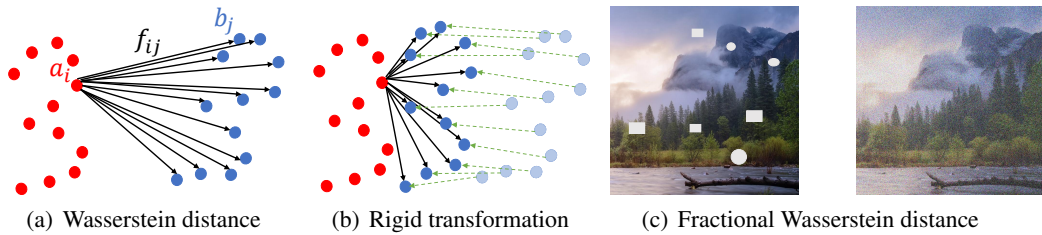(a) Wasserstein distance   (b) Rigid transformation   (c) Fractional Wasserstein distance

Fig. 1: (a) 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.

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 "$\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} = \lambda \cdot \min\{W_A, W_B\}$", we call the distance as the fractional Wasserstein distance and denote it as $\mathcal{W}_2^2(A, B, \lambda)$.*

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 $\mathcal{T}$ on P is a transformation (i.e., rotation, translation, reflection, or their combination) which 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 $\mathcal{T}$ for B so as to minimize the Wasserstein distance $\mathcal{W}_2^2(A, \mathcal{T}(B))$, or $\mathcal{W}_2^2(A, \mathcal{T}(B), \lambda)$ 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 $\texttt{Ball}(x, r) = \{p \in X \mid d_X(x, p) \leq r\}$ 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 ρ, such that for any $x \in X$ and $r \geq 0$, $\texttt{Ball}(x, 2r)$ is always covered by the union of at most 2^ρ 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 ρ-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 [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 $||a_i - b_j||^2$, 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 $O(n^{2.5}\texttt{poly}(\log U))$ 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 $\frac{1}{\epsilon^2}\texttt{poly}(\texttt{log} n)$ time using $\tilde{O}(\frac{1}{\epsilon^2}m)$ expected work.

In the community of machine learning, Cuturi [25] proposed a much faster "Sinkhorn Distance" algorithm which yields an approximation for the Wasserstein distance. Following Cuturi's work, Altschuler *et al.* [5] provided an additive approximation [3] 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 $O(n^{3/2}\epsilon^{-d}\texttt{poly}(\log U)\texttt{poly}(\log n))$. Using the idea of preconditioning [72], Khesin *et al.* [45] developed two randomized $(1 + \epsilon)$-approximation algorithms with the running times $O(n\epsilon^{-O(d)}\log(M)^{O(d)}\log n)$ and $O(n\epsilon^{-O(d)}\log(U)^{O(d)}\log n)$, 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 ρ 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 $\tilde{O}(\log n)$. 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.

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 $\mathcal{T}$ for B (note that we do not know $\mathcal{T}$ in advance), it is difficult to also preserve the Wasserstein distance between A and $\mathcal{T}(B)$. In theory, Cabello *et al.* [16] presented a $(2 + \epsilon)$-approximation solution for the 2D case, and later Klein and Veltkamp [47] achieved an $O(2^{d-1})$-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 [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 $(1 + \epsilon)$-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 paper.

1.3 Overview of The Alignment Procedure

Because the approach of [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*

---

[3] The "additive approximation" means that the computed Wasserstein distance is at most $OPT + \epsilon$, if OPT is the optimal Wasserstein distance.

Point method (ICP) 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

$$\hat{A} = \cup_{i=1}^{n_1} \{a_i^1, a_i^2, \cdots, a_i^{n_2}\}; \tag{2}$$

$$\hat{B} = \cup_{j=1}^{n_2} \{b_j^1, b_j^2, \cdots, b_j^{n_1}\}, \tag{3}$$

where each a_i^j (*resp.*, b_j^i) has the weight f_{ij} and the same spatial position of a_i (*resp.*, b_j). That is, each a_i (*resp.*, b_j) is decomposed into n_2 (*resp.*, n_1) copies. First, we take a translation vector $\overrightarrow{v}$ such that the weighted mean points of $\hat{A}$ and $\hat{B} + \overrightarrow{v}$ 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 $\hat{B} + \overrightarrow{v}$ to minimize its weighted L_2^2 difference to $\hat{A}$. For this purpose, we generate two $d \times (n_1 n_2)$ matrices $\mathbb{M}_A$ and $\mathbb{M}_B$ as follows.

We use $\mathtt{a}_i$ (*resp.*, $\mathtt{b}_j$) to denote the corresponding d-dimensional column vector of a_i (*resp.*, b_j). Each point of $\hat{A}$ (*resp.*, $\hat{B} + \overrightarrow{v}$) corresponds to an individual column of $\mathbb{M}_A$ (*resp.*, $\mathbb{M}_B$); for example, a point $a_i^j \in \hat{A}$ (*resp.*, $b_j^i + \overrightarrow{v} \in \hat{B} + \overrightarrow{v}$) corresponds to a column $\sqrt{f_{ij}}\mathtt{a}_i$ (*resp.*, $\sqrt{f_{ij}}(\mathtt{b}_j + \overrightarrow{v})$) in $\mathbb{M}_A$ (*resp.*, $\mathbb{M}_B$). Formally,

$$\mathbb{M}_A = \big[\sqrt{f_{11}}\mathtt{a}_1, \sqrt{f_{12}}\mathtt{a}_1, \cdots, \sqrt{f_{1n_2}}\mathtt{a}_1, \sqrt{f_{21}}\mathtt{a}_2, \sqrt{f_{22}}\mathtt{a}_2, \cdots, \sqrt{f_{2n_2}}\mathtt{a}_2,$$
$$\cdots, \sqrt{f_{n_1 1}}\mathtt{a}_{n_1}, \sqrt{f_{n_1 2}}\mathtt{a}_{n_1}, \cdots, \sqrt{f_{n_1 n_2}}\mathtt{a}_{n_1}\big]; \tag{4}$$
$$\mathbb{M}_B = \big[\sqrt{f_{11}}(\mathtt{b}_1 + \overrightarrow{v}), \sqrt{f_{12}}(\mathtt{b}_2 + \overrightarrow{v}), \cdots, \sqrt{f_{1n_2}}(\mathtt{b}_{n_2} + \overrightarrow{v}),$$
$$\sqrt{f_{21}}(\mathtt{b}_1 + \overrightarrow{v}), \sqrt{f_{22}}(\mathtt{b}_2 + \overrightarrow{v}), \cdots, \sqrt{f_{2n_2}}(\mathtt{b}_{n_2} + \overrightarrow{v}),$$
$$\cdots, \sqrt{f_{n_1 1}}(\mathtt{b}_1 + \overrightarrow{v}), \sqrt{f_{n_1 2}}(\mathtt{b}_2 + \overrightarrow{v}), \cdots, \sqrt{f_{n_1 n_2}}(\mathtt{b}_{n_2} + \overrightarrow{v})\big]. \tag{5}$$

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

The above idea has a drawback that the matrices $\mathbb{M}_A$ and $\mathbb{M}_B$ are too large and thus the complexity for computing the multiplication $\mathbb{M}_A \times \mathbb{M}_B^T$ can be very high (which is $O(n_1 n_2 d^2)$). Actually we do not need to really construct the large matrices $\mathbb{M}_A$ and $\mathbb{M}_B$, since many of the columns are identical (except for the scalar $\sqrt{f_{ij}}$). Instead, we can compute the multiplication $\mathbb{M}_A \times \mathbb{M}_B^T$ in $O(n_1 n_2 d + \min\{n_1, n_2\} \cdot d^2)$ time. **Note:** Both the complexities "$O(n_1 n_2 d^2)$" and "$O(n_1 n_2 d + \min\{n_1, n_2\} \cdot d^2)$" 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 paper.

Claim 1. The multiplication $\mathbb{M}_A \times \mathbb{M}_B^T$ can be computed in $O(n_1 n_2 d + \min\{n_1, n_2\} \cdot d^2)$ 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, "$F_{i,:}$" 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 $\mathtt{diag}(t)$ to denote the diagonal matrix where the i-th diagonal entry is the i-th entry of

t. For example, if $t = [t_1, t_2, \cdots, t_n]$, then $\sqrt{t} = [\sqrt{t_1}, \sqrt{t_2}, \cdots, \sqrt{t_n}]$ and

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

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

$$\begin{aligned}
\mathbb{M}_A &= \big[\sqrt{f_{11}}\mathbb{a}_1, \sqrt{f_{12}}\mathbb{a}_1, \cdots, \sqrt{f_{1n_2}}\mathbb{a}_1, \sqrt{f_{21}}\mathbb{a}_2, \sqrt{f_{22}}\mathbb{a}_2, \cdots, \sqrt{f_{2n_2}}\mathbb{a}_2, \\
&\qquad \cdots, \sqrt{f_{n_11}}\mathbb{a}_{n_1}, \sqrt{f_{n_12}}\mathbb{a}_{n_1}, \cdots, \sqrt{f_{n_1n_2}}\mathbb{a}_{n_1} \big] \\
&= \big[\mathbb{a}_1\sqrt{F_{1,:}}, \mathbb{a}_2\sqrt{F_{2,:}}, \cdots, \mathbb{a}_{n_1}\sqrt{F_{n_1,:}} \big]; \\
\mathbb{M}_B &= \big[\sqrt{f_{11}}(\mathbb{b}_1 + \overrightarrow{v}), \sqrt{f_{12}}(\mathbb{b}_2 + \overrightarrow{v}), \cdots, \sqrt{f_{1n_2}}(\mathbb{b}_{n_2} + \overrightarrow{v}), \\
&\qquad \sqrt{f_{21}}(\mathbb{b}_1 + \overrightarrow{v}), \sqrt{f_{22}}(\mathbb{b}_2 + \overrightarrow{v}), \cdots, \sqrt{f_{2n_2}}(\mathbb{b}_{n_2} + \overrightarrow{v}), \\
&\qquad \cdots, \sqrt{f_{n_11}}(\mathbb{b}_1 + \overrightarrow{v}), \sqrt{f_{n_12}}(\mathbb{b}_2 + \overrightarrow{v}), \cdots, \sqrt{f_{n_1n_2}}(\mathbb{b}_{n_2} + \overrightarrow{v}) \big] \\
&= \big[\mathbb{b}_1 + \overrightarrow{v}, \cdots, \mathbb{b}_{n_2} + \overrightarrow{v} \big] \times \big[\mathtt{diag}(\sqrt{F_{1,:}}), \mathtt{diag}(\sqrt{F_{2,:}}), \cdots, \mathtt{diag}(\sqrt{F_{n_1,:}}) \big].
\end{aligned}$$

For simplicity, we let $\mathbb{A} = \big[\mathbb{a}_1, \cdots, \mathbb{a}_{n_1} \big]$ and $\mathbb{B} = \big[\mathbb{b}_1 + \overrightarrow{v}, \cdots, \mathbb{b}_{n_2} + \overrightarrow{v} \big]$. Then,

$$\begin{aligned}
\mathbb{M}_A \times \mathbb{M}_B^T &= \sum_{i=1}^{n_1} \big(\mathbb{a}_i\sqrt{F_{i,:}} \big) \times \big(\mathtt{diag}(\sqrt{F_{i,:}})\mathbb{B}^T \big) \\
&= \sum_{i=1}^{n_1} \mathbb{a}_i F_{i,:}\mathbb{B}^T = \mathbb{A}F\mathbb{B}^T.
\end{aligned}$$

The sizes of $\mathbb{A}$, F, and $\mathbb{B}$ are $d \times n_1$, $n_1 \times n_2$, and $d \times n_2$, respectively. So it is easy to see that computing "$\mathbb{A}F\mathbb{B}^T$" takes $O(n_1n_2d + \min\{n_1, n_2\} \cdot d^2)$ time (*e.g.*, if $n_1 \geq n_2$, we compute $\mathbb{A}F$ first and then compute $(\mathbb{A}F) \times \mathbb{B}^T$).

Therefore, the time complexity for obtaining the optimal $\mathcal{R}$ is $O(n_1n_2d + \min\{n_1, n_2\} \cdot d^2 + d^3)$, where the term "$d^3$" comes from the computation of the SVD.

Proposition 1. *Each iteration of the approach of [21] takes $\Gamma(n_1, n_2, d) + O(n_1n_2d + \min\{n_1, n_2\} \cdot d^2 + d^3)$ 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 $\Gamma(n, d) + O(n^2d)$.*

The bottleneck is that the algorithm needs to repeatedly compute the Wasserstein distance and transformation, especially when n and d are large (usually $\Gamma(n, d) = \Omega(n^2d)$). 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 $S = \{c_1, \cdots, c_k\}$, and then P is covered by the k balls $\texttt{Ball}(c_1, r), \cdots, \texttt{Ball}(c_k, r)$ with

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

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 ρ. The diameter of P is denoted by Δ, *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, 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 Δ/r from (6). Now, we need the following Claim 2 from [48, 74]. Actually, the claim can be obtained by recursively applying the definition of doubling dimension.

Claim 2. *Let (X, d_X) be a metric space with the doubling dimension ρ, and $Y \subset X$. If the aspect ratio of Y is upper bounded by some positive value α, then $|Y| \leq 2^{\rho\lceil \log_2 \alpha \rceil}$.*

Replacing X and Y by P and S respectively in the above claim, we have

$$|S| \leq 2^{\rho\lceil \log_2 \Delta/r \rceil} \leq 2^{\rho(1 + \log_2 \Delta/r)}. \tag{7}$$

Since $|S| = k$, (7) implies $r \leq \frac{2}{k^{1/\rho}}\Delta$.

Corollary 1. *If we let $k = (\frac{2}{\epsilon})^\rho$ with some small $\epsilon > 0$, 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 Δ_A and Δ_B respectively. Let $\Delta = \max\{\Delta_A, \Delta_B\}$. We also assume that they both have the doubling dimension at most ρ. 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 $S_A = \{c_1^A, \cdots, c_k^A\}$ and $S_B = \{c_1^B, \cdots, c_k^B\}$ the obtained sets of k-cluster centers with $k = (\frac{2}{\epsilon})^\rho$. For each cluster center c_j^A (*resp.,* 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 (*resp.,* S_B) is equal to W_A (*resp.,* W_B). For the sake of convenience, we name this compression method as KCENTER. 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 1. *Suppose $\epsilon > 0$ is a small parameter in Corollary 1. Given any $c \geq 1$, let $\tilde{\mathcal{T}}$ be a rigid transformation yielding c-approximation for minimizing $\mathcal{W}_2^2(S_A, \mathcal{T}(S_B))$ in Definition 4. Then,*

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

$$= c(1 + O(\epsilon)) \cdot \min_{\mathcal{T}} \mathcal{W}_2^2(A, \mathcal{T}(B)) + 2\epsilon(1 + O(\epsilon))(c + 1)\Delta^2. \tag{8}$$

Proof. First, we denote by $\mathcal{T}_{\texttt{opt}}$ the optimal rigid transformation achieving $\min_{\mathcal{T}} \mathcal{W}_2^2(A, \mathcal{T}(B))$. Since $\tilde{\mathcal{T}}$ yields c-approximation for minimizing $\mathcal{W}_2^2(S_A, \mathcal{T}(S_B))$, we have

$$\mathcal{W}_2^2(S_A, \tilde{\mathcal{T}}(S_B)) \leq c \cdot \min_{\mathcal{T}} \mathcal{W}_2^2(S_A, \mathcal{T}(S_B))$$

$$\leq c \cdot \mathcal{W}_2^2(S_A, \mathcal{T}_{\texttt{opt}}(S_B)). \tag{9}$$

8

Recall that each point c_j^A (*resp.*, c_j^B) has the weight equal to the total weights of the points in the corresponding cluster. For instance, if the cluster contains $\{a_{j(1)}, a_{j(2)}, \cdots, a_{j(h)}\}$, the weight of c_j^A should be $\sum_{l=1}^h \alpha_{j(l)}$; actually, we can view c_j^A as h overlapping points $\{a'_{j(1)}, a'_{j(2)}, \cdots, a'_{j(h)}\}$ with each $a'_{j(l)}$ having the weight $\alpha_{j(l)}$. Therefore, for the sake of convenience, we use another representation for S_A and S_B in our proof below:

$$S_A = \{a'_1, \cdots, a'_{n_1}\} \text{ and } S_B = \{b'_1, \cdots, b'_{n_2}\}, \tag{10}$$

where each a'_j (*resp.*, b'_j) has the weight α_j (*resp.*, β_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 $||a'_i - a_i||, ||b'_j - b_j|| \le \epsilon\Delta$ for any $1 \le i \le n_1$ and $1 \le j \le n_2$, and these bounds are invariant under any rigid transformation in the space. Consequently, for any pair (i, j) and any rigid transformation $\mathcal{T}$, we have

$$
\begin{aligned}
&||a_i - \mathcal{T}(b_j)||^2 \\
&\le \left(||a_i - a'_i|| + ||a'_i - \mathcal{T}(b'_j)|| + ||\mathcal{T}(b'_j) - \mathcal{T}(b_j)||\right)^2 \\
&\le \left(||a'_i - \mathcal{T}(b'_j)|| + 2\epsilon\Delta\right)^2 \\
&= ||a'_i - \mathcal{T}(b'_j)||^2 + 4\epsilon\Delta ||a'_i - \mathcal{T}(b'_j)|| + 4\epsilon^2\Delta^2 \\
&\le ||a'_i - \mathcal{T}(b'_j)||^2 + 2\epsilon\left(\Delta^2 + ||a'_i - \mathcal{T}(b'_j)||^2\right) + 4\epsilon^2\Delta^2 \\
&= (1 + 2\epsilon)||a'_i - \mathcal{T}(b'_j)||^2 + (2\epsilon + 4\epsilon^2)\Delta^2
\end{aligned}
\tag{11}
$$

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

$$||a'_i - \mathcal{T}(b'_j)||^2 \le (1 + 2\epsilon)||a_i - \mathcal{T}(b_j)||^2 + (2\epsilon + 4\epsilon^2)\Delta^2. \tag{12}$$

Based on Definition 1, we denote by $\tilde{F} = \{\tilde{f}_{ij}\}$ the induced flow of $\mathcal{W}_2^2(S_A, \tilde{\mathcal{T}}(S_B))$ (using the representations (10) for S_A and S_B). Then (11) directly implies that

$$
\begin{aligned}
&\mathcal{W}_2^2(A, \tilde{\mathcal{T}}(B)) \\
&\le \frac{1}{\min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij} ||a_i - \tilde{\mathcal{T}}(b_j)||^2 \\
&\le \frac{1 + 2\epsilon}{\min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij} ||a'_i - \tilde{\mathcal{T}}(b'_j)||^2 + (2\epsilon + 4\epsilon^2)\Delta^2 \\
&= (1 + 2\epsilon)\mathcal{W}_2^2(S_A, \tilde{\mathcal{T}}(S_B)) + (2\epsilon + 4\epsilon^2)\Delta^2.
\end{aligned}
\tag{13}
$$

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

$$\mathcal{W}_2^2(S_A, \mathcal{T}_{\mathsf{opt}}(S_B)) \le (1 + 2\epsilon)\mathcal{W}_2^2(A, \mathcal{T}_{\mathsf{opt}}(B)) + (2\epsilon + 4\epsilon^2)\Delta^2. \tag{14}$$

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

$$
\begin{aligned}
\mathcal{W}_2^2(A, \tilde{\mathcal{T}}(B)) &\le (1 + 2\epsilon)\mathcal{W}_2^2(S_A, \tilde{\mathcal{T}}(S_B)) + (2\epsilon + 4\epsilon^2)\Delta^2 \\
&\le (1 + 2\epsilon) \cdot c \cdot \mathcal{W}_2^2(S_A, \mathcal{T}_{\mathsf{opt}}(S_B)) + (2\epsilon + 4\epsilon^2)\Delta^2 \\
&\le c(1 + 2\epsilon)^2 \cdot \mathcal{W}_2^2(A, \mathcal{T}_{\mathsf{opt}}(B)) + 2\epsilon(c + 1 + 2c\epsilon)(1 + 2\epsilon)\Delta^2,
\end{aligned}
\tag{15}
$$

and the proof is completed.

When ϵ is small enough, Theorem 1 shows that $\mathcal{W}_2^2(A, \tilde{\mathcal{T}}(B)) \approx c \cdot \mathcal{W}_2^2(A, \mathcal{T}_{\mathsf{opt}}(B))$. That is, $\tilde{\mathcal{T}}$, 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 would like to emphasize that though we use the algorithm from [21] in Step 3, Theorem 1 is an independent result; that is, any alignment method with the same objective function in Definition 4 can benefit from Theorem 1.

Algorithm 1 Geometric Alignment with KCENTER

Input: An instance (A, B) of the geometric alignment problem in Definition 4 with bounded doubling dimension ρ in $\mathbb{R}^d$; $k \in \mathbb{Z}^+$.

Output: A rigid transformation $\mathcal{T}$ of B and the Wasserstein distance flow between A and $\mathcal{T}(B)$.

1: Run the Gonzalez's k-center clustering algorithm on A and B as described in Theorem 1, and obtain the sets of cluster centers S_A and S_B respectively.
2: Apply the existing alignment algorithm, e.g., the algorithm of [21], on (S_A, S_B).
3: Obtain the rigid transformation $\mathcal{T}$ from Step 2, and compute the corresponding Wasserstein distance flow between A and $\mathcal{T}(B)$.

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 ρ, if we require the compression error (*i.e.,* the radius) to be no larger than $\epsilon\Delta$. However, the exact doubling dimension ρ 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 $(2/\epsilon)^\rho$. In the worst case, k can be as large as $(4/\epsilon)^\rho$ (so the value k is enlarged by a factor 2^ρ).

First, we need to estimate the diameter Δ. In practice we often avoid to compute the exact value of Δ 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

$$\frac{1}{2}\Delta \le \tilde{\Delta} \le \Delta. \tag{16}$$

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

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

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

$$|S| \le 2^{\rho\lceil \log_2 2/\epsilon \rceil} \le 2^{\rho(1+\log_2 2/\epsilon)} = (4/\epsilon)^\rho. \tag{17}$$

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

2.2 Extension II: k-means + k-center

In Theorem 1, 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 1. 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, the obtained clusters may have large radii and thus (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 $\le \epsilon\Delta$ if $k \ge (\frac{2}{\epsilon})^\rho$). 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 "$2\epsilon(c+1+2c\epsilon)(1+2\epsilon)\Delta^2$" in (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 by $\epsilon\Delta$; but if we take a more careful analysis, the formula (11) can be rewritten as follows:

$$
\begin{aligned}
&||a_i - \mathcal{T}(b_j)||^2 \\
&\leq \left(||a_i - a_i'|| + ||a_i' - \mathcal{T}(b_j')|| + ||\mathcal{T}(b_j') - \mathcal{T}(b_j)||\right)^2 \\
&= ||a_i' - \mathcal{T}(b_j')||^2 + 2(||a_i - a_i'|| + ||\mathcal{T}(b_j') - \mathcal{T}(b_j)||) \times ||a_i' - \mathcal{T}(b_j')|| \\
&\quad + (||a_i - a_i'|| + ||\mathcal{T}(b_j') - \mathcal{T}(b_j)||)^2 \\
&\leq ||a_i' - \mathcal{T}(b_j')||^2 + 4\epsilon\Delta \times ||a_i' - \mathcal{T}(b_j')|| + 2||a_i - a_i'||^2 + 2||\mathcal{T}(b_j') - \mathcal{T}(b_j)||^2 \\
&\leq ||a_i' - \mathcal{T}(b_j')||^2 + 2\epsilon\left(\Delta^2 + ||a_i' - \mathcal{T}(b_j')||^2\right) + 2||a_i - a_i'||^2 + 2||\mathcal{T}(b_j') - \mathcal{T}(b_j)||^2 \\
&= (1+2\epsilon)||a_i' - \mathcal{T}(b_j')||^2 + 2\epsilon\Delta^2 + \boxed{2||a_i - a_i'||^2 + 2||\mathcal{T}(b_j') - \mathcal{T}(b_j)||^2}.
\end{aligned}
\tag{18}
$$

Comparing (18) with (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, the accumulated additive errors in (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 $\mathcal{W}_2^2(A, B, \lambda)$.

Algorithm 2 Fractional Wasserstein distance

Input: An instance (A, B) of the Wasserstein distance problem in Definition 2 with the fraction λ ($0 < \lambda \leq 1$).
Output: The fractional Wasserstein distance $\mathcal{W}_2^2(A, B, \lambda)$ 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., [4] or [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 [4] or [25]*
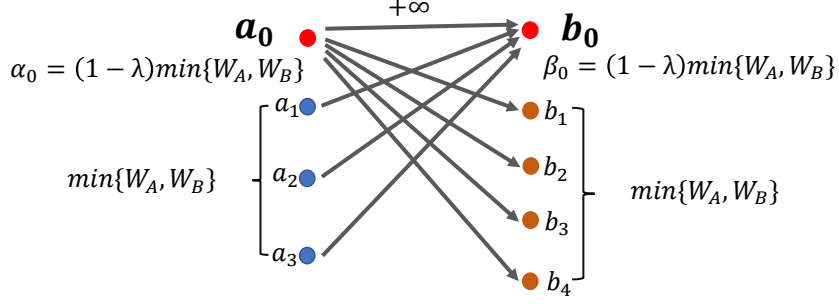
Fig. 2: We add two dummy points a_0 and b_0, where their weights are both $(1 - \lambda) \cdot \min\{W_A, W_B\}$. The point a_0 connects all the b_js with the edge cost 0, and so does b_0; the cost of the edge connecting a_0 and b_0 is $+\infty$. The total flow from the left to the right is $\min\{W_A, W_B\} + (1 - \lambda) \cdot \min\{W_A, W_B\}$. 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.

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 $f_{i_0 j_0}$, from $F \setminus \{f_{00}\}$, and perform the following modification: let $\delta = \min\{f_{00}, f_{i_0 j_0}\}$, and update

$$
\begin{aligned}
f_{00} &\longrightarrow f_{00} - \delta; \\
f_{i_0 j_0} &\longrightarrow f_{i_0 j_0} - \delta; \\
f_{0 j_0} &\longrightarrow f_{0 j_0} + \delta; \\
f_{i_0 0} &\longrightarrow f_{i_0 0} + \delta.
\end{aligned}
\tag{19}
$$

From the edge costs in Figure 2 (the edge cost connecting a_0 and b_0 is ∞), 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: $F' \longrightarrow F' \cup \{f'_{00}, f'_{0j}, f'_{i0} \mid 1 \leq i \leq n_1, 1 \leq j \leq n_2\}$, where

$$
\begin{aligned}
f'_{00} &= 0; \\
f'_{0j} &= \frac{(1 - \lambda) \cdot \min\{W_A, W_B\}}{W_B - \lambda \cdot \min\{W_A, W_B\}} (\beta_j - \sum_{i=1}^{n_1} f'_{ij}); \\
f'_{i0} &= \frac{(1 - \lambda) \cdot \min\{W_A, W_B\}}{W_A - \lambda \cdot \min\{W_A, W_B\}} (\alpha_i - \sum_{j=1}^{n_2} f'_{ij}).
\end{aligned}
\tag{22}
$$

It is easy to verify that the augmented F' is a feasible flow set from $\tilde{A}$ to $\tilde{B}$ 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

12

(21), we know that the augmented F' yields a lower transportation cost between $\tilde{A}$ and $\tilde{B}$, 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 [4] or [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 $\mathcal{T}(S_B)$ (*resp.*, A and $\mathcal{T}(B)$). Similar with Theorem 1, we also have the following Theorem 2 for the partial alignment with any given parameter $\lambda \in [0, 1]$. The proof (which is very similar with that of Theorem 1) is placed in the appendix.

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

$$
\begin{aligned}
&\mathcal{W}_2^2\big(A, \tilde{\mathcal{T}}(B), \lambda\big) \\
&\leq c(1 + 2\epsilon)^2 \cdot \min_{\mathcal{T}} \mathcal{W}_2^2\big(A, \mathcal{T}(B), \lambda\big) + 2\epsilon(c + 1 + 2c\epsilon)(1 + 2\epsilon)\Delta^2 \\
&= c\big(1 + O(\epsilon)\big) \cdot \min_{\mathcal{T}} \mathcal{W}_2^2\big(A, \mathcal{T}(B), \lambda\big) + 2\epsilon\big(1 + O(\epsilon)\big)(c + 1)\Delta^2.
\end{aligned}
\tag{23}
$$

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 [39] for more details. Also note that if we use KCENTER+, 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 $O\big(h\big(\Gamma(k, d) + k^2 d + k d^2 + d^3\big)\big)$ 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 $(\mathcal{R}_l, \overrightarrow{v}_l)$ 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 $\mathcal{R}_l B + \overrightarrow{v}_l$ 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 $(\mathcal{R}, \overrightarrow{v})$ be the orthogonal matrix and translation vector of $\mathcal{T}$. Then*

$$
\begin{aligned}
\mathcal{R} &= \Pi_{l=1}^{\lambda} \mathcal{R}_l, \\
\overrightarrow{v} &= (\Pi_{l=2}^{\lambda} \mathcal{R}_l)\overrightarrow{v}_1 + (\Pi_{l=3}^{\lambda} \mathcal{R}_l)\overrightarrow{v}_2 + \cdots + \mathcal{R}_\lambda \overrightarrow{v}_{\lambda-1} + \overrightarrow{v}_\lambda,
\end{aligned}
\tag{24}
$$

and $\mathcal{T}(B)$ can be obtained in $O(hd^3 + nd^2)$ time.

Proof. The equations (24) can be easily verified by simple calculations and we only need to focus on the time complexity. We can recursively compute the multiplications $\Pi_{l=i}^{\lambda} \mathcal{R}_l$ for $i = h, h-1, \cdots, 1$. Consequently, the orthogonal matrix $\mathcal{R}$ and translation vector $\overrightarrow{v}$ can be obtained in $O(hd^3)$ time. In addition, the complexity for computing $\mathcal{T}(B) = \mathcal{R}B + \overrightarrow{v}$ 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 $\mathcal{T}(B)$, we can compute $\mathcal{W}_2^2(A, \mathcal{T}(B))$ in $\Gamma(n, d)$ time. **Note** that the complexity $\Gamma(n, d)$ usually is $\Omega(n^2 d)$, 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 3. *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 $O\Big(h\big(\Gamma(k, d) + k^2 d + kd^2 + d^3\big)\Big) + \Gamma(n, d)$, where $k = (\frac{2}{\epsilon})^\rho$.*

If we run the same number of rounds on the original instance (A, B) by the approach [21], the total running time will be $O\Big(h\big(\Gamma(n, d) + n^2 d\big)\Big)$ 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 2.4GHz Intel CPUs and 256GB 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.

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 *crystal growth (CG)* model [46] generates the networks by simulating the physics of growing protein crystals. We use 3 pairs of PPI networks from these 3 models, where each network is a graph containing 3000 to 10000 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 [54], we assign a unit weight to each vector.

(2) For unsupervised bilingual lexicon induction, we have 5 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 [85], each language has a vocabulary list containing 3000 to 13000 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 [34] that was widely studied before. The Caltech-Office dataset contains 10 categories of images across 4 different domains: *Amazon* (A), *Caltech10* (C), *DSLR* (D), and *Webcam* (W). Each dataset contains 3000 to 7000 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*. Similar 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 (k-NN) 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 (*resp., B*) has the weight $\frac{W_A}{k}$ (*resp.,* $\frac{W_B}{k}$).
– **RANDOM+:** sample k points uniformly at random from A and B separately; we assign each point of A (*resp., B*) to its nearest sampled point; each sampled point of A (*resp., 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, 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 γ 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 Figure 4 and Figure 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 Figure 6, Figure 7, and Figure 8, respectively. We vary the fraction parameter λ from 0.9 to 1, and fix the compression rate γ 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 Figure 9 and Figure 10, respectively. The compression rate γ is fixed to be 0.1, and the fraction value λ 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.

Unsupervised bilingual lexicon induction. The parameters γ and λ 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 Figure 11 and Figure 12 respectively. Similar with the experiments on PPI networks, our KCENTER+ 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 Figure 13, Figure 14, Figure 15, and Figure 16, respectively; the experimental results on fractional Wasserstein distance are shown in Figure 17, Figure 18, Figure 19, and Figure 20, respectively.

Domain adaptation. The results of Wasserstein distance, normalized time and accuracy for different compression rates are shown in Tables 1, 2, 3, respectively. We illustrate the best results with bold fonts

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, 5, and 6, where their performances are similar to the results in Tables 1-3. We vary the value of λ from 0.9 to 1.0, and the compression rate γ 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 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 paper, 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 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.
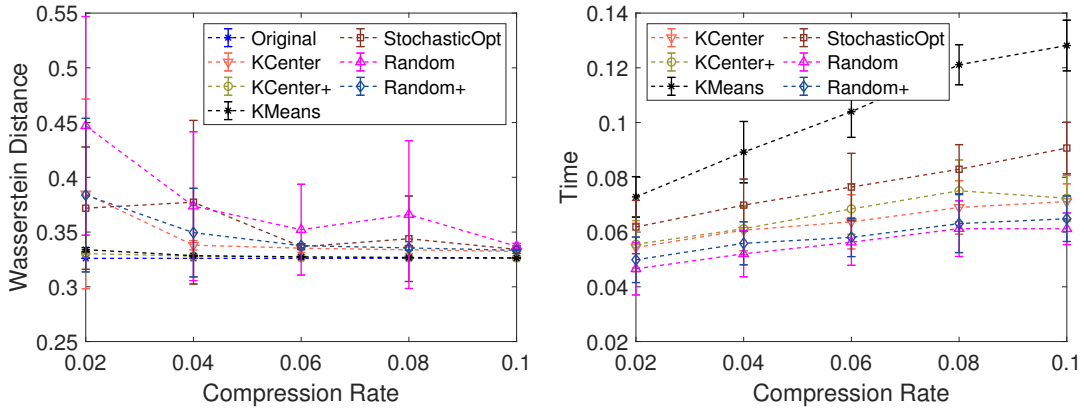

Fig. 3: The Wasserstein distance and normalized running time on CG for PPI network alignment.

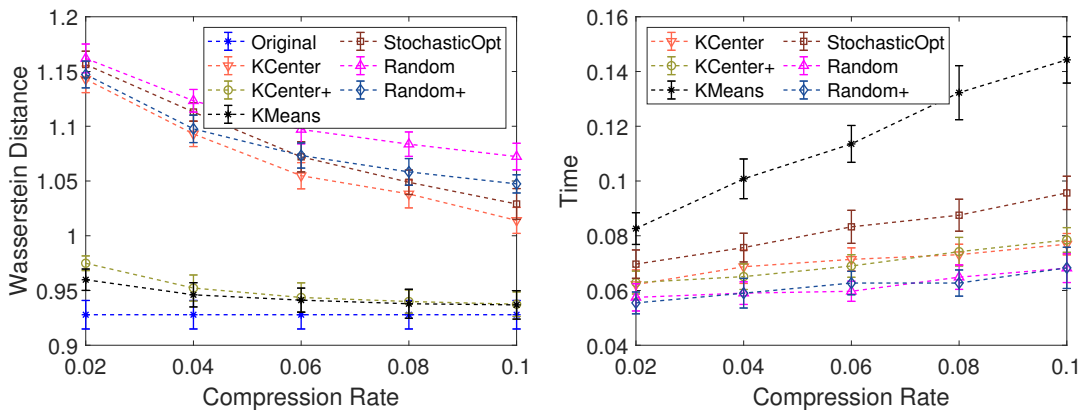Fig. 11: The Wasserstein distance and normalized running time on ES-EN for bilingual lexicon induction.

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 λ.

Fig. 7: The Wasserstein distance and normalized running time on DMC for PPI network alignment with different fraction λ.

Fig. 8: The Wasserstein distance and normalized running time on DMR for PPI network alignment with different fraction λ.

Fig. 9: The normalized Wasserstein distance and normalized running time (over ORIGNIAL) on CG for PPI network alignment with different dimensions.

Fig. 10: The normalized fractional Wasserstein distance and normalized running time (over ORIGNIAL) on CG for PPI network alignment with different dimensions. The fraction λ is 0.9.

Fig. 12: The Wasserstein distance and normalized running time on ES-EN for bilingual lexicon induction with different fraction λ.

Fig. 13: The Wasserstein distance and normalized running time on IT-EN for bilingual lexicon induction.

Fig. 14: The Wasserstein distance and normalized running time on JA-ZH for bilingual lexicon induction.
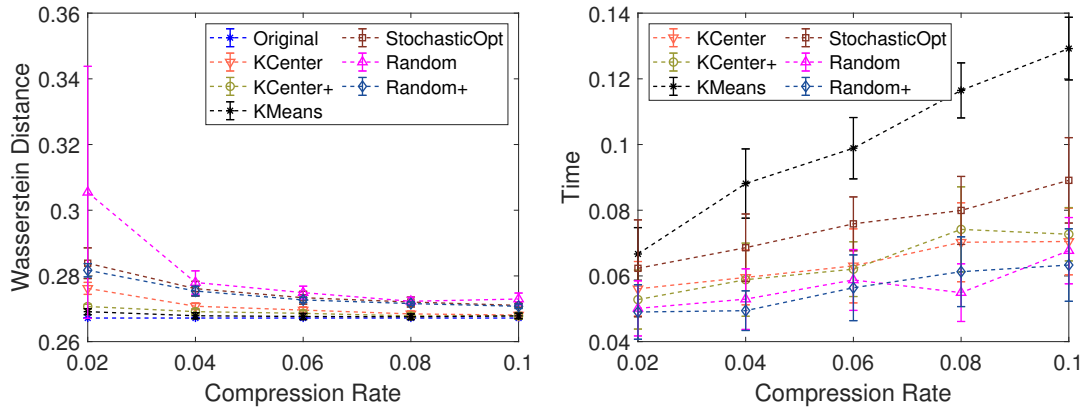

Fig. 15: The Wasserstein distance and normalized running time on TR-EN for bilingual lexicon induction.

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 λ.

Fig. 18: The Wasserstein distance and normalized running time on JA-ZH for bilingual lexicon induction with different fraction λ.

Fig. 19: The Wasserstein distance and normalized running time on TR-EN for bilingual lexicon induction with different fraction λ.

Fig. 20: The Wasserstein distance and normalized running time on ZH-EN for bilingual lexicon induction with different fraction λ.

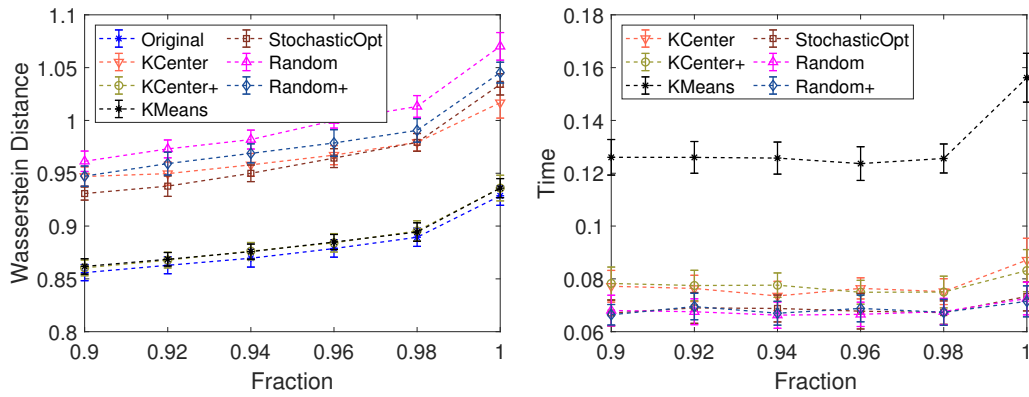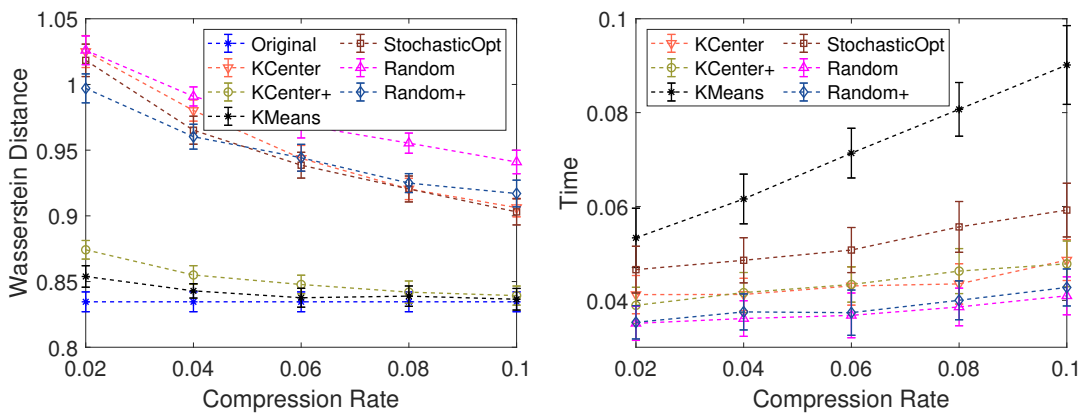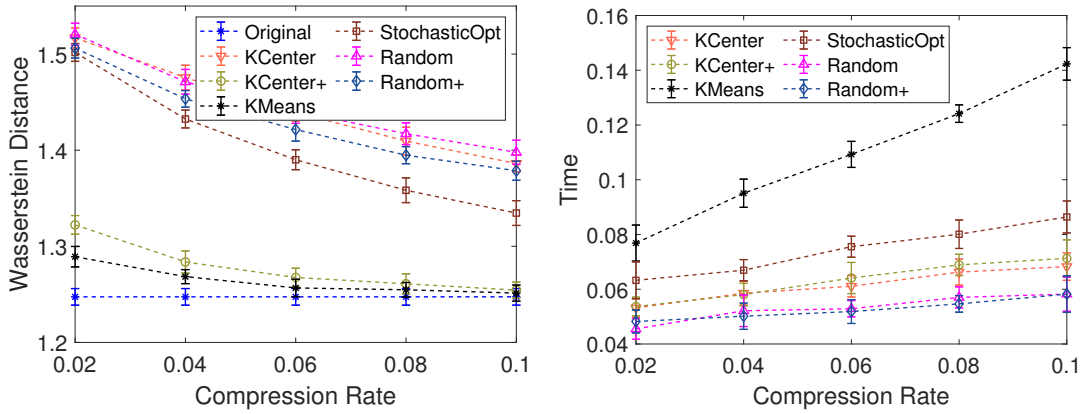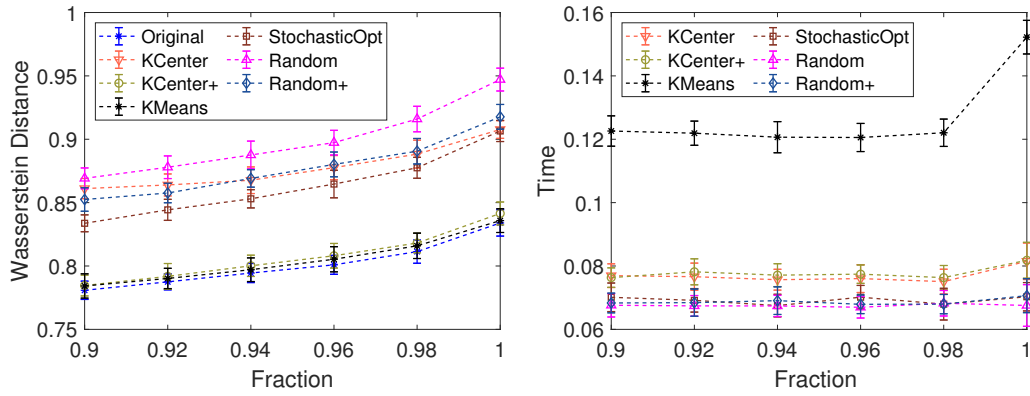Table 1: Wasserstein distance for DA with different compression level

	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	**18.698**	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	19.958	**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**	21.891	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

Table 2: The Normalized Time for DA with different compression level

	Compression	KCENTER	KCENTER+	KMEANS	STOCHASTICOPT	RANDOM	RANDOM+
	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
D→A	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
	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
D→C	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
	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
D→W	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
	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
W→A	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
	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
W→C	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
	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
W→D	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

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	0.783	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	0.8	0.79
	0.04	0.927	**0.859**	0.857	0.807	0.750	0.82	0.816
	0.06	0.927	0.890	**0.892**	0.864	0.779	0.853	0.844
	0.08	0.927	**0.925**	0.922	0.896	0.775	0.87	0.869
	0.10	0.927	0.922	**0.928**	0.914	0.785	0.862	0.868
W→A	0.02	0.678	0.612	**0.659**	0.649	0.609	0.634	0.589
	0.04	0.678	0.622	**0.660**	0.653	0.643	0.63	0.61
	0.06	0.678	0.627	**0.660**	0.660	0.649	0.64	0.62
	0.08	0.678	0.668	**0.683**	0.664	0.643	0.644	0.626
	0.10	0.678	0.661	**0.692**	0.668	0.653	0.656	0.629
W→C	0.02	0.600	0.530	0.541	**0.588**	0.538	0.563	0.536
	0.04	0.600	0.544	0.566	**0.598**	0.551	0.57	0.517
	0.06	0.600	0.563	0.600	**0.607**	0.569	0.582	0.538
	0.08	0.600	0.563	0.593	**0.600**	0.568	0.579	0.546
	0.10	0.600	0.572	0.599	**0.602**	0.579	0.587	0.544
W→D	0.02	0.911	0.642	**0.779**	0.749	0.571	0.664	0.641
	0.04	0.911	0.771	**0.832**	0.810	0.644	0.753	0.737
	0.06	0.911	0.853	**0.856**	0.848	0.662	0.794	0.783
	0.08	0.911	**0.909**	0.899	0.894	0.708	0.819	0.822
	0.10	0.911	0.907	**0.911**	0.901	0.726	0.835	0.838

Table 4: Wasserstein distance for DA with different fraction λ

		ORIGINAL	KCENTER	KCENTER+	KMEANS	STOCHASTICOPT	RANDOM	RANDOM+
D→A	$\lambda = 1.0$	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	19.081	18.258	18.477
D→C	$\lambda = 1.0$	18.911	19.353	**18.943**	19.146	20.711	19.954	20.187
	$\lambda = 0.98$	18.829	19.272	**18.883**	18.995	20.534	19.775	20.077
	$\lambda = 0.96$	18.726	19.182	**18.746**	18.860	20.35	19.599	19.913
	$\lambda = 0.94$	18.553	19.052	**18.611**	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	19.886	19.111	19.516
D→W	$\lambda = 1.0$	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
W→A	$\lambda = 1.0$	12.960	13.523	**13.063**	13.413	16.657	15	15.225
	$\lambda = 0.98$	12.815	13.379	**12.903**	13.252	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
W→C	$\lambda = 1.0$	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
W→D	$\lambda = 1.0$	13.554	13.540	**13.525**	13.527	19.164	15.986	15.864
	$\lambda = 0.98$	13.217	13.291	**13.197**	13.239	18.971	15.807	15.609
	$\lambda = 0.96$	12.998	13.052	12.984	**12.962**	18.595	15.349	15.286
	$\lambda = 0.94$	12.670	12.807	12.681	**12.672**	18.216	15.011	15.032
	$\lambda = 0.92$	12.380	12.501	12.363	**12.341**	18.143	14.667	14.786
	$\lambda = 0.90$	12.037	12.175	12.043	**12.030**	17.909	14.33	14.391

Table 5: The Normalized Time for DA with different fraction λ

		KCENTER	KCENTER+	KMEANS	STOCHASTICOPT	RANDOM	RANDOM+
D→A	$\lambda = 1.0$	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
D→C	$\lambda = 1.0$	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
D→W	$\lambda = 1.0$	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
W→A	$\lambda = 1.0$	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
W→C	$\lambda = 1.0$	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
W→D	$\lambda = 1.0$	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

Table 6: Accuracy for DA with different fraction λ

		ORIGINAL	KCENTER	KCENTER+	KMEANS	STOCHASTICOPT	RANDOM	RANDOM+
D→A	$\lambda = 1.0$	0.779	0.773	**0.794**	0.778	0.764	0.773	0.772
	$\lambda = 0.98$	0.779	0.774	**0.791**	0.774	0.766	0.785	0.765
	$\lambda = 0.96$	0.777	0.774	**0.787**	0.774	0.767	0.778	0.769
	$\lambda = 0.94$	0.775	0.772	**0.795**	0.774	0.765	0.78	0.761
	$\lambda = 0.92$	0.780	0.776	**0.800**	0.774	0.758	0.784	0.766
	$\lambda = 0.90$	0.767	0.776	**0.801**	0.781	0.757	0.78	0.766
D→C	$\lambda = 1.0$	0.748	0.730	0.736	0.754	**0.755**	0.747	0.734
	$\lambda = 0.98$	0.756	0.730	0.746	**0.755**	0.752	0.746	0.73
	$\lambda = 0.96$	0.760	0.733	0.747	**0.761**	0.75	0.751	0.736
	$\lambda = 0.94$	0.758	0.733	0.757	**0.758**	0.753	0.745	0.73
	$\lambda = 0.92$	0.754	0.733	**0.755**	**0.755**	**0.755**	0.747	0.736
	$\lambda = 0.90$	0.749	0.734	0.755	**0.758**	0.753	0.756	0.74
D→W	$\lambda = 1.0$	0.927	0.921	**0.928**	0.917	0.786	0.867	0.885
	$\lambda = 0.98$	0.925	0.924	**0.927**	0.911	0.792	0.867	0.88
	$\lambda = 0.96$	0.919	**0.926**	0.917	0.915	0.768	0.871	0.874
	$\lambda = 0.94$	0.921	**0.925**	0.922	0.914	0.784	0.875	0.88
	$\lambda = 0.92$	0.925	0.923	**0.924**	0.917	0.779	0.876	0.872
	$\lambda = 0.90$	0.927	**0.926**	0.925	0.917	0.78	0.879	0.87
W→A	$\lambda = 1.0$	0.678	0.664	**0.697**	0.670	0.66	0.657	0.631
	$\lambda = 0.98$	0.675	0.661	**0.691**	0.666	0.65	0.642	0.625
	$\lambda = 0.96$	0.672	0.662	**0.693**	0.674	0.647	0.656	0.629
	$\lambda = 0.94$	0.664	0.664	**0.695**	0.669	0.656	0.649	0.626
	$\lambda = 0.92$	0.660	0.668	**0.696**	0.665	0.64	0.66	0.615
	$\lambda = 0.90$	0.660	0.666	**0.690**	0.667	0.648	0.634	0.622
W→C	$\lambda = 1.0$	0.600	0.571	0.597	**0.601**	0.576	0.578	0.544
	$\lambda = 0.98$	0.588	0.571	0.589	**0.601**	0.58	0.577	0.545
	$\lambda = 0.96$	0.597	0.568	0.581	**0.597**	0.579	0.591	0.554
	$\lambda = 0.94$	0.598	0.572	0.584	**0.599**	0.575	0.582	0.551
	$\lambda = 0.92$	0.588	0.572	0.583	**0.597**	0.572	0.572	0.537
	$\lambda = 0.90$	0.585	0.576	0.588	**0.599**	0.578	0.573	0.55
W→D	$\lambda = 1.0$	0.911	0.903	**0.911**	0.903	0.724	0.835	0.844
	$\lambda = 0.98$	0.911	0.910	**0.911**	0.906	0.737	0.828	0.84
	$\lambda = 0.96$	0.917	**0.917**	**0.917**	0.909	0.732	0.825	0.847
	$\lambda = 0.94$	0.917	0.915	**0.917**	0.912	0.715	0.833	0.83
	$\lambda = 0.92$	0.924	0.917	**0.924**	0.913	0.734	0.828	0.848
	$\lambda = 0.90$	0.924	0.918	**0.924**	0.914	0.72	0.831	0.845

6 Acknowledgements

References

1. P. O. Aboagye, Y. Zheng, C. M. Yeh, J. Wang, W. Zhang, L. Wang, H. Yang, and J. M. Phillips. Normalization of language embeddings for cross-lingual alignment. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

2. P. K. Agarwal, K. Fox, D. Panigrahi, K. R. Varadarajan, and A. Xiao. Faster algorithms for the geometric transportation problem. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, pages 7:1–7:16, 2017.

3. P. K. Agarwal and K. R. Varadarajan. A near-linear constant-factor approximation for euclidean bipartite matching? In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 247–252, 2004.

4. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

5. J. Altschuler, J. Weed, and P. Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In *Annual Conference on Neural Information Processing Systems*, pages 1964–1974, 2017.

6. D. Alvarez-Melis and T. S. Jaakkola. Gromov-wasserstein alignment of word embedding spaces. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1881–1890. Association for Computational Linguistics, 2018.

7. A. Andoni, K. Do Ba, P. Indyk, and D. Woodruff. Efficient sketches for earth-mover distance, with applications. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 324–330. IEEE, 2009.

8. A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583, 2014.

9. A. Andoni, C. Stein, and P. Zhong. Parallel approximate undirected shortest paths via low hop emulators. In K. Makarychev, Y. Makarychev, M. Tulsiani, G. Kamath, and J. Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 322–335. ACM, 2020.

10. D. Arthur and S. Vassilvitskii. K-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

11. M. Belkin. *Problems of learning on manifolds*. The University of Chicago, 2003.

12. S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010.

13. P. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

14. G. Beugnot, A. Genevay, K. Greenewald, and J. Solomon. Improving approximate optimal transport distances using quantization. In *Uncertainty in artificial intelligence*, pages 290–300. PMLR, 2021.

15. J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Wortman. Learning bounds for domain adaptation. In *Proc. of the 21st Annual Conference on Neural Information Processing Systems*, pages 129–136, 2007.

16. S. Cabello, P. Giannopoulos, C. Knauer, and G. Rote. Matching point sets with respect to the earth mover's distance. *Computational Geometry*, 39(2):118–133, 2008.

17. X. Cao, Y. Wei, F. Wen, and J. Sun. Face alignment by explicit shape regression. *International Journal of Computer Vision*, 107(2):177–190, 2014.

18. L. Chapel, R. Flamary, H. Wu, C. Févotte, and G. Gasso. Unbalanced optimal transport through non-negative penalized linear regression. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 23270–23282, 2021.

19. L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, and S. Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. *arXiv preprint arXiv:2203.00671*, 2022.

20. X. Chen, R. Jayaram, A. Levi, and E. Waingarten. New streaming algorithms for high dimensional emd and mst. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 222–233, 2022.

21. S. Cohen and L. Guibas. The earth mover's distance under transformation sets. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, page 1, 1999.

22. N. D. Cornea, M. F. Demirci, D. Silver, S. Dickinson, and P. Kantor. 3d object retrieval using many-to-many matching of curve skeletons. In *Shape Modeling and Applications, 2005 International Conference*, pages 366–371. IEEE, 2005.

23. N. Courty, R. Flamary, D. Tuia, and A. Rakotomamonjy. Optimal transport for domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1853–1865, 2016.

24. N. Courty, R. Flamary, D. Tuia, and A. Rakotomamonjy. Optimal transport for domain adaptation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(9):1853–1865, 2017.

25. M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 2292–2300, 2013.

26. S. Dasgupta and K. Sinha. Randomized partition trees for exact nearest neighbor search. In *Conference on Learning Theory*, pages 317–337, 2013.

27. S. Dev, S. Hassan, and J. M. Phillips. Closed form word embedding alignment. *Knowl. Inf. Syst.*, 63(3):565–588, 2021.

28. H. Ding, T. Chen, F. Yang, and M. Wang. A data-dependent algorithm for querying earth mover's distance with low doubling dimensions. In C. Demeniconi and I. Davidson, editors, *Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021*, pages 630–638. SIAM, 2021.

29. H. Ding and J. Xu. FPTAS for minimizing the earth mover's distance under rigid transformations and related problems. *Algorithmica*, 78(3):741–770, 2017.

30. H. Ding and M. Ye. On geometric alignment in low doubling dimension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1460–1467, 2019.

31. D. Feldman. Core-sets: An updated survey. *WIREs Data Mining Knowl. Discov.*, 10(1), 2020.

32. K. Fox and J. Lu. A near-linear time approximation scheme for geometric transportation with arbitrary supplies and spread. *J. Comput. Geom.*, 13(1), 2022.

33. A. V. Goldberg and R. E. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989.

34. B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE conference on computer vision and pattern recognition*, pages 2066–2073. IEEE, 2012.

35. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

36. E. Grave, A. Joulin, and Q. Berthet. Unsupervised alignment of embeddings with wasserstein procrustes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1880–1890. PMLR, 2019.

37. A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

38. J. Ham, D. D. Lee, and L. K. Saul. Semisupervised alignment of manifolds. In *AISTATS*, pages 120–127, 2005.

39. S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

40. P. Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 39–42. Society for Industrial and Applied Mathematics, 2007.

41. P. Indyk and N. Thaper. Fast color image retrieval via embeddings. In *Workshop on Statistical and Computational Theories of Vision (at ICCV)*, 2003.

42. K. Jin, C. Liu, and C. Xia. Two-sided wasserstein procrustes analysis. In *IJCAI*, pages 3515–3521, 2021.

43. I. Jubran, A. Maalouf, R. Kimmel, and D. Feldman. Provably approximated ICP. *CoRR*, abs/2101.03588, 2021.

44. D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 741–750. ACM, 2002.

45. A. B. Khesin, A. Nikolov, and D. Paramonov. Preconditioning for the geometric transportation problem. In *35th International Symposium on Computational Geometry*, pages 15:1–15:14, 2019.

46. W. K. Kim and E. M. Marcotte. Age-dependent evolution of the yeast protein interaction network suggests a limited role of gene duplication and divergence. *PLoS computational biology*, 4(11):e1000232, 2008.

47. O. Klein and R. C. Veltkamp. Approximation algorithms for computing the earth mover's distance under transformations. In *International Symposium on Algorithms and Computation*, pages 1019–1028. Springer, 2005.

48. R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

49. M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

50. T. J. Laakso. Plane with a_∞-weighted metric not bilipschitz embeddable to r^n. *Bulletin of the London Mathematical Society*, 34(6):667–676, 2002.

51. F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd annual symposium on foundations of computer science*, pages 514–523. IEEE, 2012.

52. Y. T. Lee and A. Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science,*, pages 424–433, 2014.

53. S. Li. On constant factor approximation for earth mover distance over doubling metrics. *CoRR*, abs/1002.4034, 2010.

54. Y. Liu, H. Ding, D. Chen, and J. Xu. Novel geometric approach for global alignment of PPI networks. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 31–37, 2017.

55. S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

56. N. Malod-Dognin, K. Ban, and N. Pržulj. Unified alignment of protein-protein interaction networks. *Scientific Reports*, 7(1):953, 2017.

57. D. Maltoni, D. Maio, A. K. Jain, and S. Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

58. T. Mikolov, Q. V. Le, and I. Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013.

59. D. Mukherjee, A. Guha, J. M. Solomon, Y. Sun, and M. Yurochkin. Outlier-robust optimal transport. In *International Conference on Machine Learning*, pages 7850–7860. PMLR, 2021.

60. A. Munteanu and C. Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *Künstliche Intell.*, 32(1):37–53, 2018.

61. S. Nasser, I. Jubran, and D. Feldman. Low-cost and faster tracking systems using core-sets for pose-estimation. *CoRR*, abs/1511.09120, 2015.

62. J. B. Orlin. A faster strongly polynominal minimum cost flow algorithm. In *Proc. of the 20th Annual ACM Symposium on Theory of Computing*, pages 377–387, 1988.

63. J. B. Orlin. A polynomial time primal network simplex algorithm for minimum cost flows. *Mathematical Programming*, 78(2):109–129, 1997.

64. J. B. Orlin, S. A. Plotkin, and É. Tardos. Polynomial dual network simplex algorithms. *Mathematical programming*, 60(1-3):255–276, 1993.

65. S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.*, 22(10):1345–1359, 2010.

66. M. Perrot, N. Courty, R. Flamary, and A. Habrard. Mapping estimation for discrete optimal transport. *Advances in Neural Information Processing Systems*, 29, 2016.

67. Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vis.*, 40(2):99–121, 2000.

68. E. S. Sayed Mohammad and B.-J. Yoon. A network synthesis model for generating protein interaction network families. *PloS one*, 7, August 2012.

69. P. H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

70. R. Sharathkumar and P. K. Agarwal. Algorithms for the transportation problem in geometric settings. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 306–317, 2012.

71. R. Sharathkumar and P. K. Agarwal. A near-linear time ϵ-approximation algorithm for geometric bipartite matching. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 385–394, 2012.

72. J. Sherman. Generalized preconditioning and undirected minimum-cost flow. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 772–780, 2017.

73. R. V. Solé, R. Pastor-Satorras, E. Smith, and T. B. Kepler. A model of large-scale proteome evolution. *Advances in Complex Systems*, 5(01):43–54, 2002.

74. K. Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 281–290, 2004.

75. É. Tardos. A strongly polynomial minimum cost circulation algorithm. *Combinatorica*, 5(3):247–256, 1985.

76. S. Todorovic and N. Ahuja. Region-based hierarchical image matching. *International Journal of Computer Vision*, 78(1):47–66, 2008.

77. P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.

78. K. R. Varadarajan and P. K. Agarwal. Approximation algorithms for bipartite and non-bipartite matching in the plane. In *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 805–814, 1999.

79. A. Vázquez, A. Flammini, A. Maritan, and A. Vespignani. Modeling of protein interaction networks. *Complexus*, 1(1):38–44, 2003.

80. C. Villani. Topics in optimal transportation. *American Mathematical Society*, 58, 2008.

81. G. Wahba. A least squares estimate of satellite attitude. *SIAM review*, 7(3):409–409, 1965.

82. C. Wang, P. Krafft, and S. Mahadevan. Manifold alignment, 2011.

83. Z. Yin, H. Lan, G. Tan, M. Lu, A. V. Vasilakos, and W. Liu. Computing platforms for big biological data analytics: perspectives and challenges. *Computational and structural biotechnology journal*, 15:403–411, 2017.

84. H. Youn, L. Sutton, E. Smith, C. Moore, J. F. Wilkins, I. Maddieson, W. Croft, and T. Bhattacharya. On the universal structure of human lexical semantics. *Proceedings of the National Academy of Sciences*, 113(7):1766–1771, 2016.

85. M. Zhang, Y. Liu, H. Luan, and M. Sun. Earth mover's distance minimization for unsupervised bilingual lexicon induction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 1934–1945, 2017.

A Proof of Theorem 2

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

$$\mathcal{W}_2^2(S_A, \tilde{\mathcal{T}}(S_B), \lambda) \leq c \cdot \min_{\mathcal{T}} \mathcal{W}_2^2(S_A, \mathcal{T}(S_B), \lambda)$$
$$\leq c \cdot \mathcal{W}_2^2(S_A, \mathcal{T}_{\mathsf{opt}}(S_B), \lambda). \tag{25}$$

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

$$\mathcal{W}_2^2(A, \tilde{\mathcal{T}}(B), \lambda)$$
$$= \frac{1}{\lambda \min\{W_A, W_B\}} \min_{\tilde{F}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij} \|a_i - \tilde{\mathcal{T}}(b_j)\|^2$$
$$\leq \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij}^s \|a_i - \tilde{\mathcal{T}}(b_j)\|^2$$
$$\underbrace{\leq}_{\text{by (11)}} \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij}^s \left((1 + 2\epsilon) \|a_i' - \tilde{\mathcal{T}}(b_j')\|^2 + (2\epsilon + 4\epsilon^2)\Delta^2 \right)$$
$$= \frac{(1 + 2\epsilon)}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij}^s (\|a_i' - \tilde{\mathcal{T}}(b_j')\|^2) + (2\epsilon + 4\epsilon^2)\Delta^2 \frac{\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \tilde{f}_{ij}^s}{\lambda \min\{W_A, W_B\}}$$
$$= (1 + 2\epsilon)\mathcal{W}_2^2(S_A, \tilde{\mathcal{T}}(S_B), \lambda) + (2\epsilon + 4\epsilon^2)\Delta^2. \tag{26}$$

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

$$\mathcal{W}_2^2(S_A, \mathcal{T}_{\mathsf{opt}}(S_B), \lambda)$$
$$= \frac{1}{\lambda \min\{W_A, W_B\}} \min_{F^s} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij}^s \|a_i' - \mathcal{T}_{\mathsf{opt}}(b_j')\|^2$$
$$\leq \frac{1}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} f_{ij} \|a_i' - \mathcal{T}_{\mathsf{opt}}(b_j')\|^2$$
$$\underbrace{\leq}_{\text{by(12)}} \frac{1 + 2\epsilon}{\lambda \min\{W_A, W_B\}} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \left(f_{ij} \|a_i - \mathcal{T}_{\mathsf{opt}}(b_j)\|^2 + (2\epsilon + 4\epsilon^2)\Delta^2 \right)$$
$$= (1 + 2\epsilon)\mathcal{W}_2^2(A, \mathcal{T}_{\mathsf{opt}}(B), \lambda) + (2\epsilon + 4\epsilon^2)\Delta^2. \tag{27}$$

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

$$\mathcal{W}_2^2(A, \tilde{\mathcal{T}}(B), \lambda)$$
$$\underbrace{\leq}_{\text{by(26)}} (1 + 2\epsilon)\mathcal{W}_2^2(S_A, \tilde{\mathcal{T}}(S_B), \lambda) + (2\epsilon + 4\epsilon^2)\Delta^2$$
$$\underbrace{\leq}_{\text{by(25)}} (1 + 2\epsilon) \cdot c \cdot \mathcal{W}_2^2(S_A, \mathcal{T}_{\mathsf{opt}}(S_B), \lambda) + (2\epsilon + 4\epsilon^2)\Delta^2$$
$$\underbrace{\leq}_{\text{by(27)}} c(1 + 2\epsilon) \cdot ((1 + 2\epsilon)\mathcal{W}_2^2(A, \mathcal{T}_{\mathsf{opt}}(B), \lambda) + (2\epsilon + 4\epsilon^2)\Delta^2) + (2\epsilon + 4\epsilon^2)\Delta^2$$
$$= c(1 + 2\epsilon)^2 \cdot \mathcal{W}_2^2(A, \mathcal{T}_{\mathsf{opt}}(B), \lambda) + 2\epsilon(c + 1 + 2c\epsilon)(1 + 2\epsilon)\Delta^2, \tag{28}$$

and the proof is completed.