Scalable Nearest Neighbor Search for Optimal Transport

Arturs Backurs [1 2] Yihe Dong [3] Piotr Indyk [4] Ilya Razenshteyn [5] Tal Wagner [4]

Abstract

The Optimal Transport (a.k.a. Wasserstein) distance is an increasingly popular similarity measure for rich data domains, such as images or text documents. This raises the necessity for fast nearest neighbor search algorithms according to this distance, which poses a substantial computational bottleneck on massive datasets. In this work we introduce Flowtree, a fast and accurate approximation algorithm for the Wasserstein-1 distance. We formally analyze its approximation factor and running time. We perform extensive experimental evaluation of nearest neighbor search algorithms in the W_1 distance on real-world dataset. Our results show that compared to previous state of the art, Flowtree achieves up to 7.4 times faster running time.

1. Introduction

Given a finite metric space $\mathcal{M} = (X, d_X)$ and two distributions μ and ν on X, the Wasserstein-1 distance (a.k.a. Earth Mover's Distance or Optimal Transport) between μ and ν is defined as

$$W_1(\mu,\nu) = \min_{\tau} \sum_{x_1,x_2 \in X} \tau(x_1,x_2) \cdot d_X(x_1,x_2), \quad (1)$$

where the minimum is taken over all distributions τ on $X \times X$ whose marginals are equal to μ and ν. The Wasserstein-1 distance and its variants are heavily used in applications to measure similarity in structured data domains, such as images (Rubner et al., 2000) and natural language text (Kusner et al., 2015). In particular, (Kusner et al., 2015) proposed the *Word Mover Distance (WMD)* for text documents. Each document is seen as a uniform distribution over the words it contains, and the underlying metric between words is given by high-dimensional word embeddings such as word2vec (Mikolov et al., 2013) or GloVe (Pennington et al., 2014). It is shown in (Kusner et al., 2015) (see also (Le et al., 2019; Yurochkin et al., 2019; Wu et al., 2018)) that the Wasserstein-1 distance between the two distributions is a high-quality measure of similarity between the associated documents.

To leverage the Wasserstein-1 distance for classification tasks, the above line of work uses the k-nearest neighbor classifier. This poses a notorious bottleneck for large datasets, necessitating the use of fast approximate similarity search algorithms. While such algorithms are widely studied for ℓ_p distances (chiefly ℓ_2; see (Andoni et al., 2018a) for a survey), much less is known for Wasserstein distances, and a comprehensive study appears to be lacking. In particular, two properties of the W_1 distance make the nearest neighbor search problem very challenging. First, the W_1 distance is fairly difficult to compute (the most common approaches are combinatorial flow algorithms (Kuhn, 1955) or approximate iterative methods (Cuturi, 2013)). Second, the W_1 distance is strongly incompatible with Euclidean (and more generally, with ℓ_p) geometries (Bourgain, 1986; Khot & Naor, 2006; Naor & Schechtman, 2007; Andoni et al., 2008; 2015; 2018b), which renders many of the existing techniques for nearest neighbor search inadequate (e.g., random projections).

In this work, we systematically study the k-nearest neighbor search (k-NNS) problem with respect to the W_1 distance. In accordance with the above applications, we focus on the case where the ground set X is a finite subset of $\mathbb{R}^d$, endowed with the Euclidean distance, where d can be a high dimension, and each distribution over X has finite support of size at most s.[1] Given a dataset of n distributions $\mu_1, \mu_2, \ldots, \mu_n$, the goal is to preprocess it, such that given a query distribution ν (also supported on X), we can quickly find the k distributions μ_i closest to ν in the W_1 distance. To speed up search, the algorithms we consider rely on efficient estimates of the distances $W_1(\mu_i, \nu)$. This may lead to retrieving approximate nearest neighbors rather than the exact ones, which is often sufficient for practical

[1]Author names are ordered alphabetically. Code available at `https://github.com/ilyaraz/ot_estimators`. [2]TTIC [3]Microsoft [4]MIT [5]Microsoft Research. Correspondence to: Tal Wagner <talw@mit.edu>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

[1]In the application to (Kusner et al., 2015), X is the set word embeddings of (say) all terms in the English language, and s is the maximum number of terms per text document.

applications.

1.1. Prior work

(Kusner et al., 2015) sped up k-NNS for WMD by designing two approximations of W_1. The first algorithm estimates $W_1(\mu, \nu)$ as the Euclidean distance between their respective means. The second algorithm, called "Relaxed WMD" (abbrev. R-WMD), assigns every point in the support of μ to its closest point in the support of ν, and vice versa, and returns the maximum of the two assignments. Both of these methods produce an estimate no larger than the true distance $W_1(\mu, \nu)$. The former is much faster to compute, while the latter has a much better empirical quality of approximation. The overall k-NNS pipeline in (Kusner et al., 2015) consists of the combination of both algorithms, together with exact W_1 distance computation. Recently, (Atasu & Mittelholzer, 2019) proposed modifications to R-WMD by instating additional capacity constraints, resulting in more accurate estimates that can be computed almost as efficiently as R-WMD.

(Indyk & Thaper, 2003) studied the approximate NNS problem for the W_1 distance in the context of image retrieval. Their approach capitalizes on a long line of work of *tree-based* methods, in which the given metric space is embedded at random into a tree metric. This is a famously fruitful approach for many algorithmic and structural statements (Bartal, 1996; 1998; Charikar et al., 1998; Indyk, 2001; Gupta et al., 2003; Fakcharoenphol et al., 2004; Calinescu et al., 2005; Mendel & Naor, 2006). It is useful in particular for Wasserstein distances, since the optimal flow (τ in (1)) on a tree can be computed in linear time, and since a tree embedding of the underlying metric yields an ℓ_1-embedding of the Wasserstein distance, as shown by (Kleinberg & Tardos, 2002; Charikar, 2002). This allowed (Indyk & Thaper, 2003) to design an efficient NNS algorithm for W_1 based on classical locality-sensitive hashing (LSH). Recently, (Le et al., 2019) introduced a kernel similarity measure based on the same approach, and showed promising empirical results for additional application domains.

1.2. Our results

Flowtree. The tree-based method used in (Indyk & Thaper, 2003; Le et al., 2019) is a classic algorithm called *Quadtree*, described in detail Section 2. In this method, the ground metric X is embedded into a random tree of hypercubes, and the cost of the optimal flow is computed with respect to the tree metric. We suggest a modification to this algorithm, which we call *Flowtree*: It computes the optimal *flow* on the same random tree, but evaluates the *cost* of that flow in the original ground metric.

While this may initially seem like a small modification, it in fact leads to an algorithm with vastly different properties. On one hand, while both algorithms run asymptotically in time $O(s)$, Quadtree is much faster in practice. The reason is that the *cost* of the optimal flow on the tree can be computed very efficiently, without actually computing the flow itself. On the other hand, Flowtree is *dramatically more accurate*. Formally, we prove it has an asymptotically better approximation factor than Quadtree. Empirically, our experiments show that Flowtree is as accurate as state-of-the-art $O(s^2)$ time methods, while being much faster.

Theoretical results. A key difference between Flowtree and Quadtree is that the approximation quality of Flowtree is *independent of the dataset size*, i.e., of the number n of distributions $\mu_1, \ldots, \mu_n$ that need to be searched. Quadtree, on the other hand, degrades in quality as n grows. We expose this phenomenon in two senses:

- *Worst-case analysis:* We prove that Flowtree reports an $O(\log^2 s)$-approximate nearest neighbor w.h.p if the input distributions are uniform, and an $O(\log(d\Phi) \cdot \log s)$-approximate nearest neighbor (where d is the dimension and Φ is the coordinate range of X) even if they are non-uniform. Quadtree, on the other hand, reports an $O(\log(d\Phi) \cdot \log(sn))$-approximate nearest neighbor, and we show the dependence on n is *necessary*.

- *Random model:* We analyze a popular random data model, in which both Flowtree and Quadtree recover the exact nearest neighbor with high probability. Nonetheless, here too, we show that Flowtree's success probability is independent of n, while Quadtree's degrades as n grows.

Empirical results. We evaluate Flowtree, as well as several baselines and state-of-the-art methods, for nearest neighbor search in the W_1 distance on real-world datasets.

Our first set of experiments evaluates each algorithm individually. Our results yield a sharp divide among existing algorithms: The linear time ones are very fast in practice but only moderately accurate, while the quadratic time ones are much slower but far more accurate. Flowtree forms an intermediate category: it is slower and more accurate than the other linear time algorithms, and is at least 5.5 (and up to 30) times faster than the quadratic time algorithms, while attaining similar or better accuracy.

The above results motivate a sequential combination of algorithms, that starts with a fast and coarse algorithm to focus on the most promising candidates nearest neighbors, and gradually refines the candidate list by slower and more accurate algorithms. Such pipelines are commonly used in practice, and in particular were used in (Kusner et al., 2015) (termed "prefetch and prune"). Our second set of experiments evaluates pipelines of various algorithms. We show that incorporating Flowtree into pipelines substantially improves the overall running times, by a factor of up to 7.4.

2. Preliminaries: Quadtree

In this section we describe the classic Quadtree algorithm. Its name comes from its original use in two dimensions (cf. (Samet, 1984)), but it extends to—and has been successfully used in—various high-dimensional settings (e.g. (Indyk, 2001; Indyk et al., 2017; Backurs et al., 2019)). It enjoys a combination of appealing theoretical properties and amenability to fast implementation. As it forms the basis for Flowtree, we now describe it in detail.

Generic Quadtree. Let $X \subset \mathbb{R}^d$ be a finite set of points. Our goal is to embed X into a random tree metric, so as to approximately preserve each pairwise distance in X. To simplify the description, suppose that the minimum pairwise distance in X is exactly 1, and that all points in X have coordinates in $[0, \Phi]$.[2]

The first step is to obtain a randomly shifted hypercube that encloses all points in X. To this end, let $H_0 = [-\Phi, \Phi]^d$ be the hypercube with side length 2Φ centered at the origin. Let $\sigma \in \mathbb{R}^d$ be a random vector with i.i.d. coordinates uniformly distributed in $[0, \Phi]$. We shift H_0 by σ, obtaining the hypercube $H = [-\Phi, \Phi]^d + \sigma$. Observe that H has side length 2Φ and encloses X. The random shift is needed in order to obtain formal guarantees for arbitrary X.

Now, we construct a tree of hypercubes by letting H be the root, halving H along each dimension, and recursing on the resulting sub-hypercubes. We add to the tree only those hypercubes that are non-empty (i.e., contain at least one point from X). Furthermore, we do not partition hypercubes that contain exactly one point from X; they become leaves. The resulting tree has at most $O(\log(d\Phi))$ levels and exactly $|X|$ leaves, one per point in X.[3] We number the root level as $\log \Phi + 1$, and the rest of the levels are numbered downward accordingly ($\log \Phi, \log \Phi - 1, \ldots$). We set the weight of each tree edge between level $\ell + 1$ and level ℓ to be 2^ℓ.

The resulting quadtree has $O(|X|d \cdot \log(d\Phi))$ nodes, and it is straightforward to build it in time $\widetilde{O}(|X|d \cdot \log(d\Phi))$.[4]

Wasserstein-1 on Quadtree. The tree distance between each pair $x, x' \in X$ is defined as the total edge weight on the unique path between their corresponding leaves in the quadtree. Given two distributions μ, ν on X, the Wasserstein-1 distance with this underlying metric (as a proxy for the Euclidean metric on X) admits the closed-form $\sum_v 2^{\ell(v)}|\mu(v) - \nu(v)|$, where v ranges over all nodes in the tree, $\ell(v)$ is the level of v, $\mu(v)$ is the total μ-mass of points enclosed in the hypercube associated with v, and $\nu(v)$ is defined similarly for the ν-mass. If μ, ν have supports of size at most s, then this quantity can be computed in time $O(s \cdot \log(d\Phi))$.

The above closed-form implies, in particular, that W_1 on the quadtree metric embeds isometrically into ℓ_1, as originally observed by (Charikar, 2002) following (Kleinberg & Tardos, 2002). Namely, the ℓ_1 space has a coordinate associated with each tree node v, and a distribution μ is embedded in that space by setting the value of each coordinate v to $2^{\ell(v)}\mu(v)$, where $\mu(v)$ is defined as above. Furthermore, observe that if μ has support size at most s, then its corresponding ℓ_1 embedding w.r.t the tree metric has at most sh non-zero entries, where h is the height of the tree. Thus, computing W_1 on the tree metric amounts to computing the ℓ_1 distance between sparse vectors, which further facilitates fast implementation in practice.

3. Flowtree

The Flowtree algorithm for k-NNS w.r.t. the W_1 distance is as follows. In the preprocessing stage, we build a quadtree T on the ground set X, as described in Section 2. Let $t(x, x')$ denote the quadtree distance between every pair $x, x' \in X$. In the query stage, in order to estimate $W_1(\mu, \nu)$ between two distributions μ, ν, we compute the optimal flow f w.r.t. the tree metric, that is,

$$f = \operatorname{argmin}_{\tilde{f}} \sum_{x,x' \in X} \tilde{f}(x, x') \cdot t(x, x'),$$

where the argmin is taken over all distributions on $X \times X$ with marginals μ, ν. Then, the estimate of the distance between μ and ν is given by

$$\widetilde{W}_1(\mu, \nu) = \sum_{x,x' \in X} f(x, x') \cdot \|x - x'\|.$$

Note that if the support sizes of μ and ν are upper-bounded by s, then the Flowtree estimate of their distance can be computed in time linear in s (see proof in appendix).

Lemma 1. *$\widetilde{W}_1(\mu, \nu)$ can be computed in time $O(s(d + \log(d\Phi)))$.*

Unlike Quadtree, Flowtree does not reduce to sparse ℓ_1 distance computation. Instead, one needs to compute the optimal flow tree f explicitly by bottom-up greedy algorithm, and then use it to compute $\widetilde{W}_1(\mu, \nu)$. On the other hand, Flowtree has the notable property mentioned earlier: its NNS approximation factor is *independent* of the

[2]This is without loss of generality, as we can set the minimum distance to 1 by scaling, and we can shift all the points to have non-negative coordinates without changing internal distances.

[3]This is since the diameter of the root hypercube H is $\sqrt{d}\Phi$, and the diameter of a leaf is no less than $1/2$, since by scaling the minimal distance in X to 1 we have assured that a hypercube of diameter $1/2$ contains a single point and thus becomes a leaf. Since the diameter is halved in each level, there are at most $O(\log(d\Phi))$ levels.

[4]Note that although the construction partitions each hypercube into 2^d sub-hypercubes, eliminating empty hypercubes ensures that the tree size does not depend exponentially on d.

dataset size n. In comparison, the classic Quadtree does not possess this property, and its accuracy deteriorates as the dataset becomes larger. We formally establish this distinction in two senses: first by analyzing worst-case bounds, and then by analyzing a popular random data model.

3.1. Worst-case bounds

We start with an analytic worst-case bound on the performance of quadtree. Let us recall notation: X is a finite subset of $\mathbb{R}^d$, and $\Phi > 0$ is the side length of a hypercube enclosing X. We are given a dataset of n distributions $\mu_1, \ldots, \mu_n$, and a query distribution ν, where each of these distributions is supported on a subset of X of size at most s. Our goal is to find a near neighbor of ν among $\mu_1, \ldots \mu_n$. A distribution μ_i is called a *c-approximate nearest neighbor* of ν if $W_1(\mu_i, \nu) \leq c \cdot \min_{i^*} W_1(\mu_{i^*}, \nu)$.

The following theorem is an adaptation of a result by (Andoni et al., 2008) (where it is proven for a somewhat different algorithm, with similar analysis). All proofs are deferred to the appendix.

Theorem 1 (Quadtree upper bound)**.** *With probability* ≥ 0.99*, the nearest neighbor of* ν *among* $\mu_1, \ldots \mu_n$ *in the Quadtree distance is an* $O(\log(\min\{sn, |X|\}) \log(d\Phi))$*-approximate nearest neighbor in the* W_1 *distance.*

Next, we show that the $\log n$ factor in the above upper bound is *necessary* for Quadtree.

Theorem 2 (Quadtree lower bound)**.** *Suppose c is such that Quadtree is guaranteed to return a c-approximate nearest neighbor, for any dataset, with probability more than (say)* $1/2$*. Then* $c = \Omega(\log n)$*.*

In contrast, Flowtree attains an approximation factor that does not depend on n.

Theorem 3 (Flowtree upper bound)**.** *With probability* ≥ 0.99*, the nearest neighbor of* ν *among* $\mu_1, \ldots \mu_n$ *in the Flowtree distance is an* $O(\log(s) \log(d\Phi))$*-approximate nearest neighbor for the* W_1 *distance.*

Finally, we combine ideas from (Andoni et al., 2008) and (Bačkurs & Indyk, 2014) to prove another upper bound for Flowtree, which is also independent of the dimension d and the numerical range Φ. No such result is known for Quadtree (nor does it follow from our techniques).

Theorem 4 (Flowtree upper bound for uniform distributions[5])**.** *For an integer s, assume that for every distribution there exists an integer $s' \leq s$ such that the weights of all elements in the support are integer multiples of* $1/s'$*. With probability* ≥ 0.99*, the nearest neighbor of* ν *among* $\mu_1, \ldots \mu_n$ *in the Flowtree distance is an* $O(\log^2 s)$*-approximate nearest neighbor for the* W_1 *distance.*

[5]For simplicity, Theorem 4 is stated for uniform distribution (or close to uniform), such as documents in (Kusner et al., 2015). A similar result holds for any distribution, with additional dependence on the numerical range of mass values.

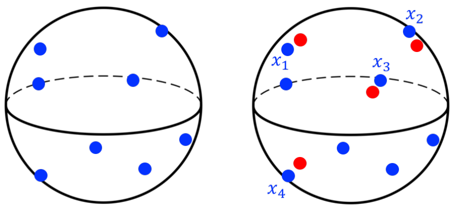


Figure 1: Random model illustration with $s = 4$. Left: The blue points are the N random data points. The data distributions are all subsets of 4 points. Right: The red points form a query distribution whose planted nearest neighbor is the distribution supported on $\{x_1, x_2, x_3, x_4\}$.

3.2. Random model

The above worst-case results appear to be overly pessimistic for real data. Indeed, in practice we observe that Quadtree and especially Flowtree often recover the exact nearest neighbor. This motivates us to study their performance on a simple model of random data, which is standard in the study of nearest neighbor search.

The data is generated as follows. We choose a ground set X of N points i.i.d. uniformly at random on the d-dimensional unit sphere $\mathcal{S}^{d-1}$. For each subset of N of size s, we form a uniform distribution supported on that subset. These distributions make up the dataset $\mu_1, \ldots, \mu_n$ (so $n = \binom{N}{s}$).

To generate a query, pick any μ_i as the "planted" nearest neighbor, and let $x_1, \ldots, x_s$ denote its support. For $k = 1, \ldots, s$, choose a uniformly random point y_k among the points on $\mathcal{S}^{d-1}$ at distance at most ϵ from x_k, where ϵ is a model parameter. The query distribution ν is defined as the uniform distribution over $y_1, \ldots, y_s$. By known concentration of measure results, the distance from y_k to every point in X except x_k is $\sqrt{2} - o(1)$ with high probability. Thus, the optimal flow from ν to μ_i is the perfect matching $\{(x_k, y_k)\}_{k=1}^s$, and μ_i is the nearest neighbor of ν. The model is illustrated in Figure 1.

Theorem 5. *In the above model, the success probability of Quadtree in recovering the planted nearest neighbor decays exponentially with N, while the success probability of Flowtree is independent of N.*

4. Experiments

In this section we empirically evaluate Flowtree and compare it to various existing methods.

Table 1: Dataset properties. Avg. s is the average support size of the distributions in the dataset.

Name	Size	Queries	Underlying metric	Avg. s
20news	$11,314$	$1,000$	Word embedding	115.9
Amazon	$10,000$	$1,000$	Word embedding	57.44
MNIST	$60,000$	$10,000$	2D Euclidean	150.07

4.1. Synthetic data

We implement the random model from Section 3.2. The results are in Figure 2. The x-axis is N (the number of points in the ground metric), and the y-axis is the fraction of successes over 100 independent repetitions of planting a query and recovering its nearest neighbor. As predicted by Theorem 5, Quadtree's success rate degrades as N increases (and we recall that $n = \binom{N}{s}$), while Flowtree's does not.

4.2. Real data

Datasets. We use three datasets from two application domains. Their properties are summarized in Table 1.

- *Text documents:* We use the standard benchmark 20news dataset of news-related online discussion groups, and a dataset of Amazon reviews split evenly over 4 product categories. Both have been used in (Kusner et al., 2015) to evaluate the Word-Move Distance. Each document is interpreted as a uniform distribution supported on the terms it contains (after stopword removal). For the underlying metric, we use GloVe word embeddings (Pennington et al., 2014) with $400,000$ terms and 50 dimensions.
- *Image recognition:* We use the MNIST dataset of handwritten digits. As in (Cuturi, 2013), each image is interpreted as a distribution over 28×28 pixels, with mass proportional to the greyscale intensity of the pixel (normalized so that the mass sums to 1). Note that the distribution is supported on only the non-white pixels in the image. The underlying metric is the 2-dimensional Euclidean distance between the 28×28 pixels, where they are identified with the points $\{(i,j)\}_{i,j=1}^{28}$ on the plane.

Algorithms. We evaluate the following algorithms:

- *Mean:* $W_1(\mu,\nu)$ is estimated as the Euclidean distance between the means of μ and ν. This method has been suggested and used in (Kusner et al., 2015).[6]
- *Overlap:* A simple baseline that estimates $W_1(\mu,\nu)$ by the size of the intersection of their supports.
- *TF-IDF:* A well-known similarity measure for text documents. It is closely related to Overlap.[7] For MNIST we omit this baseline since it is not a text dataset.
- *Quadtree:* See Section 2.
- *Flowtree:* See Section 3.
- *R-WMD:* The Relaxed WMD method of (Kusner et al., 2015), described in Section 1.1. We remark that this method does not produce an admissible flow (i.e., it does not adhere to the capacity and demand constraints of W_1).
- *ACT-1*: The Approximate Constrained Transfers method of (Atasu & Mittelholzer, 2019) gradually adds constraints to R-WMD over i iterations, for a parameter i. The $i = 0$ case is identical to R-WMD, and increasing i leads to increasing both the accuracy and the running time. Like R-WMD, this method does not produce an admissible flow. In our experiments, the optimal setting for this method is $i = 1$,[8] which we denote by ACT-1. The appendix contains additional results for larger i.
- *Sinkhorn with few iterations:* The iterative Sinkhorn method of (Cuturi, 2013) is designed to converge to a near-perfect approximation of W_1. Nonetheless, it can be adapted into a fast approximation algorithm by invoking it with a fixed small number of iterations. We use 1 and 3 iterations, referred to as Sinkhorn-1 and Sinkhorn-3 respectively. Since the Sinkhorn method requires tuning certain parameters (the number of iterations as well as the regularization parameter), the experiments in this section evaluate the method at its optimal setting, and the appendix includes experiments with other parameter settings.

As mentioned in Section 1.2, these methods can be grouped by their running time dependence on s:

- *"Fast" linear-time:* Mean, Overlap, TF-IDF, Quadtree
- *"Slow" linear-time:* Flowtree
- *Quadratic time:* R-WMD, ACT-1, Sinkhorn

The difference between "fast" and "slow" linear time is that the former algorithms reduce to certain simple cache-efficient operations, and furthermore, Mean greatly benefits from SIMD vectorization. In particular, Overlap, TF-IDF and Quadtree require computing a single ℓ_1 distance between sparse vectors, while Mean requires computing a single Euclidean distance in the ground metric. This renders them an order of magnitude faster than the other methods, as our empirical results will show.

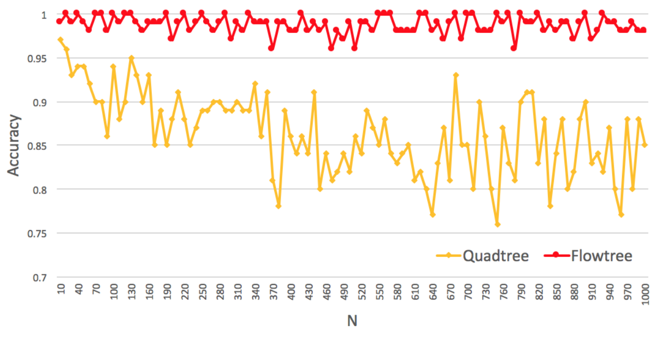
Runtime measurement. All running times are measured on a "Standard F72s_v2" Microsoft Azure instance equipped with Intel Xeon Platinum 8168 CPU. In our implementations, we use NumPy linked with OpenBLAS, which is used in a single-threaded mode.

Implementation. We implement R-WMD, ACT and Sinkhorn in Python with NumPy, as they amount to standard matrix operations which are handled efficiently by the

[6]There it is called Word Centroid Distance (WCD).

[7]Namely, it is a weighted variant of Overlap, where terms are weighted according to their frequency in the dataset.

[8]This coincides with the results reported in (Atasu & Mittelholzer, 2019).



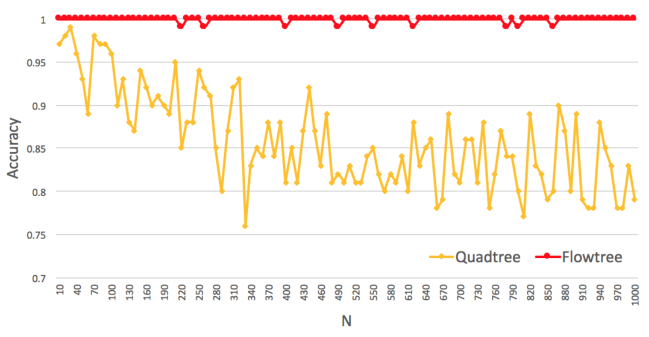


Figure 2: Results on random data. Left: $d = s = 10$, $\epsilon = 0.25$. Right: $d = 10$, $s = 100$, $\epsilon = 0.4$.

underlying BLAS implementation. We implement Mean, Overlap, TF-IDF, Quadtree and Flowtree in C++ (wrapped in Python for evaluation). For Mean we use the Eigen library to compute dense ℓ_2 distances efficiently. For Exact W_1 we use the POT library in Python, which in turn calls the Lemon graph library written in C++. The accuracy and pipeline evaluation code is in Python.

4.3. Individual accuracy experiments

Our first set of experiments evaluates the runtime and accuracy of each algorithm individually. The results are depicted in Figures 3 to 5. The plots report the recall@m accuracy as m grows. The recall@m accuracy is defined as the fraction of queries for which the true nearest neighbors is included in the top-m ranked neighbors (called *candidates*) by the evaluated method.

For each dataset, the left plot reports the accuracy of all of the methods for large values of m. The right plot reports the accuracy of the high-accuracy methods for smaller values of m (since they cannot be discerned in the left plots). The high-accuracy methods are Flowtree, R-WMD, ACT, Sinkhorn, and on MNIST also Quadtree. For Quadtree and Flowtree, which are randomized methods, we report the mean and standard deviation (shown as error bars) of 5 executions. The other methods are deterministic. The legend of each plot is annonated with the running time of each method, also summarized in Table 2.

Results. The tested algorithms yield a wide spectrum of different time-accuracy tradeoffs. The “fast” linear time methods (Mean, TF-IDF, Overlap and Quadtree) run in order of milliseconds, but are less accurate than the rest. The quadratic time methods (R-WMD, ACT-1 and Sinkhorn) are much slower, running in order of seconds, but are dramatically more accurate.

Flowtree achieves comparable accuracy to the quadratic time baselines, while being faster by a margin. In particular, its accuracy is either similar to or better than R-WMD, while being 5.5 to 6 times faster. Compared to ACT-1, Flowtree is either somewhat less or more accurate (depending on the dataset), while being at least 8 times faster. Compared to Sinkhorn, Flowtree achieves somewhat lower accuracy, but is at least 13.8 and up to 30 times faster.

4.4. Pipeline experiments

The above results exhibit a sharp divide between fast and coarse algorithms to slow and accurate ones. In practical nearest neighbor search system, both types of algorithms are often combined sequentially as a *pipeline* (e.g., (Sivic & Zisserman, 2003; Jegou et al., 2008; 2010)). First, a fast and coarse method is applied to all points, pruning most of them; then a slower and more accurate method is applied to the surviving points, pruning them further; and so on, until finally exact computation is performed on a small number of surviving points. In particular, (Kusner et al., 2015) employ such a pipeline for the Word Mover Distance, which combines Mean, R-WMD, and exact W_1 computation.

In this section, we systematically evaluate pipelines built of the algorithms tested above, on the 20news dataset.

Experimental setup. We perform two sets of experiments: In one, the pipeline reports one candidate, and its goal is to output the true nearest neighbor (i.e., recall@1). In the other, the pipeline reports 5 candidates, and its goal is to include the true nearest neighbor among them (i.e., recall@5). We fix the target accuracy to 0.9 (i.e., the pipeline must achieve the recall goal on 90% of the queries), and report its median running time over 3 identical runs.

Evaluated pipelines. The baseline pipelines we consider contain up to three methods:

- First: Mean, Overlap or Quadtree.
- Second: R-WMD, ACT-1, Sinkhorn-1, or Sinkhorn-3.
- Third: Exact W_1 computation. For recall@5 pipelines whose second method is Sinkhorn, this third step is omitted, since they already attain the accuracy goal without it.

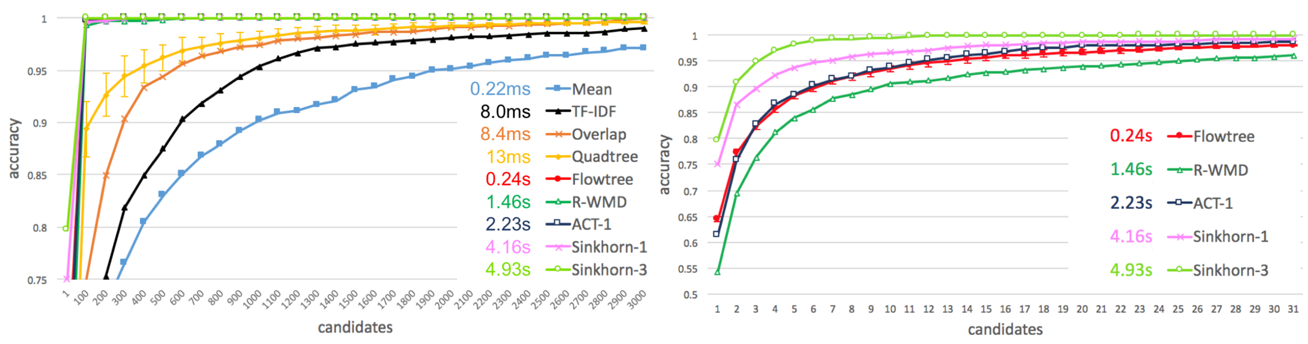


Figure 3: Individual accuracy and runtime results on 20news

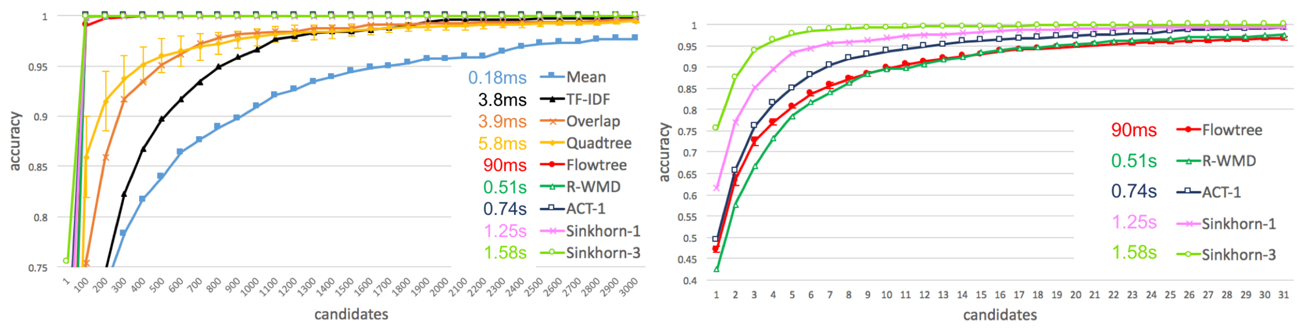


Figure 4: Individual accuracy and runtime results on Amazon

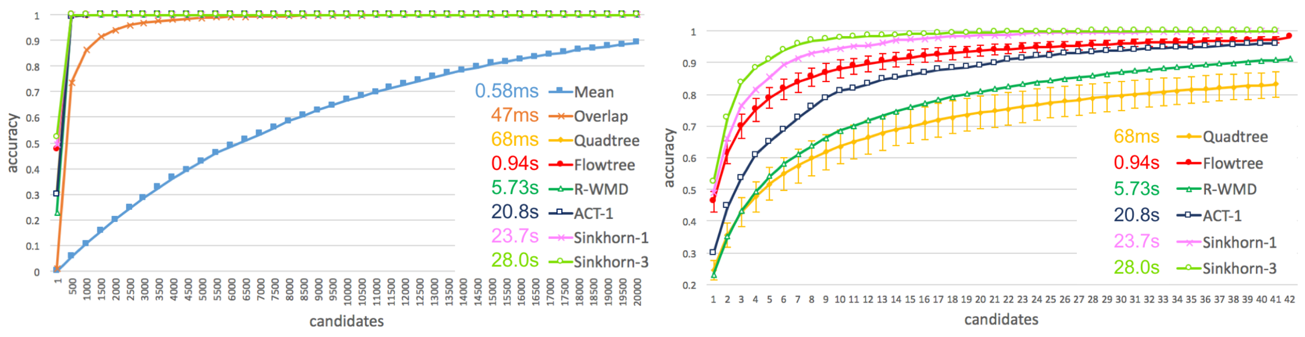


Figure 5: Individual accuracy and runtime results on MNIST*

Table 2: Running times

Dataset	Mean	TF-IDF	Overlap	Quadtree	Flowtree	R-WMD	ACT-1**	Sinkhorn-1	Sinkhorn-3	Exact W_1
20news	0.22ms	8.0ms	8.4ms	13ms	0.24s	1.46s	2.23s	4.16s	4.93s	41.5s
Amazon	0.18ms	3.8ms	3.9ms	5.8ms	90ms	0.51s	0.74s	1.25s	1.58s	4.23s
MNIST*	0.58ms	—	47ms	68ms	0.94s	5.73s	20.8s	23.7s	28.0s	154.0s

* On MNIST, the accuracy of R-WMD, ACT and Sinkhorn is evaluated on $1,000$ random queries. The running time of R-WMD, ACT, Sinkhorn and Exact W_1 is measured on 100 random queries. The running time of Flowtree is measured $1,000$ random queries.
** ACT takes a faster form when applied to uniform distributions. We use a separate implementation for this case. This accounts for the large difference in its performance on 20news and Amazon (where distributions are uniform) compared to MNIST (where they are not).

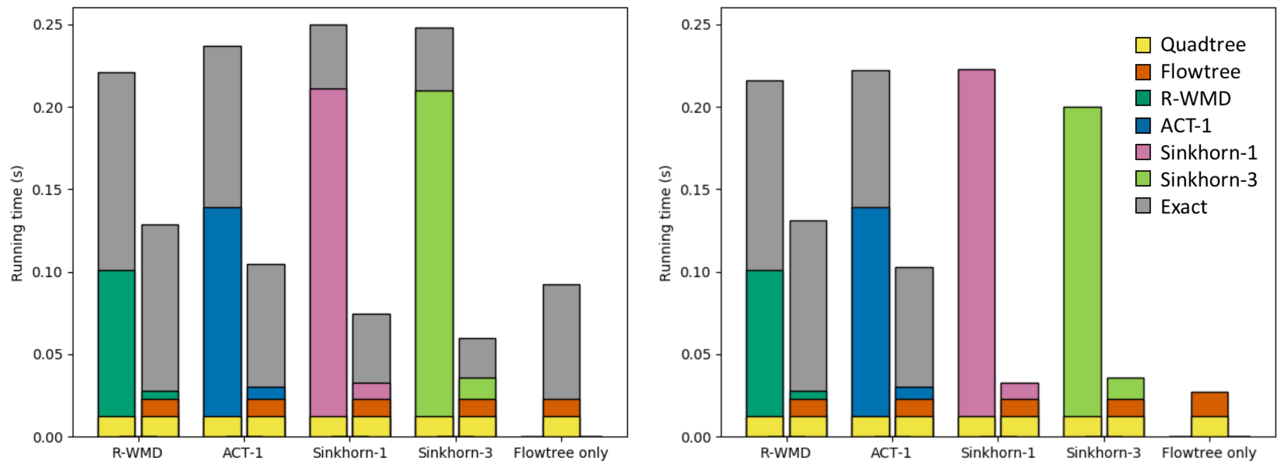


Figure 6: Best performing pipelines for recall@1 ≥ 0.9 (on the left) and recall@5 ≥ 0.9 (on the right). Each vertical bar denotes a pipeline, built of the methods indicated by the color encoding, bottom-up. The y-axis measures the running time up to each step of the pipeline. The plot depicts 4 baseline pipelines, consisting of Quadtree, then the method X indicated on the x-axis (R-WMD, ACT-1, Sinkhorn-1 or Sinkhorn-3), and then (optionally) Exact W_1. Next to each baseline bar we show the bar obtained by adding Flowtree to the pipeline as an intermediate algorithm between Quadtree and X. The rightmost bar in each plot shows the pipeline obtained by using Flowtree instead of X.

Table 3: Best pipeline runtime results

	Recall@1 ≥ 0.9	Recall@5 ≥ 0.9
Without Flowtree	0.221s	0.200s
With Flowtree	0.059s	0.027s

To introduce Flowtree into the pipelines, we evaluate it both as an intermediate stage between the first and second methods, and as a replacement for the second method.

Pipeline parameters. A pipeline with ℓ algorithms has parameters $c_1, \ldots, c_{\ell-1}$, where c_i is the number of output candidates (non-pruned points) of the i^{th} algorithm in the pipeline.[9] We tune the parameters of each pipeline optimally on a random subset of 300 queries (fixed for all pipelines). The optimal parameters are listed in the appendix.

Results. We found that in the first step of the pipeline, Quadtree is significantly preferable to Mean and Overlap, and the results reported in this section are restricted to it. More results are included in the appendix.

Figure 6 shows the runtimes of pipelines that start with Quadtree. Note that each pipeline is optimized by different parameters, not depicted in the figure. For example, Sinkhorn-3 is faster than Sinkhorn-1 on the right plot, even though it is generally a slower algorithm. However, it is also more accurate, which allows the preceding Quadtree step to be less accurate and report fewer candidates, while still attaining overall accuracy of 0.9. Specifically, in the optimal recall@5 setting, Sinkhorn-3 runs on 227 candidates reported by Quadtree, while Sinkhorn-1 runs on 295.

The best runtimes are summarized in Table 3. The results show that introducing Flowtree improves the best runtimes by a factor of 3.7 for recall@1 pipelines, and by a factor of 7.4 for recall@5 pipelines.

In the recall@1 experiments, the optimally tuned baseline pipelines attain runtimes between 0.22 to 0.25 seconds. Introducing Flowtree before the second method in each pipeline improves its running time by a factor of 1.7 to 4.15. Introducing Flowtree instead of the second method improves the runtime by a factor of 2.4 to 2.7.

Once Flowtree is introduced into the recall@1 pipelines, the primary bottleneck becomes the final stage of exact W_1 computations. In the recall@5 experiments, this step is not always required, which enables larger gains for Flowtree. In these experiments, the optimally tuned baseline pipelines attain runtimes between 0.2 to 0.22 seconds. Introducing Flowtree before the second method in each pipeline improves its running time by a factor of 1.64 to 6.75. Introducing Flowtree instead of the second method improves the runtime by a factor of 7.4 to 8.

Overall, Flowtree significantly improves the running time of every pipeline, both as an addition and as a replacement.

[9] The final algorithm always outputs either 1 or 5 points, according to the recall goal.

References

Andoni, A., Indyk, P., and Krauthgamer, R. Earth mover distance over high-dimensional spaces. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 343–352. Society for Industrial and Applied Mathematics, 2008.

Andoni, A., Naor, A., and Neiman, O. Snowflake universality of wasserstein spaces. *arXiv preprint arXiv:1509.08677*, 2015.

Andoni, A., Indyk, P., and Razenshteyn, I. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 2018a.

Andoni, A., Krauthgamer, R., and Razenshteyn, I. Sketching and embedding are equivalent for norms. *SIAM Journal on Computing*, 47(3):890–916, 2018b.

Atasu, K. and Mittelholzer, T. Linear-complexity data-parallel earth movers distance approximations. In *International Conference on Machine Learning*, pp. 364–373, 2019.

Bačkurs, A. and Indyk, P. Better embeddings for planar earth-mover distance over sparse sets. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pp. 280–289, 2014.

Backurs, A., Indyk, P., Onak, K., Schieber, B., Vakilian, A., and Wagner, T. Scalable fair clustering. In *International Conference on Machine Learning*, pp. 405–413, 2019.

Bartal, Y. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pp. 184–193. IEEE, 1996.

Bartal, Y. On approximating arbitrary metrices by tree metrics. In *STOC*, volume 98, pp. 161–168, 1998.

Bourgain, J. The metrical interpretation of superreflexivity in banach spaces. *Israel Journal of Mathematics*, 56(2): 222–230, 1986.

Calinescu, G., Karloff, H., and Rabani, Y. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2005.

Charikar, M., Chekuri, C., Goel, A., Guha, S., and Plotkin, S. Approximating a finite metric by a small number of tree metrics. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pp. 379–388. IEEE, 1998.

Charikar, M. S. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pp. 380–388. ACM, 2002.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pp. 2292–2300, 2013.

Fakcharoenphol, J., Rao, S., and Talwar, K. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.

Gupta, A., Krauthgamer, R., and Lee, J. R. Bounded geometries, fractals, and low-distortion embeddings. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pp. 534–543. IEEE, 2003.

Indyk, P. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pp. 10–33. IEEE, 2001.

Indyk, P. and Thaper, N. Fast image retrieval via embeddings. In *3rd international workshop on statistical and computational theories of vision*, volume 2, pp. 5, 2003.

Indyk, P., Razenshteyn, I., and Wagner, T. Practical data-dependent metric compression with provable guarantees. In *Advances in Neural Information Processing Systems*, pp. 2617–2626, 2017.

Jegou, H., Douze, M., and Schmid, C. Hamming embedding and weak geometric consistency for large scale image search. In *European conference on computer vision*, pp. 304–317. Springer, 2008.

Jegou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Kalantari, B. and Kalantari, I. A linear-time algorithm for minimum cost flow on undirected one-trees. In *Combinatorics Advances*, pp. 217–223. Springer, 1995.

Khot, S. and Naor, A. Nonembeddability theorems via fourier analysis. *Mathematische Annalen*, 334(4):821–852, 2006.

Kleinberg, J. and Tardos, E. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Journal of the ACM (JACM)*, 49(5):616–639, 2002.

Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. From word embeddings to document distances. In *International conference on machine learning*, pp. 957–966, 2015.

Le, T., Yamada, M., Fukumizu, K., and Cuturi, M. Tree-sliced approximation of wasserstein distances. *arXiv preprint arXiv:1902.00342*, 2019.

Mendel, M. and Naor, A. Ramsey partitions and proximity data structures. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pp. 109–118. IEEE, 2006.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Naor, A. and Schechtman, G. Planar earthmover is not in l_1. *SIAM Journal on Computing*, 37(3):804–826, 2007.

Pennington, J., Socher, R., and Manning, C. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

Rubner, Y., Tomasi, C., and Guibas, L. J. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

Samet, H. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

Sivic, J. and Zisserman, A. Video google: A text retrieval approach to object matching in videos. In *null*, pp. 1470. IEEE, 2003.

Wu, L., Yen, I. E., Xu, K., Xu, F., Balakrishnan, A., Chen, P.-Y., Ravikumar, P., and Witbrock, M. J. Word mover's embedding: From word2vec to document embedding. *arXiv preprint arXiv:1811.01713*, 2018.

Yurochkin, M., Claici, S., Chien, E., Mirzazadeh, F., and Solomon, J. Hierarchical optimal transport for document representation. *arXiv preprint arXiv:1906.10827*, 2019.