

Scalable Nearest Neighbor Search for Optimal Transport*

Arturs Backurs[†]
TTIC

Yihe Dong
Microsoft

Piotr Indyk
MIT

Ilya Razenshteyn
Microsoft Research

Tal Wagner
MIT

September 28, 2020

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 .¹ The Wasserstein-1 distance and its variants are heavily used in applications to measure similarity in structured data domains, such as images [RTG00] and natural language text [KSKW15]. In particular, [KSKW15] 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 [MSC⁺13] or GloVe [PSM14]. It is shown in [KSKW15] (see also [LYFC19, YCC⁺19, WYX⁺18]) 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 [AIR18] 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 [Kuh55] or approximate iterative methods [Cut13]). Second, the W_1 distance is strongly incompatible with Euclidean (and more generally, with ℓ_p) geometries [Bou86, KN06, NS07, AIK08, ANN15, AKR18], 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

*Code available at https://github.com/ilyaraz/ot_estimators.

[†]Author names are ordered alphabetically.

¹For mathematical foundations of Wasserstein distances, see [Vil03].

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 .² Given a dataset of n distributions $\mu_1, \mu_2, \dots, \mu_n$, 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 applications.

1.1 Prior work

Kusner et al. [KSKW15] 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 [KSKW15] consists of the combination of both algorithms, together with exact W_1 distance computation. Recently, [AM19] 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 and Thaper [IT03] 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 [Bar96, Bar98, CCG⁺98, Ind01, GKL03, FRT04, CKR05, MN06]. 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 [KT02, Cha02]. This allowed [IT03] to design an efficient NNS algorithm for W_1 based on classical locality-sensitive hashing (LSH). Recently, [LYFC19] 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 [IT03, LYFC19] 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 $\mu_1, \dots, \mu_n$ 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*.

²In the application to [KSKW15], 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.

- *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 [KSKW15] (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. [Sam84]), but it extends to—and has been successfully used in—various high-dimensional settings (e.g. [Ind01, IRW17, BIO⁺19]). 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 $[0, \Phi]$.³

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 .⁴ We number the root level as $\log \Phi + 1$, and the rest of the levels are numbered downward accordingly ($\log \Phi, \log \Phi - 1, \dots$). 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 $\tilde{O}(|X|d \cdot \log(d\Phi))$.⁵

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

³This 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.

⁴This 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.

⁵Note 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 .

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 [Cha02] following [KT02]. 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 3.1. $\widetilde{W}_1(\mu, \nu)$ 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 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 $\mu_1, \dots, \mu_n$, 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 $\mu_1, \dots, \mu_n$. 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 [AIK08] (where it is proven for a somewhat different algorithm, with similar analysis). All proofs are deferred to the appendix.

Theorem 3.2 (Quadtree upper bound). *With probability ≥ 0.99 , the nearest neighbor of ν among $\mu_1, \dots, \mu_n$ 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 3.3 (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 $c = \Omega(\log n)$.*

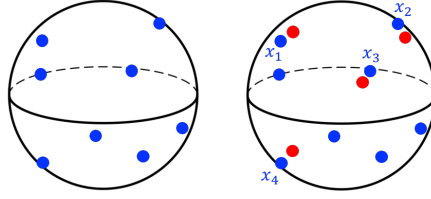


Figure 1: Random model illustration with $s = 4$. Left: The blue points are the N random data points. The data distributions are all subsets of 4 points. Right: The red points form a query distribution whose planted nearest neighbor is the distribution supported on $\{x_1, x_2, x_3, x_4\}$.

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

Theorem 3.4 (Flowtree upper bound). *With probability ≥ 0.99 , the nearest neighbor of ν among $\mu_1, \dots, \mu_n$ 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 [AIK08] and [BI14] 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 3.5 (Flowtree upper bound for uniform distributions⁶). *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 $\mu_1, \dots, \mu_n$ in the Flowtree distance is an $O(\log^2 s)$ -approximate nearest neighbor for the W_1 distance.*

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 Flow 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 $\mu_1, \dots, \mu_n$ (so $n = \binom{N}{s}$).

To generate a query, pick any μ_i as the “planted” nearest neighbor, and let $x_1, \dots, x_s$ denote its support. For $k = 1, \dots, s$, 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 $y_1, \dots, y_s$. 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 3.6. *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.

⁶For simplicity, Theorem 3.5 is stated for uniform distribution (or close to uniform), such as documents in [KSKW15]. A similar result holds for any distribution, with additional dependence on the numerical range of mass values.

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 3.6, Quadtree’s success rate degrades as N increases (and we recall that $n = \binom{N}{s}$), while Flowtree’s does not.

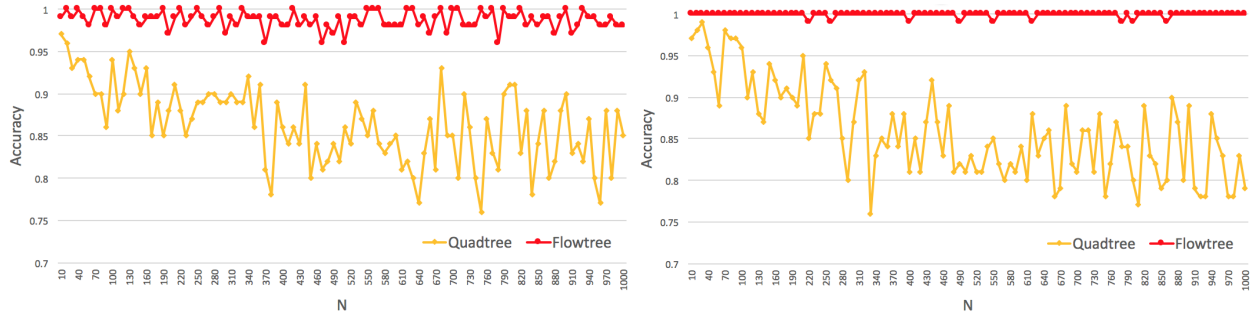


Figure 2: Results on random data. Left: $d = s = 10$, $\epsilon = 0.25$. Right: $d = 10$, $s = 100$, $\epsilon = 0.4$.

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 [KSKW15] 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 [PSM14] with 400,000 terms and 50 dimensions.
- *Image recognition:* We use the MNIST dataset of handwritten digits. As in [Cut13], 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 [KSKW15].⁷
- *Overlap:* A simple baseline that estimates $W_1(\mu, \nu)$ by the size of the intersection of their supports.

⁷There it is called Word Centroid Distance (WCD).

- *TF-IDF*: A well-known similarity measure for text documents. It is closely related to Overlap.⁸ 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 [KSKW15], 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 [AM19] 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 $i = 1$,⁹ which we denote by ACT-1. The appendix contains additional results for larger i .
- *Sinkhorn with few iterations*: The iterative Sinkhorn method of [Cut13] 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 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.

⁸Namely, it is a weighted variant of Overlap, where terms are weighted according to their frequency in the dataset.

⁹This coincides with the results reported in [AM19].

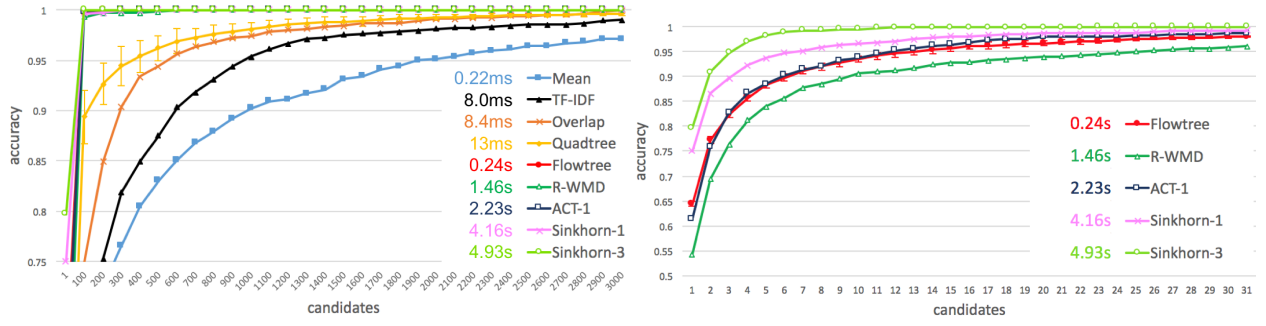


Figure 3: Individual accuracy and runtime results on 20news

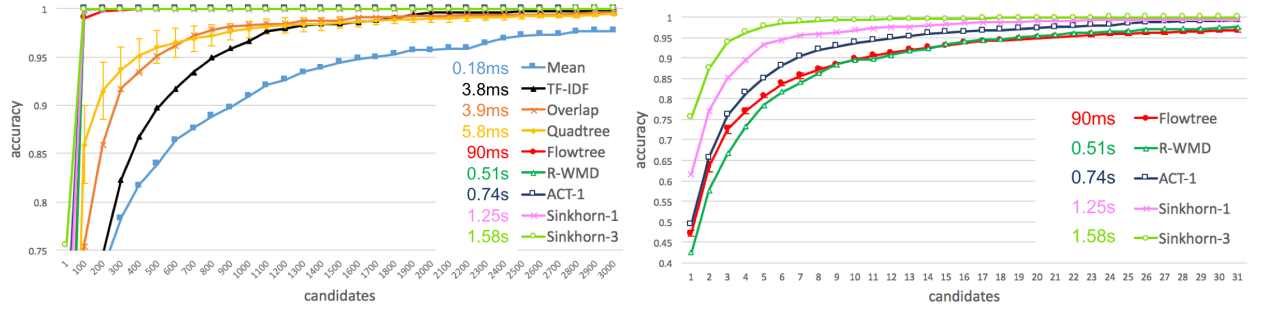


Figure 4: Individual accuracy and runtime results on Amazon

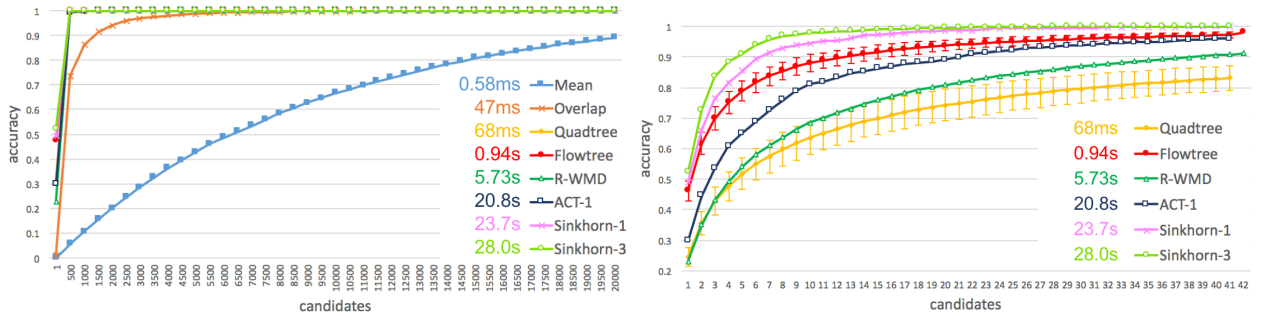


Figure 5: Individual accuracy and runtime results on MNIST*

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).

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 annotated 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., [SZ03, JDS08, JDS10]). 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, [KSKW15] 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.

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 $c_1, \dots, c_{\ell-1}$, where c_i is the number of output candidates (non-pruned points) of the i^{th} algorithm in the pipeline.¹⁰ 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.

¹⁰The final algorithm always outputs either 1 or 5 points, according to the recall goal.

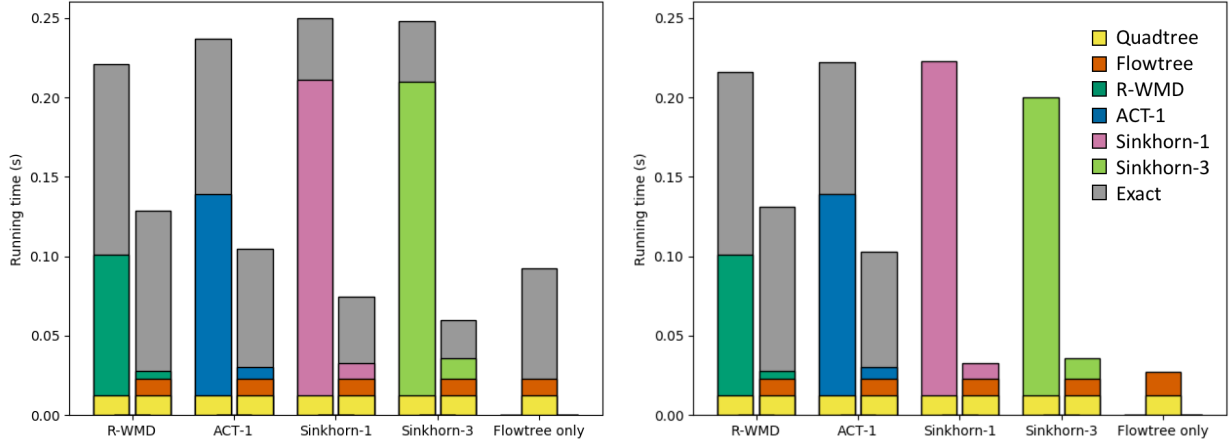


Figure 6: Best performing pipelines for $\text{recall@1} \geq 0.9$ (on the left) and $\text{recall@5} \geq 0.9$ (on the right). Each vertical bar denotes a pipeline, built of the methods indicated by the color encoding, bottom-up. The y-axis measures the running time up to each step of the pipeline. 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

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.

Acknowledgments. Supported by NSF TRIPODS awards No. 1740751 and No. 1535851, Simons Investigator Award, and MIT-IBM Watson AI Lab collaboration grant.

References

- [AIK08] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 343–352. Society for Industrial and Applied Mathematics, 2008.
- [AIR18] Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 2018.
- [AKR18] Alexandr Andoni, Robert Krauthgamer, and Ilya Razenshteyn. Sketching and embedding are equivalent for norms. *SIAM Journal on Computing*, 47(3):890–916, 2018.
- [AM19] Kubilay Atasu and Thomas Mittelholzer. Linear-complexity data-parallel earth movers distance approximations. In *International Conference on Machine Learning*, pages 364–373, 2019.
- [ANN15] Alexandr Andoni, Assaf Naor, and Ofer Neiman. Snowflake universality of wasserstein spaces. *arXiv preprint arXiv:1509.08677*, 2015.
- [Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE, 1996.
- [Bar98] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *STOC*, volume 98, pages 161–168, 1998.
- [BI14] Artūrs Bačkurs and Piotr Indyk. Better embeddings for planar earth-mover distance over sparse sets. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pages 280–289, 2014.
- [BIO⁺19] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. Scalable fair clustering. In *International Conference on Machine Learning*, pages 405–413, 2019.
- [Bou86] Jean Bourgain. The metrical interpretation of superreflexivity in banach spaces. *Israel Journal of Mathematics*, 56(2):222–230, 1986.
- [CCG⁺98] Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*, pages 379–388. IEEE, 1998.
- [Cha02] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.
- [CKR05] Gruia Calinescu, Howard Karloff, and Yuval Rabani. Approximation algorithms for the 0-extension problem. *SIAM Journal on Computing*, 34(2):358–372, 2005.
- [Cut13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013.
- [FRT04] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.
- [GKL03] Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pages 534–543. IEEE, 2003.
- [Ind01] Piotr Indyk. Algorithmic applications of low-distortion geometric embeddings. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 10–33. IEEE, 2001.

- [IRW17] Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Practical data-dependent metric compression with provable guarantees. In *Advances in Neural Information Processing Systems*, pages 2617–2626, 2017.
- [IT03] Piotr Indyk and Nitin Thaper. Fast image retrieval via embeddings. In *3rd international workshop on statistical and computational theories of vision*, volume 2, page 5, 2003.
- [JDS08] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *European conference on computer vision*, pages 304–317. Springer, 2008.
- [JDS10] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.
- [KK95] Bahman Kalantari and Iraj Kalantari. A linear-time algorithm for minimum cost flow on undirected one-trees. In *Combinatorics Advances*, pages 217–223. Springer, 1995.
- [KN06] Subhash Khot and Assaf Naor. Nonembeddability theorems via fourier analysis. *Mathematische Annalen*, 334(4):821–852, 2006.
- [KSKW15] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966, 2015.
- [KT02] Jon Kleinberg and Eva Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. *Journal of the ACM (JACM)*, 49(5):616–639, 2002.
- [Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- [LYFC19] Tam Le, Makoto Yamada, Kenji Fukumizu, and Marco Cuturi. Tree-sliced approximation of wasserstein distances. *arXiv preprint arXiv:1902.00342*, 2019.
- [MN06] Manor Mendel and Assaf Naor. Ramsey partitions and proximity data structures. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 109–118. IEEE, 2006.
- [MSC⁺13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.
- [NS07] Assaf Naor and Gideon Schechtman. Planar earthmover is not in L1. *SIAM Journal on Computing*, 37(3):804–826, 2007.
- [PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover’s distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.
- [Sam84] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.
- [SZ03] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *null*, page 1470. IEEE, 2003.
- [Vil03] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.

- [WYX⁺18] Lingfei Wu, Ian EH Yen, Kun Xu, Fangli Xu, Avinash Balakrishnan, Pin-Yu Chen, Pradeep Ravikumar, and Michael J Witbrock. Word mover’s embedding: From word2vec to document embedding. *arXiv preprint arXiv:1811.01713*, 2018.
- [YCC⁺19] Mikhail Yurochkin, Sebastian Claiç, Edward Chien, Farzaneh Mirzazadeh, and Justin Solomon. Hierarchical optimal transport for document representation. *arXiv preprint arXiv:1906.10827*, 2019.

A Proofs

A.1 Flowtree Computation

In this section we prove Lemma 3.1. We begin by specifying the greedy flow computation algorithm on the tree. Let h denote the height of the tree (for the quadtree the height is $h = O(\log(d\Phi))$). Suppose we are given a pair of distributions μ, ν , each supported on at most s leaves of the tree. For every node v in the tree, let $C_\mu(v)$ denote the set of points in $x \in X$ such that $\mu(x) > 0$ and the tree leaf that contains x is a descendant of v . Similarly define $C_\nu(v)$. Note that we only need to consider nodes for which either $C_\mu(v)$ or $C_\nu(v)$ is non-empty, and there are at most $2sh$ such nodes.

The algorithm starts with a zero flow f , and processes the nodes in a bottom-up order starting at the leaf. In each node, the unmatched demands collected from its children are matched arbitrarily, and the demands that cannot be matched are passed on to the parent. In more detail, a node is processed as follows:

1. Collect from the children the list of unmatched μ -demands for the nodes in $C_\mu(v)$ and the list of unmatched ν -demands for the nodes in $C_\nu(v)$. Let $\{\mu_v(x) : x \in C_\mu(v)\}$ denote the unmatched μ -demands and let $\{\nu_v(x) : x \in C_\nu(v)\}$ denote the unmatched ν -demands.
2. While there is a pair $x \in C_\mu(v)$ and $x' \in C_\nu(v)$ with $\mu_v(x) > 0$ and $\nu_v(x') > 0$, let $\eta = \min\{\mu_v(x), \nu_v(x')\}$, and update (i) $f(x, x') += \eta$, (ii) $\mu_v(x) -= \eta$, (iii) $\nu_v(x') -= \eta$.
3. Now either μ_v or ν_v is all-zeros. If the other one is not all-zeros (i.e., there is either remaining unmatched μ -demand or remaining unmatched ν -demand), pass it on to the parent.

A leaf v contains a single point $x \in X$ with either $\mu(x) > 0$ or $\nu(x) > 0$; it simply passes it on to its parent without processing.

It is well known that the above algorithm computes an optimal flow on the tree (with respect to tree distance costs), see, e.g., [KK95]. Let us now bound its running time. The processing time per node v in the above algorithm is $O(|C_\mu(v)| + |C_\nu(v)|)$. In every given level in the tree, if $v_1, \dots, v_k$ are the nodes in that level, then $\{C_\mu(v_1), \dots, C_\mu(v_k)\}$ is a partition of the support of μ , and $\{C_\nu(v_1), \dots, C_\nu(v_k)\}$ is a partition of the support of ν . Therefore the total processing time per level is $O(s)$, and since there are h levels, the flow computation time is $O(sh)$. Then we need to compute the Flowtree output $\widehat{W}_1(\mu, \nu)$. Observe that in the above algorithm, whenever we match demands between a pair x, x' , we fully satisfy the unmatched demand of one of them. Therefore the output flow f puts non-zero flow between at most $2s$ pairs. For each such pair we need to compute the Euclidean distance in time $O(d)$, and the overall running time is $O(s(d + h))$.

A.2 Quadtree and Flowtree Analysis

Proof of Theorem 3.2. Let $x, y \in X$. Let $p_\ell(x, y)$ be the probability that x, y fall into the same cell (hypercube) in level ℓ of the quadtree. It satisfies,

$$1 - \frac{\|x - y\|_1}{2^\ell} \leq p_\ell(x, y) \leq \exp\left(-\frac{\|x - y\|_1}{2^\ell}\right).$$

To see this, recall that in level ℓ we impose a grid with side length $2^{-\ell}$, shifted at random by an i.i.d. uniform shift in $[0, 2^{-\ell}]$ in each coordinate. The probability that x, y are separated in coordinate i is $2^{-\ell}|x_i - y_i|$, and thus $p_\ell(x, y) = \prod_{i=1}^d (1 - 2^{-\ell}|x_i - y_i|)$. The lower bound in ?? follows by a union bound, and the upper bound follows by applying the general estimate $1 - z \leq \exp(-z)$ to each term in the product.

Let t be the tree metric induced on X by the quadtree. Note that for $t(x, y)$ to be at most $O(2^\ell)$, x, y must fall into the same hypercube in level ℓ . For any $\delta > 0$, we can round $\frac{\|x-y\|_1}{\log(1/\delta)}$ to its nearest power of 2 and obtain ℓ such that $2^\ell = \Theta(\frac{\|x-y\|_1}{\log(1/\delta)})$. It satisfies,

$$\Pr \left[t(x, y) < \frac{O(1)}{\log(1/\delta)} \|x - y\|_1 \right] \leq \delta.$$

By letting $\delta = \Omega(\min\{1/|X|, 1/(s^2n)\})$, we can take union bound either over all pairwise distances in X (of which there are $\binom{|X|}{2}$), or over all distances between the support of the query ν and the union of supports of the dataset $\mu_1, \dots, \mu_n$ (of which there are at most s^2n , if every support has size at most s). Then, with probability say 0.995, all those distances are contracted by at most $O(\log(\min\{sn, |X|\}))$, i.e.,

$$t(x, y) \geq \frac{1}{O(\log(1/\delta))} \|x - y\|_1. \quad (2)$$

On the other hand,

$$\mathbb{E}[t(x, y)] = \sum_{\ell} 2^\ell \cdot (1 - p_\ell(x, y)) \leq \sum_{\ell} 2^\ell \cdot \frac{\|x - y\|_1}{2^\ell} \leq O(\log(d\Phi)) \cdot \|x - y\|_1.$$

Let μ^* be the true nearest neighbor of ν in $\mu_1, \dots, \mu_n$. Let $f_{\mu^*, \nu}^*$ be the optimal flow between them. Then by the above,

$$\mathbb{E} \left[\sum_{(x, y) \in X \times X} f_{\mu^*, \nu}^*(x, y) t(x, y) \right] \leq O(\log(d\Phi)) \sum_{(x, y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \|x - y\|_1.$$

By Markov, with probability say 0.995,

$$\sum_{(x, y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \cdot t(x, y) \leq O(\log(d\Phi)) \sum_{(x, y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \cdot \|x - y\|_1. \quad (3)$$

Let μ' be the nearest neighbor of ν in the dataset according to the quadtree distance. Let $f_{\mu', \nu}^*$ be the optimal flow between them in the true underlying metric (ℓ_1 on X), and let $f_{\mu, \nu}$ be the optimal flow in the quadtree. Finally let W_t denote the Wasserstein-1 distance on the quadtree. Then,

$$\begin{aligned} & W_1(\mu', \nu) \\ &= \sum_{(x, y) \in X \times X} f_{\mu', \nu}^*(x, y) \cdot \|x - y\|_1 \\ &\leq \sum_{(x, y) \in X \times X} f_{\mu', \nu} \cdot \|x - y\|_1 && f_{\mu', \nu}^* \text{ is optimal for } \|\cdot\|_1 \\ &\leq O(\log(\min\{sn, |X|\})) \sum_{(x, x') \in X \times X} f_{\mu', \nu} \cdot t(x, y) && \text{eq. (2)} \\ &= O(\log(\min\{sn, |X|\})) \cdot W_t(\mu', \nu) && \text{definition of } W_t \\ &\leq O(\log(\min\{sn, |X|\})) \cdot W_t(\mu^*, \nu) && \mu' \text{ is the nearest neighbor in } W_t \\ &= O(\log(\min\{sn, |X|\})) \sum_{(x, y) \in X \times X} f_{\mu^*, \nu} \cdot t(x, y) && \text{definition of } W_t \\ &\leq O(\log(\min\{sn, |X|\})) \sum_{(x, y) \in X \times X} f_{\mu^*, \nu}^* \cdot t(x, y) && f_{\mu^*, \nu}^* \text{ is optimal for } t(\cdot, \cdot) \\ &\leq O(\log(\min\{sn, |X|\}) \log(d\Phi)) \sum_{(x, y) \in X \times X} f_{\mu^*, \nu}^* \cdot \|x - y\|_1 && \text{eq. (3)} \\ &= O(\log(\min\{sn, |X|\}) \log(d\Phi)) \cdot W_1(\mu^*, \nu), \end{aligned}$$

so μ' is a $O(\log(\min\{sn, |X|\}) \log(d\Phi))$ -approximate nearest neighbor. $\square$

Proof of Theorem 3.3. It suffices to prove the claim for $s = 1$ (i.e., the standard ℓ_1 -distance). Let $d > 0$ be an even integer. Consider the d -dimensional hypercube. Our query point is the origin. The true nearest neighbor is e_1 (standard basis vector). The other data points are the hypercube nodes whose hamming weight is exactly $d/2$. The number of such points is $\Theta(2^d/\sqrt{d})$, and this is our n .

Consider imposing the grid with cell side 2 on the hypercube. The probability that 0 and 1 are uncut in a given axis is exactly $1/2$, and since the shifts in different axes are independent, the number of uncut axes is distributed as $\text{Bin}(d, 1/2)$. Thus with probability $1/2$ there are at least $d/2$ uncut dimensions. If this happens, we have a data point hashed into the same grid cell as the origin (to get such data point, put 1 in any $d/2$ uncut dimensions and 0 in the rest), so its quadtree distance from the origin is 1. On the other hand, the distance of the origin to its true nearest neighbor e_1 is at least 1, since they will necessarily be separated in the next level (when the grid cells have side 1). Thus the quadtree cannot tell between the true nearest neighbor and the one at distance $d/2$, and we get the lower bound $c \geq d/2$. Since $n = \Theta(2^d/\sqrt{d})$, we have $d/2 = \Omega(\log n)$ as desired. $\square$

Proof of Theorem 3.4. The proof is the same as for Theorem 3.2, except that in eq. (2), we take a union bound only over the s^2 distances between the supports of ν and μ^* (the query and its true nearest neighbor). Thus each distance between μ^* and ν is contracted by at most $O(\log s)$.

Let W_F denote the Flowtree distance estimate of W_1 . Let μ' be the nearest neighbor of ν in the Flowtree distance. With the same notation in the proof of Theorem 3.2,

$$\begin{aligned}
W_1(\mu', \nu) &= \sum_{(x,y) \in X \times X} f_{\mu', \nu}^*(x, y) \cdot \|x - y\|_1 \\
&\leq \sum_{(x,y) \in X \times X} f_{\mu', \nu}(x, y) \cdot \|x - y\|_1 && f_{\mu', \nu}^* \text{ is optimal for } \|\cdot\|_1 \\
&= W_F(\mu', \nu) && \text{Flowtree definition} \\
&\leq W_F(\mu^*, \nu) && \mu' \text{ is nearest in Flowtree distance} \\
&= \sum_{(x,y) \in X \times X} f_{\mu^*, \nu}(x, y) \cdot \|x - y\|_1 && \text{Flowtree definition} \\
&\leq O(\log s) \sum_{(x,y) \in X \times X} f_{\mu^*, \nu}(x, y) \cdot t(x, y) && \text{eq. (2)} \\
&\leq O(\log s) \sum_{(x,y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \cdot t(x, y) && f_{\mu^*, \nu}^* \text{ is optimal for } t(\cdot, \cdot) \\
&\leq O(\log(d\Phi) \log s) \sum_{(x,y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \cdot \|x - y\|_1 && \text{eq. (3)} \\
&= O(\log(d\Phi) \log s) \cdot W_1(\mu^*, \nu),
\end{aligned}$$

as needed. Note that the difference from the proof of Theorem 3.2 is that we only needed the contraction bound (eq. (2)) for distances between μ^* and ν . $\square$

Proof of Theorem 3.5. We set $\varepsilon = 1/\log s$. Let $t'(x, y)$ denote the quadtree distance where the weight corresponding to a cell v in level $\ell(v)$ is $2^{\ell(v)(1-\varepsilon)}$ instead of $2^{\ell(v)}$. Let $f_{\mu, \nu}$ be the optimal flow in the quadtree defined by weights t' .

Let $\delta = c/s^2$ where $c > 0$ is a sufficiently small constant. For a every x, y , let ℓ_{xy} be the largest integer such that

$$2^{\ell_{xy}} \leq \frac{\|x - y\|_1}{(\log(1/\delta))^{1/(1-\varepsilon)}}.$$

The probability that x, y are separated (i.e., they are in different quadtree cells) in level ℓ_{xy} is

$$1 - p_{\ell_{xy}}(x, y) \geq 1 - \exp\left(-\frac{\|x - y\|_1}{2^{\ell_{xy}}}\right) \geq 1 - \frac{\delta}{1 - \varepsilon}.$$

By the setting of δ , we can take a union bound over all $x \in \text{support}(\mu^*)$ and $y \in \text{support}(\nu)$ and obtain that with say 0.99 probability, simultaneously, every pair x, y is separated at level ℓ_{xy} . We denote this event by $\mathcal{E}_{lower}$ and suppose it occurs. Then for every x, y we have

$$t'(x, y) \geq 2 \cdot 2^{\ell_{xy}(1-\epsilon)} \geq 2 \cdot \left(\frac{1}{2} \cdot \frac{\|x - y\|_1}{(\log(1/\delta))^{1/(1-\epsilon)}} \right)^{1-\epsilon} \geq \frac{\|x - y\|_1^{1-\epsilon}}{\log(1/\delta)} = \frac{\|x - y\|_1^{1-\epsilon}}{\Theta(\log s)}.$$

Next we upper-bound the expected tree distance $t'(x, y)$. (Note that we are not conditioning on $\mathcal{E}_{lower}$.) Observe that

$$t'(x, y) = 2 \sum_{\ell=-\infty}^{\infty} 2^{\ell(1-\epsilon)} \cdot \mathbf{1}\{x, y \text{ are separated at level } \ell\}.$$

Let $L_{x,y}$ be the largest integer such that $2^{L_{x,y}} \leq \|x - y\|_1$. We break up $t'(x, y)$ into two terms,

$$t'_{lower}(x, y) = 2 \sum_{\ell=-\infty}^{L_{x,y}} 2^{\ell(1-\epsilon)} \cdot \mathbf{1}\{x, y \text{ are separated at level } \ell\},$$

and

$$t'_{upper}(x, y) = 2 \sum_{\ell=L_{x,y}+1}^{\infty} 2^{\ell(1-\epsilon)} \cdot \mathbf{1}\{x, y \text{ are separated at level } \ell\},$$

thus $t'(x, y) = t'_{lower}(x, y) + t'_{upper}(x, y)$. For $t'_{lower}(x, y)$ it is clear that deterministically,

$$t'_{lower}(x, y) \leq 2 \sum_{\ell=-\infty}^{L_{x,y}} 2^{\ell(1-\epsilon)} = O\left(2^{L_{x,y}(1-\epsilon)}\right) = O\left(\|x - y\|_1^{1-\epsilon}\right).$$

For $t'_{upper}(x, y)$, we have

$$\begin{aligned} \mathbb{E}[t'_{upper}(x, y)] &= 2 \sum_{\ell=L_{x,y}+1}^{\infty} 2^{\ell(1-\epsilon)} p_{\ell}(x, y) \\ &\leq 2 \sum_{\ell=L_{x,y}}^{\infty} 2^{\ell(1-\epsilon)} \cdot \frac{\|x - y\|_1}{2^{\ell}} \\ &= 2\|x - y\|_1 \sum_{\ell=L_{x,y}}^{\infty} 2^{-\epsilon\ell} \\ &= 2\|x - y\|_1 \cdot \frac{2^{-L_{x,y}\epsilon}}{1 - 2^{-\epsilon}} \\ &\leq O(\log s) \cdot \|x - y\|_1^{1-\epsilon}, \end{aligned}$$

where in the final bound we have used that $2^{L_{x,y}} = \Theta(\|x - y\|_1)$ and $1 - 2^{-\epsilon} = \Theta(\epsilon) = \Theta(\log s)$. Together,

$$\mathbb{E}[t'(x, y)] = \mathbb{E}[t'_{lower}(x, y) + t'_{upper}(x, y)] \leq \Theta(\log s) \cdot \|x - y\|_1^{1-\epsilon}. \quad (4)$$

Now we are ready to show the $O(\log^2 s)$ upper bound on the approximation factor. Below we will use the fact that every weight $f_{\mu^*, \nu}(x, y)$ in the flow is of the form $i/(s's'')$ for some integer $0 \leq i \leq s's''$. This follows from the assumption that each element in the support of every measure is an integer multiple of $1/s'$ or of $1/s''$ for some $1 \leq s', s'' \leq s$.

$$\begin{aligned} W_1(\mu', \nu) &= \sum_{(x,y) \in X \times X} f_{\mu', \nu}^*(x, y) \cdot \|x - y\|_1 & f_{\mu', \nu}^* \text{ is optimal for } \|\cdot\|_1 \\ &\leq \sum_{(x,y) \in X \times X} f_{\mu', \nu}(x, y) \cdot \|x - y\|_1 \end{aligned}$$

$$\begin{aligned}
&= W_F(\mu', \nu) && \text{Flowtree definition} \\
&\leq W_F(\mu^*, \nu) && \mu' \text{ is nearest to } \nu \text{ in Flowtree distance} \\
&= \sum_{(x,y) \in X \times X} f_{\mu^*, \nu}(x, y) \cdot \|x - y\|_1 && \text{Flowtree definition} \\
&\leq \left(\sum_{(x,y) \in X \times X} f_{\mu^*, \nu}^{1-\varepsilon}(x, y) \cdot \|x - y\|_1^{1-\varepsilon} \right)^{1/(1-\varepsilon)} && \text{subadditivity of } (\cdot)^{1-\varepsilon} \\
&\leq O(1) \left(\sum_{(x,y) \in X \times X} f_{\mu^*, \nu}(x, y) \cdot \|x - y\|_1^{1-\varepsilon} \right)^{1/(1-\varepsilon)} && f_{\mu^*, \nu}(x, y) \geq 1/(s' s'') \geq 1/s^2 \text{ or } f_{\mu^*, \nu}(x, y) = 0 \\
&\leq O(\log s) \left(\sum_{(x,y) \in X \times X} f_{\mu^*, \nu}(x, y) \cdot t'(x, y) \right)^{1/(1-\varepsilon)} && \text{eq. (4)} \\
&\leq O(\log s) \left(\sum_{(x,y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \cdot t'(x, y) \right)^{1/(1-\varepsilon)} && f_{\mu^*, \nu} \text{ is optimal for } t'(\cdot, \cdot) \\
&\leq O(\log^2 s) \left(\sum_{(x,y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \cdot \|x - y\|_1^{1-\varepsilon} \right)^{1/(1-\varepsilon)} && \text{eq. (4)} \\
&\leq O(\log^2 s) \sum_{(x,y) \in X \times X} f_{\mu^*, \nu}^*(x, y) \cdot \|x - y\|_1 && \text{concavity of } (\cdot)^{1-\varepsilon} \text{ and } \sum f_{\mu^*, \nu}^*(x, y) = 1 \\
&\leq O(\log^2 s) \cdot W_1(\mu^*, \nu),
\end{aligned}$$

as needed. $\square$

Proof of Theorem 3.6. Quadtree. For every $k = 1, \dots, s$, let H_k be the smallest hypercube in the quadtree that contains both x_k and y_k . (Note that H_k is a random variable, determined by the initial random shift in the Quadtree construction.) In order for Quadtree to correctly identify μ_i as the nearest neighbor of ν , every H_k must not contain any additional points from X . Otherwise, if say H_1 contains a point $x' \neq x_1$, the W_1 distance on the quadtree from ν to μ_i is equal to its distance to the uniform distribution over $\{x', x_2, \dots, x_s\}$. Since the points in X are chosen uniformly i.i.d. over $\mathcal{S}^{d-1}$, the probability of the above event, and thus the success probability of Quadtree, is upper bounded by $\mathbb{E}[(1 - V)^{N-s}]$, where $V = \text{volume}(\cup_{k=1}^s H_k \cap \mathcal{S}^{d-1})$. This V is a random variable whose distribution depends only on d, s, ϵ , and is independent of N . Thus the success probability decays exponentially with N .

Flowtree. On the other hand, suppose that each H_k contains no other points from $\{x_1, \dots, x_s\}$ other than x_k (but is allowed to contain any other points from X). This event guarantees that the optimal flow on the tree between μ_i and ν is the planted perfect matching, i.e., the true optimal flow, and thus the estimated Flowtree distance between them *equals* $W_1(\mu_i, \nu)$. This guarantees that Flowtree recovers the planted nearest neighbor, and this event depends only on d, s, ϵ , and is independent of N . $\square$

B Additional Experiments

B.1 Additional Sinkhorn and ACT Experiments

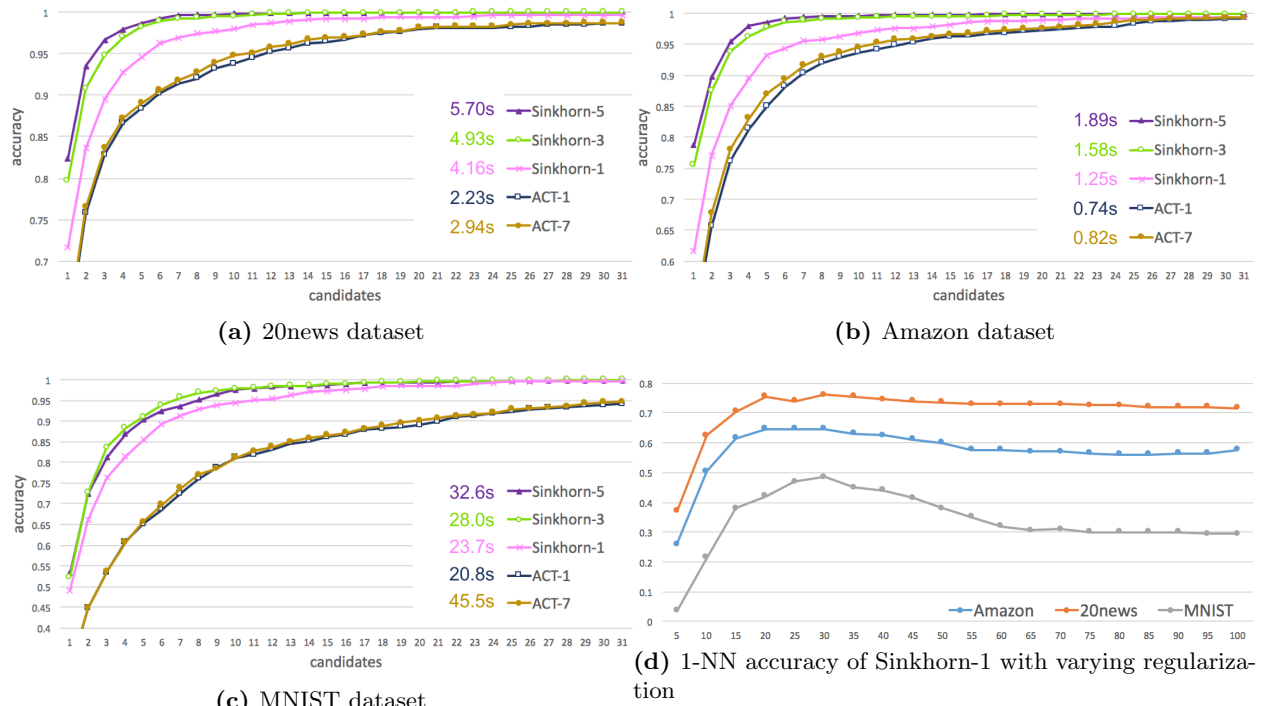
Number of iterations. Both ACT and Sinkhorn are iterative algorithms, and the number of iterations is a parameter to set. Our main experiments use ACT with 1 iteration and Sinkhorn with 1 or 3 iterations. The next experiments motivate these choices. Figures 7(a)–(c) depict the accuracy and running time of ACT-1, ACT-7, Sinkhorn-1, Sinkhorn-3 and Sinkhorn-5 on each of our datasets.¹¹ It can be seen that for

¹¹ACT-1 and ACT-7 are the settings reported in [AM19].

both algorithms, increasing the number of iterations beyond the settings used in Section 4 yields comparable accuracy with a slower running time. Therefore in Section 4 we restrict our evaluation to ACT-1, Sinkhorn-1 and Sinkhorn-3. We also remark that in the pipeline experiments, we have evaluated Sinkhorn with up to 9 iterations. In those experiments too, the best results are achieved with either 1 or 3 iterations

Sinkhorn regularization parameter. Sinkhorn has a regularization parameter λ that needs to be tuned per dataset. We set $\lambda = \eta \cdot M$, where M is the maximum value in the cost matrix (of the currently evaluated pair of distributions), and tune η . In all of our three datasets the optimal setting is $\eta = 30$, which is the setting we use in Section 4. As an example, Figure 7(d) depicts the 1-NN accuracy (y-axis) of Sinkhorn-1 per η (x-axis).

Figure 7: Additional Sinkhorn and ACT experiments



B.2 Additional Pipeline Results

The next tables summarize the running times and parameters settings of all pipelines considered in our experiments (whereas the main text focuses on pipelines that start with Quadtree, since it is superior as a first step to Mean and Overlap). The listed parameters are the number of output candidates of each step in the pipeline.

In the baseline pipelines, parameters are tuned to achieve optimal performance (i.e., minimize the running time while attaining the recall goal on at least 90% of the queries). The details of the tuning procedure is as follows. For all pipelines we use the same random subset of 300 queries for tuning. Suppose the pipeline has ℓ algorithms. For $i = 1, \dots, \ell$, let c_i the output number of candidates of the i th algorithm in the pipeline. Note that c_ℓ always equals either 1 or 5, according to the recall goal of the pipeline, so we need to set $c_1, \dots, c_{\ell-1}$. Let p_1 be the recall@1 accuracy of the first algorithm in the pipeline. Namely, p_1 is the fraction of queries such that the top-ranked c_1 candidates by the first algorithm contain the true nearest neighbor. We calculate 10 possible values of c_1 , corresponding to $p_1 \in \{0.9, 0.91, \dots, 0.99\}$. We optimize the pipeline by a full grid search over those values of c_1 and all possible values of $c_2, \dots, c_{\ell-1}$.

When introducing Flowtree into a pipeline as an intermediate method, we do not re-optimize the parameters, but rather set its output number of candidates to the maximum between 10 and twice the output number of candidates of the subsequent algorithm in the pipeline. Re-optimizing the parameters could possibly improve results.

Pipeline methods	Candidates	Time
Mean, Sinkhorn-1, Exact	1476, 11, 1	0.543
Mean, Sinkhorn-3, Exact	1476, 5, 1	0.598
Mean, R-WMD, Exact	1850, 28, 1	0.428
Mean, ACT-1, Exact	1677, 14, 1	0.420
Overlap, Sinkhorn-1, Exact	391, 6, 1	0.610
Overlap, Sinkhorn-3, Exact	391, 5, 1	0.691
Overlap, R-WMD, Exact	576, 14, 1	0.367
Overlap, ACT-1, Exact	434, 10, 1	0.429
Quadtree, Sinkhorn-1, Exact	295, 5, 1	0.250
Quadtree, Sinkhorn-3, Exact	227, 3, 1	0.248
Quadtree, R-WMD, Exact	424, 12, 1	0.221
Quadtree, ACT-1, Exact	424, 8, 1	0.236

Table 4: Recall@1, no Flowtree.

Pipeline methods	Candidates	Time
Mean, Flowtree , Sinkhorn-1, Exact	1850, 10, 5, 1	0.089
Mean, Flowtree , Sinkhorn-3, Exact	1677, 10, 4, 1	0.077
Mean, Flowtree , R-WMD, Exact	2128, 48, 24, 1	0.242
Mean, Flowtree , ACT-1, Exact	2128, 20, 10, 1	0.138
Overlap, Flowtree , Sinkhorn-1, Exact	489, 10, 5, 1	0.087
Overlap, Flowtree , Sinkhorn-3, Exact	576, 10, 3, 1	0.076
Overlap, Flowtree , R-WMD, Exact	576, 28, 14, 1	0.173
Overlap, Flowtree , ACT-1, Exact	576, 16, 8, 1	0.119
Quadtree, Flowtree , Sinkhorn-1, Exact	424, 10, 5, 1	0.074
Quadtree, Flowtree , Sinkhorn-3, Exact	424, 10, 3, 1	0.059
Quadtree, Flowtree , R-WMD, Exact	424, 22, 11, 1	0.129
Quadtree, Flowtree , ACT-1, Exact	424, 16, 8, 1	0.104
Mean, Flowtree , Exact	1850, 9, 1	0.105
Overlap, Flowtree , Exact	489, 9, 1	0.100
Quadtree, Flowtree , Exact	424, 9, 1	0.092

Table 5: Recall@1, with Flowtree.

Pipeline methods	Candidates	Time
Mean, Sinkhorn-1	1476, 5	0.464
Mean, Sinkhorn-3	1476, 5	0.549
Mean, R-WMD, Exact	1850, 28, 5	0.426
Mean, ACT-1, Exact	1677, 14, 5	0.423
Overlap, Sinkhorn-1	391, 5	0.560
Overlap, Sinkhorn-3	391, 5	0.650
Overlap, R-WMD, Exact	576, 14, 5	0.368
Overlap, ACT-1, Exact	434, 10, 5	0.428
Quadtree, Sinkhorn-1	295, 5	0.222
Quadtree, Sinkhorn-3	227, 5	0.200
Quadtree, R-WMD, Exact	424, 11, 5	0.216
Quadtree, ACT-1, Exact	424, 7, 5	0.222

Table 6: Recall@5, no Flowtree.

Pipeline methods	Candidates	Time
Mean, Flowtree , Sinkhorn-1	1850, 10, 5	0.046
Mean, Flowtree , Sinkhorn-3	1476, 10, 5	0.043
Mean, Flowtree , R-WMD, Exact	2128, 48, 24, 5	0.237
Mean, Flowtree , ACT-1	2128, 10, 5	0.048
Overlap, Flowtree , Sinkhorn-1	391, 10, 5	0.042
Overlap, Flowtree , Sinkhorn-3	391, 10, 5	0.044
Overlap, Flowtree , R-WMD, Exact	576, 28, 14, 5	0.173
Overlap, Flowtree , ACT-1	576, 10, 5	0.046
Quadtree, Flowtree , Sinkhorn-1	424, 10, 5	0.033
Quadtree, Flowtree , Sinkhorn-3	424, 10, 5	0.034
Quadtree, Flowtree , ACT-1	424, 10, 5	0.029
Mean, Flowtree	2128, 5	0.043
Overlap, Flowtree	576, 5	0.039
Quadtree, Flowtree	645, 5	0.027
Quadtree, Flowtree , R-WMD, Exact	424, 22, 11, 5	0.131
Quadtree, Flowtree , ACT-1, Exact	424, 16, 8, 5	0.103

Table 7: Recall@5, with Flowtree.