Even Faster Algorithm for the Chamfer Distance

Ying Feng[*] Piotr Indyk[†]

September 14, 2025

Abstract

For two d-dimensional point sets A, B of size up to n, the Chamfer distance from A to B is defined as $CH(A,B) = \sum_{a \in A} \min_{b \in B} \|a - b\|$. The Chamfer distance is a widely used measure for quantifying dissimilarity between sets of points, used in many machine learning and computer vision applications. A recent work of Bakshi et al, NeuriPS'23, gave the first near-linear time $(1+\varepsilon)$-approximate algorithm, with a running time of $\mathcal{O}(nd \log(n)/\varepsilon^2)$. In this paper we improve the running time further, to $\mathcal{O}(nd(\log\log n + \log \frac{1}{\varepsilon})/\varepsilon^2)$. When ε is a constant, this reduces the gap between the upper bound and the trivial $\Omega(dn)$ lower bound significantly, from $\mathcal{O}(\log n)$ to $\mathcal{O}(\log\log n)$.

[*]MIT. E-mail: yingfeng@mit.edu
[†]MIT. E-mail: indyk@mit.edu

1 Introduction

For any two d-dimensional point sets A, B of sizes up to n, the Chamfer distance from A to B is defined as

$$\mathsf{CH}(A, B) = \sum_{a \in A} \min_{b \in B} \|a - b\|$$

where $\|\cdot\|$ is the underlying norm defining the distance between the points. Chamfer distance and its variant, the Relaxed Earth Mover Distance [KSKW15, AM19], are widely used metrics for quantifying the distance between two *sets* of points. These measures are especially popular in fields such as machine learning (e.g.,[KSKW15, WCL+19]) and computer vision (e.g.,[AS03, SMFW04, FSG17, JSQJ18]). A closely related notion of "the sum of maximum similarities", where $\min_{b \in B} \|a - b\|$ is replaced by $\max_{b \in B} a \cdot b$, has been recently popularized by the ColBERT system [KZ20]. Efficient subroutines for computing Chamfer distances are provided in prominent libraries including Pytorch [pyt23], PDAL [pda23] and Tensorflow [ten23]. In many applications (e.g., see [KSKW15]), Chamfer distance is favored as a faster alternative to the more computationally intensive Earth-Mover Distance or Wasserstein Distance.

Despite the popularity of Chamfer distance, efficient algorithms for computing it haven't attracted as much attention as algorithms for, say, the Earth-Mover Distance. The first improvement to the naive $\mathcal{O}(dn^2)$-time algorithm was obtained in [SMFW04], who utilized the fact that $\mathsf{CH}(A, B)$ can be computed by performing $|A|$ nearest neighbor queries in a data structure storing B. However, even when the state of the art approximate nearest neighbor algorithms are used, this leads to an $(1 + \epsilon)$-approximate estimator with only slightly sub-quadratic running time of $\mathcal{O}\left(dn^{1+\frac{1}{2(1+\epsilon)^2-1}}\right)$ in high dimensions [AR15][1]. The first near-linear-time algorithm for any dimension was proposed only recently in [BIJ+23], who gave a $(1 + \epsilon)$-approximation algorithm with a running time of $\mathcal{O}(dn \log(n)/\varepsilon^2)$, for ℓ_1 and ℓ_2 norms. Since any algorithm for approximating the distance must run in at least $\Omega(dn)$ time[2], the the upper and lower running time bounds differed by a factor of $\log(n)/\varepsilon^2$.

Our result: In this paper we make a substantial progress towards reducing the gap between the upper and lower bounds for this problem. In particular, we show the following theorem. Assume a Word RAM model where both the input coordinates and the memory/processor word size is $\mathcal{O}(\log n)$ bits.[3] Then:

Theorem 1.1. *There is an algorithm that, given two sets A, B of d-dimensional points with coordinates in $\{1 \ldots \mathrm{poly}(n)\}$ and a parameter $\varepsilon > 0$, computes a $(1 + \varepsilon)$-approximation to the Chamfer distance from A to B under the ℓ_1 metric, in time*

$$\mathcal{O}(nd(\log\log n + \log\frac{1}{\varepsilon})/\varepsilon^2)).$$

The algorithm is randomized and is correct with a constant probability.

Thus, we reduce the gap between upper and lower bounds from $\mathcal{O}(\log(n)/\varepsilon^2)$ to $\mathcal{O}(\log\log n + \log\frac{1}{\varepsilon})/\varepsilon^2)$.

[1]All algorithms considered in this paper are randomized, and return $(1 + \varepsilon)$-approximate answers with a constant probability

[2]The Chamfer Distance could be dominated by the distance from a single point $a \in A$ to B.

[3]In the Appendix, we adopt the reduction of [BIJ+23] to extend the result to coordinates of arbitrary finite precision.

1.1 Our techniques

Our result is obtained by identifying and overcoming the bottlenecks in the previous algorithm [BIJ+23]. On a high level, that algorithm consists of two steps, described below. For the sake of exposition, in what follows we assume that the target approximation factor $1+\varepsilon$ is some constant.

Outline of the prior algorithm: In the first step, for each point $a \in A$, the algorithm computes an estimate $\mathcal{D}_a$ of the distance opt_a from a to its nearest neighbor in B. The estimate is $\mathcal{O}(\log n)$-approximate, meaning that we $\mathsf{opt}_a \leq \mathcal{D}_a \leq \mathcal{O}(\log n)\mathsf{opt}_a$. This is achieved as follows. First, the algorithm imposes $\mathcal{O}(\log n)$ grids of sidelength $1, 2, 4, \ldots$, and maps each point in B to the corresponding cells. Then, for each a, it identifies the finest grid cell containing both a and some point $b \in B$. Finally, it uses the distance between a and b as an estimate $\mathcal{D}_a$. To ensure that this process yields an $\mathcal{O}(\log n)$-approximation, each grid needs to be *independently* shifted at random. We emphasize that this independence between the shifts of different grids is *crucial* to ensure the $\mathcal{O}(\log n)$-approximation guarantee - the more natural approach of using "nested grids" does not work. The whole process takes $\mathcal{O}(nd)$ time per grid, or $\mathcal{O}(nd \log n)$ time overall.

In the second step, the algorithm estimates the Chamfer distance via *importance sampling.* Specifically, the algorithm samples T points from A, such that the probability of sampling a is proportional to the estimate $\mathcal{D}_a$. For each sampled point a, the distance opt_a from a to its nearest neighbor in B is computed directly in $\mathcal{O}(nd)$ time. The final estimate of the Chamfer distance is equal to the weighted average the T values opt_a . It can be shown that if the number of samples T is equal to the distortion $\mathcal{O}(\log n)$ of the estimates $\mathcal{D}_a$, this yields a constant factor approximation to the Chamfer distance from A to B. The overall cost of the second step is $\mathcal{O}(Tnd) = \mathcal{O}(nd \log n)$, i.e., asymptotically the same as the cost of the first step.

Intuitions behind the new algorithm: To improve the running time, we need to reduce the cost of each of the two steps. In what follows we outline the obstacles to this task and how they can be overcome.

Step 1: The main difficulty in reducing the cost of the first step is that, for each grid, the point-to-cell assignment takes $\mathcal{O}(nd)$ time to compute, so computing these T assignments separately for each grid takes $\mathcal{O}(ndT)$ time. And, since each grid is independently translated by a different random vector, the grids are not nested, i.e., a (smaller) cell of side length 2^i might contain points from many (larger) cells of side length 2^{i+1}. As a result, is unclear how to reuse the point-to-cell assignment in one grid to speedup the assignment in another grid, while computing them separately takes $\mathcal{O}(ndT)$ time.

To overcome this difficulty, we abandon independent shifts and resort to $\mathcal{O}(\log n)$ *nested* grids. Such grids can be viewed as forming a *quadtree* with $\mathcal{O}(\log n)$ levels, where any cell C at level $i+1$ (i.e., of side length 2^{i+1}) is connected to 2^d cells at level i contained in C. (Note that the root node of the quadtree has the highest level $\mathcal{O}(\log n)$). Although using a single quadtree increases the approximation error, we show that using *two* independently shifted quadtrees retains the $\mathcal{O}(\log n)$ approximation factor. That is, we repeat the process of finding the finest grid cell containing both a and some point from B twice, and return the point in B that is closer to a. This amplifies the probability of finding a point from B that is "close" to a, which translates into a better approximation factor compared to using a single quadtree.

We still need show that the point-to-cell assignments can be computed efficiently. To this end, we observe that for each point a, its assignment to all $\mathcal{O}(\log n)$ nested grids can be encoded as d words of length $\mathcal{O}(\log n)$, or a $d \times \mathcal{O}(\log n)$ bit matrix M. Each row corresponds to one of the d coordinates, and the most significant bit of a row indicates the assignment to cells at the highest level (i.e. cells with the largest side length) with respect to that coordinate. In other words, the most significant bits of all coordinates are packed into the first column, etc. We observe that two points a and b lie in the same cell of side length 2^i if and only if their matrices agree in all but the last i columns. If we *transpose* M and read the resulting matrix in the row-major order, then finding a point $b \in B$ in the finest grid cell containing a is equivalent to finding b that shares the longest common prefix with a. We show that this transposition can be done using $\mathcal{O}(\log n \cdot \log\log n)$ simple operations on words, yielding $\mathcal{O}(n \log n \cdot \log\log n) = \mathcal{O}(nd \cdot \log\log n)$ time overall.

As an aside, we note that quadtree computation is a common task in many geometric algorithms [HP11]. Although an $\mathcal{O}(n)$ algorithm for this task was known for constant dimension d [Cha08][4], to the best of our knowledge our algorithm is the first to achieve $\mathcal{O}(nd \cdot \log\log n)$ time for arbitrary dimension.

Step 2: At this point we computed estimates $\mathcal{D}_a$ such that $\mathsf{opt}_a \le \mathcal{D}_a \le \mathcal{O}(\log n)\mathsf{opt}_a$. Given these estimates, importance sampling still requires sampling $\Omega(\log n)$ points. Therefore, we improve the running time by *approximating* (up to a constant factor) the values opt_a, as opposed to computing them exactly. This is achieved by computing $\mathcal{O}(\log\log n)$ random projections of the input points, which ensures that that the distance between any fixed pair of points is well-approximated with probability $1 - 1/\mathrm{poly}(\log n)$. We then employ these projections in a variant of the tournament algorithm of [Kle97] which computes $\mathcal{O}(1)$-approximate estimates of opt_a for $\mathcal{O}(\log n)$ sampled points a in $\mathcal{O}(nd \log\log n)$ time. Since the algorithm of [Kle97] works for the ℓ_2 metric as opposed to the ℓ_1 metric, we replace Gaussian random projections with Cauchy random projections, and re-analyze the algorithm.

This completes the overview of an $\mathcal{O}(nd \log\log n)$-time algorithm for estimate the Chamfer distance up to a *constant* factor. To achieve a $(1+\varepsilon)$-approximation guarantee for any $\varepsilon > 0$, we proceed as follows. First, instead of sampling $\mathcal{O}(\log n)$ points as before, we sample $\mathcal{O}(\log(n)/\varepsilon^2)$ points a. Then, we use the tournament algorithm to compute $\mathcal{O}(1)$-approximations to opt_a, as before. [5] Then we use a technique called *rejection sampling* to simulate the process of sampling $\mathcal{O}(1/\varepsilon^2)$ points a with probability proportional to $\Theta(\mathsf{opt}_a)$. For each such point, we compute opt_a exactly in $\mathcal{O}(nd)$ time. Finally, we use the $\mathcal{O}(1/\varepsilon^2)$ sampled points a and the exact values of opt_a in importance sampling to estimate the Chamfer distance up to a factor of $1 + \varepsilon$.

This concludes the overview of our algorithm for the Chamfer distance under the ℓ_1 metric. We remark that [BIJ+23] also extends their result from the ℓ_1 metric to the ℓ_2 metric by first embedding points from ℓ_2 to ℓ_1 using random projections. This takes $\mathcal{O}(nd \cdot \log n)$ time, which exceeds the runtime of our algorithm, eliminating our improvement. However, a faster embedding method would yield an improved runtime for the Chamfer distance under the ℓ_2 metric. We leave finding a faster embedding algorithm as an open problem.

[4]Assuming that each coordinate can be represented using $\log n$ bits.

[5]Note that we could use the tournament algorithm to report $(1+\varepsilon)$-approximate answers, but then the dependence of the running time on $1/\varepsilon$ would become *quartic*, as the $1/\varepsilon^2$ term in the sample size would be multiplied by another $1/\varepsilon^2$ term in the bound for the number of projections needed to guarantee that the tournament algorithm returns $(1+\varepsilon)$-approximate answers.

2 Preliminaries

In this paper, we consider the regime where the approximation factor $\varepsilon \geq \sqrt{\frac{\log n}{n}}$. Note that otherwise, an $\mathcal{O}(nd/\varepsilon^2)$ time bound would be close to the runtime of a naive exact computation.

In the proof of Theorem 1.1, we assume a Word RAM model where both the input coordinates and the memory/processor word size is $\mathcal{O}(\log n)$ bits. This model is particularly important in procedures `Concatenate` and `Transpose`, where we rely on the fact that we can shift bits and perform bit-wise AND, ADD and OR operations in constant time.

Notation: For any integers $a \geq 1$, we use $[n]$ to denote the set of all integers from 1 to n. For any two real numbers a, b such that $a \leq b$, we use $[a, b]$ to denote the set of all reals from a to b. Let d be the dimension of points.

For any $q \in \mathbb{R}^d$, define $\mathsf{opt}_q^P := \min_{p \in P} \|q - p\|_1$ for some subset P of $\mathbb{R}^d$. We will omit the superscript P when it is clear in the context.

3 Quadtree

In Figure 1, we show an algorithm `QuadTree` that outputs crude estimations of the nearest distances simulatenously for a set of points. The estimation guarantee is the same as the `CrudeNN` algorithm in [BIJ$^+$23]. While [BIJ$^+$23] achieves this using a quadtree with $\log n$ *independent* levels, which naturally introduce a $\log n$ runtime overhead, we show that two compressed quadtrees with dependent levels suffice. Our construction of compressed quadtrees is a generalization of [Cha08] to high dimensions.

Correctness: For any $x \in [0, \alpha]^d$ and any integer k such that $0 \leq k \leq t$, let $h_k(x) := (\lceil \frac{\vec{x}_1 + \vec{z}_1}{2^k} \rceil, \lceil \frac{\vec{x}_2 + \vec{z}_2}{2^k} \rceil, \cdots, \lceil \frac{\vec{x}_d + \vec{z}_d}{2^k} \rceil)$, where z is the random point drawn on Line 1 in Figure 1. Observe that $h_k(x)$ is related to the prefix of $h(x)^\top$.

Claim 3.1. *Let $q, p \in [0, \alpha]^d$ be arbitrary. For any integer k such that $0 \leq k \leq t$, $h_k(q) = h_k(p)$ if and only if $h(q)^\top$ and $h(p)^\top$ share a common prefix of length at least $d(t - k)$.*

Proof. If $h(q)^\top$ and $h(p)^\top$ share a common prefix of length at least $d(t-k)$, then in hashes $h(q)$ and $h(p)$, the first $(t-k)$ bits of all d coordinates are the same. $h_k(q)$ and $h_k(p)$ compute exactly these bits, thus $h_k(q) = h_k(p)$. The reverse direction holds symmetrically. □

Claim 3.1 justifies using $h_k(\cdot)$'s as an alternative representation of the binary string $h(\cdot)^\top$. [BIJ$^+$23] shows that h_k has a locality-sensitive property, which will help us bound the distance between points.

Claim 3.2 (Lemma A.4 of [BIJ$^+$23])**.** *For any fixed integer k such that $0 \leq k \leq t$ and any two points $q, p \in [0, \alpha]^d$,*

$$\mathbf{Pr}\big[h_k(q) \neq h_k(p)\big] \leq \frac{\|q-p\|_1}{2^k},$$

$$\mathbf{Pr}\big[h_k(q) = h_k(p)\big] \leq \exp\big(-\frac{\|q-p\|_1}{2^k}\big),$$

where the probabilities are over the random choice of z.

`QuadTree`

Input: Two size-n subsets $Q := \{q_i\}_{i\in[n]}$ and $P := \{p_i\}_{i\in[n]}$ of a metric space $(\mathbb{R}^d, \|\cdot\|_1)$, such that $Q, P \subset [0,\alpha]^d$ for some bound $\alpha = \mathrm{poly}(n)$.
Output: A set of n values $\{\mathcal{D}_i\}_{i\in[n]}$, such that every $\mathcal{D}_i \in \mathbb{R}$ satisfies $\mathcal{D}_i \geq \mathsf{opt}^P_{q_i}$.

1. Let $t = \lceil \log(\alpha) \rceil + 1$. Sample two uniformly random points $z, z' \sim [0, 2^{t-1}]^d$. For any point $x \in [0,\alpha]^d$, define

$$h(x) := (\lceil \vec{x}_1 + \vec{z}_1 \rceil, \lceil \vec{x}_2 + \vec{z}_2 \rceil, \cdots, \lceil \vec{x}_d + \vec{z}_d \rceil),$$

$$h'(x) := (\lceil \vec{x}_1 + \vec{z'}_1 \rceil, \lceil \vec{x}_2 + \vec{z'}_2 \rceil, \cdots, \lceil \vec{x}_d + \vec{z'}_d \rceil),$$

   where $\vec{x}_i, \vec{z}_i, \vec{z'}_i$ are the i-th coordinates of x, z, z', respectively.

2. For each $x \in Q \cup P$:
   - Compute $h(x)$ and write each element of $h(x)$ as a t-bit binary string. Then $h(x)$ can be viewed as a d-by-t binary matrix stored in the row-major order, whose (i,j)-th entry is the j-th significant bit of the i-th element of $h(x)$. Transpose this matrix and concatenate the rows of the transpose. Denote the resulting binary string as $h(x)^\top$.
   - Similarly, compute $h'(x)^\top$.

3. Use $h(x)^\top$ as keys to sort all $x \in Q \cup P$. Also, use $h'(x)^\top$ as keys to sort all $x \in Q \cup P$.

4. For each $q_i \in Q$:
   - Use the sort to find a $p \in P$ that maximizes the length l of the longest common prefix of $h(q_i)^\top$ and $h(p)^\top$. Similarly, find a $p' \in P$ that maximizes the length l' of the longest common prefix of $h'(q_i)^\top$ and $h'(p')^\top$.
   - If $l \geq l'$ then output $\mathcal{D}_i := \|q_i - p\|_1$; otherwise, output $\mathcal{D}_i := \|q_i - p'\|_1$.

Figure 1: The `QuadTree` Algorithm.

We now show that if two points have the same hash h_k, then their distance is likely not too much greater than 2^k. A straight-forward bound follows from the diameter of the d-dimensional cube.

Lemma 3.3. *For all $q \in Q$, $p \in P$, and $0 \leq k \leq t$, the following always holds: If $h_k(q) = h_k(p)$ then $\|q-p\|_1 \leq 2^k \cdot d$.*

Proof. Observe that $h_k(q) = h_k(p)$ only if $q+z$ and $p+z$ are in the same d-dimensional cube of side-length 2^k. The diameter of such a cube under the ℓ_1 norm is $2^k \cdot d$. Therefore, for any q, p and $0 \leq k \leq t$, $\|q-p\|_1 \leq 2^k \cdot d$ is a necessary condition for $h_k(q) = h_k(p)$ to hold. □

Moreover, using Claim 3.2, we can bound this ratio with respect to n.

Lemma 3.4. *With probability at least $1 - \mathcal{O}(1/n)$, the following holds simultaneously for all $q \in Q$, $p \in P$, and $0 \le k \le t$: If $h_k(q) = h_k(p)$ then $\|q-p\|_1 \le 2^k \cdot 3\log n$.*

Proof. We show the contrapositive that with probability $1 - \mathcal{O}(1/n)$, $k < \log(\|q-p\|_1/3\log n)$ implies $h_k(q) \neq h_k(p)$ simultaneously for all $q \in Q$ and $p \in P$. It suffices to argue that for any fixed pair of points $q \in Q$ and $p \in P$, this holds with probability at least $1 - \mathcal{O}(1/n^3)$. The lemma then follows by a union bound over n^2 pairs.

Let k_0 denote the largest integer k that satisfies $k < \log(\|q-p\|_1/3\log n)$. Then we have

$$\mathbf{Pr}\big[h_{k_0}(q) = h_{k_0}(p)\big] \le \exp\big(-\frac{\|q-p\|_1}{2^{k_0}}\big) \le \exp(-3\log n).$$

i.e., with probability at least $1 - \mathcal{O}(1/n^3)$, $h_{k_0}(q) \neq h_{k_0}(p)$. Also, it is easy to see that if $h_{k_0}(q) \neq h_{k_0}(p)$, then for all $k \le k_0$, $h_k(q) \neq h_k(p)$, concluding the claim. $\square$

Symmetrically, if we define $h'_k(x) := (\lceil \frac{\vec{x}_1 + \vec{z'}_1}{2^k} \rceil, \lceil \frac{\vec{x}_2 + \vec{z'}_2}{2^k} \rceil, \cdots, \lceil \frac{\vec{x}_d + \vec{z'}_d}{2^k} \rceil)$, the claims and lemmas above also hold for h'_k. Using these, we show that the expected outputs of the `QuadTree` algorithm are (crude) estimations of the nearest neighbor distances.

Theorem 3.5. *With probability at least $1 - \mathcal{O}(1/n)$, it holds for all $q_i \in Q$ that $\mathbf{E}[\mathcal{D}_i] \le 5\min(d, 3\log n) \cdot \mathsf{opt}^P_{q_i}$.*

Proof. We assume the success case of Lemma 3.4 for both h_k and h'_k. Fix an arbitrary $q_i \in Q$. Recall that the `QuadTree` algorithm finds $p, p' \in P$ for q_i, which are associated with longest common prefixes of lengths l, l', respectively. For integer $k : 0 \le k \le t$, let $\mathcal{E}_k$ denote the event $d(t-k) \le \max(l, l') < d(t-k+1)$. Observe from Claim 3.1 that when $\mathcal{E}_k$ happens,

- either $l \ge l'$ and $h_k(q_i) = h_k(p)$,
- or $l' > l$ and $h'_k(q_i) = h'_k(p')$.

In both cases, we know from Lemma 3.3 and 3.4 that $\mathcal{D}_i \le 2^k \cdot \min(d, 3\log n)$.

Let $D := \min(d, 3\log n)$, $p^* := \arg\min_{p \in P} \|q_i - p\|_1$, and $k^* := \lceil \log(\mathsf{opt}_{q_i}) \rceil$. We have

$$\begin{aligned}
\mathbf{E}[\mathcal{D}_i] &\le \sum_{0 \le k \le t} \mathbf{Pr}\big[\mathcal{E}_k\big] \cdot (2^k \cdot D) \\
&\le D\Big(\sum_{0 \le k \le k^*} \mathbf{Pr}\big[\mathcal{E}_k\big] \cdot \mathsf{opt}_{q_i} + \sum_{k^* < k \le t} \mathbf{Pr}\big[h_{k-1}(q_i) \neq h_{k-1}(p^*) \wedge h'_{k-1}(q_i) \neq h'_{k-1}(p^*)\big] \cdot 2^k\Big)
\end{aligned}$$

where the second inequality holds because $\mathcal{E}_k$ implies that neither pair $\{h(q_i)^\top, h(p^*)^\top\}$ nor $\{h'(q_i)^\top, h'(p^*)^\top\}$ share a common prefix of length $\ge d(t-k+1)$. Thus $h_{k-1}(q_i) \neq h_{k-1}(p^*)$ and $h'_{k-1}(q_i) \neq h'_{k-1}(p^*)$ by Claim 3.1.

Moreover, events $\mathcal{E}_k$ for all k form a partition of a sample space, so $\sum_k \mathbf{Pr}\big[\mathcal{E}_k\big] \le 1$. Applying this and the locality sensitive properties of h_{k-1} and h'_{k-1}, we get

$$\mathbf{E}[\mathcal{D}_i] \le D\big(\mathsf{opt}_{q_i} + \sum_{k^* < k \le t} (\frac{\mathsf{opt}_{q_i}}{2^{k-1}})^2 \cdot 2^k\big) \le D\big(\mathsf{opt}_{q_i} + 2\mathsf{opt}_{q_i} \sum_{k^* < k \le t} \frac{\mathsf{opt}_{q_i}}{2^{k-1}}\big) \le 5D \cdot \mathsf{opt}_{q_i}$$

$\square$

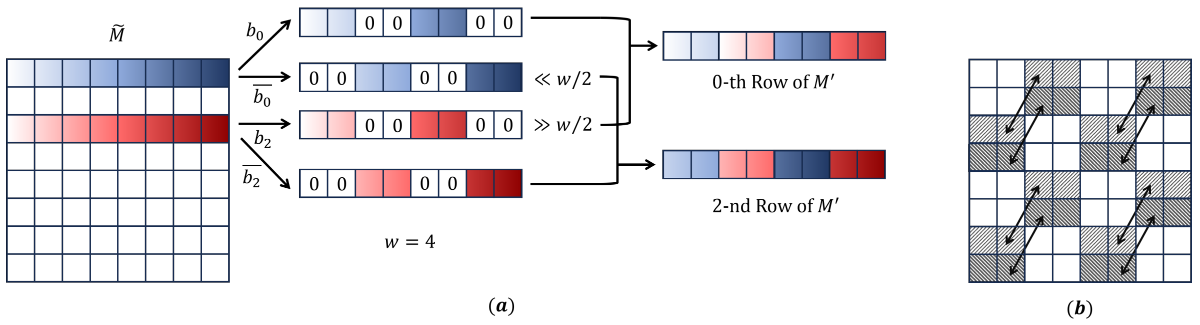

Figure 2: A pictorial example for 8-by-8 square matrix and $w = 4$. The left figure (a) shows how the `Transpose` algorithm handles the rows of $\tilde{M}$ in lines 2a and 3a. The right figure (b) illustrates the transpose outcome.

Runtime analysis:

Lemma 3.6 (Line 2)**.** *For any $x \in Q \cup P$, $h(x)^\top$ (and $h'(x)^\top$) can be computed in $\mathcal{O}(d \log\log n)$ time.*

Proof. We assume without loss of generality that both d, t are powers of 2. Computing the binary matrix representation of $h(x)$ can be done in $\mathcal{O}(d)$ time since $t = \mathcal{O}(\log n)$. Given this, we compute $h(x)^\top$ as follows.

Case 1: $d \geq t$: We partition the matrix into t-by-t square submatrices, denoted by

$$\mathsf{Matrix}(h(x)) := \left.\underbrace{\begin{bmatrix} M_1 \\ \hline M_2 \\ \hline \vdots \\ \hline M_{d/t} \end{bmatrix}}_{t}\right\} d$$

For each $i \in [d/t]$, we use a recursive subroutine `Transpose`(M_i, t) to compute $M_i^\top$. See Figure 2 for a pictorial illustration of the `Transpose` algorithm.

The correctness of the `Transpose` algorithm can be shown by induction on (the base-2 logarithm of) w. When $I = J = t = \mathcal{O}(\log n)$, Line 2a and 3a can be done using a constant number of operations on words. Thus we get the following runtime.

Claim 3.7. *Assuming $t = \mathcal{O}(\log n)$ the procedure `Transpose`(M_i, t) runs in $\mathcal{O}(t \cdot \log t)$ time.*

We execute the `Transpose` algorithm for all i, which takes $\mathcal{O}((d/t) \cdot t \log t) = \mathcal{O}(d \log\log n)$. Then we can write down $h(x)^\top$ by concatenating rows of $M_i^\top$'s, which takes $\mathcal{O}(t \cdot (d/t))$ time.

Case 2: $t > d$: We again partition the matrix into t-by-t square submatrices. In this case, we obtain $\mathsf{Matrix}(h(x)) = M := \left.\underbrace{\left[\, M_1 \mid M_2 \mid \ldots \mid M_{t/d} \,\right]}_{t}\right\} d$.

Claim 3.8. *Given $t = \mathcal{O}(\log n)$, `Transpose`(M, d) runs in $\mathcal{O}(d \log d) \leq \mathcal{O}(d \log\log n)$ time.*

Transpose

Input: An I-by-J bit matrix M where $I \leq J$ are powers of 2. An integer w that is a power of 2 and $2 \leq w \leq I$.
Output: An I-by-J matrix M' such that if it is partitioned into w-by-w square submatrices, then each submatrix is the transpose of the corresponding submatrix of M at the same coordinates.

```
1. Let
   $$\tilde{M} = \begin{cases} M & \text{if } w{=}2 \\ \texttt{Transpose}(M, w/2) & \text{otherwise} \end{cases}$$
   be zero-indexed and $\tilde{M}[i,j]$ is its $(i,j)$-th entry.
2. For each integer $i$ such that $0 \leq i < I$:
   (a) Compute a $J$-bit binary string $b_i$ such that for $j : 0 \leq j < J$, its $j$-th bit
       $b_i[j] = \begin{cases} \tilde{M}[i,j] & \text{if } (j \mod w) < w/2 \\ 0 & \text{otherwise} \end{cases}$.
       Also, compute a string $\overline{b_i}[j] = \begin{cases} 0 & \text{if } (j \mod w) < w/2 \\ \tilde{M}[i,j] & \text{otherwise} \end{cases}$.
3. Define an $I$-by-$J$ matrix $M'$, such that for each integer $0 \leq i < I$:
   (a) Let the $i$-th row of $M'$ be $\begin{cases} b_i + \overline{b_{i+(w/2)}} \gg (w/2) & \text{if } (i \mod w) < w/2 \\ \overline{b_i} + b_{i-(w/2)} \ll (w/2) & \text{if } (i \mod w) \geq w/2 \end{cases}$,
       where $\gg (w/2)$ (resp. $\ll$) denote the operation of shifting a string to the right (resp. left) by $w/2$ bits.
4. Output $M'$.
```

We execute $\texttt{Transpose}(M, d)$ and obtain $M' = \left[\, M_1^\top \;\middle|\; M_2^\top \;\middle|\; \dots \;\middle|\; M_{t/d}^\top \,\right]$. In principle, to obtain $h(x)^\top$, we just concatenate $d \cdot (t/d)$ rows of all $M_i^\top$. However, when $t \gg d$, this takes longer than $\mathcal{O}(d \log\log n)$ time. We instead use another recursive subroutine $\texttt{Concatenate}(M', d)$. An example of the Concatenate algorithm is given in Figure 3.

The correctness of the Concatenate algorithm can again be observed by inducting on the logarithm of w. Line 2a and 3a can be done using $\mathcal{O}((J/w) \cdot \lceil w/\log n \rceil)$ operations on words, and

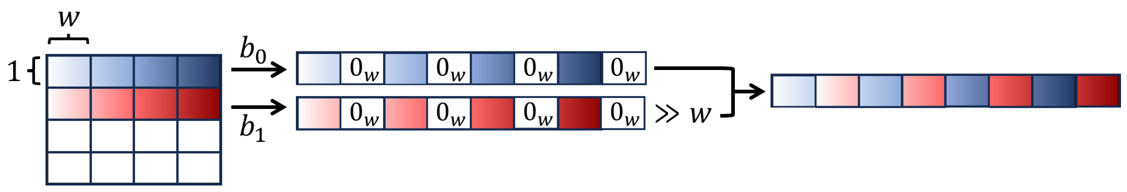

Figure 3: A pictorial example that shows the behavior of Line 2a and 3a of the `Concatenate` algorithm on a 4-by-4w matrix.

```
Concatenate

Input: An $I$-by-$J$ bit matrix $M$ where $I \leq J$ are powers of 2. An integer $w$ that is a power of 2 and $I \leq w \leq J$.
Output: An $IJ$-bit string $B$ such that if it is partitioned into $w$-bit blocks, then the $u$-th block (zero-indexed from left to right) are bits on the $(u \mod I)$-th row of $M$ from column $w \cdot \lfloor u/I \rfloor$ to $w \cdot \lfloor u/I \rfloor + w$.

  1. If $w = I$ then output $B = M$.
  2. For each integer $i$ such that $0 \leq i < I$:
      (a) Partition the $i$-th row of $M$ into $w$-bit blocks, denoted as $\overbrace{[\underbrace{b_{i,1}}_{w} \mid b_{i,2} \mid \ldots \mid b_{i,J/w}]}^{J}$.
          Compute a $2J$-bit string $b_i = [b_{i,1} \mid 0_w \mid b_{i,2} \mid 0_w \mid \ldots \mid b_{i,J/w} \mid 0_w]$, where $0_w$ is a $w$-bit all-zero string.
  3. Define an $I/2$-by-$2J$ matrix $M'$, such that for each integer $0 \leq i < I/2$:
      (a) Let the $i$-th row of $M'$ be $b_{2i} + b_{2i+1} \gg w$, where $\gg w$ is the operation of shifting a string to the right by $w$ bits.
  4. Output Concatenate$(M', 2w)$.
```

both lines are repeated for I times in each recursive call. Therefore, the total runtime is

$$\begin{aligned}\mathcal{O}(\sum_{s=1}^{\log(d)} 2^s \cdot \frac{2^{\log(d)-s}t}{2^{\log(d)-s}d} \cdot \lceil \frac{2^{\log(d)-s}d}{\log n} \rceil) &= \mathcal{O}(\sum_{s=1}^{\log(d)} 2^s \cdot \max(\frac{t}{d}, \frac{t}{d} \cdot \frac{2^{\log(d)-s}d}{\log n})) \\ &= \mathcal{O}(\log d \cdot \max(t, \frac{td}{\log n})) \\ &= \mathcal{O}(d \log\log n)\end{aligned}$$

□

Theorem 3.9. *The `QuadTree` algorithm runs in $\mathcal{O}(nd \log\log n)$ time.*

Proof. Computing $h(x)$ for all x takes $\mathcal{O}(nd)$ time. Then computing $h(x)^\top$ takes $\mathcal{O}(nd \log\log n)$ time. After that, sorting $\mathcal{O}(n)$-many $\mathcal{O}(d \log n)$-bit strings can be done in $\mathcal{O}(nd)$ time using radix sort. Finally, to find $p \in P$ with the longest common prefix for every $q \in Q$, we go through the sorted list and link each $q \in Q$ with adjacent $p \in P$, which takes $\mathcal{O}(n)$ total time. The above time bounds also hold for $h'(x)$'s, resulting in $\mathcal{O}(nd \log\log n)$ time in total.

□

4 Tournament

In this section, we compute the 2-approximation of the nearest neighbor distances for logarithmically many queries. We do so using a depth-2 tournament: we first partition input points into random groups, project them to a lower dimensional space, and collect the nearest neighbor in the projected space in every group as a set $\tilde{S}$. Then the final output of the tournament is the nearest neighbor among points in $\tilde{S}$ in the original space. Intuitively, because each random group is small, the true nearest neighbor could only lose to another near neighbor in the first step. Then in the second step, $\tilde{S}$ should contain at least one near neighbor.

Notation: We use the same notation $D := \min(d, 3\log n)$ as in the previous section. For any finite subset $T \subset \mathbb{R}$, let $\mathsf{med}T \in \mathbb{R}$ denote the median of T.

When working under the ℓ_1 norm, we use Cauchy random variables to project points. We first recall a standard bound on the median of projections, which will be useful for our analysis. (The following lemma essentially follows from Claim 2 and Lemma 2 in [Ind06]; we reprove it in the appendix for completeness.)

Lemma 4.1. *Let $x, y \in \mathbb{R}^d$ and $0 < c < 1/2$. Sample r random vectors $v_1, v_2, \cdots, v_r \sim (\mathsf{Cauchy}(0,1))^d$. With probability at least $1 - 2e^{-rc^2/50}$, $\mathsf{med}\{|v_i \cdot (x-y)| : i \in [r]\} \in (1 \pm c)\|x-y\|_1$.*

In Figure 4, we describe how to construct a data structure to find 2-approximate nearest neighbors. The construction borrows ideas from the second algorithm of [Kle97], but using a tournament of depth 2 instead of $\mathcal{O}(\log n)$.

Correctness:

We fix a query $q := q_i$.

Lemma 4.2. *With probability at least $1 - \frac{1}{10t}$, $\mathcal{D}_i \leq 2\mathsf{opt}_q$.*

We let S denote the set of all 2-approximate nearest neighbors to q, i.e., $S := \{p \in P : \|q-p\|_1 \leq 2\mathsf{opt}_q\}$, and let $p^* \in P$ denote a nearest neighbor of q, i.e. $\|q - p^*\|_1 = \mathsf{opt}_q$. To prove Lemma 4.2, we first make the following observation:

Lemma 4.3. *Let P' be an arbitrary subset of $P \setminus S$. The probability that there exists $p \in P'$ such that $\mathsf{med}_p \leq \mathsf{med}_{p^*}$, where $\mathsf{med}_p := \mathsf{med}\{|v_j \cdot (q-p)| : j \in [r]\}$ and $\mathsf{med}_{p^*} := \mathsf{med}\{|v_j \cdot (q-p^*)| : j \in [r]\}$, is at most $\frac{2(|P'|+1)}{t^2 \log n}$.*

Proof. From $P' \subseteq (P \setminus S)$ we know that $\|q-p\|_1 > 2\|q-p^*\|_1$ for any $p \in P'$. Therefore, if $\mathsf{med}_p \leq \mathsf{med}_{p^*}$ then either $\mathsf{med}_p \neq (1 \pm \frac{1}{4})\|q-p\|_1$ or $\mathsf{med}_{p^*} \neq (1 \pm \frac{1}{4})\|q-p^*\|_1$. Applying Lemma

```
Tournament

Input: A set of $t$ queries $\{q_i\}_{i\in[t]}$ and a set of $n$ points $P$, which are both subsets of a metric space $(\mathbb{R}^d, \|\cdot\|_1)$.
Output: A set of $t$ values $\{\mathcal{D}_i\}_{i\in[t]}$, such that every $\mathcal{D}_i \in \mathbb{R}$ satisfies $\mathcal{D}_i \geq \mathsf{opt}^P_{q_i}$.

Building the Data Structure.

  1. Let $r \geq 800(2\log t + \log\log n)$.

  2. For each $j \in [r]$, draw $v_j \sim (\mathsf{Cauchy}(0,1))^d$, compute $v_j \cdot p$ for all points $p \in P$, and store
     all $v_j$ and $v_j \cdot p$.

  3. Randomly partition $P$ into $n/\log n$ subsets $P_1, P_2, \cdots, P_{n/\log n}$, each of size $\log n$.

Processing the Queries.  For each query $q := q_i$ for $i \in [t]$:

  1. Compute $v_j \cdot q$ for all $j \in [r]$.

  2. Let $\tilde{S}$ be an empty set. For each $k \in [n/\log n]$:

       • Compute $\mathsf{med}_p := \mathsf{med}\{|v_j \cdot (q-p)| : j \in [r]\}$ for every $p \in P_k$.
       • Find $p := \arg\min_{p\in P_k}\{\mathsf{med}_p\}$ and add it into $\tilde{S}$.

  3. Output $\mathcal{D}_i := \min_{p\in\tilde{S}}\|q-p\|_1$ by computing and comparing all exact distances $\|q-p\|_1$
     for $p \in \tilde{S}$.
```

Figure 4: The `Tournament` Algorithm.

4.1 with $c = \frac{1}{4}$ and $r \geq 800(2\log t + \log\log n)$ and a union bound, we get that

$$\begin{aligned}\mathbf{Pr}\big[\exists p \in P' : \mathsf{med}_p \leq \mathsf{med}_{p^*}\big] &\leq \mathbf{Pr}\big[\exists p \in P' : \mathsf{med}_p \neq (1 \pm \frac{1}{4})\|q-p\|_1\big] + \\ &\quad\ \mathbf{Pr}\big[\mathsf{med}_{p^*} \neq (1 \pm \frac{1}{4})\|q-p^*\|_1\big] \\ &\leq (|P'|+1)\cdot 2e^{-(2\log t + \log\log n)} \\ &= \frac{2(|P'|+1)}{t^2\log n}.\end{aligned}$$

□

In Line 3 of the data structure building procedure, the point p^* is assigned to one of the subsets $P^* \in \{P_1, P_2, \cdots, P_{n/\log n}\}$. Focusing on this subset P^*, we can show that with high probability, either p^* is added into $\tilde{S}$, or p^* loses to another 2-approximate nearest neighbor. In both cases, the data structure is guaranteed to output an 2-approximation.

Proof (of Lemma 4.2). $P^* \setminus S$ contains at most $|P^*| = \log n$ points. Applying Lemma 4.3, we get $\mathsf{med}_{p^*} \geq \mathsf{med}_p$ simultaneously for all $p \in P^* \setminus S$ with probability at least $1 - \frac{2\log n}{t^2 \log n} \geq 1 - \frac{1}{10t}$ (as long as $t \geq 20$). Conditioned on this, $\arg\min_{p \in P^*}\{\mathsf{med}_p\}$ must be either p^* or some other element of S. Thus on line 3 of the algorithm, $\tilde{S}$ contains at least one element of S, so the final output $\min_{p\in\tilde{S}} \|q-p\|_1 \leq 2\mathsf{opt}_q$.

□

Applying a union bound on Lemma 4.2, we get the correctness guarantee:

Theorem 4.4. *Given t queries $\{q_i\}_{i\in[t]}$, with probability at least $9/10$, the* `Tournament` *algorithm outputs 2-approximate nearest neighbors simulataneously for all t queries.*

Finally, we state the runtime guarantee as follows:

Theorem 4.5. *The* `Tournament` *algorithm runs in $\mathcal{O}(n(d+t)(\log t + \log\log n) + dt^2 \log t \log n)$ time.*

Proof. For preprocessing, the algorithm projects all points in P using r projections, which takes $\mathcal{O}(n \cdot d \cdot r)$ time. To process a query q, we first take $\mathcal{O}(dr)$ time to project q. We then count the number of comparisons we make to find the minimums of medians, which is $\mathcal{O}((n/\log n)\log n \cdot r)$ using a linear-time median selection algorithm [BFP+73]. Each comparison can be done in $\mathcal{O}(1)$ time given that $v_j \cdot p$ and $v_j \cdot q$ for all $j \in [r]$ and $p \in P$ are stored. Finally, we use $\mathcal{O}(d \log n)$ time to do a linear scan over $\tilde{S}$.

We plug in $r = \mathcal{O}(\log t + \log\log n)$. For t queries, the total runtime is $\mathcal{O}(n(d+t)(\log t + \log\log n) + dt \log n)$. □

For our purpose of estimating the Chamfer distance, we will apply the `Tournament` algorithm with a number of queries $t = \Theta(D/\varepsilon^2)$ for $D = \min(d, 3\log n)$ and some $\varepsilon > 0$ satisfying $\varepsilon^{-2} = \mathcal{O}(\frac{n}{\log n})$. Under this setting, the runtime is dominated by the first additive term of Theorem 4.5, which is at most $\mathcal{O}(nd(\log\log n + \log\frac{1}{\varepsilon})/\varepsilon^2)$.

5 Rejection Sampling

Notation: All occurrences of opt in this section are with respect to the set B. Let $\varepsilon > 0$ be our target approximation factor. We call the distribution $\mathcal{P}$ an f-Chamfer distribution for some $f = f(n, d, \varepsilon)$, if it is supported on A and for every $a \in A$,

$$f \frac{\mathsf{opt}_a}{\mathsf{CH}(A,B)} \leq \mathcal{P}(a), \text{where we denote } \mathcal{P}(a) := \mathop{\mathbf{Pr}}_{x\sim\mathcal{P}}[x = a].$$

We first show a general bound for estimating the Chamfer distance using samples from a Chamfer distribution. This follows from a standard analysis of importance sampling.

Lemma 5.1. *Let $X := \{x_i\}_{i\in[t]}$ be a set of t samples drawn from a f-chamfer distribution $\mathcal{P}$. Fix $h = h(n,d,\varepsilon) \geq 1$. Given an arbitrarily $\tilde{\mathsf{opt}}_{x_i}$ for every x_i that satisfies $\mathsf{opt}_{x_i} \leq \tilde{\mathsf{opt}}_{x_i} \leq h \cdot \mathsf{opt}_{x_i}$, then for any $0 < \kappa < 1$,*

$$\mathbf{Pr}\left[\tilde{\mathsf{CH}}(A,B) \leq (1-\kappa)\mathsf{CH}(A,B)\right] + \mathbf{Pr}\left[\tilde{\mathsf{CH}}(A,B) \geq (1+\kappa)\cdot h \cdot \mathsf{CH}(A,B)\right] \leq \frac{\frac{h^2}{f} - 1}{t \cdot \kappa^2},$$

where $\tilde{\mathsf{CH}}(A,B) := \frac{\sum_{i\in[t]} \tilde{\mathsf{opt}}_{x_i}/\mathcal{P}(x_i)}{t}$.

Proof. For the purpose of analysis, assume that we additionally have arbitrary $\tilde{\mathsf{opt}}_a$ for $a \in (A \setminus X)$ that also satisfies $\mathsf{opt}_a \le \tilde{\mathsf{opt}}_a \le h \cdot \mathsf{opt}_a$. By linearity,

$$\mathbf{E}[\tilde{\mathsf{CH}}(A,B)] = \frac{\sum_{i\in[t]} \mathbf{E}[\tilde{\mathsf{opt}}_{x_i}/\mathcal{P}(x_i)]}{t} = \sum_{a\in A} \mathcal{P}(a)\cdot \frac{\tilde{\mathsf{opt}}_a}{\mathcal{P}(a)} \in [\mathsf{CH}(A,B), h\cdot \mathsf{CH}(A,B)].$$

We also bound the variance

$$\begin{aligned}\mathbf{Var}[\tilde{\mathsf{CH}}(A,B)] &\le \frac{\mathbf{E}[\tilde{\mathsf{opt}}^2_{x_1}/\mathcal{P}(x_1)^2]}{t} - \mathsf{CH}(A,B)^2 \\ &\le \frac{1}{t}(\sum_{a\in A} \frac{\tilde{\mathsf{opt}}^2_a}{\mathcal{P}(a)} - \mathsf{CH}(A,B)^2) \\ &\le \frac{1}{t}(\frac{h}{f}\mathsf{CH}(A,B)\cdot \sum_{a\in A} \tilde{\mathsf{opt}}_a - \mathsf{CH}(A,B)^2) \\ &\le \frac{1}{t}\cdot \mathsf{CH}(A,B)^2 \cdot (\frac{h^2}{f} - 1)\end{aligned}$$

where the third inequality follows from $\frac{1}{\mathcal{P}(a)} \le \frac{\mathsf{CH}(A,B)}{f\cdot \mathsf{opt}_a}$ and $\tilde{\mathsf{opt}}_a \le h\cdot \mathsf{opt}_a$. Finally, by Chebyshev's Inequality, we have

$$\mathbf{Pr}\Big[\Big|\tilde{\mathsf{CH}}(A,B) - \mathbf{E}[\tilde{\mathsf{CH}}(A,B)]\Big| \ge \kappa \cdot \mathsf{CH}(A,B)\Big] \le \frac{1}{t}\cdot \frac{\frac{h^2}{f}-1}{\kappa^2}.$$

$\square$

In this section, we aim to construct a set of samples $S = \{s_j\}_{j\in[s]}$ for some large enough s, such that each s_j is drawn from a fixed $\mathcal{O}(1)$-Chamfer distribution. Once we have S, we can compute a weighted sum of the nearest neighbor distances for $s_j \in S$, and invoke Lemma 5.1 to show that it is likely an $(1+\varepsilon)$-estimation of $\mathsf{CH}(A,B)$.

We will construct such S via a two-step sampling procedure: in the first step, we sample $\Theta(D/\varepsilon^2)$ points from A using a distribution defined by the estimations from the `QuadTree` algorithm. In the second step, we subsample these $\Theta(D/\varepsilon^2)$ points, using an acceptance probability defined by the estimations from the `Tournament` algorithm. We describe our **Chamfer-Estimate** algorithm in Figure 5.

The **Chamfer-Estimate** algorithm applies the `QuadTree` algorithm and the `Tournament` algorithm as subroutines. If they are executed successfully, their outputs should satisfy the following conditions:

Condition 5.2. *We say the `QuadTree` algorithm succeeds if for every $a \in A$, $\mathbf{E}[\mathcal{D}_a] \le 5D\cdot \mathsf{opt}_a$.*

Condition 5.3. *We say the `Tournament` algorithm succeeds if for every x_i for $i\in[q]$, $\mathcal{D}'_{x_i} \le 2\mathsf{opt}_{x_i}$.*

That is, as described in the introduction, we need `QuadTree` to provide $\mathcal{O}(\log n)$-approximation (to ensure that the sample size q can be at most logarithmic in n), and that `Tournament` provide $\mathcal{O}(1)$-approximation (to ensure that the final estimator using s samples has variance bounded by a constant).

We state some facts about the **Chamfer-Estimate** algorithm, which will be useful for our analysis.

Chamfer-Estimate

Input: Two subsets A, B of a metric space $(\mathbb{R}^d, \|\cdot\|_1)$ of size n, a parameter $\varepsilon > 0$, and a parameter $q \in \mathbb{N}$.
Output: An estimated value $\tilde{\mathsf{CH}}(A, B) \in \mathbb{R}$.

1. Execute the algorithm `QuadTree`(A, B), and let the output be a set of values $\{\mathcal{D}_a\}_{a \in A}$ which always satisfy $\mathcal{D}_a \geq \mathsf{opt}_a$. Let $\mathcal{D} := \sum_{a \in A} \mathcal{D}_a$.

2. Construct a probability distribution $\mathcal{P}$ supported on A such that for every $a \in A$, $\mathcal{P}(a) = \frac{\mathcal{D}_a}{\mathcal{D}}$. For $i \in [q]$, sample $x_i \sim \mathcal{P}$.

3. Execute the algorithm `Tournament`$(\{x_i\}_{i \in [q]}, B)$, and let the output be a set of values $\{\mathcal{D}'_{x_i}\}_{i \in [q]}$ which always satisfy $\mathcal{D}'_{x_i} \geq \mathsf{opt}_{x_i}$. Let $\mathcal{D}' := \sum_{i \in [q]} \frac{\mathcal{D}'_{x_i}}{\mathcal{P}(x_i)}/q$ and denote $\mathcal{P}'(a) := \frac{\mathcal{D}'_a}{\mathcal{D}'}$ (which is well-defined only if $a = x_i$ for some $i \in [q]$).

4. Define
$$M := \max_{i \in [q]} \frac{\mathcal{P}'(x_i)}{\mathcal{P}(x_i)}.$$
For each $i \in [q]$, mark x_i as ACCEPTED with probability $\frac{\mathcal{P}'(x_i)}{M \cdot \mathcal{P}(x_i)}$.
If the number of ACCEPTED x_i is less than $s = 10/\varepsilon^2$ then output **Fail** and exit the algorithm. Otherwise, collect the first s ACCEPTED x_i as a set $S := \{s_j\}_{j \in [s]}$.

5. Compute opt_{s_j} for each j. Output
$$\tilde{\mathsf{CH}}(A, B) := \sum_{j \in [s]} \frac{\mathsf{opt}_{s_j}}{\mathcal{P}'(s_j)}/s.$$

Figure 5: The **Chamfer-Estimate** Algorithm.

Claim 5.4 (Line 2)**.** *Under Condition 5.2, with probability at least* $9/10$*,* $\mathcal{P}$ *is a* $(1/50D)$*-Chamfer Distribution.*

Proof. With probability at least $9/10$, $\mathcal{D} \leq 50D \cdot \mathsf{CH}(A, B)$ by Markov's Inequality. Upon this condition, for any $a \in A$, $\frac{\mathsf{opt}_a}{50D \cdot \mathsf{CH}(A,B)} \leq \frac{\mathcal{D}_a}{\mathcal{D}}$. □

Claim 5.5 (Line 3)**.** *Let* $q \geq 10^4 D$*. Under Condition 5.2 and 5.3, with probability at least* $4/5$*,* $\mathcal{D}' \geq \mathsf{CH}(A, B)/2$*.*

Proof. We apply the importance sampling analysis in Lemma 5.1. We assume that Claim 5.4 holds and $\mathsf{opt}_{x_i} \leq \mathcal{D}'_{x_i} \leq 2\mathsf{opt}_{x_i}$, then
$$\mathbf{Pr}\big[\mathcal{D}' \leq (1 - \frac{1}{2})\mathsf{CH}(A, B)\big] \leq \frac{2^2 \cdot 50D - 1}{q \cdot (\frac{1}{2})^2} < \frac{1}{10}.$$
□

Analysis of S: We now show that the set S on Line 4 collects enough samples (thus the algorithm does not fail) and is equivalent to sampling from a $\mathcal{O}(1)$-Chamfer distribution $\mathcal{Q}$. We note that the algorithm, in fact, only knows a $(1/50D)$-Chamfer distribution $\mathcal{P}$ and probabilities $\mathcal{P}'(x_i)$ for $\{x_i\}_{i\in[q]}$, so it cannot explicitly sample from such $\mathcal{Q}$. Nevertheless, by a standard analysis of rejection sampling, we show that S "simulates" sampling from $\mathcal{Q}$.

Lemma 5.6. *Let $q \geq 10^4 D/\varepsilon^2$. Under Condition 5.2 and 5.3, with probability at least $3/5$, the number of* ACCEPTED x_i *is at least s, so the algorithm does not fail.*

Proof. We assume that Claim 5.4 and 5.5 hold. Then for any x_i, $\frac{1}{\mathcal{P}(x_i)} \leq \frac{50D\cdot\mathsf{CH}(A,B)}{\mathsf{opt}_{x_i}}$ and $\mathcal{P}'(x_i) = \frac{\mathcal{D}'_a}{\mathcal{D}'} \leq \frac{2\mathsf{opt}_{x_i}}{\mathsf{CH}(A,B)/2}$. Thus $M \leq 200D$. The expectation is

$$\mathbf{E}[|\{\text{ACCEPTED } x_i\}|] = \sum_{i\in[q]} \frac{\mathcal{P}'(x_i)}{M\mathcal{P}(x_i)} \geq \frac{1}{200D}\sum_{i\in[q]}\frac{\mathcal{D}'_{x_i}}{\mathcal{P}(x_i)}\cdot\frac{1}{\mathcal{D}'} = \frac{1}{200D}\cdot q\mathcal{D}'\cdot\frac{1}{\mathcal{D}'} = \frac{q}{200D}$$

where the second to last equality is due to the definition of $\mathcal{D}' := \sum_{i\in[q]}\frac{\mathcal{D}'_{x_i}}{\mathcal{P}(x_i)}/q$. The final bound holds by Markov's Inequality and our setting of q. □

Lemma 5.7. *Each s_j is independently and identically distributed, and under Condition 5.3, $\mathbf{Pr}\big[s_j = a\big] \geq \frac{\mathsf{opt}_a}{2\mathsf{CH}(A,B)}$ for any $a \in A$.*

Proof. The independence and identicality follows directly from our sampling procedure. For the probability statement, we assume (without loss of generality) that during rejection sampling on Line 4, a sample x_i is accepted and renamed as s_j. Then for any $a \in A$,

$$\begin{aligned}\mathbf{Pr}\big[s_j = a\big] &= \mathbf{Pr}\big[x_i = a \mid x_i \text{ accepted}\big] \\ &= \frac{\mathcal{P}(a)\cdot\mathbf{Pr}\big[x_i \text{ accepted} \mid x_i = a\big]}{\mathbf{Pr}\big[x_i \text{ accepted}\big]} \\ &= \frac{\mathcal{P}(a)\cdot\mathbf{Pr}\big[x_i \text{ accepted} \mid x_i = a\big]}{\sum_{a_0\in A}\mathcal{P}(a_0)\cdot\mathbf{Pr}\big[x_i \text{ accepted} \mid x_i = a_0\big]} \\ &= \frac{\mathcal{P}(a)\cdot\frac{\mathcal{P}'(a)}{M\mathcal{P}(a)}}{\sum_{a_0\in A}\mathcal{P}(a_0)\cdot\frac{\mathcal{P}'(a_0)}{M\mathcal{P}(a_0)}}\end{aligned}$$

In the final equality, because we conditioned on $x_i = a$ (resp. $x_i = a_0$) on the LHS, we know that on the RHS, $\mathcal{P}'(a) = \frac{\mathcal{D}'(a)}{\mathcal{D}'}$ is well-defined and satisfy $\mathsf{opt}_a \leq \mathcal{D}'(a) \leq 2\mathsf{opt}_a$ (resp. $\mathsf{opt}_{a_0} \leq \mathcal{D}'(a_0) \leq 2\mathsf{opt}_{a_0}$), given Condition 5.3. Therefore, we have

$$\mathbf{Pr}\big[s_j = a\big] = \frac{\mathcal{P}'(a)}{\sum_{a_0\in A}\mathcal{P}'(a_0)} = \frac{\mathcal{D}'(a)}{\sum_{a_0\in A}\mathcal{D}'(a_0)} \geq \frac{\mathsf{opt}_a}{2\mathsf{CH}(A,B)}.$$

□

Lemma 5.6 and 5.7 together say that S can be viewed as a set of s samples from a $\frac{1}{2}$-Chamfer Distribution, thus we can invoke another importance sampling analysis. In the final step of the algorithm, we compute the exact nearest neighbor distance for all s_j and then compute a weighted sum over them. With high probability, this gives an $(1 \pm \varepsilon)$-estimation of $\mathsf{CH}(A,B)$.

Theorem 5.8. *Under Condition 5.2 and 5.3,* **Chamfer-Estimate**$(A, B, \varepsilon, q \geq 10^4 D/\varepsilon^2)$ *outputs* $\tilde{\mathsf{CH}}(A, B)$ *that satisfies* $(1-\varepsilon)\mathsf{CH}(A, B) \leq \tilde{\mathsf{CH}}(A, B) \leq (1+\varepsilon)\mathsf{CH}(A, B)$ *with probability at least* $1/2$.

Proof. In the success case of Lemma 5.6, we can apply Lemma 5.1 with $f = 1/2$, $h = 1$, $t = s$, and $\kappa = \varepsilon$. Then

$$\mathbf{Pr}\left[\left|\tilde{\mathsf{CH}}(A, B) - \mathsf{CH}(A, B)\right| \geq \varepsilon \cdot \mathsf{CH}(A, B)\right] \leq \frac{1}{s \cdot \varepsilon^2}.$$

□

Theorem 5.9. Chamfer-Estimate$(A, B, \varepsilon, q = 10^4 D/\varepsilon^2)$ *runs in time* $\mathcal{O}(nd(\log\log n + \log\frac{1}{\varepsilon})/\varepsilon^2))$.

Proof. This is dominated by the runtime of `QuadTree`, `Tournament`, and the time of computing $\mathtt{opt}_{s_j}$ on Line 5. `QuadTree`(A, B) runs in $\mathcal{O}(nd \log\log n)$ time and `Tournament` $(\{x_i\}_{i\in[q]}, B)$ runs in $\mathcal{O}(nd(\log\log n + \log\frac{1}{\varepsilon})/\varepsilon^2)$ time. Finally, the brute-force search for $\mathtt{opt}_{s_j}$ for $j \in [10/\varepsilon^2]$ takes $\mathcal{O}(nd/\varepsilon^2)$ time. □

Acknowledgements

Ying Feng was supported by an MIT Akamai Presidential Fellowship. Piotr Indyk was supported in part by the NSF TRIPODS program (award DMS-2022448).The authors would like to thank Anders Aamand for helpful comments that helped simplify the algorithm in Section 4.

References

[AHNR95] Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman. Sorting in linear time? In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 427–436, New York, NY, USA, 1995. Association for Computing Machinery.

[AM19] Kubilay Atasu and Thomas Mittelholzer. Linear-complexity data-parallel earth mover's distance approximations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 364–373. PMLR, 09–15 Jun 2019.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801, 2015.

[AS03] Vassilis Athitsos and Stan Sclaroff. Estimating 3d hand pose from a cluttered image. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 2, pages II–432. IEEE, 2003.

[BFP$^+$73] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, August 1973.

[BIJ$^+$23] Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, Sandeep Silwal, and Erik Waingarten. A near-linear time algorithm for the chamfer distance, 2023.

[Cha08] Timothy M. Chan. Well-separated pair decomposition in linear time? *Information Processing Letters*, 107(5):138–141, 2008.

[FSG17] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.

[HP11] Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

[Ind06] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the ACM (JACM)*, 53(3):307–323, 2006.

[JSQJ18] Li Jiang, Shaoshuai Shi, Xiaojuan Qi, and Jiaya Jia. Gal: Geometric adversarial loss for single-view 3d-object reconstruction. In *Proceedings of the European conference on computer vision (ECCV)*, pages 802–816, 2018.

[Kle97] Jon M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 599–608, New York, NY, USA, 1997. Association for Computing Machinery.

[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. PMLR, 2015.

[KZ20] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[pda23] Pdal: Chamfer. https://pdal.io/en/2.4.3/apps/chamfer.html, 2023. Accessed: 2023-05-12.

[pyt23] Pytorch3d: Loss functions. https://pytorch3d.readthedocs.io/en/latest/modules/loss.html, 2023. Accessed: 2023-05-12.

[SMFW04] Erik B Sudderth, Michael I Mandel, William T Freeman, and Alan S Willsky. Visual hand tracking using nonparametric belief propagation. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pages 189–189. IEEE, 2004.

[ten23] Tensorflow graphics: Chamfer distance. https://www.tensorflow.org/graphics/api_docs/python/tfg/nn/loss/chamfer_distance/evaluate, 2023. Accessed: 2023-05-12.

[WCL$^+$19] Ziyu Wan, Dongdong Chen, Yan Li, Xingguang Yan, Junge Zhang, Yizhou Yu, and Jing Liao. Transductive zero-shot learning with visual structure constraint. *Advances in neural information processing systems*, 32, 2019.

A Reducing the Bit Precision of Inputs.

In our algorithm, we assumed that all points in input sets A, B are integers in $\{1, 2, \cdots, \mathrm{poly}(n)\}^d$. Here, we show that this is without loss of generality, as long as all coordinates of the original input are w-bit integers for arbitrary $w \geq \log n$ in a unit-cost RAM with a word length of w bits.

Section $A.3$ of [BIJ+23] gives an efficient reduction from real inputs to the case that

$$1 \leq \min_{a \in A, b \in B} \|a - b\|_1 \leq \max_{a \in A, b \in B} \|a - b\|_1 \leq \mathrm{poly}(n),$$

i.e., the input has a $\mathrm{poly}(n)$-bounded aspect ratio. Their reduction can be adapted to our case as follows:

Claim A.1 (Lemma $A.3$ of [BIJ+23])**.** *Given an* est *such that* $\mathsf{CH}(A, B) \leq \mathsf{est} \leq \mathrm{poly}(n) \cdot \mathsf{CH}(A, B)$, *if there exists an algorithm that computes an* $(1 + \varepsilon)$*-approximation to* $\mathsf{CH}(A, B)$ *in* $\mathcal{O}(nd(\log \log n + \log \frac{1}{\varepsilon})/\varepsilon^2))$ *time under the assumption that* A, B *contain points from* $\{1 \ldots \mathrm{poly}(n)\}^d$*, then there exists an algorithm that computes an* $(1 + \varepsilon)$*-approximation to* $\mathsf{CH}(A, B)$ *for any integer-coordinate* A, B *in asymptotically same time.*

It remains to show how to obtain a $\mathrm{poly}(n)$-approximation.

Lemma A.2. *There exists an* $\mathcal{O}(nd + n \log \log n)$*-time algorithm that computes* est *which satisfies* $\mathsf{CH}(A, B) \leq \mathsf{est} \leq \mathrm{poly}(n) \cdot \mathsf{CH}(A, B)$ *with* $1 - \frac{1}{n}$ *probability.*

Proof. Similar to (the proof of Lemma $A.3$ in) [BIJ+23], we sample a vector $v \sim \mathsf{Cauchy}(0, 1)$, which can be discretized to $\mathcal{O}(\log n)$-bit precision following [Ind06]. We then compute the inner products $\{v \cdot a\}_{a \in A}$ and $\{v \cdot b\}_{b \in B}$. The distribution of $v \cdot a - v \cdot b$ follows $\mathsf{Cauchy}(0, \|a - b\|_1)$ by the 1-stability property of Cauchy's. So we have that for every $a \in A$ and $b \in B$,

$$\frac{\|a - b\|_1}{\mathrm{poly}(n)} \leq |v \cdot a - v \cdot b| \leq \|a - b\|_1 \cdot \mathrm{poly}(n),$$

with probability $1 - 1/\mathrm{poly}(n)$. Therefore, $\mathsf{est} := \mathsf{CH}(\{v \cdot a\}_{a \in A}, \{v \cdot b\}_{b \in B})$ is a $\mathrm{poly}(n)$-approximation to $\mathsf{CH}(A, B)$. We may assume by scaling that $\{v \cdot a\}_{a \in A}, \{v \cdot b\}_{b \in B}$ contain w-bit integers, which can be sorted in $\mathcal{O}(n \log \log n)$ time [AHNR95]. Then to compute est, we find all one-dimensional nearest neighbors by going through the sorted list and link each $a' \in \{v \cdot a\}_{a \in A}$ with adjacent $b' \in \{v \cdot b\}_{b \in B}$, which takes $\mathcal{O}(n)$ time. Thus the total runtime is $\mathcal{O}(nd + n \log \log n)$ as claimed. □

B Proof of Lemma 4.1

Proof. We use the fact that for $v \sim (\mathsf{Cauchy}(0, 1))^d$ and any $x \in \mathbb{R}^d$, $(v \cdot x) \sim \mathsf{Cauchy}(0, \|x\|_1)$. Also, for any $k > 0$, if a random variable $z \sim \mathsf{Cauchy}(0, 1)$ then $kz \sim \mathsf{Cauchy}(0, k)$. Therefore, for any $v_i : i \in [r]$, $\mathbf{Pr}\big[|v_i \cdot (x - y)| > (1 + c)\|x - y\|_1\big] = \mathbf{Pr}\big[U > 1 + c\big]$ where $U \sim \mathsf{HalfCauchy}(0, 1)$. The density of U is $f_U(u) = \frac{2}{\pi} \cdot \frac{1}{1+u^2}$, thus $\mathbf{Pr}\big[U > 1\big] = 1/2$ and

$$\begin{aligned}
\mathbf{Pr}\big[U > 1 + c\big] &= \frac{1}{2} - \int_1^{1+c} f_U(u) du \\
&\leq \frac{1}{2} - c \cdot f_U(3/2) && \text{for } 0 < c < 1/2 \\
&< \frac{1}{2} - c/10
\end{aligned}$$

Similarly, we can get $\mathbf{Pr}\big[|v_i \cdot (x-y)| < (1-c)\|x-y\|_1\big] < \frac{1}{2} - c/10$. For $i \in [r]$, let $\mathcal{I}_i$ be an indicator variable that equals 1 if $|v_i \cdot (x-y)| < (1-c)\|x-y\|_1$ and equals 0 otherwise. By Hoeffding's bound,

$$\mathbf{Pr}\big[\sum_{i\in[r]} \mathcal{I}_i \geq \frac{r}{2}\big] < e^{-2rc^2/100},$$

which upper bounds the failure probability that the median is too small. We symmetrically bound the probability that the median is too large. Then

$$\mathbf{Pr}\big[\mathsf{med}\{|v_i \cdot (x-y)| : i \in [r]\} \in (1 \pm c)\|x-y\|_1\big] \geq 1 - 2e^{-rc^2/50}.$$

$\square$