Faster and Better Solution to Embed L_p Metrics by Tree Metrics

Yuxiang Zeng
The Hong Kong University of Science and Technology
Hong Kong SAR, China
yzengal@cse.ust.hk

Yongxin Tong
State Key Laboratory of Software Development Environment,
Beihang University, Beijing, China
yxtong@buaa.edu.cn

Lei Chen
The Hong Kong University of Science and Technology
Hong Kong SAR, China
leichen@cse.ust.hk

ABSTRACT

Hierarchically Separated Tree (HST) is the most popular solution to embed a metric space into a tree metric. By using HSTs, many optimization problems, which are hard on defined metrics, become easier to get good approximation bounds with respect to the effectiveness, *e.g.*, task assignment, trip planning, and facility location planning. Existing work focuses on constructing HSTs for arbitrary metric spaces, which makes a general-purpose algorithm take at least $O(n^2)$-time to get tight distortion guarantees $O(\log n)$. Here, distortion is a prevalent measurement of HSTs' effectiveness and usability. However, we observe that (1) in many applications that HSTs are applied, only L_p metrics are used (*e.g.*, Euclidean space), (2) the state-of-the-art solution is still time-consuming to construct HSTs for large-scale data, and (3) distortions of existing algorithms are only satisfactory for high-dimensional data. Thus, in this paper, we are motivated to study the Embedding L_p metrics through Tree metrics (ELT) problem. We aim to design a faster algorithm than $O(n^2)$ time to construct HSTs with not only $O(\log n)$ distortion guarantees but also good and robust empirical results. Specifically, we first present a divide-and-conquer based general framework and prove that it has a distortion guarantee of $O(\log n)$. To achieve a better time complexity than $O(n^2)$, we next design two optimization techniques: reducing to nearest neighbor search (by indexing) and sampling. Finally, extensive experiments demonstrate that our algorithm DCsam outperforms the state-of-the-art algorithms by a large margin in terms of both distortion and running time.

CCS CONCEPTS

• **Theory of computation** → **Data structures and algorithms for data management**; **Computational geometry**.

KEYWORDS

metric embedding, hierarchically separated tree

ACM Reference Format:
Yuxiang Zeng, Yongxin Tong, and Lei Chen. 2022. Faster and Better Solution to Embed L_p Metrics by Tree Metrics. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3514221.3517831

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA.

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517831

1 INTRODUCTION

Hierarchically Separated Tree (HST) [14] is a tree-based data structure (index), which is the most popular solution to embed arbitrary metric spaces ("original metrics" for short) into tree-based metric spaces. By using HSTs, many optimization problems (*e.g.*, NP-hard or online problems) become easier to get approximate results with good bounds on the performance ratio. A few selections of these problems include task assignment [18, 55, 57], trip planning [19, 63], privacy protection [24, 54], and facility location planning [12, 20].

These studies have three common things in their algorithms: (1) they reduce the problems on their defined metric spaces to instances on HSTs, (2) instances on HSTs can be tackled by good approximation guarantees (say ρ), and (3) problems on the original metrics will usually have approximation bounds of $\rho \times$ distortion. Here, the *distortion* is the maximum elongation of the distance for any two data points on the HST over their original distances. Since lower distortions imply better effectiveness (*i.e.*, smaller approximation bounds), early studies [14, 15, 25, 26, 39, 41] of HSTs focus on minimizing the distortion. This series of work culminated in the breakthrough of [25, 26] by constructing HSTs with tight distortion guarantees ($O(\log n)$). Recent work [17, 29, 32, 64] improved the time efficiency of the seminal construction method in [25, 26] from $O(n^3)$ to $O(n^2)$ in the worst case.

Up to now, the algorithm in [25, 26] takes at least $O(n^2)$ to construct an HST for n data points whose distance function needs $O(1)$ time (*e.g.*, constant dimensions). Similar to other $O(n^2)$-time algorithms (*e.g.*, [31, 61]), *it is time-consuming to construct HSTs for large-scale spatial data on 2D Euclidean spaces.* For example, our experiments show that a regular server (with Intel(R) Xeon(R) 2.40GHz CPU) takes several hours to construct HSTs for one million 2D points. Moreover, we observe another empirical phenomenon: *distortions of existing HSTs are much higher on low-dimensional spaces than high-dimensional spaces.* High distortions may cause low effectiveness of optimization solutions by HSTs, as many applications consider low-dimensional spatial data. For example, 2D Euclidean space and 2D Manhattan space are often considered in the aforementioned applications of HSTs.

Motivated by these observations, this paper focuses on *constructing HSTs more efficiently to get both tight theoretical guarantees and low empirical distortions.* Specifically, we study the Embedding L_p metrics through Tree metrics (ELT) problem. We focus this problem scope, since L_p metrics cover many popular metrics such as Manhattan distance (L_1), Euclidean distance (L_2) and Chebyshev distance (L_∞). These metrics not only "play the most prominent roles" in low-distortion embeddings (according to §8 of the textbook [58]), but also are the primary concerns in applications of HSTs (*e.g.*, the selected problems above).

To solve the ELT problem, we design a divide-and-conquer based framework. The *main idea* is to first find a good division scheme to partition a metric space into several subspaces, then construct the subtree of each subspace, and finally merge the subtrees into the final HST. Although theoretical analysis implies that this framework has a distortion guarantee no worse than the state-of-the-art method [26], it suffers from an even higher time complexity ($O(n^4 \log n)$). Thus, we design two optimization techniques: *reducing distortion computation to nearest neighbor (NN) search by indexing* and *two-phase sampling*. The former brings us the chance to reduce high time complexity by leveraging the rich literature of (approximate) NN, which is not limited to L_p metrics. The latter achieves the trade-off between distortions and running time.

Our main contributions are summarized as follows.

- We propose a divide-and-conquer based framework. Theoretical analysis implies that its distortion guarantee is no lower than that of the state-of-the-art [26], *i.e.*, $O(\log n)$.
- We design two optimization techniques to improve the time complexity while retaining $O(\log n)$ distortion guarantee. By using these optimizations, the time complexity of our algorithm DCsam is $O(n^{1.5} \log^2 n)$, which is asymptotically faster than the state-of-the-art solution ($O(n^2)$ [26]).
- Extensive experiments demonstrate that our solution outperforms the existing baselines [14, 15, 26] in both effectiveness (*i.e.*, distortion) and time efficiency. For example, distortions of DCsam are lower than these baselines by up to 16×-41× in our experiments. Moreover, DCsam is up to 783× faster than [26] and up to 4-5 orders of magnitude faster than [14, 15].

In the rest of this paper, Sec. 2 introduces the ELT problem and baselines. Then, we present our solution in Sec. 3-4, experiments in Sec. 5, and related work in Sec. 6. Finally, we conclude in Sec. 7.

2 PROBLEM STATEMENT AND BASELINE

This section introduces the problem definition and our baselines.

2.1 Problem Definition

Definition 1 (L_p Metric)**.** An L_p metric space ("metric" for short) is denoted by $S = (V, Dis)$. The set V contains n points in real d-dimensional space $\mathbb{R}^d$. $Dis = (\sum_{i=1}^{d} |x[i] - y[i]|^p)^{1/p}$ is the L_p-norm function. As a metric space, S needs to satisfy three conditions for all $x, y, z \in V$: (1) $Dis(x, x) = 0$, (2) $Dis(x, y) = Dis(y, x)$, and (3) $Dis(x, y) + Dis(y, z) \geq Dis(x, z)$ (triangle inequality).

We focus on L_p metrics, where $p \in [1, \infty]$ and d is fixed. Following the conventions in existing work [17, 25, 26, 64], the distance $Dis(x, y)$ when $x \neq y$ is defined in $[1, \infty)$ (*e.g.*, by normalization). The diameter of this metric is denoted by $\Delta = \max Dis(x, y)$. We assume Δ is bounded by $O(poly(n))$ (*e.g.*, n^d or n's power of any large constant). This assumption generally holds in real-world datasets.

Definition 2 (Tree Metric [58])**.** A tree metric $S_T = (V_T, Dis_T)$ is a tree-structured metric, where distance $Dis_T(u, v)$ between nodes u and v is the sum of edge weights along the path from u to v.

Fig. 1 depicts an example of tree metrics, where points p_1-p_6 in Table 1 are represented by the leaves u_{13}-u_{18}. On this tree, the distance between p_2 and p_5 (corresponding to leaves u_{13} and u_{15}) consists of two parts, *i.e.*, the distance of path from u_{13} to their lowest common ancestor (LCA) and the length of path from their LCA (denoted by $\mathrm{lca}(u_{13}, u_{15}) = u_4$) to u_{14}. Thus, we have $Dis_T(p_2, p_5) = W(u_{13}, u_7) + W(u_7, u_4) + W(u_4, u_9) + W(u_9, u_{15}) = 2+4+4+2 = 12$, where $W(\cdot, \cdot)$ is the edge weight (marked in the left side of Fig. 1).

Definition 3 (Embedding, Stretch and Distortion [58])**.** Given two metrics $S = (V, Dis)$ and $S_T = (V_T, Dis_T)$, a mapping $f : V \to V_T$ is an **embedding** if $Dis(x, y) \leq Dis_T(f(x), f(y))$ for all $x, y \in V$. The **stretch** of the distance between x and y is defined as

$$\mathrm{Stretch}(x, y) := \frac{Dis_T(f(x), f(y))}{Dis(x, y)}. \tag{1}$$

The **distortion** is the maximum of the pairwise stretches, *i.e.*,

$$\mathrm{Distort}(V) := \max_{x \in V, y \in V} \frac{Dis_T(f(x), f(y))}{Dis(x, y)} \tag{2}$$

Fig. 1 also depicts an embedding of the points p_1-p_6 in a 2D Euclidean space (*i.e.*, S is a L_2 metric). As the distance $Dis(p_2, p_5)$ is $\sqrt{(3-0)^2 + (4-5)^2} = 3.16$, the *stretch* between these two points is $Dis_T(p_2, p_5)/Dis(p_2, p_5) = 12/3.16 = 3.80$. After enumerating all pairwise stretches, we can derive that the *distortion* is $\mathrm{Distort}(V) = \mathrm{Stretch}(p_4, p_1) = 60/\sqrt{(5-10)^2 + (8-9)^2} = 11.77$. Next, we present the *Embedding L_p metrics by Tree metrics* (ELT) problem as follows.

Definition 4 (ELT Problem [14])**.** Given an L_p metric space $S = (V, Dis)$, we aim to embed S into a tree metric $S_T = (V_T, Dis_T)$ such that the **distortion** of this embedding can be minimized.

Key Idea of ELT Problem. *First*, the ELT problem embeds (maps) the metric space S into a simple and well-structured tree metric S_T (*i.e.*, HST introduced in Sec. 2.2). Unfortunately, the pairwise distances in S may not remain the same in S_T after embedding and some of them are stretched. Thus, ELT aims to minimize the maximum among all stretches (*i.e.*, the distortion in Def. 3).

Second, the ELT problem is useful, since it has been shown that many optimization problems on HSTs are easier to be solved than on other metric spaces (*e.g.*, a Euclidean space) [14, 17, 26, 39], *e.g.*, trip planning [19, 63] and facility location planning [12, 20].

Third, there are generally three steps to use the ELT problem: (1) constructing the tree metric S_T, (2) reducing the problem instances (*i.e.*, any point $x \in V$ is mapped into $f(x) \in V_T$ and Dis is changed into Dis_T) and (3) designing approaches for instances on $S_T = (V_T, Dis_T)$. Our work focuses on the first step since the state-of-the-art construction algorithm is slow in large-scale datasets and also has a high distortion. Due to the page limitation, please refer to the textbooks [60] (§8) and [35] (§26) for a more detailed introduction.

Finally, in Sec. 5.3, we provide a case study on using ELT in a real-world problem: online metric bipartite matching (OMBM). We also show (1) the effectiveness of OMBM is improved by using ELT and (2) a lower distortion leads to a greater improvement.

Motivation of Our Problem Scope. Our ELT problem is based on the proposed problem in [14], where the *original metric S* is an arbitrary metric space. We restrict the problem scope of S due to two reasons: (1) for L_p metric, "the cases $p = 1, 2, \infty$ play the most prominent roles" in low-distortion embeddings according to the textbook [58] (§8), and (2) these cases L_1, L_2, L_∞ are widely used as the distance metric in aforementioned applications. We also discuss the support to other metric spaces in Sec. 4.4.

Table 1: Coordinates of points p_1-p_6 in a Euclidean space

Point	p_1	p_2	p_3	p_4	p_5	p_6
Coordinate	(10,9)	(3,4)	(2,6)	(5,8)	(0,5)	(1,9)

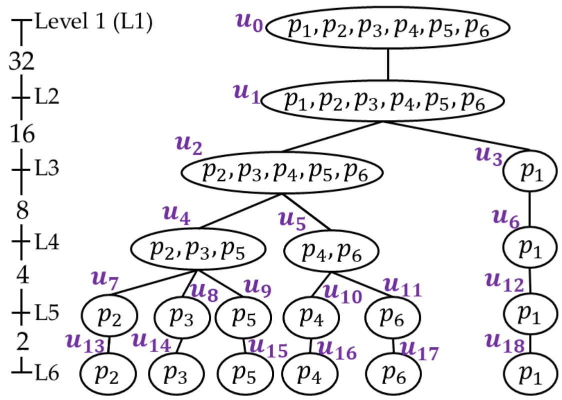


Figure 1: A 2-HST of the toy example ($k = 2$)

Theoretical Analysis Model. We follow the standard analysis model in existing work [14, 15, 25, 26, 35, 39] to analyze the theoretical guarantee of distortions (*distortion guarantee* for short). Def. 5 indicates that the distortion guarantee is the *maximum of all expected stretches*, *i.e.*, $\mathbb{E}[Dis_T(f(x), f(y))]/Dis(x, y)$.

Definition 5 (Distortion Guarantee). A tree metric S_T probabilistically approximates the original metric S if a probability distribution over S_T exists such that $\mathbb{E}[Dis_T(f(x), f(y))] \leq \rho \times Dis(x, y)$ for all $x, y \in V$, where $\mathbb{E}[\cdot]$ is the expected distance under this distribution and ρ is the distortion guarantee.

2.2 Hierarchically Separated Tree and Baseline

The *Hierarchically Separated Tree* (HST) [14] is the most popular solution due to its tight distortion guarantee ($O(\log n)$ [25]).

Definition 6 (HST). An HST is a rooted tree with three properties:
(1) All edges between adjacent levels have the same weight;
(2) From top to bottom, the edge weights are geometrically decreased by a user-defined parameter k;
(3) Each point in S is mapped into a unique leaf on the HST.

HSTs are also called k-HSTs, where k is usually 2. Fig. 1 illustrates a 2-HST. These properties are obviously satisfied based on the edge weights labelled on the left, *e.g.*, $Dis_T(u_1, u_2) = Dis_T(u_1, u_3) = 16$ (the first one) and $Dis_T(u_1, u_2) = Dis_T(u_2, u_4)/k$ (the second one). Besides, each point p_1-p_6 is mapped into a unique leaf u_{12}-u_{17}.

Baseline Selection. We select the most popular algorithms [14, 15, 26] as our baselines in experiments. FRT [26] is the state-of-the-art with $O(\log n)$ distortion guarantee and $O(n^2)$ time complexity.

Algo. 1 illustrates this baseline [26]. The *basic idea* is to construct the nodes of HST via ball partitions whose centers are iterated from a randomized permutation π and radii are computed by β and k (lines 1-2). Line 3 creates the root and calculates the tree height H. Lines 4-10 create the other nodes from top to bottom, where U_{i-1} denotes the internal nodes at the level $i - 1$. At each level i, some points are separated from the internal node u by a ball partition with the center $\pi[j]$ and the radius r_i (line 8). In line 9, we create a child node u' of u to represent these separated points. The iterations will stop until each (leaf) node represents a singleton point.

Algorithm 1: Construct HSTs by FRT [26]

1 Pick β in $[\frac{1}{k}, 1]$ randomly from a distribution $p(\beta) = \frac{1}{\beta k \ln k}$;
2 Pick a random permutation π of the n points in V;
3 Root $\leftarrow V$, height $H \leftarrow \lceil \log_k \Delta \rceil + 1$, $U_1 \leftarrow \{root\}$, $i \leftarrow 2$;
4 **while** *U_{i-1} has some node with more than one point* **do**
5   Radius $r_i \leftarrow \beta \times k^{H+1-i}$, edge weight $w_i \leftarrow r_i \times k$;
6   **for** $j \leftarrow 1$ **to** n **do**
7   **foreach** *node $u \in U_{i-1}$* **do**
8             $u' \leftarrow$ a new node consisting of all unassigned points in u closer than r_i to $\pi[j]$;
9   u' is a child of u with weight w_i, add u' into U_i;
10     $i \leftarrow i + 1$;

Table 2: Summary of the major notations in this paper

Notation	Description
S, S_T	the original metric S and the tree metric S_T
V, n	a set of n points in the original metric S (*i.e.*, $\lvert V \rvert = n$)
Dis, Dis_T	the distance functions on S and S_T respectively
Δ	the diameter of S, *i.e.*, $\Delta = \max_{x,y \in V} Dis(x, y)$
k	the parameter of the HSTs (*i.e.*, k-HSTs), where $k \geq 2$
H	the height of the constructed HSTs, $H = \lceil \log_k \Delta \rceil + 1$
$\mathrm{B}(x, r)$	a circular range centered at point x with a radius r
$\mathrm{Stretch}(x, y)$	the stretch between points x and y
$\mathrm{Distort}(V)$	the distortion of the tree metric $S_T = (V_T, Dis_T)$
$\mathrm{Stretch}(X, Y)$	the maximum stretch between point sets X and Y

Example 1. Algo. 1 constructs a 2-HST for the points in Table 1 with $\beta = 1$ and $\pi = \{p_2, p_3, p_5, p_4, p_6, p_1\}$. Since the diameter Δ is $\max_{i,j} Dis(p_i, p_j) = 10.77$, the tree height H is $\lceil \log_2 \Delta \rceil + 1 = 5$. When level $i = 2$, the radius r_2 is $1 \times 2^{5+1-2} = 16$ and the weight w_2 is $r_2 \times 2 = 32$ in line 5. In line 8, a new node u_1 is created to represent the points p_1-p_6, since all these points are closer than $r_2 = 16$ to $\pi[1] = p_2$. In line 9, we add this node u_1 as a child of the root u_0 with the edge weight $w_2 = 32$ as shown in Fig. 1. When $i = 3$, the radius r_3 is 8 and the weight w_3 is 16. When $j = 1$ at the third level, a node u_2 is created to represent p_2-p_6, since their distances to $\pi[j] = p_2$ are closer than $r_3 = 8$. Then, u_2 is appended as a child of the node u_1 with the edge weight $w_3 = 16$ in Fig. 1. When j is 4, the distance of the only unassigned point p_1 to $\pi[j] = p_4$ is closer than r_3. Hence, we create a child node u_3 of the internal node u_1 in lines 8-9. Similarly, nodes u_4-u_6 are created at the fourth level, nodes u_7-u_{12} are created at the fifth level, and nodes u_{13}-u_{18} are created at the leaf level (*i.e.*, the sixth level).

Remark. The distance function on an HST involves the computation of the lowest common ancestor, which can be answered by a range minimum query in $O(1)$ time by using $O(n)$ auxiliary space. Please refer to [28, 30, 36] for the implementation details.

Table 2 lists the major notations used in the rest of this paper.

3 OUR GENERAL FRAMEWORK

In this section, we present a divide-and-conquer based framework including the main idea (Sec. 3.1) and algorithm details (Sec. 3.2).

3.1 Main Idea

3.1.1 Understanding High Distortion of Baseline. Although the baseline FRT (*i.e.*, Algo. 1) has a tight distortion guarantee ($O(\log n)$), we observe that distortions of FRT can be high in real datasets and synthetic datasets (see Sec. 5). Sometimes, the distortions of FRT are higher than another baseline (algorithm Bar96 [14]) which has an even worse distortion guarantee ($O(\log n \log \min\{n, \Delta\})$).

Our understanding of this experimental result is as follows:

(1) Algo. 1 [26] and other different implementations [17, 25, 29, 32, 64] of FRT [26] are all randomized algorithms.

(2) In general, the effectiveness of a randomized algorithm highly depends on the choices of random variables [45]. For example, distortions of any implementation of FRT depend on two random variables β and π. Although β is a continuous variable, at most n^2 values of β matter according to [26]. By contrast, π, a random permutation of n data points, has $n!$ possible choices, where $n!$ can be extremely large for even small n (*e.g.*, 13! is over 6 billion).

(3) Such numerous choices of π can lead to low robustness. Thus, the HST constructed by FRT may have a high distortion unless sufficient choices of π have been tried. However, trying different π will inevitably exacerbate the bottleneck of time efficiency.

3.1.2 Main Idea of Our Framework. By contrast with the baseline FRT [26], our divide-and-conquer based framework relies on β only.

Rationale of Divide-and-Conquer. As shown in Fig. 1, an internal node (*e.g.*, u_2) of an HST corresponds to a subset (*e.g.*, p_2-p_6) of the point set V and the subtree rooted at this internal node can be viewed as an HST of a subspace (*e.g.*, $(\{p_2, \cdots, p_6\}, Dis)$). Moreover, the subsets of the nodes at the same level are disjoint. Based on such HSTs, our divide-and-conquer based strategy is as follows:

- **Divide.** We divide the original metric into disjoint subspaces.
- **Conquer.** We construct a subtree (HST) for each subspace.
- **Combine.** These subtrees are merged into the final HST.

Intuitively, different divisions of the original metric may lead to different effectiveness (distortions). Thus, a major *challenge* here is how to find a good division based on the distortion. To address this challenge, we first propose a *new equivalent expression of distortion* and then elaborate on how to use it in *our division scheme*. Note that the primary expression of distortion enumerates all pairwise stretches in Def. 3, which relies on the final HST after the construction procedure. In contrast, the new expression allows us to compute some intermediate results (*e.g.*, some of the pairwise stretches) to guide the division without waiting for the final HST.

New Equivalent Expression of Distortion. We assume the original metric $S = (V, Dis)$ has been divided into m disjoint subspaces $S_1 = (X_1, Dis), \cdots, S_m = (X_m, Dis)$, *i.e.*, $\bigcup_i X_i = V$ and $X_i \cap X_j = \emptyset$ for any $i \neq j$. We define a function $\mathrm{Stretch}(X_i, X_j)$ to compute the maximum stretch between any point in X_i and any point in X_j, *i.e.*,

$$\mathrm{Stretch}(X_i, X_j) \coloneqq \max_{x_i \in X_i, x_j \in X_j} \frac{Dis_T(f(x_i), f(x_j))}{Dis(x_i, x_j)}. \tag{3}$$

For brevity, $\mathrm{Stretch}(X_i, X_j)$ is also called the "maximum stretch between subspaces S_i and S_j". By using $\mathrm{Stretch}(X_i, X_j)$, we can rewrite the definition of the distortion in Eq. (2) as:

$$\mathrm{Distort}(V) = \max\left\{\max_i\{\mathrm{Distort}(X_i)\}, \max_{i<j}\{\mathrm{Stretch}(X_i, X_j)\}\right\}.$$

This is correct since it enumerates all pairwise stretches. For example, $\mathrm{Distort}(X_i)$ enumerates all pairwise stretches over point set X_i and $\max_{i<j}\{\mathrm{Stretch}(X_i, X_j)\}$ enumerates the pairwise stretches between any point in X_i and any point outside X_i. Let a point set Y_i to denote $\bigcup_{j=i+1}^{m} X_j$. A new equivalent expression of distortion is

$$\mathrm{Distort}(V) = \max_i\{\mathrm{Distort}(X_i), \mathrm{Stretch}(X_i, Y_i)\}. \tag{4}$$

This is because

$$\begin{aligned}
\max_{i<j}\{\mathrm{Stretch}(X_i, X_j)\} &= \max_{i<j} \max_{x \in X_i, y \in X_j} \frac{Dis_T(f(x), f(y))}{Dis(x, y)} \\
&= \max_{x \in X_i, y \in X_{i+1} \cup \cdots \cup X_m} \frac{Dis_T(f(x), f(y))}{Dis(x, y)} \\
&= \max_{x \in X_i, y \in Y_i} \frac{Dis_T(f(x), f(y))}{Dis(x, y)} = \max_i\{\mathrm{Stretch}(X_i, Y_i)\}
\end{aligned}$$

Algorithm 2: Divide-and-conquer based framework DC

input : a metric space $S = (V, Dis)$, current level l, and β
output: a k-HST with height H of this metric space S

Root $u_l \leftarrow$ represents V, $cp^* \leftarrow$ null, $\mathrm{stret}^* \leftarrow \infty$;
foreach *center* $cp \in V$ **do** // Find a good division
  $\mathrm{stret} \leftarrow 1$, $Y_l \leftarrow V$, $X_{H+2} \leftarrow \{cp\}$;
  for *level* $i \leftarrow l+1$ **to** $H+1$ **do**
    $r_i \leftarrow \beta \times k^{H+1-i}$, $Y_i \leftarrow$ points in Y_{i-1} within the circular range $\mathrm{B}(cp, r_i)$, $X_i \leftarrow Y_{i-1} \setminus Y_i$;
    $\mathrm{stret} \leftarrow \max\{\mathrm{stret}, \mathrm{Stretch}(X_i, Y_i)\}$;
  if *stret* $<$ *stret** **then** $\mathrm{stret}^*, cp^* \leftarrow \mathrm{stret}, cp$;
for $i \leftarrow l+1$ **to** $H+1$ **do** // Divide, Conquer, Combine
  $Y_i \leftarrow$ the points in Y_{i-1} that are located in the circular range $\mathrm{B}(cp^*, r_i)$, $X_i \leftarrow Y_{i-1} \setminus Y_i$;
  A node $u_i \leftarrow$ represents the point set Y_i and add u_i to child nodes of u_{i-1} with edge weight $r_i \times k$;
  $T_i \leftarrow$ construct the HST of subspace $S_i = (X_i, Dis)$ by this algorithm with level $(i-1)$ and same β;
  Link the child nodes of T_i's root as the child nodes of u_{i-1} with the edge weight $r_i \times k$;

Observations from New Expression. The distortion defined in Eq. (4) is determined by two factors: (1) the distortion of the subtree for each subspace (*i.e.*, $\mathrm{Distort}(X_i)$) and (2) the maximum stretch between each pair of subspaces (*i.e.*, $\mathrm{Stretch}(X_i, Y_i)$). Since the first factor $\mathrm{Distort}(X_i)$ is recursively defined by Eq. (4) (*i.e.*, subspace (X_i, Dis) is recursively divided into subspaces), we can mainly focus on the second factor. In other words, a good division scheme should minimize $\max_i \mathrm{Stretch}(X_i, Y_i)$ as much as possible.

Our Division Scheme. We can enumerate each point in V as the center cp of a circular range that is used to divide the original metric S. This circular range is denoted by $\mathrm{B}(cp, r)$, where cp is the center and r is the radius. We maintain the best center cp^* in V whose maximum stretch between the separated subspaces (*i.e.*, $\max_i \mathrm{Stretch}(X_i, Y_i)$ by this center) is the lowest.

3.2 General Framework

3.2.1 Preliminary. We do not intend to change the structure of HSTs, so we use the same height and edge weight as in FRT [26].

(1) The height H is $\lceil \log_k \Delta \rceil + 1$, where Δ is the diameter of S.

(2) For any level $i = 1, \cdots, H$, the edge weight between the ith level and the $(i+1)$th level is $\beta \times k^{H+1-i}$, where β is randomly sampled in $[1/k, 1]$ by the same distribution in FRT [26].

3.2.2 Algorithm Details. Algo. 2 illustrates the details of our framework DC. Lines 2-7 find the best center cp^* whose maximum stretch (stret*) between the separated subspaces is the lowest. Specifically, for each center cp in V, we use stret to denote the maximum stretch between the separated subspaces by cp, *i.e.*, stret = $\max_i \text{Stretch}(X_i, Y_i)$ in line 6. In line 5, we separate the points in Y_{i-1} by a circular range $\text{B}(cp, r_i)$ centered at cp with a radius r_i. At each level i, the point set Y_i contains the points in Y_{i-1} inside this circular range, while X_i contain the remaining points in Y_{i-1} which are outside this circular range. At the leaf level, Y_{H+1} contains a single point (*i.e.*, cp) since the radius $r_{H+1} = \beta \le 1$. Thus, we use X_{H+2} to denote $\{cp\}$ in line 3 so that $\bigcup_{i=l+1}^{H+2} X_i = V$. We maintain stret* and cp^* in line 7. If more than one center can result in stret*, we randomly sample one as cp^*. After finding a good division scheme, we *divide* the metric space S into several subspaces by cp^* in lines 8-9. At each level i, we create a node u_i to represent the point set Y_i and add u_i to the child nodes of the internal node u_{i-1} (line 10). For each disjoint subspace S_i, we *recursively* construct its subtree in line 11. Then, we merge (*combine*) the subtrees into the final HST by linking the children of the subtree's root as the children of u_{i-1}.

Example 2. Back to our toy example. Initially, S contains all six points p_1-p_6, l is 1 and β is 1. The tree height H is $\lceil \log_2 \Delta \rceil + 1 = 5$. As shown in Fig. 1, we create the root u_0 in line 1 to represent p_1-p_6. In line 2, we iteratively select p_1-p_6 as the center cp. In lines 3-6, we calculate the maximum stretch (*i.e.*, stret) between the separated subspaces by cp. For example, when $cp = p_2$ and the level $i = 2$, the radius r_2 is $2^4 = 16$. In line 5, the set Y_2 is p_1-p_6 and X_2 is $\emptyset$, since all the points are in a circular range centered at p_2 with a radius of 16 (*i.e.*, $\text{B}(p_2, r_2)$). So, stret is still 1. When $i = 3$ and the radius $r_3 = 8$, we have Y_3 is p_2-p_6 and $X_3 = \{p_1\}$. We calculate $\text{Stretch}(X_3, Y_3)$ by Eq. (3) in line 6. Eventually, we have $\text{Stretch}(X_3, Y_3) = Dis_T(u_{18}, u_{16})/Dis(p_1, p_4) = 60/5.10 = 11.77$, where u_{18} and u_{16} are leaves in Fig. 1. Hence, stret becomes 11.77. Similarly, we derive that (1) $Y_4 = \{p_2, p_3, p_5\}$, $X_4 = \{p_4, p_6\}$, $\text{Stretch}(X_4, Y_4) = \text{Stretch}(p_3, p_6) = 28/\sqrt{10} = 8.85$ and (2) $Y_5 = \{p_2\}$, $X_5 = \{p_3, p_5\}$, $\text{Stretch}(X_5, Y_5) = \text{Stretch}(p_2, p_3) = 12/\sqrt{5} = 5.37$. Finally, the maximum stretch between the separated subspaces by cp is stret = $\max\{11.77, 8.85, 5.37\} = 11.77$. After enumerating all the center points, we have $cp^* = p_2$ and stret$^* = 11.77$.

Lines 8-12 construct the final HST by recursions. Fig. 2 illustrates the detailed procedure, where nodes created for sets Y_i are marked by blue and subtrees created for subspaces $S_i = (X_i, Dis)$ are marked by red. Specifically, when level $i = 2$, we create u_1 in Fig. 2a to represent $Y_2 = \{p_1, \cdots, p_6\}$ and no recursion is executed since X_2 is $\emptyset$. At the 3rd level, we create u_2 to represent $Y_3 = \{p_2, \cdots, p_6\}$ and construct the subtree for the subspace $S_3 = (X_3, Dis)$ in Fig. 2b. Since there is only one point left, the subtree contains the nodes $u', u_3, u_6, u_{12}, u_{18}$ at levels 2-6, where u' is this subtree's root (*i.e.*, a fake node that will be removed later) and the other nodes are shown in Fig. 2b. In line 12, we link the node u_3 as a child node of u_1 with the edge weight $w_3 = r_3 \times 2 = 16$ and remove u'. Similarly, when the level i increases to 4, the node u_4 is created to represent the point set Y_4 and a subtree is recursively constructed for the subspace $S_4 = (X_4, Dis)$ as shown in Fig. 2c. At the 5th level, u_7 is created to represent Y_5 and a subtree is created to denote X_5 as shown in Fig. 2d. At the leaf level, we construct the HST in Fig. 1.

3.2.3 Distortion Guarantee. We prove the distortion guarantee (defined in Def. 5) of Algo. 2 is $O(\log n)$ in Theorem 1.

Theorem 1. *The distortion guarantee of Algo. 2 is $O(\log n)$.*

Proof. The *main idea* is to use our new equivalent expression of distortion and the fact that the distortion guarantee of the baseline FRT is $O(\log n)$ [26], *i.e.*, $c_\beta \log n$ for a fixed β, where c_β is a constant factor depending on β. Since Algo. 2 uses the same distribution in FRT to sample β, we only need to prove *the distortion guarantee of Algo. 2 is no higher than $c_\beta \log n$ for fixed β.*

Without loss of generality (WLOG), we assume $S_0 = (V_0, Dis)$ is an input of Algo. 2 and $|V_0| = n_0$. Here, S_0 can be either the original metric S or a separated subspace in line 11. T^* is an HST of S_0 constructed by Algo. 2 and ρ^* is its distortion. T is an HST of S_0 constructed by FRT Algo. 1 and $\rho \le c_\beta \log n_0$ is its distortion. We use $\text{Stretch}_{T^*}(x, y)$ and $\text{Stretch}_T(x, y)$ to denote the stretch between any $x, y \in V_0$ on T^* and T, respectively. From the prerequisite, we know $\text{Stretch}_T(x, y) \le \rho \le c_\beta \log n_0$ for all points $x, y \in V_0$.

Next, we use our new equivalent expression of distortion in Eq. (4) to derive the distortion of T^*. As the distortion is recursively defined in Eq. (4), we focus on the term $\max_i \text{Stretch}(X_i, Y_i)$, *i.e.*, the value of stret in line 6 of Algo. 2. Line 7 guarantees the best center cp_0^* has the minimum stret (denoted by stret$_0^*$). We consider two cases based on cp_0^* in Algo. 2 and the parameter π in FRT:

(1) $cp_0^* \ne \pi[1]$. In FRT (Algo. 1), the left-most node is separated by the center $\pi[1]$ at each level. For example, the nodes $u_0, u_1, u_2, u_4, u_7, u_{12}$ in Fig. 1 are separated by $\pi[1] = p_2$ in Example 1. Thus, the points contained in each left-most node can be viewed as X_i and the union of the points in its right siblings can be viewed as Y_i. As the $\max_i \text{Stretch}(X_i, Y_i)$ by $\pi[1]$ is larger than that by the best center cp_0^*, we can derive that $\max_i \text{Stretch}_{T^*}(X_i, Y_i) \le \max_i \text{Stretch}_T(X_i, Y_i) \le c_\beta \log n_0$ for this case.

(2) $cp_0^* = \pi[1]$. Similarly, we also derive $\max_i \text{Stretch}_{T^*}(X_i, Y_i) = \max_i \text{Stretch}_T(X_i, Y_i) \le c_\beta \log n_0$ for this case.

Since we have proved stret$_0^* = \max_i \text{Stretch}_{T^*}(X_i, Y_i) \le c_\beta \log n_0$, we then study the recursive term $\max_i \text{Distort}(X_i)$ in Eq. (4).

WLOG, we assume Algo. 2 has recursively constructed subtrees for subspaces $S_1 = (V_1, Dis), \cdots, S_m = (V_m, Dis)$, where $n_i = |V_i| < n_0$. We use cp_i^* to denote the best centers in each subspace S_i and stret$_i^*$ to denote the corresponding $\max_i \text{Stretch}(X_i, Y_i)$ by cp_i^* in this subspace. As our previous analysis holds for any input space, we have stret$_i^* \le c_\beta \log n_i < c_\beta \log n_0$ for any i in $[1, m]$. By Eq. (4), the distortion of our HST T^* is $\max_{i=0}^m \text{stret}_i \le c_\beta \log n_0$.

We complete this proof by substituting $S_0 = S$ and $n_0 = n$. □

By substituting $c_\beta = 8$, the proof above also shows Algo. 2 gets the same distortion guarantee ($8 \log n$) for 2-HSTs as in FRT [26].

3.2.4 Time Complexity. We assume the dimension d of the original metric is constant and hence $Dis(x, y)$ takes $O(1)$ time. We use n_i to denote the number of points in X_i. In Algo. 2, line 2 takes $O(n)$ time. Line 4 and 8 take $O(H)$ time. Line 5 takes $O(|X_i||Y_i|) = O(n \cdot n_i)$ time to calculate $\text{Stretch}(X_i, Y_i)$. Line 9 takes $O(n)$ time. Line 11 is the recursion procedure over X_i with n_i points. Thus, we can derive

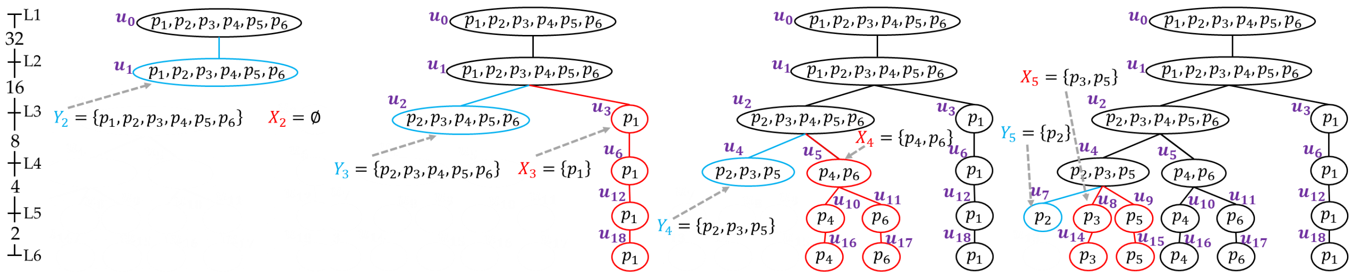


Figure 2: Illustrations of the construction procedure by our divide-and-conquer based framework DC

Table 3: Comparisons between our algorithms and FRT

Algorithm	Time	Space	Distortion Guarantee
FRT [26]	$O(n^2)$	$O(n)$	$O(\log n)$
DC (Sec. 3)	$O(n^4 \log n)$	$O(n)$	$O(\log n)$
DCnn (Sec. 4.2)	$O(n^3 \log n)$	$O(n)$	$O(\log n)$
DCsam (Sec. 4.3)	$O(n^{1.5} \log^2 n)$	$O(n)$	$O(\log n)$

the worst-case running time $T(n)$ by the recursion below,

$$T(n) = \sum_{i=l+1}^{H+1} T(n_i) + O(n^2 H \sum_{i=l+1}^{H+1} n_i), \tag{5}$$

where $\sum_{i=l+1}^{H+1} n_i = O(n)$. Algo. 2 has totally n recursions, since each recursion creates one unique leaf (*i.e.*, cp^*) and an HST has n leaves. Thus, Eq. (5) is bounded by $O(n \cdot n^3 H) = O(n^4 H)$. As $\Delta \leq O(poly(n))$ and $H = O(\log \Delta) = O(\log n)$, the worst-case time complexity of Algo. 2 is $O(n^4 \log n)$.

4 OUR OPTIMIZATION METHODS

A naive implementation of our framework takes $O(n^4 \log n)$ time, which is much slower than the baseline FRT ($O(n^2)$). Thus, we present our optimization techniques in Sec. 4.1-Sec. 4.3 to improve the time efficiency and discuss practical issues (*e.g.*, achieving linear space of HSTs) in Sec. 4.4. Table 3 lists the comparisons between our proposed algorithms and the baseline FRT [26].

4.1 Main Idea

Understanding Inefficiency of Our Algorithm DC. Based on the time complexity analysis of Algo. 2, two factors lead to the inefficiency of DC: (1) it is time-consuming to compute the maximum stretch between two separated subspaces, *i.e.*, Stretch(X_i, Y_i) in the new expression of distortion (Eq. (4)), and (2) all the points are tested to be candidate centers to find a good division.

Main Idea of Optimization Techniques. To alleviate the *first factor*, we identify that the distortion computation in our new expression can be reduced to the nearest neighbor (NN) search in Sec. 4.2. We also show that the approximate nearest neighbor (ANN) can be used to handle multi-dimensional data. This reduction brings us the chance to reduce the time complexity, since efficient indexes to NN and ANN have been widely studied. By indexing, Sec. 4.2 also designs a pruning strategy to avoid using all the points as centers, which alleviates the second factor to some extent.

To fully overcome the *second factor*, we use sampling to pick a certain number of points as centers in Sec. 4.3. Intuitively, this sample number makes the trade-off between distortion and running time. Thus, we derive a lower bound of this sample number ($O(\log n)$) based on the distortion guarantee in Lemma 4. By using $O(\log n)$ samples, our algorithm (DCsam in Table 3) takes $O(n^{1.5} \log^2 n)$ time, which is asymptotically faster than the baseline FRT [26].

4.2 Optimization By Indexing

This subsection presents our indexing based optimization, which reduces the time complexity from $O(n^4 \log n)$ to $O(n^3 \log n)$.

4.2.1 Reducing Distortion Computation to NN Search. We have presented a new expression of distortion in Eq. (4), where recursively computes the distortion by the maximum stretch between separated subspaces (*i.e.*, $\max_i$ Stretch(X_i, Y_i)). Thus, we focus on reducing Stretch(X_i, Y_i) to NN search as follows.

$$\text{Stretch}(X_i, Y_i) := \max_{x \in X_i, y \in Y_i} \frac{Dis_T(\text{leaf}(x), \text{leaf}(y))}{Dis(x, y)} \tag{6}$$

$$= \max_{x \in X_i, y \in Y_i} \frac{Dis_T(\text{leaf}(x), u_{i-1}) + Dis_T(u_{i-1}, \text{leaf}(y))}{Dis(x, y)} \tag{7}$$

$$= \max_{x \in X_i, y \in Y_i} \frac{2 \times Dis_T(\text{leaf}(x), u_{i-1})}{Dis(x, y)} \tag{8}$$

$$= \max_{x \in X_i, y \in Y_i} \frac{2(r_{i-1} + \cdots + r_H)}{Dis(x, y)} = \frac{2(r_{i-1} + \cdots + r_H)}{\min_{x \in X_i, y \in Y_i} Dis(x, y)} \tag{9}$$

$$= \frac{2\beta(k + k^2 \cdots + k^{H-i+2})}{\min_{x \in X_i, y \in Y_i} Dis(x, y)} = \frac{2\beta(k^{H-i+3} - k)/(k-1)}{\min_{x \in X_i} \{\min_{y \in Y_i} Dis(x, y)\}} \tag{10}$$

Eq. (6) is due to the definition of Stretch$(\cdot, \cdot)$. In Eq. (7), an internal node u_{i-1} is created to be the lowest common ancestor (LCA) of x and y, *i.e.*, lca(leaf(x), leaf(y)) $= u_{i-1}$. Thus, the distance between leaf(x) and leaf(y) equals to the total distance from leaf(x)/leaf(y) to their LCA u_{i-1}, *i.e.*, $Dis_T(\text{leaf}(x), \text{leaf}(y)) = Dis_T(\text{leaf}(x), u_{i-1}) + Dis_T(u_{i-1}, \text{leaf}(y))$. Based on the edge weights of HSTs, the distances from leaf(x) or leaf(y) to u_{i-1} are equal. Finally, we derive Eq. (10), where $\min_{y \in Y_i} Dis(x, y)$ asks for the *nearest neighbor (NN)* of x over Y_i. Since the point number in Y_i decides the time of NN search, we present a relatively tight upper bound of Stretch(X_i, Y_i) to reduce the point number from $|Y_i|$ to $|X_{i+1}|$.

Lemma 1. *Given X_i and $Y_i = \bigcup_{j=i+1}^{H+2} X_j$ in Algo. 2, we have*

$$\textit{Stretch}(X_i, Y_i) \leq \left\{ \frac{2(k^3 - k^{i+1-H})}{(k-1)^2}, \frac{2\beta(k^{H-i+3} - k)/(k-1)}{\min_{x \in X_i} \min_{y \in X_{i+1}} Dis(x, y)} \right\}$$

Proof. By the definitions of Y_i, Y_{i+1} and Eq. (10), we only need to prove the following inequality to derive this lemma.

$$\frac{2\beta(k^{H-i+3} - k)/(k-1)}{\min_{x \in X_i} \{\min_{y \in Y_i \setminus X_{i+1}} Dis(x, y)\}} \leq \frac{2(k^3 - k^{i+1-H})}{(k-1)^2}. \tag{11}$$

Based on the line 7 of Algo. 2, we know (1) $Dis(cp, y) < r_{i+1}$ for any point y in $Y_i \setminus X_{i+1} = Y_{i+1}$ and (2) $r_i \leq Dis(cp, x) < r_{i-1}$ for any point x in X_i. Since $r_i = \beta k^{H-(i-1)}$, we have

$$Dis(x,y) \geq Dis(cp,x) - Dis(cp,y) \geq r_i - r_{i+1} = \beta(k-1)k^{H-i} \quad (12)$$

By substituting Eq. (12) into the left-hand side (LHS) of Eq. (11), we have the upper bound of the LHS as

$$\frac{2\beta(k^{H-i+3}-k)/(k-1)}{\beta(k-1)k^{H-i}} = \frac{2(k^{H-i+3}-k)}{(k-1)^2 k^{H-i}} = \frac{2(k^3 - k^{i+1-H})}{(k-1)^2}.$$

Note that this upper bound is usually much smaller than the distortion guarantee $8 \log n$. For example, when $k = 2$ (the most popular parameter selection), it is always smaller than 16. Thus, we have derived a relatively tight upper bound for $\text{Stretch}(X_i, Y_i)$. □

Pruning via Indexing. Let $\text{Stretch}^{\uparrow}(X_i, Y_i)$ denote the upper bound of $\text{Stretch}(X_i, Y_i)$ defined in Lemma 1. Since X_i and Y_i are divided by cp at the level i, we use $\text{UB}[cp][i]$ to denote the maximum of these upper bounds between levels i and $H+1$, *i.e.*,

$$\text{UB}[cp][i] := \max_{i \leq j \leq H+1} \text{Stretch}^{\uparrow}(X_i, Y_i) \quad (13)$$

By dynamic programming (DP), we can rewrite Eq. (13) as

$$\text{UB}[cp][i] = \max\{\text{UB}[cp][i+1], \text{Stretch}^{\uparrow}(X_i, Y_i)\} \quad (14)$$

Let LB denote the maximum of $\text{stret}^* = \max_i \text{Stretch}^{\uparrow}(X_i, Y_i)$ with respect to (*w.r.t.*) the best center cp^* during *all the recursions* of Algo. 2 (lines 6, 7 and 11). Thus, LB can be viewed as the lower bound of the distortion. Once we have found a division whose maximum stretch between separated subspaces is lower than LB, we can stop enumerating the next center as the current one is good enough to retain the distortion. Our pruning lemma is as follows.

Lemma 2. *We are given a subspace $S' = (V', Dis)$ at current level l during the recursions of Algo. 2. For any center point $cp \in V'$, if $UB[cp][l+1] \leq LB$ or $stret \leq LB$, we can directly use cp as the best center point cp^* without changing the distortion guarantee.*

Proof. Since LB is the maximum of stret^* *w.r.t.* the best center cp^* during all the recursions of Algo. 2, we know the distortion defined in Eq. (4) is larger than LB. We use stret to be the maximum stretch of the separated subspaces by the current center point cp. When stret $\leq$ LB, it indicates that the current division scheme cannot increase the distortion. Thus, the current center point cp is good enough to keep the distortion guarantee $O(\log n)$.

When $\text{UB}[cp][l+1] \leq \text{LB}$, $\text{UB}[cp][l+1]$ is the upper bound of stret for the center point cp to divide a larger space ($S = (V, Dis)$), where S' is separated from this space. In other words, the maximum stretch to divide S at level l by cp is $\text{UB}[cp][l+1]$. As $V' \subseteq V$, we can infer that stret $\leq \text{UB}[cp][l+1] \leq \text{LB}$ by Eq. (3). □

Remark. In our framework (Algo. 2), we can also use an index to compute the separated subspaces (lines 7 and 11) by range queries. However, such an implementation involves H (exact) range queries over the point set Y_{i-1}, which can be slow for multi-dimensional data. Instead, we use a faster implementation that takes $O(n)$ time. The *basic idea* is to directly compute the level at which a point $x \in X_i$ is separated from the set Y_{i-1} (*i.e.*, outside the circular range $\text{B}(cp, r_i)$). This level is denoted by $\text{lev}(x, cp)$. Since $r_i \leq Dis(x, cp) < r_{i-1}$ and $r_i = k^{H-(i-1)}\beta$, we can derive that

$$\text{lev}(x, cp) = \begin{cases} H + 1 - \lfloor \log_k \left(Dis(x, cp)/\beta\right) \rfloor, & \text{if } x \neq cp \\ H + 2, & \text{if } x = cp \end{cases} \quad (15)$$

Algorithm 3: Our algorithm DCnn

input : a metric space $S = (V, Dis)$, current level l, and β
output: a k-HST with height H of this metric space S

1 Root $u_l \leftarrow$ represents V, $cp^* \leftarrow$ null, $\text{stret}^* \leftarrow \infty$;
2 $V \leftarrow$ sort each point v in V by $\text{UB}[v][l+1]$;
3 **if** $UB[V[0]][l+1] \leq LB$ **then** Use $V[0]$ as cp^* to execute lines 8-12 of Algo. 2 and **return**; // Prune
4 **foreach** *center* $cp \in V$ **do** // Find a good division
  // Compute separated subspaces
5   $X \leftarrow$ sort points in V by $\text{lev}(x, cp)$ in Eq. (15);
6   $\forall i \in (l, H+2], X_i \leftarrow$ all $x \in X$ whose $\text{lev}(x, cp) = i$;
  // Reduce to nearest neighbor (NN) search
7   **for** *level* $i \leftarrow H+1$ **to** $l+1$ **do**
8     $\text{Stretch}^{\uparrow}(X_i, Y_i) \leftarrow$ upper bound of $\text{Stretch}(X_i, Y_i)$ in Lemma 1 by NN or ANN search;
9     $\text{stret} \leftarrow \max\{\text{stret}, \text{Stretch}^{\uparrow}(X_i, Y_i)\}$;
10     $\text{UB}[cp][i] \leftarrow \min\{\text{UB}[cp][i+1], \text{Stretch}^{\uparrow}(X_i, Y_i)\}$;
11   **if** $stret > stret^*$ **then break**; // Prune
12   **if** $stret < stret^*$ **then** $\text{stret}^*, cp^* \leftarrow \text{stret}, cp$;
13   **if** $stret^* \leq LB$ **then break**; // Prune
14 $\text{LB} \leftarrow \max\{\text{LB}, \text{stret}^*\}$, execute lines 8-12 of Algo. 2;

After getting these levels, we use a counting sort to derive the separated subspaces. We use the counting sort (instead of other sorting methods), since the level here is no larger than $H + 2 = O(\log n)$ and a counting sort takes linear time and linear space.

4.2.2 *Algorithm Details.* Algo. 3 illustrates a divide-and-conquer based method (DCnn) with the optimization above. Line 1 is same as that of Algo. 2. In line 2, we sort each point $v \in V$ based on its upper bound ($\text{UB}[v][l+1]$) of the maximum stretch between their separated subspaces (when v is used as the center). Line 3 is a pruning by Lemma 2. We find a good division in lines 4-13. Specifically, we compute the separated point sets X_i in lines 5-6 by a counting sort. In lines 7-11, we calculate the upper bound of the maximum stretch between the separated subspaces $S_i = (X_i, Dis)$. First, we enumerate each level in line 7. Then, we compute $\text{Stretch}^{\uparrow}(X_i, Y_i)$ by answering NN queries over the points X_{i+1} based on Lemma 1. In line 10, we update the current upper bound $\text{UB}[cp][i]$ by Eq. (14). We maintain the best center cp^* and the corresponding upper bound (stret^*) in line 12. Line 13 is another pruning based on Lemma 2. In line 14, we maintain the lower bound (*i.e.*, LB) of the final distortion. After getting a good division, we run lines 8-12 of Algo. 2 to execute the divide-and-conquer procedure. Here, we still use lines 5-6 to compute the separated subspaces by cp^*.

Example 3. Back to Example 2. Algo. 3 also iterates each point in p_1-p_6 as the center point cp. When $cp = p_2$, we first calculate $\text{lev}(x, cp)$ by Eq. (15) for each $x \in \{p_1, \cdots, p_6\}$. For example, $\text{lev}(p_1, p_2) = H + 1 - \lfloor \log_2 (Dis(p_1, p_2)/\beta) \rfloor = 5 + 1 - \lfloor \log_2 \sqrt{74} \rfloor = 3$ (*i.e.*, the first case of Eq. (15)), and $\text{lev}(p_2, p_2) = H + 2 = 7$ (*i.e.*, the second case of Eq. (15)). The other values of $\text{lev}(x, cp)$ are listed in Table 4 and these values represent the level at which each point x is separated from Y_{i-1} in Fig. 1. In line 5, we use a counting sort to obtain an ordered sequence of the points $X = \{p_1, p_4, p_6, p_3, p_5, p_2\}$. In line 6, we can process that $X_2 = \emptyset$, $X_3 = \{p_1\}$, $X_4 = \{p_4, p_6\}$,

Table 4: **Values of lev(x, cp) when $cp = p_2$ in Example 3**

x	p_1	p_2	p_3	p_4	p_5	p_6
lev(x, p_2)	3	7	5	4	5	4

$X_5 = \{p_3, p_5\}$, $X_6 = \emptyset$ and $X_7 = \{p_2\}$. We can also verify that the separated point sets are same as the results in Example 2. Lines 7-11 calculate the upper bound (stret) by Lemma 1. Specifically, when level $i = H + 1 = 6$, we need to find out the NN of each point in X_6 over the dataset X_7. Since X_6 is an empty set, we have $\text{Stretch}^{\uparrow}(X_6, Y_6) = 2(2^3 - 2^{6+1-5})/1^2 = 8$ by Lemma 1. Next, we set stret = 8 and UB$[cp][i] = 8$ in lines 10-11. When i decreases to 3, we answer the NN of $X_3 = \{p_1\}$ over the dataset $X_4 = \{p_4, p_6\}$. By Lemma 1, we derive $\text{Stretch}^{\uparrow}(X_3, Y_3)$ as

$$\max\left\{\frac{2(2^3 - 2^{3+1-5})}{(2-1)^2}, \frac{2(2^{5-3+3} - 2)}{Dis(p_1, p_4)}\right\} = \max\left\{15, \frac{60}{\sqrt{26}}\right\} = 15,$$

where p_4 is the NN of p_1 over X_4. Eventually, we have $cp^* = p_2$.

4.2.3 Distortion Guarantee. In Lemma 3, we prove the distortion guarantee of Algo. 3 is still $O(\log n)$ by using either NN or ANN search. This lemma holds for any ANN algorithm that reports any point within c times the nearest distance (*i.e.*, c-approximate NN). For those ANN algorithms that achieve this c-approximation with high probability, $O(\log n)$ will hold with high probability.

Lemma 3. *By using either (exact) NN search or c-approximate NN search, the distortion guarantee of Algo. 3 is still $O(\log n)$.*

Proof. For exact NN search, the correctness of this statement can be directly derived from the proofs of Eq. (10)-(15) and Lemma 1-2 and the distortion guarantee of Algo. 2 in Theorem 1.

For c-approximate NN search, we have $Dis(x, y')/c \le Dis(x, y^*) \le Dis(x, y')$, where y^* is the NN of x in X_{i+1} and y' is the ANN. Let cp' be the center found by ANN and the corresponding upper bound stret$'$ is obtained by substituting $Dis(x, y^*)$ with $Dis(x, y')/c$ in line 9. Thus, $\text{stret}' = \max_i\{\frac{2(k^3 - k^{i+1-H})}{(k-1)^2}, \frac{2\beta(k^{H-i+3} - k)/(k-1)}{\min_{x \in X_i} Dis(x, y')/c}\} \le c \times \text{stret}$ where stret is the upper bound of $\max_i \text{Stretch}(X_i, Y_i)$ in line 9 by exact NN. This is due to the definition of this upper bound in Lemma 1 and $Dis(x, y') \ge Dis(x, y^*)$ for any point $x \in X_i$. Thus, the distortion by ANN is at most c times the distortion by exact NN. Since c is a constant error bound, we complete the proof. □

4.2.4 Time Complexity. Our ELT problem focuses on $\mathbb{R}^d$ for fixed dimension d. Under this scope, there exists algorithms [5, 34, 35, 58] that take $O(n \log n)$ pre-processing time and $O(n)$ space to answer c-approximate NN query in $O(\log n)$ time. For low dimension (*e.g.*, $d = 2$), exact NN with an $O(\log n)$ query time also needs $O(n)$ space and $O(n \log n)$ pre-processing time [22, 52]. Accordingly, we analyze the time complexity of Algo. 3 as follows. Line 2 needs $O(n \log n)$ time to sort. Lines 5-6 only need $O(n)$ time by a counting sort, since lev$(y, cp) \le H + 2$. In each iteration of line 7, we take $O(|X_{i+1}| \log |X_{i+1}|)$ time to construct the index for NN/ANN search and $O(|X_i| \log |X_{i+1}|)$ time to find NN/ANN of any point in X_i over the point set X_{i+1}. Thus, for each center point cp, the total time cost of lines 7-11 is bounded by $\sum_{i=l+1}^{H+1} O((|X_{i+1}| + |X_i|) \log |X_{i+1}|) = O(n \log n)$, where $\bigcup_{i=l+1}^{H+2} |X_i| = n$. Line 3 and 14 are recursions, which are similar to lines 8-12 of Algo. 2. Thus, based on the analysis of Algo. 2, the time complexity of Algo. 3 is $O(n^3 \log n)$.

4.3 Optimization By Sampling

This subsection presents our sampling based optimization, which reduces the time complexity from $O(n^3 \log n)$ to $O(n^{1.5} \log^2 n)$.

4.3.1 Our Sampling Scheme. In Algo. 3, enumerating all centers in V leads to the worst case and we use sampling to address this issue.

Our Two-phase Sampling. To beat $O(n^2)$ time, the *basic idea* is to (1) do partitions by a two-phase sampling, (2) construct the subtrees of partitioned subspaces by Algo. 3 with the aforementioned simple sampling, and (3) merge the subtrees into the final HST. Our two-phase sampling scheme is as follows:

Phase 1. We randomly pick m centers from the point set V (without replacement). For each center cp, we keep partitioning the point set as in line 5 of Algo. 3 until the number of remaining points is bounded by a parameter α. We mark these remaining points to be taken out from the sample set V, because cp and its closed points (*i.e.*, the remaining ones) may result in similar divisions at top levels and experiments show that the distortion defined in our new expression is often decided by the divisions at top levels.

Phase 2. We again sample m centers from the marked remaining points of the best center from the first phase. Since the number of these remaining points is bounded by $O(\alpha)$, we will try m samples from $O(\alpha)$ points and maintain the best center point cp^*.

Rationale of Two-phase Sampling. We first assume the original metric space $S = (V, Dis)$ is partitioned into several subspaces $\{S_i = (V_i, Dis)\}$, where the number of points in each subspace is bounded by a parameter α (*i.e.*, $|V_i| = O(\alpha)$). Hence, each subtree of these subspaces can be constructed in $O(m\alpha^2 \log \alpha)$ time by Algo. 3 based on its time complexity in Sec. 4.2.4. Thus, it takes $O(\frac{n}{\alpha} m\alpha^2 \log \alpha) = O(mn\alpha \log \alpha)$ time to construct all the subtrees. When $m = O(\log n)$ and $\alpha = O(\sqrt{n})$, this time cost is $O(n^{1.5} \log^2 n)$, which is asymptotically faster than $O(n^2)$. As long as partitioning also takes $O(n^{1.5} \log^2 n)$ time via our two-phase sampling, the overall time complexity will beat the state-of-the-art [26]. Thus, we present the algorithm details of partitioning in Sec. 4.3.2 and analyze the parameter selections of m and α in Sec. 4.3.3-4.3.4. Note that the parameter selections ($m = O(\log n)$ and $\alpha = O(\sqrt{n})$) are based on the theoretical analysis in Sec. 4.3.3 and Sec. 4.3.4.

4.3.2 Algorithm Details. Algo. 4 illustrates the recursive procedure to divide the original metric into several subspaces each with less than $O(\alpha)$ points. Lines 2-11 are the *first phase*. Specifically, we randomly pick $O(m)$ center points from V (without replacement) in lines 2-3. Lines 4-5 compute the separated subspaces by cp. Line 6 finds the largest level i^* in $[l+1, H+2]$ such that the total number of remaining points is still bounded by $O(\alpha)$. In other words, the point set $Y_{i^*} = \bigcup_{j=i^*+1}^{H+2} X_j$ at the level i^* has $O(\alpha)$ points. Lines 7-9 calculate the upper bound of the maximum stretch between separated subspaces by the same way in Algo. 3. In line 11, we mark the remaining points to be taken out from the possible samples. Line 12 is the *second phase*, where we enumerate each center from the marked points of cp^* (in line 11) and maintain the best center cp^*. Line 13 computes the separated subspaces by cp^*. Lines 14-17 create a node u_i to represent the point set Y_i. Each subspace X_i is then recursively partitioned by Algo. 4 in line 16. Line 17 merges the subtree of each subspace into the final HST. In line 18, we can obtain a tree/subtree of space/subspace S, where each leaf node represents a disjoint subspace SS_i each with $O(\alpha)$ points. After

Algorithm 4: Our algorithm DCsam

1 **if** $|V| \le O(\alpha)$ **then return** *an empty subtree and* S *itself*;
2 **for** *sampleID* $\leftarrow 1$ **to** $O(m)$ **do** // Phase 1
3     $cp \leftarrow$ a random center from unmarked points in V;
4     $X \leftarrow$ sort points x in V by $\text{lev}(x, cp)$ in Eq. (15);
5     $\forall i \in (l, H+2], X_i \leftarrow$ all $x \in X$ whose $\text{lev}(x, cp) = i$;
6     $i^* \leftarrow \arg\max\{i \in [l+1, H+2] \mid \sum_{j=i+1}^{H+2} |X_i| = O(\alpha)\}$;
7   **for** *level* $i \leftarrow i^*$ **to** $l+1$ **do**
8         $\text{Stretch}^{\uparrow}(X_i, Y_i) \leftarrow$ upper bound of $\text{Stretch}(X_i, Y_i)$ in Lemma 1 by NN or ANN search;
9         $\text{stret} \leftarrow \max\{\text{stret}, \text{Stretch}^{\uparrow}(X_i, Y_i)\}$;
10   **if** *stret* $<$ *stret** **then** $\text{stret}^*, cp^* \leftarrow \text{stret}, cp$;
11   Mark the points separated in levels i^*+1-$H+2$;
12 **foreach** $O(m)$ *samples from the marked points of* cp^* **do** run lines 3-10 ; // Phase 2
13 Run lines 3-6 with cp^*, $SS_{i^*+1} = (\bigcup_{j=i^*+1}^{H+2} X_j, Dis)$;
14 **for** $i \leftarrow l+1$ **to** i^* **do** // Divide-and-Conquer
15   Node $u_i \leftarrow$ represent points $Y_i = \bigcup_{j=i+1}^{H+2} X_j$ and add u_i to the child nodes of u_{i-1} with edge weight $r_i \times k$;
16     $T_i, SS_i \leftarrow$ partition subspace $S_i = (X_i, Dis)$ by this algorithm recursively at the level $i-1$;
17   Link the child nodes of T_i's root node as the child nodes of u_{i-1} with the edge weight $r_i \times k$;
18 **return** *a subtree created by* $\{u_i\}$ *and a set of subspaces* SS_i;

that, we use Algo. 3 (with $O(m)$ sampled centers) to process all the subspaces in SS and merge their subtrees into the final HST.

Example 4. Back to Example 2. We assume the number of sampled centers $m = \log n \approx 3$ and the parameter $\alpha = \sqrt{n} \approx 3$. We also assume p_2 is the first sample in line 2. Lines 4-5 process the separated subspaces, *i.e.*, $X_2 = \emptyset$, $X_3 = \{p_1\}$, $X_4 = \{p_4, p_6\}$, $X_5 = \{p_3, p_5\}$, $X_6 = \emptyset$ and $X_7 = \{p_2\}$. Line 6 derives $i^* = 4$ and lines 7-9 calculate stret $= 11.77$. The detailed procedure of lines 4-6 is referred to Example 3. Then we have $\text{stret}^* = 11.77$ and $cp^* = p_2$ in line 10. We mark the remaining points p_2, p_3, p_5 in line 11 and they are removed from the candidate samples. At the end of the first phase, we have $cp^* = p_2$. Line 12 is the second phase of our sampling scheme and we sample $m = 3$ points from the marked points (*e.g.*, p_3, p_5) of p_2. Since they both lead to larger stret than 11.77, cp^* remains to be p_2. In line 13, we have $SS_5 = (\{p_2, p_3, p_5\}, Dis)$. When level $i = 2$ in line 14, we create the internal node u_1 in Fig. 1 to represent $Y_2 = \{p_1, \cdots, p_6\}$. When $i = 3$, we create the internal node u_2 to represent $Y_3 = \{p_2, \cdots, p_6\}$. The subspace $S_3 = (X_3, Dis)$ is recursively partitioned in line 16, and we obtain an empty tree T_3 and a subspace $SS_3 = (X_3, Dis)$ (since $|X_3| \le \alpha$). Similarly, when $i = i^* = 4$, we create the internal node u_4 to represent $Y_4 = \{p_2, p_3, p_5\}$. In line 16, we get an empty subtree T_4 and a subspace $SS_4 = (X_4, Dis)$. Finally, we use Algo. 3 to construct the subtrees for each subspace SS_i and merge them into the final HST by the red edges in Fig. 1.

4.3.3 Distortion Guarantee.

We prove the distortion guarantee of Algo. 4 is still $O(\log n)$ when $m = O(\log n)$ in Lemma 4.

Lemma 4. *When the number of sampled centers $m \ge O(\log n)$, the distortion guarantee of Algo. 4 is still $O(\log n)$.*

Proof. Let m be the number of samples and $\rho_1, \cdots, \rho_m$ be the *maximum stretch* between the separated subspaces by the sampled centers. We need to prove $m = O(\log n)$ sample is enough to achieve the distortion guarantee $O(\log n)$ with a high probability (*e.g.*, $1 - 1/n$). We assume the desired value is $8\log n + \delta$ as the distortion guarantee of the baseline FRT is proved to be $8 \log n$ in [26]. We can derive Eq. (16) by Markov's inequality [44].

$$\Pr\left[\rho_i \ge 8\log n + \delta\right] \le \frac{\mathbb{E}[\rho_i]}{8\log n + \delta} \le \frac{8\log n}{8\log n + \delta} \tag{16}$$

Thus, the probability that no samples have the desired *maximum stretch* is bounded by $(\frac{8\log n}{8\log n + \delta})^m$. If this probability is lower than $1/n$, we can derive the bound of m as follows.

$$\left(\frac{8\log n}{8\log n + \delta}\right)^m \le \frac{1}{n} \implies m \ge \frac{\log n}{\log\left(8\log n + \delta\right) - \log\left(8\log n\right)} \tag{17}$$

By choosing a proper $\delta = O(\log n)$, we have $m = O(\log n)$. □

4.3.4 Time Complexity.

Since Algo. 4 improves over Algo. 3 by using sampling, we can recursively define the running time of Algo. 4 as follows based on the previous complexity analysis.

$$T(n) = \begin{cases} O(1), & \text{if } n \le c \\ \sum_{i=l+1}^{H+1} T(n_i) + O(mn\log n), & \text{otherwise} \end{cases} \tag{18}$$

As line 16 has $O(n/\alpha)$ partitions and $m = \log n$ by Lemma 4, we can derive the time complexity by mathematical induction.

$$T(n) = O(\max\{H, n/\alpha\} \times mn \log n) \tag{19}$$

Besides, it takes $O(\frac{n}{\alpha} \times \alpha^2 \log^2 \alpha)$ time to handle the partitioned subspaces by using only $O(\log \alpha)$ samples in Algo. 3. Thus, we can achieve the optimal time complexity when $T(n)$ in Eq. (19) asymptotically equals $O(\frac{n}{\alpha}\alpha^2 \log^2 \alpha)$. Finally, we have $\alpha = O(\sqrt{n})$ and the time complexity of Algo. 4 is $O(n^{1.5}\log^2 n)$.

Remark. To achieve the time complexity of $O(n^{1.5}\log^2 n)$, we use the following lemma to derive the tree height.

Lemma 5. *Let the upper bound $\Delta^{\uparrow}$ of the diameter Δ be the sum of the top-2 longest distances to a point $z \in V$ (which takes $O(n)$ time). The tree height H equals to either $\lceil \log_k \Delta^{\uparrow} \rceil + 1$ or $\lceil \log_k \Delta^{\uparrow} \rceil$.*

Proof. Suppose the diameter Δ equals to $Dis(x, y)$. We first prove the upper bound $\Delta^{\uparrow}$ is between Δ and 2Δ based on the triangle inequality and the definitions of $\Delta, \Delta^{\uparrow}$: (1) $\Delta = Dis(x, y) \le Dis(x, z) + Dis(z, y) \le \Delta^{\uparrow}$; and (2) $\Delta^{\uparrow} \le 2\max_{v \in V} Dis(v, z) \le 2\Delta$.

We next show how to derive the proper tree height. Let H' be $\lceil \log_k \Delta^{\uparrow} \rceil + 1$. Since $k \ge 2$ and $\Delta^{\uparrow} \in [\Delta, 2\Delta]$, we have $H' \in [H, H+1]$, where H' and H are integers. Then, we can first use H' to be the initial tree height and construct the HST by Algo. 4. If the constructed HST has only one node at the 2nd level, we will safely remove its root without changing the distortion. This is true since (1) the remaining subtree is also an HST and (2) the distortion defined in Eq. (3) remains the same (*i.e.*, the distance function $Dis_T(\cdot, \cdot)$ between any two leaves on the HST remains the same). For example, as shown in Fig. 1, we can safely remove the root u_0 without changing the distances between any two leaves and the distortion (11.77). Moreover, when $H' = H + 1$, we can prove this root must have only child, since the radius at the 2nd level, $r_2' = \beta \times k^{H'+1-2} = \beta k^H$, is longer than Δ. Thus, all the points are contained in only one node at the 2nd level. □

4.4 Discussion

This subsection discusses the following practical issues.

Achieving Linear Space. The standard HST constructed by either the baseline FRT or our solution takes $O(nH) = O(n \log n)$ spaces. To achieve a linear size space, we can use the compressing strategy proposed in [64]. The basic idea of this compressing strategy is to remove any redundant node that has only one child during the construction. This ensures that the total number of tree nodes is bounded by $O(n)$. For example, we can execute this operation after line 10 of Algo. 2 and line 15 of Algo. 4 (when $X_i = \emptyset$).

Beyond L_p Metrics. To extend our solution to non-L_p metrics, we only need to replace the (approximate) nearest neighbor search algorithm for these metrics. This extension benefits from the rich studies on the approximate nearest neighbor (ANN) search. For instance, we can apply a popular ANN library called FLANN [47, 48] for some non-L_p metrics such as chi-square histogram distance [49] and Hellinger distance [53]. The experiment in Appendix A of our full paper [2] shows this extension still achieves a notably lower distortion and 10.5× faster time efficiency than the state-of-the-art method FRT [26, 64]. In fact, existing work has proposed many ANN algorithms for other metrics that take (1) $O(\log n)$ query time, (2) $O(n \log n)$ pre-processing time, and (3) $O(n)$ space. Please refer to the recent work [5, 46] for a comprehensive survey.

Insertions and Deletion. In this scenario, points in the original metric are no longer static and the HST needs to be updated when some points are inserted or deleted. To handle this scenario, our construction methods can be extended by a data structure called Hierarchically Separated Forest (HSF) in [64]. We only need to replace the construction routine (FRT) with our algorithm DCsam. Our experiment in our full paper [2] demonstrates that this extension is 16× faster than the method in [64] and also has a lower distortion.

5 EXPERIMENTAL STUDY

In the experiment, we present our setup in Sec. 5.1 and results in Sec. 5.2. We also conduct a case study in Sec. 5.3 to demonstrate the motivations of using HSTs with low distortions.

5.1 Experimental Setup

Datasets. We use four **real datasets**, *NYC*, *Tokyo*, *Chengdu*, and *Haikou*, which are commonly used in existing work [56, 65–67]. *NYC* and *Tokyo* are collected by Foursquare [3] in New York and Tokyo. Their raw data [62] has 227,428 and 573,703 check-in records respectively and each record is associated with its user ID, a timestamp, the location and category of the check-in venue. *Chengdu* and *Haikou* are collected by Didi Chuxing [1] in Chengdu and Haikou. Their raw data [23] has 209,423 and 7,340,025 taxi-calling records and each record contains its appearance time, completion time, the pickup location and delivery location of the passenger. We extract the locations from these datasets and remove the duplicated ones since they are mapped into the same leaf on an HST. Table 5 lists the number of unique locations in these datasets.

In Table 6, we also generate four **synthetic datasets** (*Nor*, *Exp*, *Uni*, and *Skew*) to test the effect of the dimensionality and scalability. The dimension d is up to 100 and the number of points n is up to 100 million. The first three synthetic datasets are generated following the uniform, normal and exponential distributions, respectively. The range of each coordinate is $[-10^7, 10^7]$. The fourth dataset, skewed

Table 5: Real datasets in 2D Euclidean spaces (*i.e.*, L_2 metric)

Dataset	*NYC*	*Tokyo*	*Chengdu*	*Haikou*
#(points) (n)	42,981	67,123	227,447	319,419

Table 6: Synthetic datasets (default setting is underlined)

Parameter	Setting
Distribution	uniform, normal, exponential, skewed
#(points) (n)	5, 10, <u>50</u>, 100, $\cdots$, 100000 ($\times 10^3$)
#(dimensions) (d)	<u>2</u>, 3, 4, 5, 10, 20, 100

data, is generated from uniform data by raising the coordinates of 2nd-100th dimensions to their powers (*e.g.*, y to y^γ and $\gamma = 2$ by default), following the existing work [8, 50, 51].

Due to the page limitation, we focus on the most popular HST and distance metric in existing work: 2-HST (k=2) for L_2 metric ($p = 2$). In general, larger p will lead to longer running time and larger k will decrease the tree height and change the edge weight.

Compared Algorithms. We compare our algorithms, DC (Algo. 2), DCnn (Algo. 3), and DCsam (Algo. 4), with the following baselines.

- **Bar96** [14] and **Bar98** [15]. They were designed by Bartal to construct HSTs with different distortion guarantees: $O(\log n \log(\min\{n, \Delta\}))$ and $O(\log n \log\log n)$.
- **FRT** [26]. FRT [26] is the state-of-the-art solution with a distortion guarantee of $O(\log n)$. We use the $O(n^2)$-time implementation in [64] since it is faster than the other sequential implementations [17, 25, 26] in these datasets. Note that these implementations *always* have the same distortion.

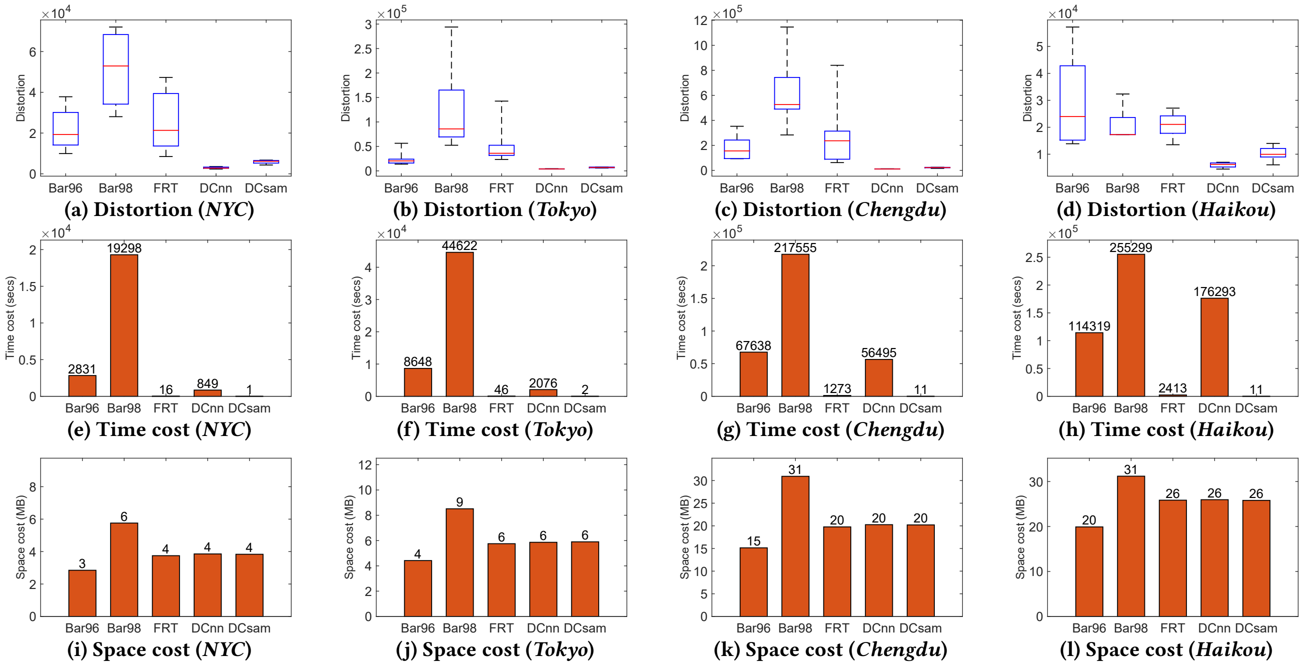
Implementation. All the algorithms are coded in C++. DCnn and DCsam use R-trees [16, 33] in the Boost library [21] to answer NN queries. Unlike DCnn, DCsam uses Arya and Mount's ANN library [9, 10] in ANN search for multi-dimensions (d = 3-20) and an LSH based algorithm QALSH [37, 38] for high-dimensions (d = 100). The error bound c for ANN is 3.0 for $d \leq 10$ and 3.5 for $d > 10$.

Metrics. The algorithms are evaluated in terms of *distortion*, *time cost*, and *space cost* of the constructed HST. Experiments are conducted on a server with Intel Xeon(R) 2.40GHz processors with 128GB RAM. Each setting is repeated 10 times and each time corresponds to a test case with the data points and parameters (*e.g.*, n and d). Finally, their average result is reported.

5.2 Experimental Result on Constructing HST

5.2.1 Result on Real Datasets. Fig. 3 shows the experimental results on the real datasets. These results fluctuate in Fig. 3a-3d since all the algorithms are randomized solutions. The distortions of our algorithms (DCnn and DCsam) are much lower and more stable than those of existing baselines. For example, the average distortions of DCnn and DCsam are up to 24.8× and 12.9× lower than that of FRT. The gap between DCnn and DCsam is up to 1.9×, which indicates our sampling based optimization retains low distortions.

In terms of *time efficiency*, our algorithm DCsam is the fastest and FRT is the runner-up. For example, DCsam is up to 224×, 10673×, and 23836× faster than FRT, Bar96, and Bar98, respectively. DCnn is often faster than Bar96 and Bar98, but slower than FRT. DCsam is faster than DCnn by up to 16439×, which demonstrates our sampling based optimization significantly reduces the time cost. The result of DC is ignored, since it is extremely slow. For example, DC takes almost 10 days to handle the smallest real dataset *NYC*.



(a) Distortion (*NYC*) (b) Distortion (*Tokyo*) (c) Distortion (*Chengdu*) (d) Distortion (*Haikou*)
(e) Time cost (*NYC*) (f) Time cost (*Tokyo*) (g) Time cost (*Chengdu*) (h) Time cost (*Haikou*)
(i) Space cost (*NYC*) (j) Space cost (*Tokyo*) (k) Space cost (*Chengdu*) (l) Space cost (*Haikou*)

Figure 3: Performance on the real datasets

As for *space cost*, all the algorithms are relatively efficient, since the space usages are less than 32MB. Bar96 takes the lowest space cost since its HSTs have lower heights than the others. The results of FRT and our algorithms are close due to the same tree height.

5.2.2 Result on Multi-Dimensional Synthetic Datasets. Fig. 4 depicts the results on multi-dimensional datasets. In terms of *distortion*, our algorithms are always more effective than the existing baselines. For example, DC/DCnn and DCsam have up to 14.6×-88.1× and 9.5×-41.7× lower distortions than the baselines. Our DCsam always has a lower distortion than FRT under these four distributions. For instance, the improvement of DCsam over FRT is up to 10% when $d = 100$. We also observe the distortions decrease with the increase of the dimension. It implies it is easier to handle the high-dimension data than low-dimension data. This may because pairwise distances are sparse on high-dimensional spaces due to the curse of dimensionality [58, 59]. Based on our expressions of the distortion in Eq. (4) and (10), the sparsity makes it easier to get low distortions (*i.e.*, nearest neighbors get farther).

As for *time cost*, DCsam is still the fastest and FRT is the runner-up. For instance, DCsam is faster than FRT by up to 5.3×. Moreover, it is faster than Bar98 and Bar96 by 3-5 orders of magnitude. Our algorithm DCnn is often more efficient than Bar98 and Bar96, while our naive implementation DC is the least efficient. Some results of Bar98, Bar96, DC, and DCnn are not provided, since their construction cannot be finished in 24 hours.

In terms of *memory usage*, all the algorithms need less than 8.5MB space. Bar96 is the most efficient and Bar98 is the least efficient. The space costs of FRT, DC, DCnn and DCsam are close.

5.2.3 Result on Scalability Tests. Fig. 5 illustrates the experimental results of the scalability tests. In terms of *distortion*, our proposed algorithms are notably more effective than the baselines. For example, DCsam has up to 16.9×-34.4× lower distortions than the baselines. Among the baselines, FRT and Bar96 are always better than Bar98. We also observe that the distortions of all the algorithms increase with the expansion of the data scale. This is reasonable since their distortion guarantees all increase with the data size (*i.e.*, n).

In terms of *time cost*, DCsam is always the fastest and FRT is the runner-up. For instance, DCsam is up to 783× faster than FRT, and DCsam is faster than DCnn by up to 4 orders of magnitude due to our sampling technique. DCnn is faster than DC by up to 565× because of our indexing based optimization. These results verify that DCsam is more capable of handling large-scale datasets. Those results, whose algorithms cannot terminate in 1 day, are omitted.

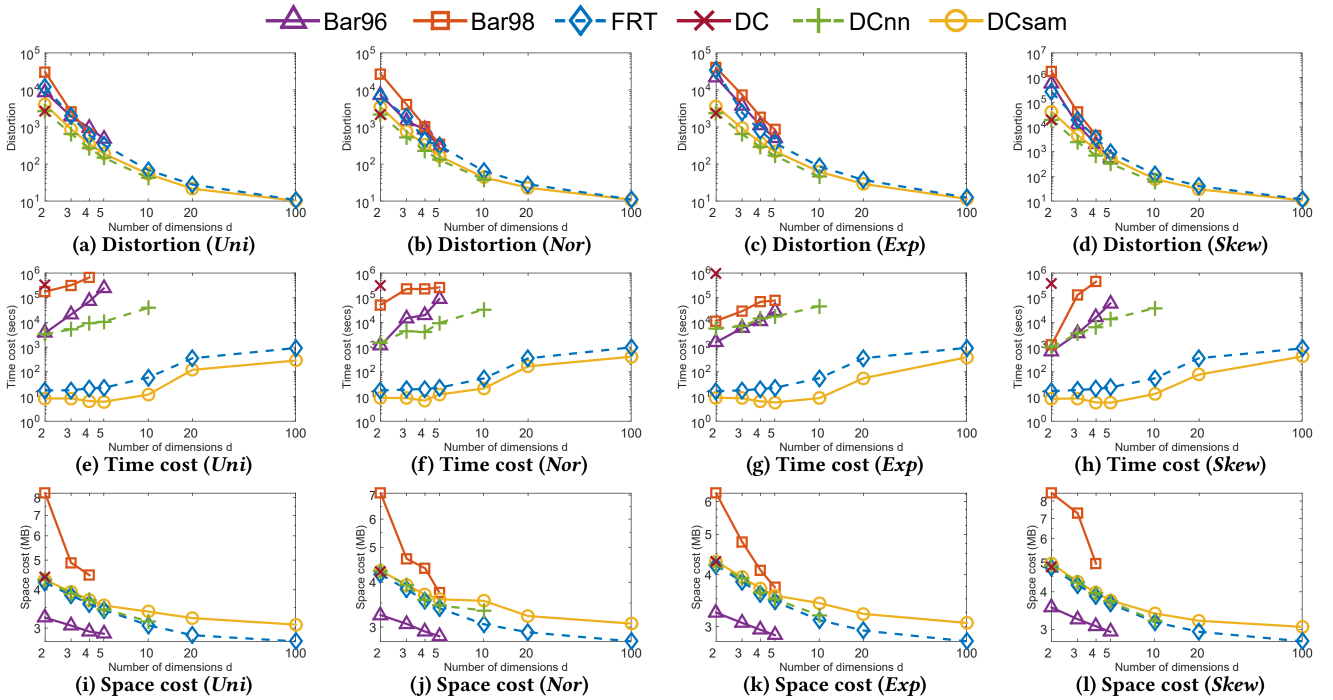
We omit the results of *space cost* due to the page limitation. The overall pattern is very similar to the previous results. The space cost of our DCsam is 9.2GB when n is 100 million.

5.2.4 Summary. The major experimental findings are as follows.

(1) In terms of *effectiveness*, our algorithms always have lower distortions than the existing baselines. For example, distortions of DCsam are up to 16×, 25×, and 41× lower than FRT, Bar96, and Bar98, respectively. Another observation is that distortions of existing baselines for high-dimensional data are better than for low-dimensional data due to the curse of dimensionality [58, 59].

(2) In terms of *efficiency*, DCsam is always the fastest. In real datasets, it is up to 224×, 10673×, and 23836× faster than FRT (runner-up), Bar96 and Bar98, respectively. Moreover, comparisons between DC, DCnn and DCsam validate our optimization techniques, which jointly reduces the running time by over 5 orders of magnitude. As for *space cost* of HSTs, the space of our algorithms is often close to that of FRT which is comparably efficient.

(3) Among the baselines, although FRT has the best distortion guarantee, its distortions are not robust enough and sometimes higher than Bar96 in real datasets. As for efficiency, FRT is the fastest, Bar96 takes the least space and Bar98 is the least efficient.

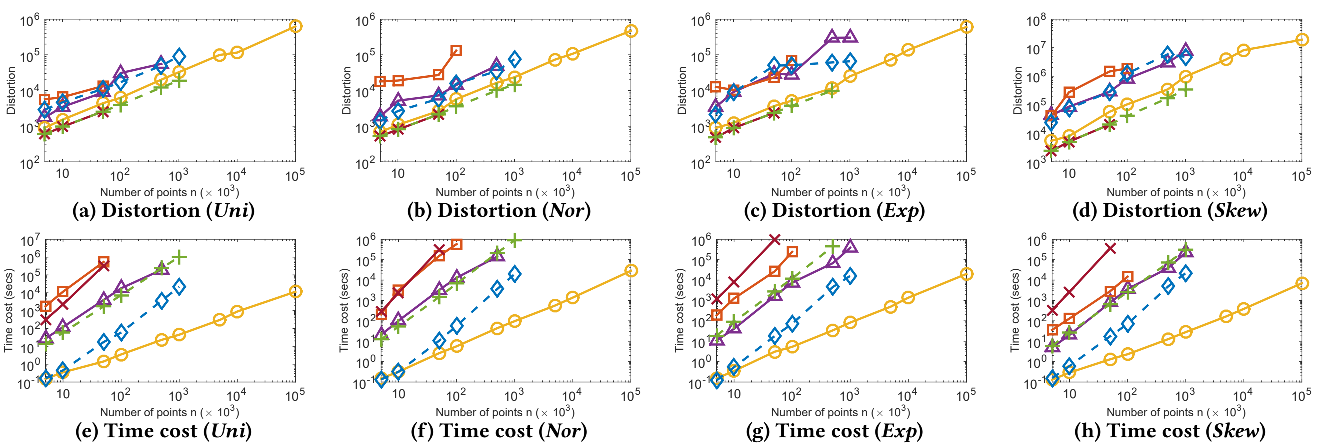


(a) Distortion (*Uni*) (b) Distortion (*Nor*) (c) Distortion (*Exp*) (d) Distortion (*Skew*)

(e) Time cost (*Uni*) (f) Time cost (*Nor*) (g) Time cost (*Exp*) (h) Time cost (*Skew*)

(i) Space cost (*Uni*) (j) Space cost (*Nor*) (k) Space cost (*Exp*) (l) Space cost (*Skew*)

Figure 4: Performance on the multi-dimensional synthetic datasets (both coordinate axes are in log scale)



(a) Distortion (*Uni*) (b) Distortion (*Nor*) (c) Distortion (*Exp*) (d) Distortion (*Skew*)

(e) Time cost (*Uni*) (f) Time cost (*Nor*) (g) Time cost (*Exp*) (h) Time cost (*Skew*)

Figure 5: Performance the scalability tests (both coordinate axes are in log scale)

5.3 A Case Study on Using HST

5.3.1 Motivation. We conduct a case study on using HSTs in a real problem to verify two things: (1) whether an algorithm can get better performance with HSTs than without HSTs, and (2) whether HSTs with lower distortions can further improve the performance.

5.3.2 Experimental Setup. Our case study focuses on the *Online Metric Bipartite Matching (OMBM)* problem [40], which has been widely used as the model for task assignment in spatial crowdsourcing [57]. In a bipartite graph, OMBM assumes the location of each left-hand vertex (worker's location) are known in advance and the location of each right-hand vertex (task's location) is only known when it appears. The edge weight is the distance between the locations. Here, we focus on the objective called *bottleneck distance* [7]. In other words, we aim to find a matching in this bipartite graph to minimize the longest distance between the matched pairs.

Dataset. We use the real dataset called *Shenzhou* in [55], since [55] has also studied the OMBM problem with a different objective in spatial crowdsourcing. *Shenzhou* is a taxi-calling dataset collected by UCAR Inc. [4]. It has four test groups and the number of vertices is up to 10292 in these tests. Please refer to [55] for more details.

Compared Algorithms. We compare the following algorithms:

(1) Greedy. For each right-hand vertex, it matches the nearest left-hand vertex by their distance on the 2D Euclidean space.

(2) Greedy-FRT and Greedy-DCsam. They also use the greedy strategy above by using distances on the HSTs of this Euclidean space, *i.e.*, 2-HSTs constructed by FRT and DCsam respectively.

We pick Greedy as an example since both Greedy [40] and HST-based Greedy [43] are widely used to solve the OMBM problem.

Metrics. In addition to the previous metrics, we also consider the objective (*i.e.*, bottleneck distance in the matching of OMBM).

Table 7: Experimental results of our case study

Algorithm	Distortion	Construction Time (Secs)	Bottleneck Distance
Greedy	N/A	N/A	151.3
Greedy-FRT	732.5	0.241	127.6
Greedy-DCsam	517.0	0.058	115.8

5.3.3 Experimental Result. Table 7 presents the average results on all the test cases of the *Shenzhou* dataset. We can easily observe that the objective (*i.e.*, bottleneck distance) is improved by using HSTs. For example, Greedy-FRT can reduce the bottleneck distance by 15.6% and Greedy-DCsam can improve the objective by 23.4%. It also demonstrates that HSTs with lower distortions can result in better effectiveness in practice. Although the result of distortion indicates that the original (Euclidean) distance is stretched on the HST, the objective (*i.e.*, the bottleneck distance) is still reasonably decreased, because (1) it is the actual moving distance (*i.e.*, Euclidean distance) between a worker and a task and (2) Greedy is myopic while Greedy-FRT/Greedy-DCsam has a good theoretical guarantee to find a better assignment [40, 43]. Besides, our algorithm DCsam still performs better than the baseline FRT in terms of both distortion and time efficiency when constructing HSTs. The space cost of the HSTs is always less than 0.43MB and the difference of the space cost between DCsam and FRT is very little (< 10KB).

5.3.4 Discussion. We have two observations from the case study.

- A lower distortion can lead to better effectiveness for the optimization problem that uses HSTs. This is reasonable since the theoretical guarantee of an HST-based solution for the original metric (*e.g.*, Euclidean metric for OMBM) is $\rho \times$ distortion, where ρ is the theoretical guarantee of this solution for HSTs. In other words, when the distortion of the constructed HST gets lower, this effectiveness guarantee will get closer to the optimum.
- it is important to construct an HST efficiently in some applications. For example, killer applications of the OMBM problem include task assignment in spatial crowdsourcing, taxi dispatching and food delivery [55, 57]. In these applications, the left-hand vertices of OMBM represent the locations of workers or drivers, which are periodically changed in practice. In Greedy-FRT and Greedy-DCsam, HSTs are constructed based on the locations of left-hand vertices. Thus, the construction of an HST should be scalable enough and cannot be done in advance. Otherwise, the high time cost in constructing an HST (by FRT) will become the time efficiency bottleneck of the HST-based algorithm (*e.g.*, Greedy-FRT).

In general, the first observation potentially holds in most of the optimization problems mentioned in Sec. 1, such as privacy protection and facility location planning. The second observation often holds in online problems such as task assignment in spatial crowdsourcing [57] and real-time trip planning [19, 63].

6 RELATED WORK

Our paper is closely related to the *Embedding Arbitrary metrics by Tree metrics* (EAT) problem. For other metric embedding problems, please refer to the surveys and textbooks [6, 35, 39, 58].

The EAT problem was first studied by Bartal in [14]. It aimed to embed arbitrary metrics by tree metrics with low distortions. Bartal also first proposed the *Hierarchically Separated Tree* (HST) to solve it. So far, HSTs have been widely used in many applications, such as task assignment [18, 55, 57], trip planning [19, 63], privacy preservation [24, 54], facility location planning [12, 20], distributed query processing [42], and clustering [13].

To minimize the *distortion* of an HST, early studies [14, 15, 25, 26, 39, 41] mainly focus on improving the distortion guarantees. Specifically, Bartal [14] proposed a graph-based algorithm with a distortion guarantee of $O(\log n \log(\min\{n, \Delta\}))$ and further improved the distortion guarantee to $O(\log n \log\log n)$ [15]. Konjevod *et al.* [41] followed [14, 15] and discovered a distortion guarantee of $O(\log \Delta)$. Indyk [39] converted a quadtree [27] into an HST with a distortion guarantee of $O(\log^4 n)$. Among these studies, Fakcharoenphol *et al.* [25, 26] proposed the state-of-the-art construction algorithm FRT with the tight distortion guarantee ($O(\log n)$).

Recent studies [17, 29, 32, 64] focused on improving the efficiency of FRT. Specifically, Zeng *et al.* [64] proposed a $O(n^2)$-time implementation with $O(n)$ space cost. Blelloch *et al.* [17] and Friedrichs *et al.* [29] studied the parallel versions to get low average-case time complexity. Gao *et al.* [32] aimed to minimize the communication cost in a distributed sensor network during the construction.

7 CONCLUSION

This paper studies the Embedding L_p metrics through Tree metrics (ELT) problem. Although solutions have been proposed to solve this problem with optimal theoretical guarantees ($O(\log n)$), they are still not effective and efficient enough in large-scale datasets. To achieve a low distortion, we first present a divide-and-conquer based framework, which has a high time cost. We next propose two optimization techniques (indexing and sampling) and design an algorithm called DCsam with the optimal theoretical guarantee and a low time complexity ($O(n^{1.5} \log^2 n)$). Finally, extensive experiments demonstrate that DCsam outperforms the state-of-the-art methods by a large margin in both distortion and running time.

ACKNOWLEDGMENTS

We are very grateful to the anonymous reviewers for their constructive and insightful comments. Yuxiang Zeng and Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2018AAA0101100, the Hong Kong RGC GRF Project 16202218, CRF Project C6030-18G, C1031-18G, C5026-18G, AOE Project AoE/E-603/18, RIF Project R6020-19, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, HKUST-NAVER/LINE AI Lab, Didi-HKUST joint research lab, HKUST-Webank joint research lab grants and HKUST Global Strategic Partnership Fund (2021 SJTU-HKUST). Yongxin Tong's work is partially supported by the National Key Research and Development Program of China under Grant No. 2018AAA0101100, the National Science Foundation of China (NSFC) under Grant Nos. U21A20516, 61822201, U1811463 and 62076017, and the State Key Laboratory of Software Development Environment Open Funding No. SKLSDE-2020ZX-07.

REFERENCES

[1] 2021. *Didi Chuxing*. Retrieved Oct 21, 2021 from https://www.didiglobal.com/

[2] 2021. *Faster and Better Solution to Embed L_p Metrics by Tree Metrics (Full Paper)*. Technical Report. https://github.com/yzengal/SIGMOD22_Git

[3] 2021. *Foursquare*. Retrieved Oct 21, 2021 from https://foursquare.com/

[4] 2021. *UCAR Inc.* Retrieved Oct 21, 2021 from https://www.10101111.com/

[5] Ahmed Abdelkader, Sunil Arya, Guilherme Dias da Fonseca, and David M. Mount. 2019. Approximate Nearest Neighbor Searching with Non-Euclidean and Weighted Distances. In *SODA*. 355–372.

[6] Ittai Abraham, Yair Bartal, and Ofer Neiman. 2006. Advances in metric embedding theory. In *STOC*. 271–286.

[7] Barbara M. Anthony and Christine Chung. 2014. Online bottleneck matching. *Journal of Combinatorial Optimization* 27, 1 (2014), 100–114.

[8] Lars Arge, Mark de Berg, Herman J. Haverkort, and Ke Yi. 2008. The priority R-tree: A practically efficient and worst-case optimal R-tree. *ACM Trans. Database Syst.* 4, 1 (2008), 9:1–9:30.

[9] Sunil Arya and David M. Mount. 1993. Approximate Nearest Neighbor Queries in Fixed Dimensions. In *SODA*. 271–280.

[10] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. 1998. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *Journal of the ACM* 45, 6 (1998), 891–923.

[11] V Asha, Nagappa U Bhajantri, and P Nagabhushan. 2011. GLCM–based chi–square histogram distance for automatic detection of defects on patterned textures. *International Journal of Computational Vision and Robotics* 2, 4 (2011), 302–313.

[12] Yossi Azar and Noam Touitou. 2019. General Framework for Metric Optimization Problems with Delay or with Deadlines. In *FOCS*. 11–22.

[13] Arturs Backurs, Piotr Indyk, Krzysztof Onak, Baruch Schieber, Ali Vakilian, and Tal Wagner. 2019. Scalable Fair Clustering. In *ICML*. 405–413.

[14] Yair Bartal. 1996. Probabilistic Approximations of Metric Spaces and Its Algorithmic Applications. In *FOCS*. 184–193.

[15] Yair Bartal. 1998. On Approximating Arbitrary Metrices by Tree Metrics. In *STOC*. 161–168.

[16] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*. 322–331.

[17] Guy E. Blelloch, Anupam Gupta, and Kanat Tangwongsan. 2012. Parallel probabilistic tree embeddings, k-median, and buy-at-bulk network design. In *SPAA*. 205–213.

[18] Zhao Chen, Peng Cheng, Yuxiang Zeng, and Lei Chen. 2019. Minimizing Maximum Delay of Task Assignment in Spatial Crowdsourcing. In *ICDE*. 1454–1465.

[19] Christian Coester and Elias Koutsoupias. 2019. The online k-taxi problem. In *STOC*. 1136–1147.

[20] Marek Cygan, Artur Czumaj, Marcin Mucha, and Piotr Sankowski. 2018. Online Facility Location with Deletions. In *ESA*. 21:1–21:15.

[21] Beman Dawes, David Abrahams, and Rene Rivera. 2021. *Boost C++ Libraries*. Retrieved Oct 21, 2021 from https://www.boost.org/

[22] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. 2008. *Computational geometry: algorithms and applications, 3rd Edition*. Springer.

[23] Didi Chuxing. 2021. *GAIA Initiative*. Retrieved Oct 21, 2021 from http://gaia.didichuxing.com

[24] Yunus Esencayi, Marco Gaboardi, Shi Li, and Di Wang. 2019. Facility Location Problem in Differential Privacy Model Revisited. In *NeurIPS*. 8489–8498.

[25] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2003. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*. 448–455.

[26] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. 2004. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences* 69, 3 (2004), 485–497.

[27] Raphael A. Finkel and Jon Louis Bentley. 1974. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica* 4 (1974), 1–9.

[28] Johannes Fischer and Volker Heun. 2011. Space-Efficient Preprocessing Schemes for Range Minimum Queries on Static Arrays. *SIAM J. Comput.* 40, 2 (2011), 465–492.

[29] Stephan Friedrichs and Christoph Lenzen. 2018. Parallel Metric Tree Embedding Based on an Algebraic View on Moore-Bellman-Ford. *Journal of the ACM* 65, 6 (2018), 43:1–43:55.

[30] Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. 1984. Scaling and Related Techniques for Geometry Problems. In *STOC*. 135–143.

[31] Junhao Gan and Yufei Tao. 2015. DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In *SIGMOD*. 519–530.

[32] Jie Gao, Leonidas J. Guibas, Nikola Milosavljevic, and Dengpan Zhou. 2009. Distributed resource management and matching in sensor networks. In *IPSN*. 97–108.

[33] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. 47–57.

[34] Sariel Har-Peled. 2001. A Replacement for Voronoi Diagrams of Near Linear Size. In *FOCS*. 94–103.

[35] Sariel Har-Peled. 2011. *Geometric approximation algorithms*. Number 173. American Mathematical Society.

[36] Dov Harel and Robert Endre Tarjan. 1984. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J. Comput.* 13, 2 (1984), 338–355.

[37] Qiang Huang, Jianlin Feng, Qiong Fang, Wilfred Ng, and Wei Wang. 2017. Query-aware locality-sensitive hashing scheme for L_p norm. *The VLDB Journal* 26, 5 (2017), 683–708.

[38] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *PVLDB* 9, 1 (2015), 1–12.

[39] Piotr Indyk. 2001. Algorithmic Applications of Low-Distortion Geometric Embeddings. In *FOCS*. 10–33.

[40] Bala Kalyanasundaram and Kirk Pruhs. 1993. Online Weighted Matching. *Journal of Algorithms* 14, 3 (1993), 478–488.

[41] Goran Konjevod, R. Ravi, and F. Sibel Salman. 2001. On approximating planar metrics by tree metrics. *Information Processing Letters* 80, 4 (2001), 213–219.

[42] Jian Li, Amol Deshpande, and Samir Khuller. 2009. Minimizing Communication Cost in Distributed Multi-query Processing. In *ICDE*. 772–783.

[43] Adam Meyerson, Akash Nanavati, and Laura J. Poplawski. 2006. Randomized online algorithms for minimum metric bipartite matching. In *SODA*. 954–959.

[44] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[45] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.

[46] David M. Mount. 2019. New Directions in Approximate Nearest-Neighbor Searching. In *CALDAM*. 1–15.

[47] Marius Muja and David G. Lowe. 2009. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*. 331–340.

[48] Marius Muja and David G. Lowe. 2021. *FLANN: Fast Library for Approximate Nearest Neighbors*. Retrieved Oct 21, 2021 from https://github.com/flann-lib/flann

[49] Ofir Pele and Michael Werman. 2010. The quadratic-chi histogram distance family. In *ECCV*. 749–762.

[50] Jianzhong Qi, Guanli Liu, Christian S. Jensen, and Lars Kulik. 2020. Effectively Learning Spatial Indices. *PVLDB* 13, 11 (2020), 2341–2354.

[51] Jianzhong Qi, Yufei Tao, Yanchuan Chang, and Rui Zhang. 2020. Packing R-trees with Space-filling Curves: Theoretical Optimality, Empirical Efficiency, and Bulk-loading Parallelizability. *ACM Trans. Database Syst.* 45, 3 (2020), 14:1–14:47.

[52] Hanan Samet. 2006. *Foundations of multidimensional and metric data structures*. Academic Press.

[53] James S Tanton. 2005. *Encyclopedia of mathematics*. Infobase Publishing.

[54] Qian Tao, Yongxin Tong, Zimu Zhou, Yexuan Shi, Lei Chen, and Ke Xu. 2020. Differentially Private Online Task Assignment in Spatial Crowdsourcing: A Tree-based Approach. In *ICDE*. 517–528.

[55] Yongxin Tong, Jieying She, Bolin Ding, Lei Chen, Tianyu Wo, and Ke Xu. 2016. Online Minimum Matching in Real-Time Spatial Data: Experiments and Analysis. *PVLDB* 9, 12 (2016), 1053–1064.

[56] Yongxin Tong, Yuxiang Zeng, Bolin Ding, Libin Wang, and Lei Chen. 2021. Two-Sided Online Micro-Task Assignment in Spatial Crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering* 33, 5 (2021), 2295–2309.

[57] Yongxin Tong, Zimu Zhou, Yuxiang Zeng, Lei Chen, and Cyrus Shahabi. 2020. Spatial crowdsourcing: a survey. *The VLDB Journal* 29, 1 (2020), 217–250.

[58] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. 2017. *Handbook of discrete and computational geometry*. Chapman and Hall/CRC.

[59] Kilian Weinberger. 2021. *Lecture 2: K-Nearest Neighbors (Curse of Dimensionality)*. Retrieved Oct 21, 2021 from https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html

[60] David P Williamson and David B Shmoys. 2011. *The design of approximation algorithms*. Cambridge university press.

[61] Raymond Chi-Wing Wong, Yufei Tao, Ada Wai-Chee Fu, and Xiaokui Xiao. 2007. On Efficient Spatial Matching. In *VLDB*. 579–590.

[62] Dingqi Yang, Daqing Zhang, Vincent. W. Zheng, and Zhiyong Yu. 2015. Modeling User Activity Preference by Leveraging User Spatial Temporal Characteristics in LBSNs. *IEEE Trans. Syst. Man Cybern. Syst.* 45, 1 (2015), 129–142.

[63] Yuxiang Zeng, Yongxin Tong, and Lei Chen. 2019. Last-Mile Delivery Made Practical: An Efficient Route Planning Framework with Theoretical Guarantees. *PVLDB* 13, 3 (2019), 320–333.

[64] Yuxiang Zeng, Yongxin Tong, and Lei Chen. 2021. HST+: An Efficient Index for Embedding Arbitrary Metric Spaces. In *ICDE*. 648–659.

[65] Yuxiang Zeng, Yongxin Tong, Lei Chen, and Zimu Zhou. 2018. Latency-Oriented Task Completion via Spatial Crowdsourcing. In *ICDE*. 317–328.

[66] Yuxiang Zeng, Yongxin Tong, Yuguang Song, and Lei Chen. 2020. The Simpler The Better: An Indexing Approach for Shared-Route Planning Queries. *PVLDB* 13, 13 (2020), 3517–3530.

[67] Boming Zhao, Pan Xu, Yexuan Shi, Yongxin Tong, Zimu Zhou, and Yuxiang Zeng. 2019. Preference-Aware Task Assignment in On-Demand Taxi Dispatching: An Online Stable Matching Approach. In *AAAI*. 2245–2252.