Comparison-Based Nearest Neighbor Search

Siavash Haghiri   **Debarghya Ghoshdastidar**   **Ulrike von Luxburg**
Department of Computer Science, University of Tübingen, Germany
{haghiri,ghoshdas,luxburg}@informatik.uni-tuebingen.de

Abstract

We consider machine learning in a comparison-based setting where we are given a set of points in a metric space, but we have no access to the actual distances between the points. Instead, we can only ask an oracle whether the distance between two points i and j is smaller than the distance between the points i and k. We are concerned with data structures and algorithms to find nearest neighbors based on such comparisons. We focus on a simple yet effective algorithm that recursively splits the space by first selecting two random pivot points and then assigning all other points to the closer of the two (comparison tree). We prove that if the metric space satisfies certain expansion conditions, then with high probability the height of the comparison tree is logarithmic in the number of points, leading to efficient search performance. We also provide an upper bound for the failure probability to return the true nearest neighbor. Experiments show that the comparison tree is competitive with algorithms that have access to the actual distance values, and needs less triplet comparisons than other competitors.

1 INTRODUCTION

In many machine learning problems, data is given in the form of points and similarity or distance values between these points. Recently, comparison-based settings have become increasingly popular (Schultz and Joachims, 2003, Agarwal et al., 2007, van der Maaten and Weinberger, 2012, Amid and Ukkonen, 2015, Ukkonen et al.,

2015, Balcan et al., 2016). Here the assumption is that points come from some metric space $(\mathcal{X}, d)$, but the metric d is unknown. We only have indirect access to the metric in the form of **triplet comparisons**: for a particular triplet of points (x, y, z) we can ask whether

$$d(x, y) \leq d(x, z) \tag{1}$$

is true or not. Such settings are popular in the crowd sourcing literature (Tamuz et al., 2011, Heikinheimo and Ukkonen, 2013, Ukkonen et al., 2015).

Assume that we are given a set of objects together with the answers to triplet comparisons for some (or even all) triplets of objects, and our task is to solve machine learning problems such as clustering or classification. There are two possible strategies we could pursue. The first is to construct an ordinal embedding of the objects into the Euclidean space $\mathbb{R}^d$, that is an embedding such that the answers to the triplet comparisons are preserved (Agarwal et al., 2007, van der Maaten and Weinberger, 2012, Terada and von Luxburg, 2014, Kleindessner and von Luxburg, 2014, Arias-Castro, 2015, Amid and Ukkonen, 2015, Jain et al., 2016). Subsequently, the machine learning problem can be solved by standard methods in Euclidean spaces. The second approach would be to use the triplet comparisons to find the k nearest neighbors (kNN) of the points directly. We could then use kNN classification, clustering methods on the kNN graph, etc. In our paper, we focus on the second approach, and in particular on the question how many triplet comparisons are needed in order to find the kNNs of data points.

Many algorithms for exact or approximate nearest neighbor search use data structures based on space partitioning. Most popular is the setting where points live in the Euclidean space. KD-Tree (Bentley, 1975), PA-Tree (McNames, 2001), Spill-Tree (Liu et al., 2004), RP-Tree (Dasgupta and Freund, 2008), and MM-Tree (Ram and Gray, 2013) are among the algorithms in Euclidean setting. In addition, the setting where data points only lie on a more abstract metric space has been explored considerably. Metric Skip List (Karger and Ruhl, 2002), Navigating Net (Krauthgamer and

Lee, 2004), and Cover-Tree (Beygelzimer et al., 2006) are some of the methods in this category. However, in nearly all these algorithms, we either need to know the vector representation of the points or the distance values between the points. Few exceptional methods exist that even work in the setting where we only have access to triplet comparisons (Goyal et al., 2008, Lifshits and Zhang, 2009, Tschopp et al., 2011, Houle and Nett, 2015).

To our taste, the most appealing comparison-based data structure is what we call the comparison tree. The structure has been introduced as "metric tree based on a generalized hyperplane decomposition" in Uhlmann (1991). For the sake of simplicity we refer to it as the comparison tree. To partition the space into smaller subsets, two pivot points are picked uniformly at random from the given dataset. The space is then separated into two "generalized half-spaces", namely the two sets of points that are closer to the two respective pivots. This is done recursively until the resulting subsets become smaller than a given size. Once this tree structure has been constructed, the nearest neighbor search proceeds by comparing the query point to the two pivot points. On an intuitive level, comparison trees are promising: (i) the splits seem to adapt to the geometry of the data, (ii) it seems that the splits are not extremely unbalanced, (iii) there is "enough randomness" in the construction. Unfortunately, none of these intuitions has been proved or formally investigated yet.

The first contribution of our paper is to analyze the performance of comparison trees in general metric spaces. Under certain assumptions on the expansion rates of the space, we prove that comparison trees are nicely balanced and their height is of the order $\Theta(\log n)$. This means that to construct a comparison tree, we only need of the order $n \log n$ triplet comparisons. Moreover, we can bound the probability that the nearest neighbor algorithm finds the correct nearest neighbor. Our second contribution consists of simulations that compare the behavior of comparison trees to standard data structures in Euclidean spaces and other comparison-based data structures on metric spaces. We find that the comparison tree performs surprisingly well even if compared to competitors that access vector representations in Euclidean spaces (KD-Tree, RP-Tree and PA-Tree), and favorably in comparison to one of the recent comparison-based algorithms proposed in Houle and Nett (2015).

2 COMPARISON TREE

Let S be a set of n points in some metric space $(\mathcal{X}, d)$. **To construct a comparison tree** on S, we proceed as follows (see Algorithm 1 below). The root of the tree

$T.root$ consists of the whole set S, and each of the subsequent nodes represents a partition of the set S. In each step of the tree construction, the elements of the current node are partitioned into two disjoint sets, which in turn are the root nodes for the left and right subtrees denoted by $T.leftchild$ and $T.rightchild$. More concretely, to form a partition of the current node of the tree, we randomly choose two pivot elements among its current elements, denoted by $T.leftpivot$ and $T.rightpivot$. Then we group the remaining elements according to whether they are closer to the left or right pivot. Observe that this step does not require actual distance values, but just triplet comparisons. Then we recurse, until the current set of elements has at most size n_0 for some pre-specified n_0.

Algorithm 1 $CompTree(S, n_0)$: **Comparison tree construction**

Input: $S \subseteq \mathcal{X}$, and maximum leaf size n_0
Output: Comparison tree T
1: $T.root \leftarrow S$
2: **if** $|S| > n_0$ **then**
3:     Uniformly sample distinct points $x_1, x_2 \in S$
4:     $S_1 \leftarrow \{x \in S : d(x, x_1) \leq d(x, x_2)\}$
5:     $T.leftpivot \leftarrow x_1, \ T.rightpivot \leftarrow x_2$
6:     $T.leftchild \leftarrow CompTree(S_1, n_0)$
7:     $T.rightchild \leftarrow CompTree(S \backslash S_1, n_0)$
8: **end if**
9: **return** T

The computational complexity of the tree construction is governed by the number of triplet comparisons required in the procedure, which depends on the height of tree. In the next section, we show that under certain growth assumptions on the metric, the comparison tree has height $h = O(\log n)$ with high probability.

To use the comparison tree for (approximate) nearest neighbor search, we employ the obvious greedy procedure (sometimes called the defeatist search; see Algorithm 2): starting at the root, we compare the query element q to the current two pivot elements, and using a triplet comparison we decide whether to proceed in the left or right branch. When we reach a leaf, we carry out an exhaustive search among all its elements to determine the one that is closest to the given query. This step requires n_0 triplet comparisons. Overall, the nearest neighbor search in a comparison tree of height h requires at most $h + n_0$ triplet comparisons, which boils down to $O(\log n)$.

3 THEORETICAL ANALYSIS

In this section, we analyze the complexity and performance of the tree construction as well as the nearest

Algorithm 2 NN(q, T): **Nearest neighbor search**

Input: Comparison tree T, and query q.
Output: $\hat{x}_q$ =approximate nearest neighbor of q in S
1: **if** $T.leftchild \neq null$ **then**
2:     **if** $d(q, T.leftpivot) \leq d(q, T.rightpivot)$ **then**
3:         $\hat{x}_q \leftarrow$ NN$(q, T.leftchild)$
4:     **else** $\hat{x}_q \leftarrow$ NN$(q, T.rightchild)$
5:     **end if**
6: **else**   choose $\hat{x}_q$ s.t. $d(q, \hat{x}_q) \leq d(q, x) \; \forall x \in T.root$
7: **end if**
8: **return** $\hat{x}_q$

---

neighbor search. We first provide a high probability bound on the height of the comparison tree, which in turn bounds the number of triplet comparisons required for both tree construction and nearest neighbor search. In addition, we derive an upper bound on the probability that the above approach fails to return the exact nearest neighbor of a given query.

3.1 Expansion Conditions

Finding the nearest neighbor for a query point q in a general metric space $(\mathcal{X}, d)$ can require up to $\Omega(n)$ comparisons in the worst case, using any data structure built on the given set S (Beygelzimer et al., 2006). Hence, most similarity search methods are analyzed under the natural assumption that the metric d is growth-restricted (Karger and Ruhl, 2002, Krauthgamer and Lee, 2004). Informally, such restrictions imply that the volume of a closed ball in the space does not increase drastically when its radius is increased by a certain factor. Various related notions are used to characterize the growth rate of such metrics, for instance Assouad dimension (Assouad, 1979), doubling dimension (Gupta et al., 2003), homogeneity (Luukkainen and Saksman, 1998), and expansion rate (Karger and Ruhl, 2002) among others. The analysis of most tree based search methods in the Euclidean setting requires only a doubling property of the metric (Dasgupta and Sinha, 2015). Such results hold for metric spaces with finite doubling dimension or Assouad dimension. When dealing with general metric spaces, it is more convenient to consider the expansion rate, which is an empirical variant of the doubling property defined for a given finite set $S \subseteq \mathcal{X}$. Typically, the analysis of data dependent tree constructions requires even stronger restrictions.

In this work, we use a slightly weaker variant of the strong expansion rate condition used in Ram and Gray (2013). Intuitively, we need bounds on the expansion rate for all the finite point sets that can possibly occur in the non-leaf nodes of the comparison tree.

Let $(\mathcal{X}, d)$ be a metric space, $S \subseteq \mathcal{X}$ and $n_0 < |S|$. We construct a collection $\mathcal{C}_S \subseteq 2^S$ as follows:

1. $S \in \mathcal{C}_S$,
2. If $A \in 2^S$ and there exist $x, y \in S$ such that $d(z, x) \leq d(z, y) \; \forall z \in A$, then $A \in \mathcal{C}_S$, $S \backslash A \in \mathcal{C}_S$,
3. If $A, B \in \mathcal{C}_S$, then $A \cap B \in \mathcal{C}_S$.

We finally remove all $A \in \mathcal{C}_S$ with size $|A| \leq n_0$. Observe that $\mathcal{C}_S$ characterizes the collection of all possible non-leaf nodes of the tree[1]. We define the **strong expansion rate** of S as the smallest $\tilde{c} \geq 1$ such that

$$|B(x, 2r) \cap A| \leq \tilde{c}|B(x, r) \cap A| \qquad (2)$$

for all $A \in \mathcal{C}_S$, $x \in A$ and $r > 0$, where $B(x, r)$ is the closed ball in $(\mathcal{X}, d)$ centered at x with radius r.

Inequality (2) states that every $A \in \mathcal{C}_S$ has an expansion rate at most $\tilde{c}$ similar to the definition in Karger and Ruhl (2002). This requirement for all $A \in \mathcal{C}_S$ is strong, but seems unavoidable due to the data dependent, yet random, tree construction.

3.2 Main Results

The following theorem provides an upper bound on the height of the comparison tree.

Theorem 1 (Height of a comparison tree). *Consider a set S of size n in a metric space that satisfies the strong expansion rate condition with constant $\tilde{c}$. Fix some $n_0 \in \mathbb{N}$. Then for any $\varepsilon > 0$, with probability $1 - \varepsilon$, the comparison tree construction algorithm returns a tree with height smaller than*

$$h^* = 3\log\left(\frac{e}{\varepsilon}\right) + 96\tilde{c}^2 \log\left(\frac{n}{n_0}\right) . \qquad (3)$$

We prove the theorem later in the section. Theorem 1 implies that if the expansion rate $\tilde{c} = O(1)$, then the height of the randomly constructed tree tends to be bounded by $O(\log n)$. In particular, one can expect this to happen if the set of points is sampled from an "evenly" spread distribution in a growth-restricted space $\mathcal{X}$. As a consequence of Theorem 1, one can comment on the number of triplet comparisons required for tree construction and nearest neighbor search. We state this in the following corollary.

Corollary 2 (Number of triplet comparisons). *For $\varepsilon > 0$, let h^* be defined as in (3). Then with probability $1 - \varepsilon$, the comparison tree construction algorithm requires at most nh^* triplet comparisons to construct the comparison tree. Furthermore, for any $q \in \mathcal{X}$, with probability $1 - \varepsilon$, nearest neighbor search algorithm uses at most $h^* + n_0$ triplet comparisons to find an approximate nearest neighbor of q.*

---

[1]Technically, $\mathcal{C}_S$ is a subset of the algebra generated by the "generalized half spaces" of the induced space (S, d).

The proof is a simple consequence of Theorem 1.

Other applicable methods in our setting made various assumptions on the dataset, thus the upper bound on the required number of triplets is hardly comparable with them. If we neglect this fact and only compare the dependency to n, we can summarize the asymptotic bounds on the required number of comparisons in Table 1. We ignore all dependencies on the constants describing the geometric properties of the space (such as doubling, expansion, or disorder constants).

Table 1: Theoretical comparison with existing methods

Method	Construction	Query
Comparison Tree	$n \log n$	$\log n$
(Goyal et al., 2008)	$n^2 \log n$	$\log n$
(Lifshits and Zhang, 2009)	$n \log^2 n$	$\log n$
(Tschopp et al., 2011)	$n \log^2 n$	$\log^2 n$
(Houle and Nett, 2015)	$n \log^3 n$	$\log^3 n$

While the above discussion sheds light on the required number of triplet comparisons, it still leaves one wondering about the quality of the nearest neighbor obtained from the comparison tree. In the following result, we show that under certain conditions on the behavior of the metric in a neighborhood of a given query q, the search method succeeds in finding the true nearest neighbor of q. We use $x_q \in S$ to denote the true nearest neighbor of q, while $\hat{x}_q$ is the element returned by the nearest neighbor search. We write $B(x, r)$ and $B^\circ(x, r)$ to denote closed and open balls, respectively.

Theorem 3 (Exact nearest neighbor). *Given $S \subseteq \mathcal{X}$ and $q \in \mathcal{X}$. If there exist constants $C > 0$ and $\alpha \in (0, 1]$ such that for every $A \in \mathcal{C}_S$ containing x_q, and for all $x \in A \backslash \{x_q\}$,*

$$\left| B\big(q, d(q, x) + 2d(q, x_q)\big) \cap A \right|$$
$$\leq \left| B^\circ\big(q, d(q, x)\big) \cap A \right| + C|A|^{1-\alpha} , \quad (4)$$

then

$$\mathsf{P}\left(\hat{x}_q \neq x_q\right) \leq \frac{360 C \tilde{c}^2}{\alpha} n_0^{-\alpha} , \quad (5)$$

where the probability is with respect to the random construction of the tree.

The condition on q implies that there are not many points that have the same distance to q as the nearest neighbor. Under the local restrictions defined in (4) on the query q, the error bound (5) states that one can achieve an arbitrarily small error probability if n_0 is chosen large enough, depending on the strong expansion rate $\tilde{c}$.

Remark 4. *The assumption in (4) can also be substituted by alternative conditions. For instance, the error bound in (5) holds (up to constants) if there exists*

$D > 1$ *such that*

$$|B(q, \lambda r) \cap A| \leq \lambda^D |B(q, r) \cap A| \quad (6)$$

for all $A \in \mathcal{C}_S$, $\lambda > 1$, and $r > d(q, x_q)$. One can see the inherent resemblance of (6) to the notion of strong expansion rate (2).

We remark again on the required conditions. The notion of strong expansion rate, though used in Ram and Gray (2013), is stronger than standard conditions used in many works (Dasgupta and Sinha, 2015, Karger and Ruhl, 2002). Our main reason for resorting to this notion is because of the data dependent random splits used in comparison tree construction. While projection-based methods also use random hyperplanes for splitting each node, such hyperplanes are independent of the given set, making the analysis simpler (Dasgupta and Sinha, 2015). On the other hand, prior works in non-Euclidean setting construct data structures that naturally adhere to the structure of the metric balls (Karger and Ruhl, 2002). Unlike both these works, in the present setting, one cannot guarantee that a condition defined on the whole set will also hold for each of the partitions obtained during splits. Hence, the condition of strong expansion rate has been used in our analysis. The additional assumption on q seems essential since one can construct trivial examples where the nearest neighbor search is quite likely to fail.

3.2.1 Proof of Theorem 1

We now prove Theorem 1 using two lemmas.

Lemma 5 (Probability of unbalanced split). *For any $A \in \mathcal{C}_S$ and $\delta \in (0, 1)$, the probability that the random split in the comparison tree construction algorithm creates a child of A with less than $\delta|A|$ elements is at most $4\tilde{c}^2 \delta$.*

Thus, each split in the tree is reasonably balanced, and hence, it is likely that the size of the nodes decays rapidly with their depth. This fact is formalized below.

Lemma 6 (Maximum node size at depth h). *Let $A \in \mathcal{C}_S$ be a node at depth h of the tree. If $4\tilde{c}^2 \delta \leq 1$, then the probability that A has more than m elements is at most $(8\tilde{c}^2 \delta)^{h-1} (n/m)^{(1/\delta) \log(1/4\tilde{c}^2 \delta)}$.*

We finish the proof of Theorem 1 by observing that the height of the tree is greater than h only if there is a node at depth h of size greater than n_0. By taking a union over all possible 2^h nodes at depth h, one can see that the probability of this event is at most $2(16\tilde{c}^2 \delta)^{h-1} (n/n_0)^{(1/\delta) \log(1/4\tilde{c}^2 \delta)}$. This probability is less than ε if we fix $\delta = 1/32\tilde{c}^2$, and

$$h = \left(1 + 2\log\left(\frac{e}{\varepsilon}\right) + 96\tilde{c}^2 \log\left(\frac{n}{n_0}\right)\right) \leq h^* .$$

We now prove the above two lemmas.

Proof of Lemma 5. Let $|A| = m$, and $\mathbf{1}(\cdot)$ denote the indicator function. Then the probability of splitting A to create a child of size smaller than δm is at most

$$\frac{1}{m(m-1)} \sum_{\substack{x_1 \in A \\ x_2 \in A \setminus \{x_1\}}} \mathbf{1}\Big(\{|\{x \in A : d(x, x_2) \leq d(x, x_1)\}| \leq \delta m\}$$

$$\cup \{|\{x \in A : d(x, x_1) < d(x, x_2)\}| \leq \delta m\}\Big)$$

$$\leq \frac{1}{\binom{m}{2}} \sum_{\substack{x_1 \in A \\ x_2 \in A \setminus \{x_1\}}} \mathbf{1}\Big(|\{x \in A : d(x, x_1) < d(x, x_2)\}| \leq \delta m\Big).$$

Note that the set $\{x \in A : d(x, x_1) < d(x, x_2)\}$ contains the set $B\left(x_1, \frac{1}{4} d(x_1, x_2)\right) \cap A$, where $B(\cdot, \cdot)$ is the closed ball. Thus, one may bound the above probability by the fraction of x_1, x_2 pairs for which this ball contains less than δm elements. Moreover, using the condition of strong expansion rate (2), one has

$$\left| B\left(x_1, d(x_1, x_2)\right) \cap A \right| \leq \tilde{c}^2 \left| B\left(x_1, \frac{1}{4} d(x_1, x_2)\right) \cap A \right|.$$

Thus, one may only count the x_1, x_2 pairs for which $B(x_1, d(x_1, x_2)) \cap A$ contains at most $\tilde{c}^2 \delta m$ elements. Now, for every x_1, if one sorts x_2 in the increasing order of $d(x_1, x_2)$, then the indicator is true only for the first $\tilde{c}^2 \delta m$ of x_2's. Thus, the probability of an unbalanced split is at most $2\tilde{c}^2 \delta m^2 / m(m-1) \leq 4\tilde{c}^2 \delta$. $\qquad\square$

Proof of Lemma 6. We denote the path from the root of the tree to A by $S = A_1 \supset A_2 \supset \ldots \supset A_{h-1} \supset A_h = A$. Let A_j' denote the sibling of A_j for $j \geq 2$. By the Markov inequality, one can write for any $t > 0$,

$$\mathsf{P}(|A| > m) \leq \left(\frac{n}{m}\right)^t \mathsf{E}\left[\frac{|A_h|^t}{|A_1|^t}\right]$$

$$= \left(\frac{n}{m}\right)^t \mathsf{E}\left[\frac{|A_{h-1}|^t}{|A_1|^t} \mathsf{E}\left[\frac{|A_h|^t}{|A_{h-1}|^t}\middle| A_{h-1}\right]\right].$$

One can bound the inner conditional expectation as

$$\mathsf{E}\left[\frac{|A_h|^t}{|A_{h-1}|^t}\middle| A_{h-1}\right]$$

$$= \mathsf{E}\left[\frac{|A_h|^t}{|A_{h-1}|^t}\mathbf{1}(|A_h| > (1-\delta)|A_{h-1}|)\middle| A_{h-1}\right]$$

$$+ \mathsf{E}\left[\frac{|A_h|^t}{|A_{h-1}|^t}\mathbf{1}(|A_h| \leq (1-\delta)|A_{h-1}|)\middle| A_{h-1}\right]$$

$$\leq \mathsf{P}(|A_h'| \leq \delta|A_{h-1}|) + (1-\delta)^t,$$

where the inequality follows by replacing the ratio by 1 in the first expectation, and $|A_h|$ by its upper bound in the second one. Due to Lemma 5, one can see that this bound is at most $\left(4\tilde{c}^2\delta + (1-\delta)^t\right)$. For $t = (1/\delta)\log(1/4\tilde{c}^2\delta)$, one can use the fact that

$(1/\delta)\log(1/(1-\delta)) \geq 1$ to show that the above expectation is at most $8\tilde{c}^2\delta$. Subsequently, we use the same technique of conditioning with every $A_j, j = 2, \ldots, h-2$ to obtain

$$\mathsf{P}(|A| > m) \leq \left(\frac{n}{m}\right)^{\frac{1}{\delta}\log(\frac{1}{4\tilde{c}^2\delta})} (8\tilde{c}^2\delta)^{h-1}.$$

$\qquad\square$

3.2.2 Proof of Theorem 3

The nearest neighbor search for a query point q is done by traversing the tree from the root to one of the leaves. Let us denote the visited path by $S = A_1 \supset A_2 \supset \ldots \supset A_{k-1} \supset A_k$, where A_k is the leaf node containing $\hat{x}_q$. We assume that $q \notin S$, as otherwise the nearest neighbor search algorithm returns the query. By simple reasoning, it follows that $\hat{x}_q \neq x_q$ only if $x_q \notin A_k$, which happens if there is $l \in \{1, 2, \ldots, k-1\}$ such that $x_q \in A_l \setminus A_{l+1}$. Hence,

$$\mathsf{P}(\hat{x}_q \neq x_q) = \mathsf{P}\left(\bigcup_{l=1}^{k-1}\{x_q \in A_l, x_q \notin A_{l+1}\}\right)$$

$$\leq \sum_{l=1}^{k-1} \mathsf{P}(x_q \notin A_{l+1}|x_q \in A_l)\mathsf{P}(x_q \in A_l)$$

$$\leq \sum_{l=1}^{k-1} \mathsf{P}\left(x_q \notin A_{l+1}\,|\,x_q \in A_l, |A_l| \geq m_l\right)\mathsf{P}\left(|A_l| \geq m_l\right) +$$

$$\mathsf{P}\left(x_q \notin A_{l+1}\,|\,x_q \in A_l, |A_l| < m_l\right)\mathsf{P}\left(|A_l| < m_l\right) \qquad (7)$$

where $m_l = n_0/(1-\gamma)^{k-1-l}$ for some $\gamma \in (0, 1)$. The first inequality is due to union bound, while the second one uses $\mathsf{P}(x_q \in A_l) \leq 1$ and further decomposes based on $|A_l|$.

Lemma 7 (Probability of missing nearest neighbor in one branch). *Under the condition on q stated in Theorem 3, for any $l = 1, 2, \ldots, k-1$,*

$$\mathsf{P}(x_q \notin A_{l+1}|A_l, x_q \in A_l) \leq C|A_l|^{-\alpha}.$$

Note that for the two conditional probabilities in (7), $|A_l|$ is at least m_l and n_0, respectively. Using Lemma 7, one can bound these probabilities. To obtain a bound on $\mathsf{P}(|A_l| < m_l)$, we follow Lemma 6.

Observe that Lemma 6 implies that after repeated splits, it is less likely that the ratio of the final node to the root node will be large. In the present context, we know that $|A_{k-1}| \geq n_0$. Thus, for any $l < k-1$, the bound in Lemma 6 can be used to argue that $|A_{k-1}|/|A_l|$ cannot be large. Formally,

$$\mathsf{P}\left(|A_l| < m_l\right) = \mathsf{P}\left(\frac{|A_{k-1}|}{|A_l|} > \frac{n_0}{m_l}\right)$$

$$\leq (8\tilde{c}^2\delta)^{k-1-l}\left(\frac{1}{(1-\gamma)^{k-1-l}}\right)^{\frac{1}{\delta}\log(\frac{1}{4\tilde{c}^2\delta})}$$

$$= \left(\frac{0.25}{(1-\gamma)^{96\tilde{c}^2\log 2}}\right)^{k-1-l},$$

using $\delta = 1/32\tilde{c}^2$ as in Theorem 1. Substituting the above bound in (7) and using Lemma 7, we have

$$P(\hat{x}_q \neq x_q) \leq \sum_{l=1}^{k-1} Cn_0^{-\alpha}(1-\gamma)^{\alpha(k-1-l)}$$

$$+ Cn_0^{-\alpha}\left(\frac{0.25}{(1-\gamma)^{96\tilde{c}^2 \log 2}}\right)^{k-1-l}.$$

Choosing $\gamma = 1 - (0.25)^{1/(\alpha+96\tilde{c}^2 \log 2)}$, one can see that the second term is smaller than the first, and hence,

$$P(\hat{x}_q \neq x_q) \leq \frac{2Cn_0^{-\alpha}}{1-(1-\gamma)^{\alpha}} \leq \frac{2Cn_0^{-\alpha}}{1-(0.25)^{\alpha/67.5\tilde{c}^2}}.$$

From above, we obtain the bound in (5) by using the relation $1-a \leq b(1-a^{1/b})$, which holds for any $a \in (0,1)$ and $b > 1$. This proves Theorem 3.

We end the section with the proof of Lemma 7.

Proof of Lemma 7. Let $|A_l| = m$, and let us order the elements such that $d(q, x_i) \leq d(q, x_j)$ for $i < j$. Since $x_q \in A_l$, we have $x_1 = x_q$. Note that if $x_q \notin A_{l+1}$, then it is certainly not a pivot element. Moreover, if $x_i, x_j \in A_l$ are the pivot elements for $i < j$, then $x_q \notin A_{l+1}$ implies $d(x_q, x_i) \geq d(x_q, x_j)$. The inequality can even be strict depending on which is chosen as the left pivot. Hence, we have

$$P(x_q \notin A_{l+1}|A_l, x_q \in A_l)$$
$$\leq \frac{1}{m(m-1)} \sum_{2 \leq i < j \leq m} \mathbf{1}\big(d(x_q, x_i) \geq d(x_q, x_j)\big).$$

By the triangle inequality,

$$d(x_q, x_i) \leq d(q, x_i) + d(q, x_q) \text{ and}$$
$$d(x_q, x_j) \geq d(q, x_j) - d(q, x_q).$$

Hence, one may count the pairs $i < j$ for which

$$d(q, x_j) \leq d(q, x_i) + 2d(q, x_q).$$

As a consequence, we can write

$$P(x_q \notin A_{l+1}|A_l, x_q \in A_l) \leq \frac{1}{m(m-1)}.$$
$$\sum_{i=2}^{m} \big|\{x \in A : d(q, x_i) \leq d(q, x) \leq d(q, x_i)+2d(q, x_q)\}\big|,$$

where each term in the sum is at most $Cm^{1-\alpha}$ due to the assumption on q. Thus, the claim of the lemma is true. □

4 EXPERIMENTS

4.1 Euclidean Setting

In this section we compare the performance of comparison trees to standard space partitioning trees in Euclidean spaces. Note that the latter have access to the vector representation of the points (and thus also to all pairwise distances), whereas the comparison tree only has access to triplet comparisons. Thus, the purpose of this comparison cannot be to show that the comparison tree "outperforms" the other ones, but to examine whether it is much worse or not. There are numerous tree constructions in Euclidean spaces. Based on the results in Ram and Gray (2013) we decided to compare with KD-Tree (Bentley, 1975), RP-Tree (Dasgupta and Freund, 2008) and PA-Tree (McNames, 2001).

A description of the datasets is presented in Table 2. MNIST is a dataset of hand-written digits (LeCun et al., 1998). Gisette, CoverType and Chess (King-Rook vs. King) are from the UCI repository (Lichman, 2013). Corel is a subset of histograms as it is used in Liu et al. (2004). CoAuth is the collaboration network of Arxiv High Energy Physics from Davis and Hu (2011). We used the largest connected component of the graph and the shortest path as metric. MSC (Boeing/msc10848) is a similar but weighted graph from Davis and Hu (2011)[2].

Table 2: Description of datasets

Dataset	Size	Dimension	Distance
MNIST	70000	784	Euclidean
Gisette	12500	5000	Euclidean
CoverType	50000	53	Euclidean
Corel	19787	44	Euclidean
Chess	28056	6	Mismatch
CoAuth	11204	-	Shortest Path
MSC	10848	-	Shortest Path

We assess the performance of the nearest neighbor search by the leave-one-out method. We report the empirical probability of missing the nearest neighbor: $(1/|S|) \sum_{q \in S} \mathbf{1}(\text{NN}_{alg}(q) \neq \text{NN}(q))$ as a performance measure. Here S is the whole dataset, NN_{alg} denotes the result of nearest neighbor search by the algorithm while the true nearest neighbor is $\text{NN}(q)$.

Figure 1 shows the performance of the comparison tree versus other methods in the Euclidean space. The comparison tree has less error compared to RP-Tree and KD-Tree, and has slightly worse performance comparing with the PA-Tree. However, the differences are not

---

[2]There are negative edge weights in the graph, however we used absolute values of edge weights to have a metric by using shortest path distances.
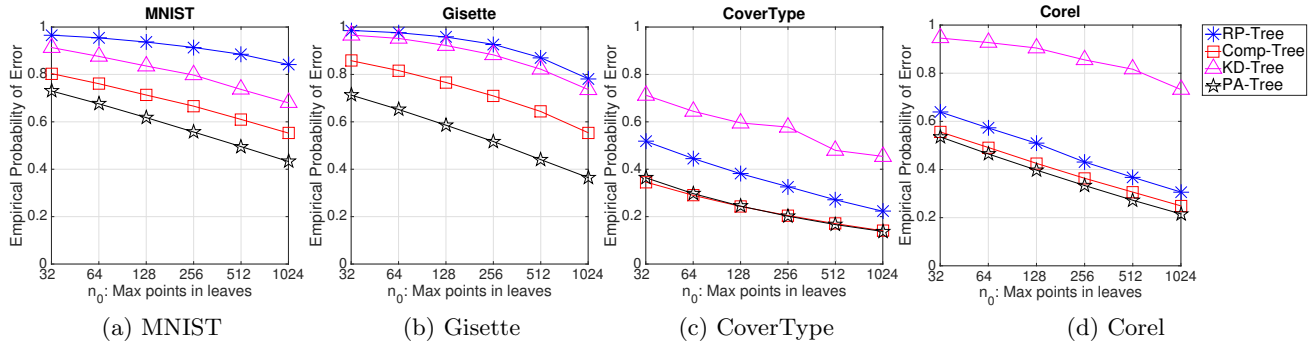
Figure 1: Performance comparison of the comparison tree versus other binary space partitioning trees with respect to the maximum leaf size, n_0.

huge, and we find the behavior of the comparison tree quite satisfactory, given that it receives much less input information than the other methods.

4.2 Comparison-Based Setting

Among the few comparison-based methods cited in the introduction, many are not practical or have already been shown to perform sub-optimally. The most promising competitor to our method is the Rank Cover Tree (RCT) (Houle and Nett, 2015). As the original implementation of the authors was not available, we implemented the method ourselves in Matlab.

We have two objectives when comparing the two comparison-based trees: the number of required triplet questions, and the accuracy they achieve in the nearest neighbor search. While the latter is easy to compare, the former is more of a challenge. It is impossible to construct an RCT with the same number of triplets that the comparison tree requires in construction phase, since the RCT needs orders of magnitude more triplets in construction. Thus, we decided to construct both trees in such a way that the number of triplet comparisons in the *query phase* is matched. We then compare the search performance, but also the number of triplets in the tree construction phase. For the RCT, the performance and the number of comparisons can be balanced by adjusting the coverage parameter ω, see Houle and Nett (2015) Section 4. For comparison trees, n_0 plays a similar role. By varying these two parameters we match the number of comparisons in the query phase.

We randomly choose 1000 data points in each experiment as test set for the query phase and the rest of the dataset for the tree construction. The empirical error defined in previous section is not well-defined for some of datasets in this section. In CoAuth and Chess dataset, many points have more than one nearest neighbor. Thus, we report the average relative distance error defined as $(1/|S|) \sum_{q \in S} \left(\frac{d_{alg}}{d_{NN}} - 1 \right)$ (Liu et al., 2004).

Here d_{alg} denotes the distance of query to the predicted nearest neighbor by the algorithm and d_{NN} denotes the distance of the query to the true nearest neighbor.

Figure 2 shows the performance of the comparison tree compared to the RCT on four datasets from Table 2. The results on the remaining Eulidean datasets are very similar to MNIST, hence we do not present them. We consider different parameter settings and match the average number of triplets used in the query phase. In terms of the relative distance errors, the RCT works slightly better in datasets with low intrinsic dimension, specially when we are provided with more triplets in query phase. However, as the bottom row in Figure 2 shows, to achieve this performance the RCT needs orders of magnitude more triplet comparisons in the tree construction phase. Therefore, if answering triplet comparisons is expensive, then the comparison tree clearly is a good alternative to the RCT.

4.3 Expansion Rate Approximation

In our theoretical analysis, we used the strong expansion condition defined in Equation (2). It is an obvious question to find out how strong these conditions really are and what the corresponding constants in our datasets would be. To this end, we provide a method to estimate the expansion rates for our datasets. We fix a dataset, for each point x we look for the smallest $\tilde{c}$ such that Equation (2) holds for that particular point. We find the smallest value with respect to various radii r. In this way we estimate an empirical pointwise $\tilde{c}(x)$ value for each point. Since the definition depends on the number of points in the dataset, we randomly choose 10000 points from each dataset for these experiments. The distribution of empirical expansion rates $\tilde{c}(x)$ are plotted by box-and-whisker plots in Figure 3.

For our theoretical analysis, we used the smallest possible $\tilde{c}$ for the whole dataset. However the distribution of pointwise values $\tilde{c}(x)$ is a more practical criterion to

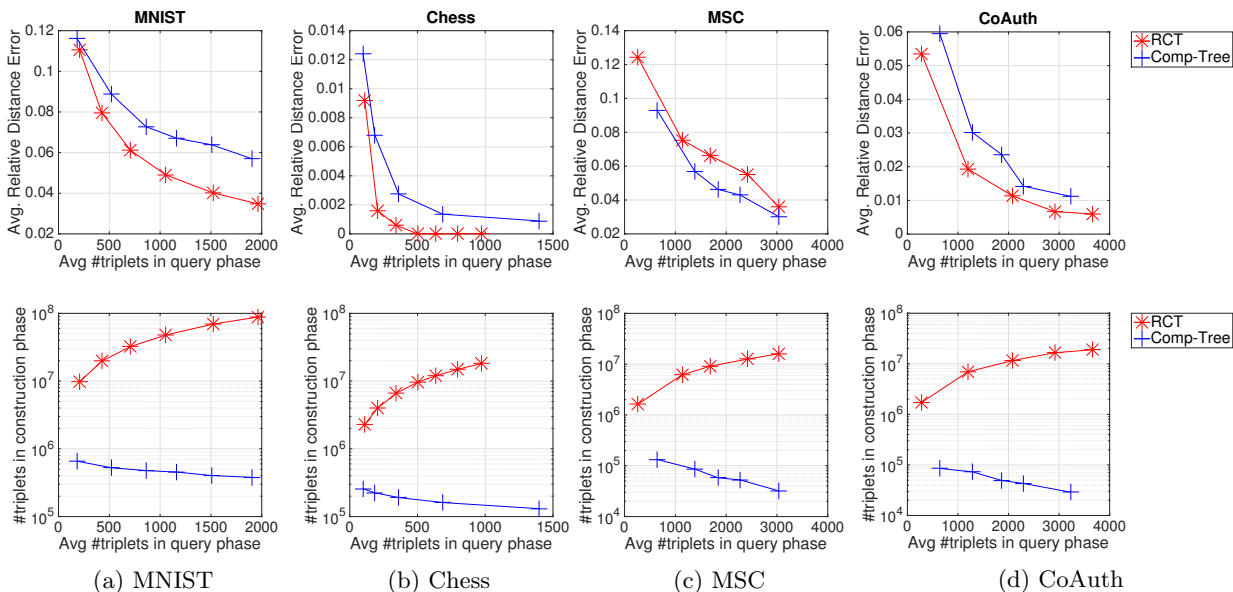(a) MNIST   (b) Chess   (c) MSC   (d) CoAuth

Figure 2: Comparing the performance of the comparison tree versus the RCT with various parameters in the construction phase. The top row corresponds to the average relative distance error with respect to the average number of triplets that each method used in the query phase. The bottom row shows the number of triplets used in the construction phase for both methods. Note that x-axis is the same in both rows.
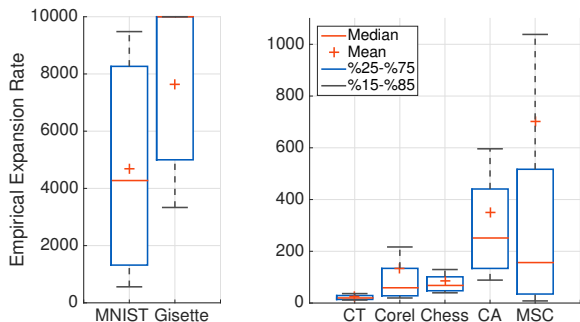

Figure 3: Distribution of empirical expansion rates $\tilde{c}(x)$ estimated for various datasets. Each bar represents the distribution of pointwise expansion rates for the corresponding dataset. CoAuth and CoverType datasets are abbreviated as "CA" and "CT" respectively. Note that the range of the expansion rates is 10 times higher for the first two datasets, thus we plotted them separately.

consider. Except for MNIST and Gisette, these values are reasonably small. Therefore, the values can justify the validity of the assumption on real datasets.

5 CONCLUSIONS

Comparison-based nearest neighbor search is a fundamental ingredient in machine learning algorithms in the comparison-based setting. Because triplet comparisons are expensive, we investigate the query complexity of comparison-based nearest neighbor algorithms. In particular, we study the comparison tree, which leads to a nice and simple, yet adaptive data structure. We prove that under strong conditions on the underlying metric, the comparison tree has logarithmic height, and we can bound the error of the nearest neighbor search. We also show in simulations that comparison trees perform not much worse than Euclidean data structures (albeit using much less information about the data), and perform favorably to other comparison-based methods if we take both the number of triplet comparisons and the nearest neighbor errors into account.

There are still a number of interesting open questions to address. The conditions we use in our analysis are rather strong, and this seems to be the case for all other papers in this area as well. Can they be considerably weakened? Can we prove that our conditions will be satisfied with small constants if we sample point from a nice metric or Euclidean space? Finally, all the above work assumes that a ground truth for the triplet comparisons exists and that the answers to the triplet queries are always correct. It would be interesting to see how the query complexity of the comparison tree increases if the error in the triplets increases.

Acknowledgements

References

S. Agarwal, J. Wills, L. Cayton, G. Lanckriet, D. Kriegman, and S. Belongie. Generalized non-metric multidimensional scaling. In *AISTATS*, pages 11–18, 2007.

E. Amid and A. Ukkonen. Multiview triplet embedding: Learning attributes in multiple maps. In *ICML*, pages 1472–1480, 2015.

E. Arias-Castro. Some theory for ordinal embedding. arXiv:1501.02861, 2015.

P. Assouad. Étude d'une dimension métrique liéea la possibilité de plongements dans rn. *CR Acad. Sci. Paris Sér. AB*, 288(15):A731–A734, 1979.

M.F. Balcan, E. Vitercik, and C. White. Learning combinatorial functions from pairwise comparisons. In *COLT*, pages 310–335, 2016.

J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006.

S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *STOC*, pages 537–546, 2008.

S. Dasgupta and K. Sinha. Randomized partition trees for nearest neighbor search. *Algorithmica*, 72(1):237–263, 2015.

T.A. Davis and Y. Hu. The university of florida sparse matrix collection, 2011. URL http://www.cise.ufl.edu/research/sparse/matrices.

N. Goyal, Y. Lifshits, and H. Schütze. Disorder inequality: a combinatorial approach to nearest neighbor search. In *WSDM*, pages 25–32, 2008.

A. Gupta, R. Krauthgamer, and J.R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *FOCS*, pages 534–543, 2003.

H. Heikinheimo and A. Ukkonen. The crowd-median algorithm. In *HCOMP*, 2013.

M.E. Houle and M. Nett. Rank-based similarity search: Reducing the dimensional dependence. *PAMI*, 37(1):136–150, 2015.

L. Jain, KG. Jamieson, and R. Nowak. Finite sample prediction and recovery bounds for ordinal embedding. In *NIPS*, pages 2703–2711, 2016.

D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *STOC*, pages 741–750, 2002.

M. Kleindessner and U. von Luxburg. Uniqueness of ordinal embedding. In *COLT*, pages 40–67, 2014.

R. Krauthgamer and J.R. Lee. Navigating nets: simple algorithms for proximity search. In *SODA*, pages 798–807, 2004.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Y. Lifshits and S. Zhang. Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design. In *SODA*, pages 318–326, 2009.

T. Liu, A.W. Moore, K. Yang, and A. Gray. An investigation of practical approximate nearest neighbor algorithms. In *NIPS*, pages 825–832, 2004.

J. Luukkainen and E. Saksman. Every complete doubling metric space carries a doubling measure. *Proceedings of the American Mathematical Society*, 126 (2):531–534, 1998.

J. McNames. A fast nearest-neighbor algorithm based on a principal axis search tree. *PAMI*, 23(9):964–976, 2001.

P. Ram and A. Gray. Which space partitioning tree to use for search? In *NIPS*, pages 656–664, 2013.

M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS*, pages 41–48, 2003.

O. Tamuz, C. Liu, S. Belongie, O. Shamir, and A. Kalai. Adaptively learning the crowd kernel. In *ICML*, pages 673–680, 2011.

Y. Terada and U. von Luxburg. Local ordinal embedding. In *ICML*, pages 847–855, 2014.

D. Tschopp, S. Diggavi, P. Delgosha, and S. Mohajer. Randomized algorithms for comparison-based search. In *NIPS*, pages 2231–2239, 2011.

J.K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, 40(4):175–179, 1991.

A. Ukkonen, B. Derakhshan, and H. Heikinheimo. Crowdsourced nonparametric density estimation using relative distances. In *HCOMP*, 2015.

L. van der Maaten and K. Weinberger. Stochastic triplet embedding. In *MLSP*, pages 1–6, 2012. Code available on http://homepage.tudelft.nl/19j49/ste.