Nearest Neighbor Search Under Uncertainty

Blake Mason[1] **Ardhendu Tripathy**[2] **Robert Nowak**[1]

[1]Department of Electrical and Computer Engineering., University of Wisconsin, Madison, Wisconsin, USA
[2]Computer Science Department., Missouri University of Science and Technology, Rolla, Missouri, USA

Abstract

Nearest Neighbor Search (NNS) is a central task in knowledge representation, learning, and reasoning. There is vast literature on efficient algorithms for constructing data structures and performing exact and approximate NNS. This paper studies NNS under Uncertainty (NNSU). Specifically, consider the setting in which an NNS algorithm has access only to a stochastic distance oracle that provides a noisy, unbiased estimate of the distance between any pair of points, rather than the exact distance. This models many situations of practical importance, including NNS based on human similarity judgements, physical measurements, or fast, randomized approximations to exact distances. A naive approach to NNSU could employ any standard NNS algorithm and repeatedly query and average results from the stochastic oracle (to reduce noise) whenever it needs a pairwise distance. The problem is that a sufficient number of repeated queries is unknown in advance; e.g., a point may be distant from all but one other point (crude distance estimates suffice) or it may be close to a large number of other points (accurate estimates are necessary). This paper shows how ideas from cover trees and multi-armed bandits can be leveraged to develop an NNSU algorithm that has optimal dependence on the dataset size and the (unknown) geometry of the dataset.

1 INTRODUCTION

This paper considers Nearest Neighbor Search under Uncertainty (NNSU). To motivate the NNSU problem, consider the following application. Suppose we have a database of N genomic sequences of length L of different species and wish to query the database to find the closest relative of a newly discovered species. Computing the exact distance in $\mathbb{R}^L$ between two sequences requires $O(L)$ operations. Using efficient NNS algorithms, one can construct a data structure in $O(NL\log N)$ time and perform a NN search in $O(L\log N)$. To reduce this complexity, one can randomly subsample the sequences at $\ell \ll L$ locations and compute an unbiased estimate of distance in $O(\ell)$ time and find nearest neighbors in $O(\ell\log(n))$ operations. If an algorithm can manage the added uncertainty of this procedure, this can improve computational complexity greatly.

In other settings, uncertainty is naturally present in the measurements due to the presence of noise. For instance, researchers gathering data to map the topology of the internet, rely on noisy one-way-delay measurements to infer distance between servers Eriksson et al. [2010] and, detecting the closest server to a given host can be challenging due to this noise. In preference learning, researchers gather pairwise comparisons from people and attempt to learn which items are most preferred Jamieson and Nowak [2011], Yue and Joachims [2009]. To simplify such problems, human judgments are frequently modelled as distance comparisons Shepard [1962], Kruskal [1964], Mason et al. [2017] with the smallest distance representing the most preferred item, but the noise inherent to human judgements makes these problems challenging. In general, we define the NNSU problem as follows:

NNSU - Problem Statement: Consider a set of n points $\mathscr{X} = \{x_1, \cdots, x_n\}$ in a metric space $(\mathscr{M}, d)$. The metric is unknown, but for any pair of points we can query a stochastic oracle for a noisy, unbiased estimate of their distance. The problem is to use the oracle to efficiently build a data structure such that for any new query point q, it returns its nearest neighbor $x_q := \min_{x_i \in \mathscr{X}} d(q, x_i)$ with probability at least $1-\delta$ in as few additional oracle queries as possible.

The NNSU problem is very different from the standard Nearest Neighbor Search (NNS) problem[1]. To develop

[1]Note that NNSU is distinct from the problem of Approximate

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

noise-tolerant methods, it is tempting to simply extend an NNS algorithm by repeatedly calling the stochastic oracle a *pre-specified* r times each time a distance measurement is needed and averaging the results. The simplicity of this idea masks the difficulty of doing it correctly. If r is too small, an algorithm will make errors, but if r is too large, the number of distance queries and hence the computational cost will be unnecessarily high. To control the probability of error, r can be no smaller than $(d(q,x_q)-d(q,x_{q'}))^{-2}$ where $x_{q'}$ is the second nearest neighbor to q. Since this quantity depends on knowledge of the nearest neighbor distances, it is impossible to specify r in practice. Additionally, even if $(d(q,x_q)-d(q,x_{q'}))^{-2}$ were known, setting r in proportion to it is only necessary to handle the worst case; far fewer calls to the stochastic oracle are sufficient to eliminate points that are far from q. Our proposed algorithm is *adaptive* to the level of noise and the geometry of $\mathscr{X}$. It is guaranteed to minimize the number of calls to the distance oracle for a specified probability of error.

1.1 RELATED WORK

One of the first algorithms for the NNS problem is the classical `KD-Tree` algorithm by Bentley [1975]. `KD-Trees` first build a tree-based data structure in $O(n\log(n))$ computations. By paying this cost up front, when presented with a query point, the tree can be used to return the nearest neighbor in only $O(\log(n))$ computations. The core drawback of the `KD-Tree` is that it lacks a uniform accuracy guarantee and may fail to return the correct nearest neighbor for certain query points Dasgupta and Sinha [2013]. A variety of methods attempt to achieve similar computational complexity for the NNS problem while also providing robust accuracy guarantees Dasgupta and Sinha [2013], Krauthgamer and Lee [2004], Beygelzimer et al. [2006], and we refer the reader to Bhatia et al. [2010] for a survey of techniques and theoretical results. Another line of work attempts to instead return an approximate nearest neighbor that is almost as close as the true nearest neighbor. We refer the reader to Wang and Banerjee [2014], Andoni et al. [2018] for an overview.

A related problem to NNSU is the noisy nearest neighbor graph problem. In this setting, one wishes to learn the graph that connects each node in $\mathscr{X}$ to its nearest neighbor. Mason et al. [2019] provide an adaptive method that utilizes multi-armed bandits to find the nearest neighbor graph from a set of n points in $O(n\log(n)\Delta^2)$ samples in favorable settings and $O(n^2\Delta^2)$ at worst where Δ^2 is a problem dependent parameter quantifying the effect of the noise. Their method is adaptive to noise, but requires additional assumptions to achieve the optimal rate of $O(n\log(n))$ samples for that problem. Bagaria et al. [2017] study nearest neighbor search in high-dimensional data. In their setting, the distance function is a known, componentwise function. Their algorithm subsamples to approximate distances. This improves dependence on dimension, though the dependence on n is linear.

1.2 MAIN CONTRIBUTIONS

In this paper, we leverage recent developments in multi-armed bandits to solve the nearest neighbor search problem using noisy measurements. We design a novel extension of the `Cover-Tree` algorithm from Beygelzimer et al. [2006] for the NNSU problem. A main innovation is reducing the problem of learning a cover of a set from noisy data to all-ε-good identification for multi-armed bandits studied in Mason et al. [2020]. Additionally, we make use of efficient methods for adaptive hypothesis testing to minimize calls to the stochastic oracle Jamieson and Jain [2018]. We refer to the resulting algorithm as the `Bandit-Cover-Tree`. We show that it requires $O(n\log^2(n)\kappa)$ calls to the distance oracle for construction and $O(\log(n)\kappa)$ calls for querying the nearest neighbor of a new point, where κ captures the effect of noise and the geometry of $\mathscr{X}$. This nearly matches the state of the art for the NNS problem despite the added challenge of uncertainty in the NNSU problem. Furthermore, we show how to extend `Bandit-Cover-Tree` for approximate nearest neighbor search instead. Finally, we demonstrate that `Bandit-Cover-Tree` can be used to learn a nearest neighbor graph in $O(n\log^2(n)\kappa)$ distance measurements, nearly matching the optimal rate given in Mason et al. [2019] but without requiring the additional assumptions from that work.

1.3 NOTATION

Let $(\mathscr{M},d)$ be a metric space with distance function d satisfying the standard axioms. Given $x_i,x_j\in\mathscr{X}$, let $d(x_i,x_j)=d_{i,j}$. For a query point q define $x_q := \arg\min_{x\in\mathscr{X}} d(x_i,q)$. For a query point q and $x_i\in\mathscr{X}$, let $d_{q,i}$ denote $d(q,x_i)$. Additionally, for a set $\mathscr{S}$, define the distance from any point $x\in(\mathscr{M},d)$ to $\mathscr{S}$ as $d(x,\mathscr{S}) := \inf_{z\in S} d_{x,z}$, the smallest distance a point in $\mathscr{S}$. Though the distances are unknown, we are able to draw independent samples of its true value according to a stochastic distance oracle, i.e. querying

$$\mathrm{Q}(i,j) \quad \text{yields a realization of} \quad d_{i,j}+\eta, \tag{1}$$

where η is a zero-mean subGaussian random variable assumed to have scale parameter $\sigma=1$[2]. We let $\hat{d}_{i,j}(s)$ denote the empirical mean of the s queries of the distance oracle, $\mathrm{Q}(i,j)$. The number of $\mathrm{Q}(i,j)$ queries made until time t is

Nearest Neighbor identification as in this case, the measurements themselves are corrupted by noise, but one wishes to find an exact nearest neighbor. We extend to finding an approximate nearest neighbor under uncertainty as well in this work.

[2] $\sigma=1$ can be relaxed to any subGuassian distribution. The subGaussian assumption itself can be relaxed by considering other confidence widths in our algorithm.

denoted as $T_{i,j}(t)$. All guarantees will be shown to hold with probability $1-\delta$ where we refer to $\delta > 0$ as the failure probability.

2 COVER TREES FOR NEAREST NEIGHBOR SEARCH

Before presenting our method and associated results, we the review `Cover Tree` algorithm from Beygelzimer et al. [2006] for the NNS problem. As the name suggests, a cover tree is a tree-based data structure where each level of the tree forms a *cover* of $\mathscr{X}$.

Definition 1. *Given a set $\mathscr{X} \subset (\mathscr{M}, d)$, a set $C \subset \mathscr{X}$ is a* cover *of $\mathscr{X}$ with resolution ε if for all $x \in \mathscr{X}$, there exists a $c \in C$ such that $d(x,c) \leq \varepsilon$.*

Each level is indexed by an integer i which decreases as one descends the tree. To avoid additional notation, we will represent the top of the tree as level ∞ and the bottom as $-\infty$ though in practice one would record integers i_{top} and i_{bottom} denoting the top and bottom level of the tree and need only explicitly store the tree between these levels. Each node in the tree corresponds to a point in $\mathscr{X}$, but points in $\mathscr{X}$ may correspond to multiple nodes in the tree. Reviewing Beygelzimer et al. [2006], let C_i denote the set of nodes at level i. The cover tree algorithm is designed so that each level of the tree i obeys three invariants:

1. **nesting:** $C_i \subset C_{i-1}$. Hence, the points corresponding to nodes at level i are also correspond to nodes in all lower levels.
2. **covering tree:** For every $p \in C_{i-1}$, there exists a $q \in C_i$ such that $d(p,q) \leq 2^i$, and child node p is connected to parent node q in the cover tree.
3. **Separation:** For any $p, q \in C_i$ with $p \neq q$, $d(p,q) > 2^i$.

These invariants are originally derived from Krauthgamer and Lee [2004]. Beygelzimer et al. [2006] show that their routines obey these invariants, and we will make use of them in our proofs and to build intuition. The core idea of this method is that the i^{th} level of the tree is a cover of resolution 2^i. Given a query point q, when navigating the tree at level i, one identifies all possible ancestor nodes of x_q at that level. When descending the tree, this set is refined until only a single parent is possible, x_q itself. The *nesting* invariance allows one to easily traverse the tree from parent to child. The *covering tree* invariance connects x_q to its parents and ancestors so that one may traverse the tree with query point q and end at x_q. Lastly, the *separation* invariance ensures that there is a single parent to each node which avoids redundancy and improves memory complexity.

In order to quantify the sample complexity of cover trees, we require a notion of the effective dimensionality of the points $\mathscr{X}$. In this paper, we will make use of the *expansion constant* as in [Haghiri et al., 2017, Krauthgamer and Lee, 2004, Beygelzimer et al., 2006]. Let $B(x,r)$ denote the ball of radius $r > 0$ centered at $x \in (\mathscr{M}, d)$ according to the distance measure d.

Definition 2. *The expansion constant of a set of points $\mathscr{S}$ is the smallest $c \geq 2$ such that $|\mathscr{S} \cap B(x,2r)| \leq c|\mathscr{S} \cap B(x,r)|$ for any $x \in \mathscr{S}$ and any $r > 0$.*

The expansion constant is sensitive to the geometry in $\mathscr{X}$. For example, for points in a low-dimensional subspace, $c = O(1)$ independent of the ambient space. By contrast, if points that are spread out on the surface of a sphere in $\mathbb{R}^d$, $c = O(2^d)$. In general c is smaller if the pairwise distances between points in $\mathscr{X}$ are more varied.

3 THE BANDIT COVER TREE ALGORITHM FOR NNSU

The `Bandit Cover Tree` algorithm is comprised of three methods. Given a cover tree and query point q, `Noisy-Find-Nearest` finds q's nearest neighbor. `Noisy-Insert` allows one to insert points into a tree and can be used to construct a new tree. `Noisy-Remove` can remove points from the tree and is deferred to the appendix.

3.1 FINDING NEAREST NEIGHBORS WITH A COVER TREE

We begin by discussing how to identify the nearest neighbor of a query point q in the set $\mathscr{X}$ given a cover tree $\mathscr{T}$ on $\mathscr{X}$. Throughout, we take $\mathscr{T}$ to denote the cover tree. We may identify $\mathscr{T}$ by the covers at each level: $\mathscr{T} := \{C_\infty, \cdots, C_i, C_{i-1}, \cdots, C_{-\infty}\}$. Assume that we are given a fixed query point q, and the expansion constant of the set $\mathscr{X} \cup \{q\}$ is bounded by c. Our method to find x_q is `Noisy-Find-Nearest` and is given in Algorithm 1. It is inspired by Beygelzimer et al. [2006], but a crucial difference is how the algorithm handles noise. At a high level, as it proceeds down each level i, it keeps track of a set $Q_i \subset C_i$ of all possible ancestors of x_q. Given Q_i, to proceed to level $i-1$, the algorithm first computes the set of children of nodes in Q_i given by the set $Q \subset C_{i-1}$. $Q_{i-1} \subset Q$ is then computed by forming a *cover* of Q at resolution 2^{i-1}. Ideally, this is the set

$$\tilde{Q}_{i-1} := \left\{ j \in Q : d(q,j) \leq \min_{k \in Q} d(q,k) + 2^{i-1} \right\} \qquad (2)$$

However, constructing this set requires calling the stochastic distance oracle and presents a trade-off:

1. By using many repeated queries to the stochastic oracle, we can more confidently declare if $j \in \tilde{Q}_{i-1}$ at the expense of higher sample complexity.

Algorithm 1 `Noisy-Find-Nearest`

Require: Cover tree $\mathscr{T}$, failure probability δ, expansion constant c if known, query point q, callable distance oracle $\mathsf{Q}(\cdot,\cdot)$, subroutine `Identify-Cover`.

Let $Q_\infty = C_\infty$, $\alpha = n+1$
for $i = \infty$ down to $i = -\infty$ **do**
  Let $Q = \bigcup_{p \in Q_i} \text{children}(p)$
  if c is known: **then**
    Let $\alpha = \min\left\{ \left\lceil \frac{\log(n)}{\log(1+1/c^2)} + 1 \right\rceil, n \right\} + 1$
  \\ Identify the set: $\{j \in Q : d(q,j) \leq \min_{k \in Q} d(q,k) + 2^{i-1}\}$
  $Q_{i-1} =$ `Identify-Cover`$(Q, \delta/\alpha, \mathsf{Q}(\cdot,\cdot), q, i)$
return `find-smallest-in-set`$(Q_{-\infty}, \delta/\alpha)$

2. By using fewer calls to the oracle, we can more quickly declare if $j \in \tilde{Q}_{i-1}$, at the expense of lower accuracy.

To handle the noise, we make use of anytime confidence widths, $C_\delta(t)$ such that for an empirical estimate of the distance $d_{q,j}$: $\hat{d}_{q,j}(t)$ of t samples, we have $\mathbb{P}(\bigcup_{t=1}^{\infty} |\hat{d}_{q,j}(t) - d_{q,j}| > C_\delta(t)) \leq \delta$. For this work, we take $C_\delta(t) = \sqrt{\frac{4\log(\log_2(2t)/\delta)}{t}}$ akin to Howard et al. [2018]. This allows us to quantify how certain or uncertain an estimate of any distance is. Furthermore, to guard against settings where the number of samples needed to detect the set $\tilde{Q}_{i-1}$ in Eq (2) is large, we introduce some slack and instead detect a set Q_i such that it contains every $j \in Q$ such that $d_{q,j} \leq d(q,Q) + 2^{i-1}$ and no j worse than $d_{q,j} \leq d(q,Q) + 2^{i-1} + 2^{i-2}$. In particular, this contains all points in $\tilde{Q}_{i-1}$ and none that are much worse. This has the advantage of controlling the number of calls to the stochastic oracle. As the algorithm descends the tree, i decreases and the slack of 2^{i-2} goes to zero ensuring that the algorithm returns the correct nearest neighbor. To handle both of these challenges, we reduce the problem of learning $\tilde{Q}_{i-1}$ to the problem of finding all ε-good arms in mutli-armed bandits studied in Mason et al. [2020] and refer to this method as `Identify-Cover` given in Alg. 2.

This process of exploring children nodes and computing cover sets terminates when the algorithm reaches level $i = -\infty$ and $Q_{-\infty}$ contains x_q. Note that in practice the algorithm could terminate when it reaches the bottom of the cover tree, $i = i_{\text{bottom}}$. At this level, it calls a simple elimination scheme `find-smallest-in-set` given in the appendix that queries the stochastic oracle and returns $x_q \in Q_{-\infty}$. To control the overall probability of error in `Noisy-Find-Nearest`, each call to `Identify-Cover` and `find-smallest-in-set` is run with failure probability δ/α for α chosen to control the family-wise error rate.

Algorithm 2 `Identify-Cover`

Require: Failure probability δ, query point q, Oracle $\mathsf{Q}(\cdot,\cdot)$, and set Q, Cover resolution 2^i

Query oracle once for each point in Q
Initialize $T_j \leftarrow 1$, update $\hat{d}_{q,j}$ for each $j \in Q$
Empirical cover set: $\widehat{G} = \{j : \hat{d}_{q,j} \leq \max_k \hat{d}_{q,k} + 2^i\}$
$U_t = \min_k \hat{d}_{q,k}(T_k) + C_{\delta/|Q|}(T_k) + 2^i$
$L_t = \min_k \hat{d}_{q,k}(T_k) - C_{\delta/|Q|}(T_k) + 2^i + 2^{i-1}$
Known points: $K = \{j : \hat{d}_{q,j}(T_j) - C_{\delta/|Q|}(T_j) > U_t \text{ or } \hat{d}_{q,j}(T_j) + C_{\delta/|Q|}(T_j) < L_t\}$
while $K \neq Q$ **do**
  $j_1(t) = \arg\min_{j \in \widehat{G} \setminus K} \hat{d}_{q,j}(T_j) + C_{\delta/|Q|}(T_j)$
  $j_2(t) = \arg\max_{j \in \widehat{G}^c \setminus K} \hat{d}_{q,j}(T_j) - C_{\delta/|Q|}(T_j)$
  $j^*(t) = \arg\min_j \hat{d}_{q,j}(T_j) - C_{\delta/|Q|}(T_j)$
  Call oracle $\mathsf{Q}(q, j_1)$, $\mathsf{Q}(q, j_2)$, and $\mathsf{Q}(q, j^*)$
  Update $T_{j_1}, \hat{d}_{q,j_1}, T_{j_2}, \hat{d}_{q,j_2}, T_{j^*}, \hat{d}_{q,j^*}$
  Update bounds L_t, U_t, sets $\widehat{G}$, K
return The set cover set with resolution 2^i: $\{j : \hat{d}_{q,j}(T_j) + C_{\delta/|Q|}(T_j) < L_t\}$

3.1.1 Approximate NNSU

In some applications, finding an exact nearest neighbor may be unnecessary and an *approximate* nearest neighbor may be sufficient. In the noiseless case, this has been shown to improve the complexity of nearest neighbor search Andoni et al. [2018]. An ε-approximate nearest neighbor is defined as any point in the set

$$\{i : d(q,i) \leq (1+\varepsilon) \min_j d(q,j)\}.$$

In this section, we show to how to extend `Noisy-Find-Nearest` to return an ε-approximate nearest neighbor instead of the exact nearest neighbor and provide psuedocode for this method in the appendix. To do so, add an additional line that exits the for loop if $d(q,Q_i) \geq 2^{i+1}(1+1/\varepsilon)$. Then, return $x_{q_\varepsilon} = \arg\min_{j \in Q_i} d(q,x_j)$. To see why this condition suffices, note that the nesting and covering tree invariances jointly imply that $d(x_q,Q_i) \leq 2^{i+1}$. Hence, by the triangle inequality

$$d(q,Q_i) \leq d(q,x_q) + d(x_q,Q_i) \leq d(q,x_q) + 2^{i+1}.$$

Since we exit when $d(q,Q_i) \geq 2^{i+1}(1+1/\varepsilon)$,

$$2^{i+1}(1+1/\varepsilon) \leq d(q,x_q) + 2^{i+1} \iff 2^{i+1} \leq \varepsilon d(q,x_q).$$

Therefore,

$$d(q,Q_i) \leq d(q,x_q) + 2^{i+1} \leq (1+\varepsilon) d(q,x_q).$$

Hence, there exists an ε-approximate nearest neighbor in Q_i. If this condition is never satisfied, which occurs when

ε is vanishingly small, the algorithm terminates normally and returns the exact nearest neighbor. We give pseudocode for this procedure in the appendix. The algorithm is similar to `Noisy-Find-Nearest` except that it makes use of a simple thresholding bandit, akin to the one we use in `Noisy-Insert` to check if $d(q, Q_i) \geq 2^{i+1}(1+1/\varepsilon)$ by querying the stochastic oracle.

3.2 BUILDING AND ALTERING A COVER TREE

In this section we demonstrate how to insert points into a cover tree by calling the stochastic oracle. Constructing a cover tree can be achieved by simply beginning with an empty tree and inserting points one at a time. Removing points from a cover tree is similar operationally to insertion with slight complications and we defer it to the appendix.

3.2.1 Insertion

Suppose we have access to a cover tree $\mathscr{T}$ on a set $\mathscr{S}$. We wish to insert a point p into $\mathscr{T}$ such that we now have a cover tree on the set $\mathscr{S} \cup \{p\}$. Intuitively, the insertion algorithm can be thought of as beginning at the highest resolution cover, at level $-\infty$ and climbing back up the tree, inserting p in each cover set C_i for all i until it reaches a level i_p such that $\min_{j \in C_{i_p}} d(p, j) \leq 2^{i_p}$ where a suitable parent node exists. The algorithm then chooses a parent $p' \in C_{i_p}$ for p such that $d(p, p') \leq 2^{i_p}$ and terminates. As trees are traditionally traversed via their roots not their leaves, we state this algorithm recursively beginning at the top.

We provide pseudocode in Algorithm 3. The algorithm draws on ideas from Beygelzimer et al. [2006] but includes additional logic to handle uncertainty. In particular, lines 2-8 implement a simple thresholding bandit based on the techniques of Jamieson and Jain [2018] for adaptive hypothesis testing with family-wise probability of error control. This allows us to identify all points within 2^i of the nearest in the set Q. This must be done for every candidate level i that p might be inserted into until it reaches level i_p.

This requires careful handling to ensure the algorithm succeeds with high probability. Each time the thresholding operation is called, there is a chance that the algorithm makes a mistake. `Noisy-Insert` is recursive and performs a this operation in every recursive call. Hence, if it makes an error on any call, the algorithm may fail. Thus, we must ensure that the probability of `Noisy-Insert` making an error in any recursive call is at most δ. Since the level i_p where p is added is unknown and depends on the query point p, the number of recursive calls before success is unknown to the algorithm. Therefore, it is not a priori obvious how to perform the appropriate Bonferroni correction to ensure the probability of an error in any recursive call is bounded by δ. A seemingly attractive approach is to use a summable

Algorithm 3 `Noisy-Insert`

Require: Cover tree $\mathscr{T}$ on n points, cover set Q_i, failure probability δ, point p to be inserted, callable distance oracle $\mathsf{Q}(\cdot,\cdot)$, level i

Optional: Empirical estimates of $\hat{d}_{p,i}$ and T_i for any $i \in C_i$

1: Let $Q = \bigcup_{j \in Q_i} \text{children}(j)$
2: Query oracle once for each point in $Q \cap \{j : T_j = 0\}$
3: Set $T_i \leftarrow 1$, update $\hat{d}_{p,i}$ for each $j \in Q \cap \{j : T_j = 0\}$
4: \\ compute the set $\{j \in Q : d(p, j) \leq 2^i\}$
5: Known points: $K = \{j : \hat{d}_{p,j}(T_j) + C_{\delta/n}(T_j) \leq 2^i$ or $\hat{d}_{p,j}(T_j) - C_{\delta/n}(T_j) > 2^i\}$
6: **while** $|K| \neq |Q|$ **do**
7: Compute $j^*(t) = \arg\min_{j \notin K} \hat{d}_{p,j}(T_j) - C_{\delta/n}(T_j)$
8: Call oracle $\mathsf{Q}(p, j^*)$ and update $T_{j^*}, \hat{d}_{p,j^*}, K$
9: \\ If $d(p, Q) > 2^i$
10: $Q_{i-1} = \{j \in Q : \hat{d}_{p,j}(T_j) + C_{\delta/n}(T_j) \leq 2^i\}$
11: **if** $Q_{i-1} = \emptyset$ **then**
12: **Return:**"no parent found"
13: **else**
14: lower = `Noisy-Insert`$(p, \mathscr{T}, Q_{i-1}, i-1, \delta, \mathsf{Q}(\cdot,\cdot), \{j \in Q : \hat{d}_{p,j}(T_j)\}, \{j \in Q : T_j\})$
15: **if** $Q_i \cap Q_{i-1} \neq \emptyset$ and lower = "no parent found" **then**
16: Choose any $p' \in Q_i \cap Q_{i-1}$
17: Insert p in children(p') \\ Assign a parent to p
18: **Return:**"parent found"
19: **else**
20: **Return:**"no parent found"

sequence of δ_i depending on the level i such that $\sum_i \delta_i = \delta$. For instance, $\delta_i = \delta \cdot 2^{-i}$ would be suitable if the root of $\mathscr{T}$ is at level 1. However, due to repeated calls to lines 2-8, this would lead to an additional multiplicative factor of the height of the cover tree affecting the sample complexity.

Instead, `Noisy-Insert` *shares* samples between rounds. By the nesting invariance, we have that $C_i \subset C_{i-1}$. Therefore, when we descend the tree from level i to $i-1$, we already have samples of the distance of some points in C_{i-1} to p. We simply reuse these samples and share them from round to round. Furthermore, since $\mathscr{T}$ is assumed to be a cover tree on n points, we trivially union bound each confidence width to hold with probability $1 - \delta/n$ such that all bounds for all recursive calls holds with probability at least $1 - \delta$.

4 THEORETICAL GUARANTEES OF `BANDIT COVER TREE`

We measure performance along several axes: 1) how much memory is necessary to store the data structure, 2) how many calls to the distance oracle are needed at query time, 3) How many calls to the distance oracle are needed to for construction, and 4) the accuracy of the data structure in returning the correct nearest neighbor and performing other

operations. Since we only have access to a stochastic oracle, ensuring the accuracy of the `Bandit Cover Tree` (`BCT`) is especially delicate.

4.1 MEMORY

We begin by showing that a cover tree can efficiently be stored. Naively, a cover tree $\mathscr{T}$ on $\mathscr{X}$ can be stored using $O(n(i_{\text{top}} - i_{\text{bottom}}))$ memory where $n = |\mathscr{X}|$ and $i_{\text{top}} - i_{\text{bottom}}$ is the height of the tree. This follows from each level having at most n nodes trivially and there being $i_{\text{top}} - i_{\text{bottom}}$ levels. For a well balanced tree, we expect that $i_{\text{top}} - i_{\text{bottom}} = O(\log(n))$ leading to an overall memory complexity of $O(n\log(n))$. In fact, it is possible to do better.

Lemma 3. *A bandit cover tree requires $O(n)$ space to be stored.*

$O(n)$ memory is possible due to the nesting and covering tree invariants. By the nesting invariant, if a point p is present in the i^{th} cover C_i, then it is present in the cover sets of all lower levels. By the covering tree invariant, each point has a unique parent in the tree. Therefore, to store a cover tree, one need only store 1) which level of the tree each point first appears in 2) each point's parent node in the level above where it appears. After a node first appears in the tree, it may be represented implicitly in all lower levels. In particular, when all methods query the children of any node p, we define $p \in \text{children}(p)$. Formally, we define an implicit and explicit node as follows:

Definition 4. *A node $p \in C_i$ in level i of cover tree $\mathscr{T}$ is* explicit *if $p \notin \{C_\infty, \cdots, C_{i+1}\}$ (i.e. it is not present in all previous levels.) Otherwise, any $p \in C_i$ is referred to as* implicit.

4.2 ACCURACY

Next, we show that `BCT` is accurate with high probability. This requires three guarantees:

1. Search accuracy: Given a cover tree $\mathscr{T}$ on set $\mathscr{X}$ and query point q, `Noisy-Find-Nearest` correctly identifies q's nearest neighbor with high probability.
2. Insertion accuracy: Given a cover tree $\mathscr{T}$ and a point p to be inserted, `Noisy-Insert` returns a valid cover tree that includes p with high probability.
3. Removal accuracy: Given a cover tree $\mathscr{T}$ and a point p to be removed, `Noisy-Remove` returns a valid cover tree that without p (deferred to appendix).

4.2.1 Search Accuracy

We begin by showing that for any $q \in (\mathscr{M}, d)$ `Noisy-Find-Nearest` returns x_q with high probability. The proof is deferred to the appendix, but the argument is sketched as follows. First, by appealing to results from Mason et al. [2020], we can guarantee that `Identify-Cover` succeeds with probability $1 - \delta/\alpha$. Second, we show that for the choice of α in `Noisy-Find-Nearest`, the probability that any call to `Identify-Cover` fails is bounded by $1 - \delta$. Finally, we show that if we correctly identify the necessary cover sets, which happens with probability at least $1 - \delta$, we my appeal to Theorem 2 of Beygelzimer et al. [2006] to ensure that our method returns x_q.

Lemma 5. *Fix any $\delta \leq 1/2$ and a query point q. Let $\mathscr{T}$ be a cover tree on a set $\mathscr{X}$. `Noisy-Find-Nearest` returns $x_q \in \mathscr{X}$ with probability at least $1 - \delta$.*

4.2.2 Insertion Accuracy

The proof that `Noisy-Insert` succeeds with high probability in adding any point p to a cover tree $\mathscr{T}$ follows the argument of Lemma 5 similarly. In particular, given a set Q, it hinges on the idea that we can use a threshold bandit to identify the set $\{j \in Q : d(p,j) \leq 2^i\}$ correctly with high probability by calling the stochastic oracle. If this procedure succeeds, then `Noisy-Insert` has correctly identified the subset of Q that are within 2^i of the point p to be inserted. If $\{j \in Q : d(p,j) \leq 2^i\} = \emptyset$, we may instead verify that $d(p,Q) > 2^i$ and can compute a new cover set Q_{i-1}. The following lemma ensures that `Noisy-Insert` succeeds with high probability.

Lemma 6. *Fix any $\delta > 0$. Let $\mathscr{T}$ be a cover tree on a set $\mathscr{X}$ and p be a point to insert. `Noisy-Insert` correctly returns a cover tree on $\mathscr{X} \cup \{p\}$ with probability at least $1 - \delta$.*

To construct a cover tree from scratch given a set $\mathscr{X}$, one need only call `Noisy-Insert` n times, once for each point in $\mathscr{X}$ beginning with an empty tree and run each call with failure probability δ/n. The following corollary ensures that this leads to a cover tree satisfying the three invariances with probability $1 - \delta$.

Corollary 7 (Construction of a Cover Tree)**.** *Fix any $\delta > 0$ and a set $\mathscr{X} \in (\mathscr{M}, d)$. Calling `Noisy-Insert` with failure probability δ/n iteratively over the points in $\mathscr{X}$ yields a cover tree that satisfies the nesting, covering tree, and separation invariances with probability $1 - \delta$.*

4.3 QUERY TIME COMPLEXITY

In the previous section, we proved that the algorithm succeeds with probability $1 - \delta$ for search and insertion, with removal deferred to the appendix. In this section we begin to answer the question of how many calls to the stochastic distance oracle it requires to perform these operations.

We begin by analyzing the query time complexity of `Bandit Cover Tree`: the number of calls to the distance oracle made when answering a nearest neighbor query. To do so, we will make use of the *expansion constant*, a data-dependent measure of dimensionality. In particular, for a query point q. We assume that the set $\mathscr{X} \cup \{q\}$ has an expansion constant of c as defined in Definition 2. Note that this quantity is for analysis purposes only and is not required by the algorithm. `Noisy-Find-Nearest` can take in the expansion constant or a bound on it if it is known, but this is not required by the algorithm. In particular, knowing a bound on c helps control the height of the tree. This allows `Noisy-Find-Nearest` to use a smaller union bound and leads to a tighter dependence on n.

To bound query time, we appeal to the concept of explicit and implicit nodes as discussed in Section 4.1 and Definition 4. Each point in $\mathscr{X}$ may correspond to multiple nodes in the tree. The first time a point appears as a node at the highest level where it is present, we say that it is *explicitly represented* and is *implicity represented* in all levels thereafter by the nesting invariant of cover trees. Recall that the set of nodes at each level i of the cover tree is denoted C_i. `Noisy-Find-Nearest` proceeds by computing a cover set $Q_i \subset C_i$ at each level of the tree. Extending the concept of explicit and implicit representations of nodes, we say that a cover set Q_i is implicitly represented if it only contains nodes that are implicitly represented. This plays an important role in our computation of query time. We may use the size of the last explicitly represented cover to help bound to complexity of the `Identify-Cover` subroutine in Algorithm 2. This routine is called for every level of the tree. Then, by bounding the height of the tree, we can control the total number of calls to the distance oracle.

Theorem 8. *Fix $\delta < 1/2$, a cover tree $\mathscr{T}$ on set $\mathscr{X}$, and a query point q. Let $|\mathscr{X}| = n$ and assume that the expansion rate of $\mathscr{X} \cup \{q\}$ is c (unknown to the algorithm). If `Noisy-Find-Nearest` succeeds, which occurs with probability $1-\delta$, then `Noisy-Find-Nearest` returns q's nearest neighbor in at most*

$$O\left(c^{17}\log(n)\log\left(\frac{n}{\delta}\right)\kappa\right)$$

calls to the stochastic distance oracle where parameter κ is defined in the proof and satisfies

$$\kappa \;\leq\; B\max\left\{\left[d(q,x_q)-d(q,x_{q'})\right]^{-2}, d_{\min}^{-2}\right\}$$

where $x_{q'}$ is q's second nearest neighbor, $d_{\min} := \min_{j,k}(x_j,x_k)$ is the smallest pairwise distance in $\mathscr{X}$, and $B > 0$ is a universal constant.[3]

Remark 9. *The dependence of c^{17} can be improved to c^7 matching Beygelzimer et al. [2006] by redefining $L_t = \min_j \hat{d}_{q,j}(T_j) - C_{\delta/|Q|}(T_j) + 2^i$ in the pseudocode of `Identify-Cover`. For most instances, this will lead to better performance, though a potentially worse worst-case bound on κ in pathological cases.*

The above result scales as $O\left(c^{O(1)}\log^2(n)\right)$. The following corollary highlights that if c is known, this can be improved to $O\left(c^{O(1)}\log(n)\log(\log(n))\right)$.

Corollary 10. *Under the same conditions as Theorem 8, if any bound $\tilde{c} \geq c$ on the expansion rate c is given to the algorithm, the number of queries to the distance oracle is at most*

$$O\left(\tilde{c}^{17}\log(n)\log\left(\frac{\tilde{c}\log(n)}{\delta}\right)\kappa\right)$$

calls to the stochastic oracle.

The term κ captures the average effect of noise on this problem. It is similar to the term $\overline{\Delta^{-2}}$ in given in Mason et al. [2019] for identifying the Nearest Neighbor Graph of $\mathscr{X}$ from a stochastic distance oracle. As the noise variance goes to 0, this term becomes 1. Furthermore, κ is adaptive to the geometry of $\mathscr{X}$ and in most cases is much smaller than its bound in Theorem 8. Intuitively, the bound on κ states that the number of repeated samples is never worse that what is necessary to 1) distinguish between q's nearest neighbor and second nearest neighbor and 2) distinguish between any two points in $\mathscr{X}$. Crucially, however, `Noisy-Find-Nearest` *adapts* to the level of noise, the geometry of $\mathscr{X}$ and need not know these values a priori.

4.4 INSERTION TIME AND BUILD TIME COMPLEXITY

Next, we bound the number of calls to the distance oracle necessary to insert a new point p into a cover tree $\mathscr{T}$.

Theorem 11. *Fix $\delta > 0$, a cover tree $\mathscr{T}$ on set $\mathscr{X}$, and a point to insert p. Let $|\mathscr{X}| = n$ and assume that the expansion rate of $\mathscr{X} \cup \{p\}$ is c. Run `Noisy-Insert` with failure probability $1-\delta$ and pass it the root level cover set: $C_{i_{top}}$ and level $i = i_{top}$. If `Noisy-Insert` succeeds, which occurs with probability $1-\delta$, then it returns a cover tree on $\mathscr{X} \cup \{p\}$ in at most*

$$O\left(c^7\log(n)\log\left(\frac{n}{\delta}\right)\overline{\kappa_p}\right)$$

calls to the noisy distance oracle where parameter $\overline{\kappa_p}$ is defined in the proof and depends on $\mathscr{X}$ and p.

Remark 12. *As in the statement of Theorem 8, the term $\overline{\kappa_p}$ captures the average effect of noise on this problem, and as the noise variance goes to 0, this term becomes 1.*

[3] The dependence on c is large theoretically, however Beygelzimer et al. [2006] suffer similar theoretical dependencies on c but show that in practice the dependence is more mild.

As discussed in Section 3.2.1, to construct a cover tree from scratch, one need only call `Noisy-Insert` on each point in $\mathscr{X}$ and add them the tree one at a time. The following theorem bounds the complexity of this process.

Theorem 13. *Fix $\delta > 0$ and set n points $\mathscr{X}$. Assume that the expansion rate of $\mathscr{X}$ is c. Calling `Noisy-Insert` with failure probability δ/n on each point in $\mathscr{X}$ one at a time, returns a cover tree $\mathscr{T}$ on $\mathscr{X}$ correctly with probability at least $1-\delta$ in at most*

$$O\left(c^7 n \log(n) \log\left(\frac{n}{\delta}\right) \widetilde{\kappa}\right)$$

calls to the noisy distance oracle where $\widetilde{\kappa} := \frac{1}{n}\sum_{i\in\mathscr{X}} \overline{\kappa}_i$ for $\overline{\kappa}_i$ defined in the proof of Theorem 11.

4.5 EXTENSION TO THE NEAREST NEIGHBOR GRAPH PROBLEM

In this section we extend the results of Mason et al. [2019] for learning *nearest neighbor graphs* from a stochastic distance oracle. In this setting, given a set $\mathscr{X} \in (\mathscr{M}, d)$, one wishes to learn the directed graph $G(\mathscr{X})$ that connects each $x \in \mathscr{X}$ to its nearest neighbor in $\mathscr{X}\backslash\{x\}$. While Mason et al. [2019] also leverage a bandit subroutine, the bandit algorithm considered in this work is markedly different from the one considered in Mason et al. [2019]. This problem is well studied in the noiseless regime and at times referred to as the all nearest neighbor problem Clarkson [1983], Sankaranarayanan et al. [2007], Vaidya [1989]. Mason et al. [2019] provide the first algorithm that learns the nearest neighbor graph using only a stochastic oracle and show that in special cases it achieves a rate of $O(n\log(n)\Delta^{-2})$ which is matches the optimal rate for the noiseless problem with only a multiplicative penalty of Δ^{-2} accounting for the effect of noise. Unfortunately, the condition necessary to show that result is stringent. In fact, it can be shown that the separation condition of Mason et al. [2019] is equivalent to the special case that $c = 2$. The authors conjectured that the condition was not necessary and showed empirically that it should be possible to achieve $O(n\log(n))$ complexity in the noisy setting without their separation condition. This motivates the question of if similar performance can be achieved under more general conditions than those needed in Mason et al. [2019]. We answer this question in the affirmative and show how to extend the `Bandit-Cover-Tree` to achieve near optimal performance on the nearest neighbor graph problem without the need for any additional assumptions on the data. This proceeds in two steps:

1. Build a cover tree $\mathscr{T}$ on $\mathscr{X}$ with probability $1-\delta/2$.
2. For each $x \in \mathscr{X}$, find its nearest neighbor in $\mathscr{X}\backslash\{x\}$ using $\mathscr{T}$ with probability $1-\delta/2n$.

The above is sufficient to specify each edge of the nearest neighbor graph. Note that $\mathscr{T}$ is a cover tree on $\mathscr{X}$ not $\mathscr{X}\backslash\{x\}$ so we cannot blindly use `Noisy-Find-Nearest` as it will return x itself. To find x's nearest neighbor in $\mathscr{X}\backslash\{x\}$, one may instead modify `Noisy-Find-Nearest` so that `Identify-Cover` is called on the set $Q_i\backslash\{x\}$ instead of Q_i. Then, when the algorithm terminates, the final set $Q_{i_{\text{bottom}}}$ will have 2 points: x and its nearest neighbor in $\mathscr{X}\backslash\{x\}$. A simple union bound ensures that this process succeeds with probability $1-\delta$. The following Lemma bounds the total number of samples.

Lemma 14. *Via the above procedure, `Bandit-Cover-Tree` returns the nearest neighbor graph of $\mathscr{X}$ with probability $1-\delta$ in*

$$O\left(c^7 n \log(n) \log\left(\frac{n}{\delta}\right) \widetilde{\kappa} + \sum_{x\in\mathscr{X}} c^7 \log(n) \log\left(\frac{n}{\delta}\right) \kappa_x\right)$$

for $\widetilde{\kappa}$ defined in Theorem 13 and κ_x as in Theorem 8.

The proof follows by combining the guarantees of Theorems 8 and 13. In particular, the above bound scales as $O(c^7 n \log^2(n)\kappa)$ which matches the rate of Mason et al. [2019] with an additional log factor. Importantly, the above result makes *no assumptions* on the set $\mathscr{X}$, and instead the bound scales with the expansion rate, c. Hence, we achieve near optimal performance for the nearest neighbor graph problem with under *far more general* conditions.

5 CONCLUSION

In this paper, we introduced the `Bandit Cover Tree` framework for the Nearest Neighbor Search under Uncertainty (NNSU) problem. `BCT` builds on top of the `Cover Tree` algorithm by Beygelzimer et al. [2006]. We present three methods, `Noisy-Find-Nearest`, `Noisy-Insert`, and `Noisy-Remove`, and bound their accuracy, memory footprint, build complexity, insertion and removal complexities, and query complexities. In particular, we show a query complexity that is $O(\log(n))$, insertion complexity that is $O(\log^2(n))$, removal complexity that is $O(\log^2(n))$, and a construction complexity of $O(n\log^2(n))$. The query complexity matches the state of the art n dependence for the NNS problem of $O(\log(n))$ despite the added uncertainty of the NNSU problem. The additional $\log(n)$ term present in the insertion and removal guarantee stems from a union bound which is necessary when dealing with a stochastic oracle, though it may be possible that by giving the algorithm access to the expansion constant, this can be improved to an additional doubly logarithmic term instead. Hence, the insertion, construction, and removal complexities are also near the state of the art for NNS. Lastly a memory footprint of $O(n)$ and accuracy of $1-\delta$ are both optimal in this problem. In particular, a tree with n leaves requires $\Omega(n)$ memory to store, and though a dependence on δ is unavoidable when dealing with a stochastic oracle,

BCTs enjoy a probability $1-\delta$ probability of success for any specified $\delta > 0$.

Note that we focus on controlling the number of calls to the distance oracle in this work and assume that an individual call requires $O(1)$ work. We expect that for practical problems the bulk of the computational effort will be many repeated calls to the oracle. It is an open question for future work to control the computational complexity in the noisy regime considering all operations, not just calls to the stochastic oracle.

Furthermore, the results depend strongly on the expansion rate, c. Some works such as Haghiri et al. [2017], Dasgupta and Sinha [2013] trade accuracy for improved dependence on c or other measures of dimension. Instead, these algorithms guarantee that x_q is correctly returned with probability at least $1-\delta_{c,n}$ for any q. Though $\delta_{c,n}$ is tunable, it often depends on c or other parameters potentially unknown to the practitioner. These methods often achieve good empirical performance, however. It may be interesting to develop methods that achieve lower theoretical accuracy but enjoy a softer dependence on c.

References

Alexandr Andoni, Piotr Indyk, and Ilya Razenshteyn. Approximate nearest neighbor search in high dimensions. *arXiv preprint arXiv:1806.09823*, 7, 2018.

Vivek Bagaria, Govinda M Kamath, Vasilis Ntranos, Martin J Zhang, and David Tse. Medoids in almost linear time via multi-armed bandits. *arXiv preprint arXiv:1711.00817*, 2017.

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

Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104. ACM, 2006.

Nitin Bhatia et al. Survey of nearest neighbor techniques. *arXiv preprint arXiv:1007.0085*, 2010.

Kenneth L Clarkson. Fast algorithms for the all nearest neighbors problem. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 226–232. IEEE, 1983.

Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for exact nearest neighbor search. In *Conference on Learning Theory*, pages 317–337, 2013.

Brian Eriksson, Paul Barford, Joel Sommers, and Robert Nowak. A learning-based approach for ip geolocation. In *International Conference on Passive and Active Network Measurement*, pages 171–180. Springer, 2010.

Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Pac bounds for multi-armed bandit and markov decision processes. In *International Conference on Computational Learning Theory*, pages 255–270. Springer, 2002.

Siavash Haghiri, Debarghya Ghoshdastidar, and Ulrike von Luxburg. Comparison based nearest neighbor search. *arXiv preprint arXiv:1704.01460*, 2017.

Steven R Howard, Aaditya Ramdas, Jon McAuliffe, and Jasjeet Sekhon. Uniform, nonparametric, non-asymptotic confidence sequences. *arXiv preprint arXiv:1810.08240*, 2018.

Kevin Jamieson and Lalit Jain. A bandit approach to multiple testing with false discovery control. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3664–3674, Red Hook, NY, USA, 2018. Curran Associates Inc.

Kevin G Jamieson and Robert Nowak. Active ranking using pairwise comparisons. In *Advances in Neural Information Processing Systems*, pages 2240–2248, 2011.

Robert Krauthgamer and James R Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

Joseph B Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

Blake Mason, Lalit Jain, and Robert Nowak. Learning low-dimensional metrics. In *Advances in neural information processing systems*, pages 4139–4147, 2017.

Blake Mason, Ardhendu Tripathy, and Robert Nowak. Learning nearest neighbor graphs from noisy distance samples. In *Advances in Neural Information Processing Systems*, pages 9586–9596, 2019.

Blake Mason, Lalit Jain, Ardhendu Tripathy, and Robert Nowak. Finding all $\{\varepsilon\}$-good arms in stochastic bandits. *Advances in Neural Information Processing Systems*, 2020.

Jagan Sankaranarayanan, Hanan Samet, and Amitabh Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 31 (2):157–174, 2007.

Roger N Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function. i. *Psychometrika*, 27(2):125–140, 1962.

Pravin M Vaidya. Ano (n logn) algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4(2):101–115, 1989.

Huahua Wang and Arindam Banerjee. Randomized block coordinate descent for online and stochastic optimization. *arXiv preprint arXiv:1407.0107*, 2014.

Yisong Yue and Thorsten Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1201–1208, 2009.