Journal of Machine Learning Research 16 (2015) 3269-3297 Submitted 1/15; Published 12/15

Plug-and-Play Dual-Tree Algorithm Runtime Analysis

Ryan R. Curtin RYAN@RATML.ORG
School of Computational Science and Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0250, USA

Dongryeol Lee DRSELEE@GMAIL.COM
Yahoo Labs
Sunnyvale, CA 94089

William B. March MARCH@ICES.UTEXAS.EDU
Institute for Computational Engineering and Sciences
University of Texas, Austin
Austin, TX 78712-1229

Parikshit Ram P.RAM@GATECH.EDU
Skytree, Inc.
Atlanta, GA 30332

Editor: Nando de Freitas

Abstract

Numerous machine learning algorithms contain pairwise statistical problems at their core—that is, tasks that require computations over all pairs of input points if implemented naively. Often, tree structures are used to solve these problems efficiently. Dual-tree algorithms can efficiently solve or approximate many of these problems. Using cover trees, rigorous worst-case runtime guarantees have been proven for some of these algorithms. In this paper, we present a *problem-independent* runtime guarantee for *any* dual-tree algorithm using the cover tree, separating out the problem-dependent and the problem-independent elements. This allows us to just plug in bounds for the problem-dependent elements to get runtime guarantees for dual-tree algorithms for any pairwise statistical problem without re-deriving the entire proof. We demonstrate this plug-and-play procedure for nearest-neighbor search and approximate kernel density estimation to get improved runtime guarantees. Under mild assumptions, we also present the first linear runtime guarantee for dual-tree based range search.

Keywords: dual-tree algorithms, adaptive runtime analysis, cover tree, expansion constant, nearest neighbor search, kernel density estimation, range search

1. Dual-tree Algorithms

A surprising number of machine learning algorithms have computational bottlenecks that can be expressed as pairwise statistical problems. By this, we mean computational tasks that can be evaluated directly by iterating over all pairs of input points. Nearest neighbor search is one such problem, since for every query point, we can evaluate its distance to every reference point and keep the closest one. This naively requires $O(N)$ time to answer a single query in a reference set of size N; answering $O(N)$ queries subsequently requires

prohibitive $O(N^2)$ time. Kernel density estimation is also a pairwise statistical problem, since we compute a sum over all reference points for each query point. This again requires $O(N^2)$ time to answer $O(N)$ queries if done directly. The reference set is typically indexed with spatial data structures to accelerate this type of computation (Finkel and Bentley, 1974; Beygelzimer et al., 2006); these result in $O(\log N)$ runtime per query under favorable conditions.

Building upon this intuition, Gray and Moore (2001) generalized the fast multipole method from computational physics to obtain dual-tree algorithms. These are extremely useful when there are large query sets, not just a few query points. Instead of building a tree on the reference set and searching with each query point separately, Gray and Moore suggest also building a query tree and traversing both the query and reference trees simultaneously (a *dual-tree traversal*, from which the class of algorithms takes its name).

Dual-tree algorithms can be easily understood through the recent framework of Curtin et al. (2013b): two trees (a query tree and a reference tree) are traversed by a *pruning dual-tree traversal*. This traversal visits combinations of nodes from the trees in some sequence (each combination consisting of a query node and a reference node), calling a problem-specific `Score()` function to determine if the node combination can be pruned. If not, then a problem-specific `BaseCase()` function is called for each combination of points held in the query node and reference node. This has significant similarity to the more common single-tree branch-and-bound algorithms, except that the algorithm must recurse into child nodes of *both* the query tree and reference tree.

There exist numerous dual-tree algorithms for problems as diverse as kernel density estimation (Gray and Moore, 2003), mean shift (Wang et al., 2007), minimum spanning tree calculation (March et al., 2010), n-point correlation function estimation (March et al., 2012), max-kernel search (Curtin et al., 2013c), particle smoothing (Klaas et al., 2006), variational inference (Amizadeh et al., 2012), range search (Gray and Moore, 2001), and embedding techniques (Van Der Maaten, 2014), to name a few.

Some of these algorithms are derived using the cover tree (Beygelzimer et al., 2006), a data structure with compelling theoretical qualities. When cover trees are used, dual-tree all-nearest-neighbor search and approximate kernel density estimation have $O(N)$ runtime guarantees for $O(N)$ queries (Ram et al., 2009a); minimum spanning tree calculation scales as $O(N \log N)$ (March et al., 2010). Other problems have similar worst-case guarantees (Curtin and Ram, 2014; March, 2013).

In this work we combine the generalization of Curtin et al. (2013b) with the theoretical results of Beygelzimer et al. (2006) and others in order to develop a worst-case runtime bound for any dual-tree algorithm when the cover tree is used.

Section 2 lays out the required background, notation, and introduces the cover tree and its associated theoretical properties. Readers familiar with the cover tree literature and dual-tree algorithms (especially Curtin et al., 2013b) may find that section to be review. Following that, we introduce an intuitive measure of cover tree imbalance, an important property for understanding the runtime of dual-tree algorithms, in Section 3. This measure of imbalance is then used to prove the main result of the paper in Section 4, which is a worst-case runtime bound for generalized dual-tree algorithms. We apply this result to three specific problems: nearest neighbor search (Section 5), approximate kernel density estimation (Section 6), and range search / range count (Section 7), showing linear runtime

Symbol	Description
$\mathscr{N}$	A tree node
$\mathscr{C}_i$	Set of child nodes of $\mathscr{N}_i$
$\mathscr{P}_i$	Set of points held in $\mathscr{N}_i$
$\mathscr{D}_i^n$	Set of descendant nodes of $\mathscr{N}_i$
$\mathscr{D}_i^p$	Set of points contained in $\mathscr{N}_i$ and $\mathscr{D}_i^n$
μ_i	Center of $\mathscr{N}_i$
λ_i	Furthest descendant distance from μ_i

Table 1: Notation for trees. See Curtin et al. (2013b) for details.

bounds for each of those algorithms. Each of these bounds is an improvement on the state-of-the-art, and in the case of range search, is the first such bound. Despite the intuition this provides for the scaling properties of all dual-tree algorithms[1], it must be kept in mind that these worst-case bounds only apply to dual-tree algorithms that use the cover tree and the standard cover tree traversal.

2. Preliminaries

For simplicity, the algorithms considered in this paper will be presented in a tree-independent context, as in Curtin et al. (2013b), but the only type of tree we will consider is the cover tree (Beygelzimer et al., 2006), and the only type of traversal we will consider is the cover tree pruning dual-tree traversal, which we will describe later.

As we will be making heavy use of trees, we must establish notation (taken from Curtin et al., 2013b). The notation we will be using is defined in Table 1.

2.1 The Cover Tree

The cover tree is a leveled hierarchical data structure originally proposed for the task of nearest neighbor search by Beygelzimer et al. (2006). Each node $\mathscr{N}_i$ in the cover tree is associated with a single point p_i. An adequate description is given in their work (we have adapted notation slightly):

> A *cover tree* $\mathscr{T}$ on a dataset S is a leveled tree where each level is a "cover" for the level beneath it. Each level is indexed by an integer scale s_i which decreases as the tree is descended. Every *node* in the tree is associated with a point in S. Each *point* in S may be associated with multiple nodes in the tree; however, we require that any point appears at most once in every level. Let C_{s_i} denote the set of points in S associated with the nodes at level s_i. The cover tree obeys the following invariants for all s_i:

1. Dual-tree algorithms using *kd*-trees and other types of trees have been observed to empirically scale linearly for tasks that take quadratic time without the use of trees; see the empirical results of Gray and Moore (2001); March et al. (2010); Vladymyrov and Carreira-Perpinán (2014); Klaas et al. (2006); Gray and Moore (2003).

- *(Nesting).* $C_{s_i} \subset C_{s_i-1}$. This implies that once a point $p \in S$ appears in C_{s_i} then *every* lower level in the tree has a node associated with p.
- *(Covering tree).* For every $p_i \in C_{s_i-1}$, there exists a $p_j \in C_{s_i}$ such that $d(p_i, p_j) < 2^{s_i}$ and the node in level s_i associated with p_j is a parent of the node in level $s_i - 1$ associated with p_i.
- *(Separation).* For all distinct $p_i, p_j \in C_{s_i}$, $d(p_i, p_j) > 2^{s_i}$.

As a consequence of this definition, if there exists a node $\mathscr{N}_i$, containing the point p_i at some scale s_i, then there will also exist a self-child node $\mathscr{N}_{ic}$ containing the point p_i at scale $s_i - 1$ which is a child of $\mathscr{N}_i$. In addition, every descendant point of the node $\mathscr{N}_i$ is contained within a ball of radius 2^{s_i+1} centered at the point p_i; therefore, $\lambda_i = 2^{s_i+1}$ and $\mu_i = p_i$ (Table 1).

Note that the cover tree may be interpreted as an infinite-leveled tree, with C_∞ containing only the root point, $C_{-\infty} = S$, and all levels between defined as above. Beygelzimer et al. (2006) find this representation (which they call the *implicit* representation) easier for description of their algorithms and some of their proofs. But clearly, this is not suitable for implementation; hence, there is an *explicit* representation in which all nodes that have only a self-child are coalesced upwards (that is, the node's self-child is removed, and the children of that self-child are taken to be the children of the node). Figure 1 shows each of the levels of an example cover tree (in the explicit representation) on a simple six-point dataset.

In this work, we consider only the explicit representation of a cover tree, and do not concern ourselves with the details of tree construction[2].

2.2 Expansion Constant

The explicit representation of a cover tree has a number of useful theoretical properties based on the expansion constant (Karger and Ruhl, 2002); we restate its definition below.

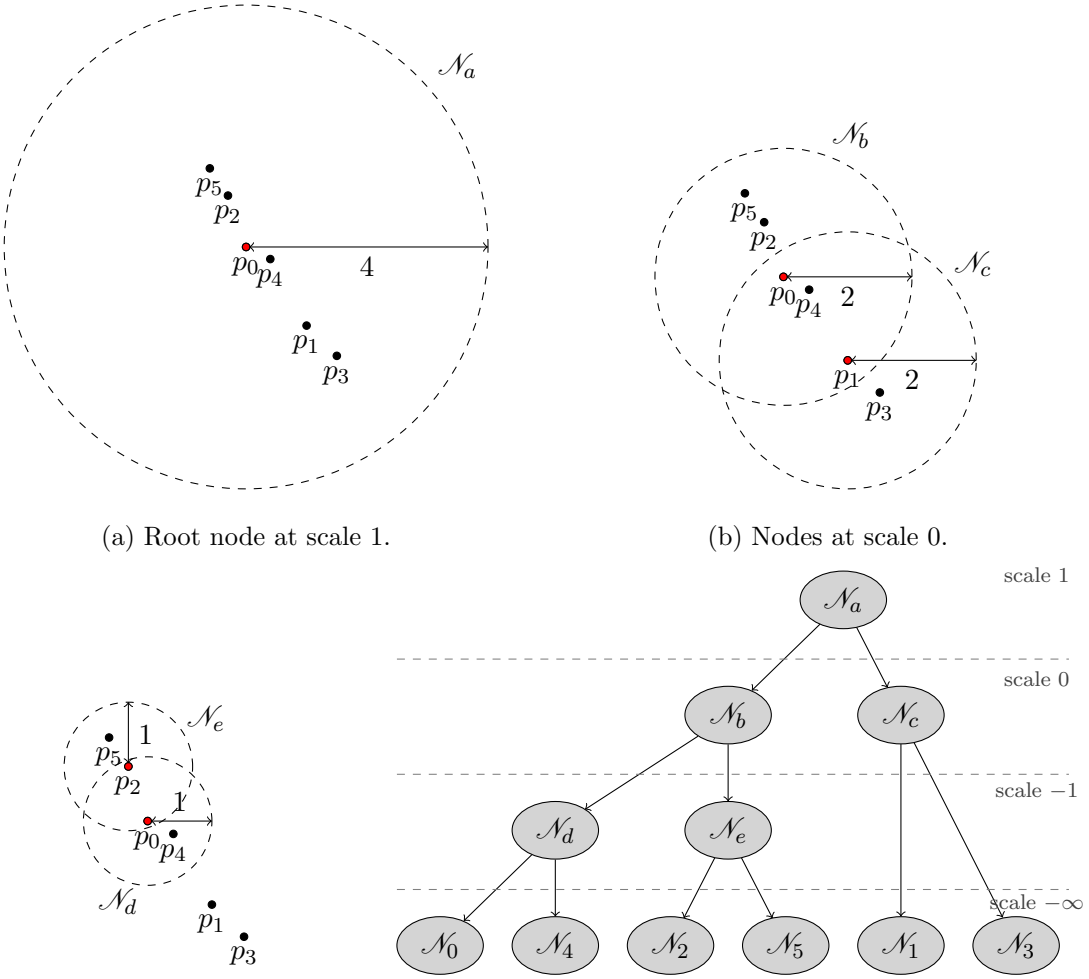
Definition 1 *Let $B_S(p, \Delta)$ be the set of points in S within a closed ball of radius Δ around some $p \in S$ with respect to a metric d: $B_S(p, \Delta) = \{r \in S : d(p, r) \leq \Delta\}$. Then, the* **expansion constant** *of S with respect to the metric d is the smallest $c \geq 2$ such that*

$$|B_S(p, 2\Delta)| \leq c|B_S(p, \Delta)| \;\forall\; p \in S,\; \forall\; \Delta > 0. \tag{1}$$

The expansion constant is used heavily in the cover tree literature. It is, in some sense, a notion of instrinic dimensionality, most useful in scenarios where c is independent of the number of points in the dataset (Karger and Ruhl, 2002; Beygelzimer et al., 2006; Krauthgamer and Lee, 2004; Ram et al., 2009a). Note also that if points in $S \subset \mathcal{H}$ are being drawn according to a stationary distribution $f(x)$, then c will converge to some finite value c_f as $|S| \to \infty$. To see this, define c_f as a generalization of the expansion constant for distributions. $c_f \geq 2$ is the smallest value such that

$$\int_{\mathcal{B}_{\mathcal{H}}(p,2\Delta)} f(x)dx \leq c_f \int_{\mathcal{B}_{\mathcal{H}}(p,\Delta)} f(x)dx \tag{2}$$

2. A batch construction algorithm is given by Beygelzimer et al. (2006), called `Construct`.



(a) Root node at scale 1.

(b) Nodes at scale 0.

(c) Nodes at scale −1.

(d) Abstract representation.

Figure 1: Example cover tree on six points in $\mathcal{R}^2$. (a) $\mathcal{N}_a$ is centered at p_0 with scale 1. (b) $\mathcal{N}_b$ and $\mathcal{N}_c$ are centered at p_0 and p_1, respectively, and have scale 0. (c) $\mathcal{N}_d$ and $\mathcal{N}_e$ are centered at p_0 and p_2, respectively, and have scale -1. The leaves, $\mathcal{N}_0$ through $\mathcal{N}_6$, are centered at each of the six points, with scale $-\infty$ (and therefore radius 0). Note that although node $\mathcal{N}_b$ in subfigure (b) overlaps node $\mathcal{N}_c$, point p_1 only belongs to $\mathcal{N}_c$, not $\mathcal{N}_b$. Note also that this is only one valid cover tree that could be built on the data; other configurations are possible; for instance, selecting a different root point gives different valid cover trees.

for all $p \in \mathcal{H}$ and $\Delta > 0$ such that $\int_{\mathcal{B}_{\mathcal{H}}(p,\Delta)} f(x)dx > 0$, and with $\mathcal{B}_{\mathcal{H}}(p, \Delta)$ defined as the closed ball of radius Δ in the space $\mathcal{H}$.

As a simple example, take $f(x)$ as a uniform spherical distribution in $\mathcal{R}^d$: for any $|x| \leq 1$, $f(x)$ is a constant; for $|x| > 1$, $f(x) = 0$. It is easy to see that c_f in this situation is 2^d, and thus for some dataset S, c must converge to that value as more and more points are added to S. Closed-form solutions for c_f for more complex distributions are less easy to derive; however, empirical speedup results from Beygelzimer et al. (2006) suggest the existence of datasets where c is not strongly dependent on d. For instance, the `covtype` dataset has 54 dimensions but the expansion constant is much smaller than other, lower-dimensional datasets.

There are some other important observations about the behavior of c. Adding a single point to S may increase c arbitrarily: consider a set S distributed entirely on the surface of a unit hypersphere. If one adds a single point at the origin, producing the set S', then c explodes to $|S'|$ whereas before it may have been much smaller than $|S|$. Adding a single point may also decrease c significantly. Suppose one adds a point arbitrarily close to the origin to S'; now, the expansion constant will be $|S'|/2$. Both of these situations are degenerate cases not commonly encountered in real-world behavior; we discuss them in order to point out that although we can bound the behavior of c as $|S| \to \infty$ for S from a stationary distribution, we are not able to easily say much about its convergence behavior.

The expansion constant can be used to show a few useful bounds on various properties of the cover tree; we restate these results below, given some cover tree built on a dataset S with expansion constant c and $|S| = N$:

- **Width bound:** no cover tree node has more than c^4 children (Lemma 4.1, Beygelzimer et al., 2006).
- **Depth bound:** the maximum depth of any node is $O(c^2 \log N)$ (Lemma 4.3, Beygelzimer et al., 2006).
- **Space bound:** a cover tree has $O(N)$ nodes (Theorem 1, Beygelzimer et al., 2006).

Lastly, we introduce a convenience lemma of our own which is a generalization of the packing arguments used by Beygelzimer et al. (2006). This is a more flexible version of their argument.

Lemma 1 *Consider a dataset S with expansion constant c and a subset $C \subseteq S$ such that every two distinct points in C are separated by at least δ. Then, for any point p (which may or may not be in S), and any radius $\rho\delta > 0$:*

$$|B_S(p, \rho\delta) \cap C| \leq c^{2+\lceil \log_2 \rho \rceil}. \tag{3}$$

Proof The proof is based on the packing argument from Lemma 4.1 in Beygelzimer et al. (2006). Consider two cases: first, let $d(p, p_i) > \rho\delta$ for any $p_i \in S$. In this case, $B_S(p, \rho\delta) = \emptyset$ and the lemma holds trivially. Otherwise, let $p_i \in S$ be a point such that $d(p, p_i) \leq \rho\delta$. Observe that $B_S(p, \rho\delta) \subseteq B_S(p_i, 2\rho\delta)$. Also, $|B_S(p_i, 2\rho\delta)| \leq c^{2+\lceil \log_2 \rho \rceil} |B_S(p_i, \delta/2)|$ by the

definition of the expansion constant. Because each point in C is separated by δ, the number of points in $B_S(p, \rho\delta) \cap C$ is bounded by the number of disjoint balls of radius $\delta/2$ that can be packed into $B_S(p, \rho\delta)$. In the worst case, this packing is perfect, and

$$|B_S(p, \rho\delta)| \leq \frac{|B_S(p_i, 2\rho\delta)|}{|B_S(p_i, \delta/2)|} \leq c^{2+\lceil \log_2 \rho \rceil}. \tag{4}$$

■

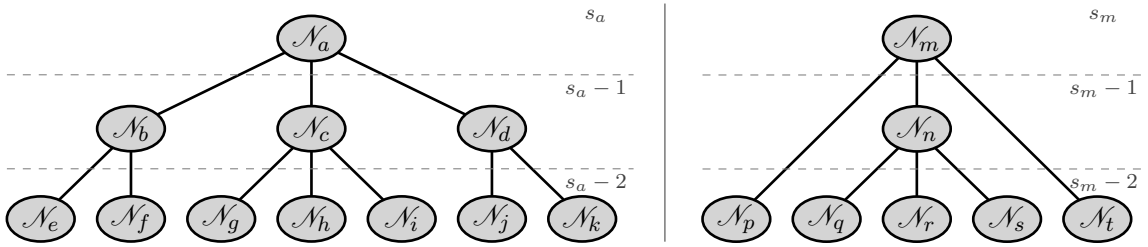
3. Tree Imbalance

It is well-known that imbalance in trees leads to degradation in performance; for instance, a *kd*-tree node with every descendant in its left child except one is effectively useless. A *kd*-tree full of nodes like this will perform abysmally for nearest neighbor search, and it is not hard to generate a pathological dataset that will cause a *kd*-tree of this sort.

This sort of imbalance applies to all types of trees, not just *kd*-trees. In our situation, we are interested in a better understanding of this imbalance for cover trees, and thus endeavor to introduce a more formal measure of imbalance which is correlated with tree performance. Numerous measures of tree imbalance have already been established; one example is that proposed by Colless (1982), and another is Sackin's index (Sackin, 1972), but we aim to capture a different measure of imbalance that uses the leveled structure of the cover tree.

We already know each node in a cover tree is indexed with an integer level (or scale). In the explicit representation of the cover tree, each non-leaf node has children at a lower level. But these children need not be strictly one level lower; see Figure 2. In Figure 2a, each cover tree node has children that are strictly one level lower; we will refer to this as a *perfectly balanced cover tree*. Figure 2b, on the other hand, contains the node $\mathscr{N}_m$ which has two children with scale two less than s_m. We will refer to this as an *imbalanced cover tree*. Note that in our definition, the balance of a cover tree has nothing to do with differing number of descendants in each child branch but instead only missing levels.

An imbalanced cover tree can happen in practice, and in the worst cases, the imbalance may be far worse than the simple graphs of Figure 2. Consider a dataset with a single



(a) Balanced cover tree.

(b) Imbalanced cover tree.

Figure 2: Balanced and imbalanced cover trees.

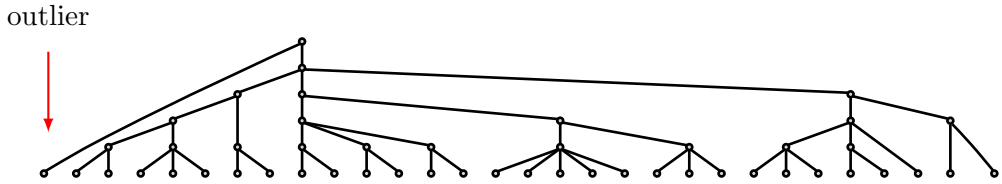


Figure 3: Single-outlier cover tree.

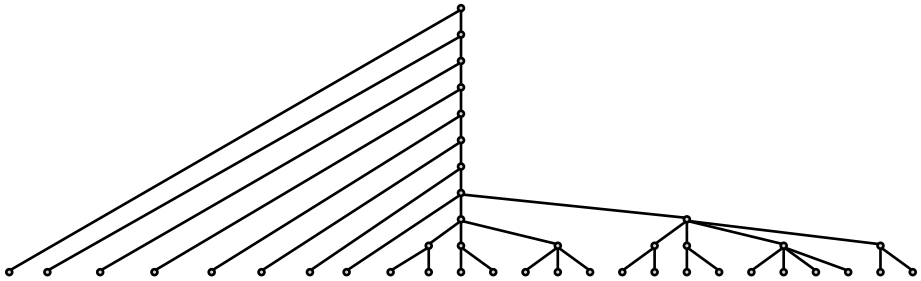


Figure 4: A multiple-outlier cover tree.

outlier which is very far away from all of the other points[3]. Figure 3 shows what happens in this situation: the root node has two children; one of these children has only the outlier as a descendant, and the other child has the rest of the points in the dataset as a descendant. In fact, it is easy to find datasets with a handful of outliers that give rise to a chain-like structure at the top of the tree: see Figure 4 for an illustration[4].

A tree that has this chain-like structure all the way down, which is similar to the *kd*-tree example at the beginning of this section, is going to perform horrendously; motivated by this observation, we define a measure of tree imbalance.

Definition 2 *The cover node imbalance $I_n(\mathscr{N}_i)$ for a cover tree node $\mathscr{N}_i$ with scale s_i in the cover tree $\mathscr{T}$ is defined as the cumulative number of missing levels between the node and its parent $\mathscr{N}_p$ (which has scale s_p). If the node is a leaf (that is, $s_i = -\infty$), then the number of missing levels is defined as the difference between s_p and $s_{\min} - 1$ where $s_{\min}$ is the smallest scale of a non-leaf node in $\mathscr{T}$. If $\mathscr{N}_i$ is the root of the tree, then the cover node imbalance is 0. Explicitly written, this calculation is*

$$I_n(\mathscr{N}_i) = \begin{cases} s_p - s_i - 1 & \text{if } \mathscr{N}_i \text{ is not a leaf and not the root node} \\ \max(s_p - s_{\min} - 1,\ 0) & \text{if } \mathscr{N}_i \text{ is a leaf} \\ 0 & \text{if } \mathscr{N}_i \text{ is the root node.} \end{cases} \tag{5}$$

3. Note also that for an outlier sufficiently far away, the expansion constant is $N - 1$, so we should expect poor performance with the cover tree anyway.
4. As a side note, this behavior is not limited to cover trees, and can happen to mean-split *kd*-trees too, especially in higher dimensions. In addition, for this scenario to arise with cover trees, it must be true that $c \sim O(N)$.

		Imbalance		
Dataset	d	$N = 5k$	$N = 50k$	$N = 500k$
lcdm	3	4.48	5.15	5.24
sdss	4	2.17	2.81	2.97
power	7	5.41	6.46	4.50
susy	18	0.74	0.76	0.86
randu	10	0.23	0.22	0.59
higgs	29	0.99	1.68	1.56
covertype	54	1.322	1.766	2.495
mnist	784	0.99	1.67	2.09

Table 2: Empirically calculated tree imbalances, normalized by N.

This simple definition of cover node imbalance is easy to calculate, and using it, we can generalize to a measure of imbalance for the full tree.

Definition 3 *The cover tree imbalance $I_t(\mathscr{T})$ for a cover tree $\mathscr{T}$ is defined as the cumulative number of missing levels in the tree. This can be expressed as a function of cover node imbalances easily:*

$$I_t(\mathscr{T}) = \sum_{\mathscr{N}_i \in \mathscr{T}} I_n(\mathscr{N}_i). \tag{6}$$

A perfectly balanced cover tree $\mathscr{T}_b$ with no missing levels has imbalance $I_t(\mathscr{T}_b) = 0$ (for instance, Figure 2a). A worst-case cover tree $\mathscr{T}_w$ which is entirely a chain-like structure with maximum scale $s_{\max}$ and minimum scale $s_{\min}$ will have imbalance $I_t(\mathscr{T}_w) \sim N(s_{\max} - s_{\min})$. Because of this chain-like structure, each level has only one node and thus there are at least N levels; or, $s_{\max} - s_{\min} \geq N$, meaning that in the worst case the imbalance is quadratic in N.[5]

However, for most real-world datasets with the cover tree implementation in **mlpack** (Curtin et al., 2013a) and the reference implementation (Beygelzimer et al., 2006), the tree imbalance is near-linear with the number of points. We have constructed cover trees on N uniformly subsampled points from a variety of datasets and calculated the imbalance; see Table 2 for the results. Ten trials were performed for each dataset and each N, and the mean imbalance is given. These results are normalized with respect to N, for which the values of 5000, 50000, and 500000 were chosen. The 'power', 'susy', 'higgs', and 'covertype' datasets are found in the UCI Machine Learning Repository (Bache and Lichman, 2013), the 'mnist' dataset is from LeCun et al. (2000), the 'lcdm' and 'sdss' datasets are Sloan Digital Sky Survey data (Adelman-McCarthy et al., 2008), and the 'randu' dataset is randomly-generated uniformly-distributed data in 10 dimensions. The imbalances on each of these datasets tend to be near-linear.

Currently, no cover tree construction algorithm specifically aims to minimize imbalance.

5. Note that in this situation, $c \sim N$ also.

Algorithm 1 The standard pruning dual-tree traversal for cover trees.

Input: query node $\mathscr{N}_q$, set of reference nodes R
Output: none

$s_r^{\max} \leftarrow \max_{\mathscr{N}_r \in R} s_r$
if $(s_q < s_r^{\max})$ **then**
  {Perform a reference recursion.}
  for each $\mathscr{N}_r \in R$ **do**
    `BaseCase(`p_q`,` p_r`)`
  end for
  $R_r \leftarrow \{\mathscr{N}_r \in R : s_r = s_r^{\max}\}$
  $R_{r-1} \leftarrow \{\mathscr{C}(\mathscr{N}_r) : \mathscr{N}_r \in R_r\} \cup (R \setminus R_r)$
  $R'_{r-1} \leftarrow \{\mathscr{N}_r \in R_{r-1} : \texttt{Score}(\mathscr{N}_q, \mathscr{N}_r) \neq \infty\}$
  recurse with $\mathscr{N}_q$ and R'_{r-1}
else
  {Perform a query recursion.}
  for each $\mathscr{N}_{qc} \in \mathscr{C}(\mathscr{N}_q)$ **do**
    $R' \leftarrow \{\mathscr{N}_r \in R : \texttt{Score}(\mathscr{N}_{qc}, \mathscr{N}_r) \neq \infty\}$
    recurse with $\mathscr{N}_{qc}$ and R'
  end for
end if

4. General Runtime Bound

Perhaps more interesting than measures of tree imbalance is the way cover trees are actually used in dual-tree algorithms. Although cover trees were originally intended for nearest neighbor search (See Algorithm `Find-All-Nearest`, Beygelzimer et al., 2006), they can be adapted to a wide variety of problems: minimum spanning tree calculation (March et al., 2010), approximate nearest neighbor search (Ram et al., 2009b), Gaussian processes posterior calculation (Moore and Russell, 2014), and max-kernel search (Curtin and Ram, 2014) are some examples. Further, through the tree-independent dual-tree algorithm abstraction of Curtin et al. (2013b), other existing dual-tree algorithms can easily be adapted for use with cover trees.

In the framework of tree-independent dual-tree algorithms, all that is necessary to describe a dual-tree algorithm is a point-to-point base case function (`BaseCase()`) and a node-to-node pruning rule (`Score()`). These functions, which are often very straightforward, are then paired with a type of tree and a pruning dual-tree traversal to produce a working algorithm. In later sections, we will consider specific examples.

When using cover trees, the typical pruning dual-tree traversal is an adapted form of the original nearest neighbor search algorithm (see `Find-All-Nearest`, Beygelzimer et al., 2006); this traversal is implemented in both the cover tree reference implementation and in the more flexible **mlpack** library (Curtin et al., 2013a). The problem-independent traversal is given in Algorithm 1 and was originally presented by Curtin and Ram (2014). Initially, it is called with the root of the query tree and a reference set R containing only the root of the reference tree.

This dual-tree recursion is a depth-first recursion in the query tree and a breadth-first recursion in the reference tree; to this end, the recursion maintains one query node $\mathscr{N}_q$ and a reference set R. The set R may contain reference nodes with many different scales; the maximum scale in the reference set is $s_r^{\max}$ (line 3). Each single recursion will descend either the query tree or the reference tree, not both; the conditional in line 4, which determines whether the query or reference tree will be recursed, is aimed at keeping the relative scales of query nodes and reference nodes close.

Keeping the query and reference scales close is both beneficial for the later theory and intuitively reasonable: recursing too quickly in the either the query or reference node will unnecessarily duplicate work. Suppose we recurse many levels down the query tree before recursing down the reference tree, giving us a set of query nodes we are considering. For *each* of these query nodes, we will then need to descend the reference tree. Because these query nodes are close together (with respect to the reference nodes we are considering, which are of larger scale and thus further apart), the pruning decisions at each level of recursion are likely to be the same for each query node. Therefore, recursing too far in the query tree may cause a large amount of duplicated work. The symmetric argument applies for recursing too far in the reference tree before recursing in the query tree. This justifies the approach of keeping the query and reference scales approximately equal.

A query recursion (lines 13–18) is straightforward: for each child $\mathscr{N}_{qc}$ of $\mathscr{N}_q$, the node combinations $(\mathscr{N}_{qc}, \mathscr{N}_r)$ are scored for each $\mathscr{N}_r$ in the reference set R. If possible, these combinations are pruned to form the set R' (line 17) by checking the output of the `Score()` function, and then the algorithm recurses with $\mathscr{N}_{qc}$ and R'.

A reference recursion (lines 4–12) is similar to a query recursion, but the pruning strategy is significantly more complicated. Given R, we calculate R_r, which is the set of nodes in R that have scale $s_r^{\max}$. We expand each node in R_r to construct R_{r-1}: this is the set of children of all nodes in R_r. This set is then combined with $R \setminus R_r$ (that is, the set of references nodes not at scale $s_r^{\max}$) to produce R_{r-1}. Each node in R_{r-1} is then scored and pruned if possible, resulting in the pruned reference set R'_{r-1}. The algorithm then recurses with $\mathscr{N}_q$ and R'_{r-1}.

The reference recursion only recurses into the top-level subset of the reference nodes in order to preserve the separation invariant. It is easy to show that every pair of points held in nodes in R is separated by at least $2^{s_r^{\max}}$:

Lemma 2 *For all distinct nodes $\mathscr{N}_i, \mathscr{N}_j \in R$ (in the context of Algorithm 1) which contain points p_i and p_j, respectively, $d(p_i, p_j) > 2^{s_r^{\max}}$, with $s_r^{\max}$ defined as in line 3.*

Proof This proof is by induction. If $|R| = 1$, such as during the first reference recursion, the result obviously holds. Now consider any reference set R and assume the statement of the lemma holds for this set R, and define $s_r^{\max}$ as the maximum scale of any node in R. Construct the set R_{r-1} as in line 10 of Algorithm 1; if $|R_{r-1}| \leq 1$, then R_{r-1} satisfies the desired property.

Otherwise, take any $\mathscr{N}_i, \mathscr{N}_j$ in R_{r-1}, with points p_i and p_j, respectively, and scales s_i and s_j, respectively. Clearly, if $s_i = s_j = s_r^{\max} - 1$, then by the separation invariant $d(p_i, p_j) > 2^{s_r^{\max}-1}$.

Now suppose that $s_i < s_r^{\max} - 1$. This implies that there exists some implicit cover tree node with point p_i and scale $s_r^{\max} - 1$ (as well as an implicit child of this node p_i with scale

$s_r^{\max} - 2$ and so forth until one of these implicit nodes has child p_i with scale s_i). Because the separation invariant applies to both implicit and explicit representations of the tree, we conclude that $d(p_i, p_j) > 2^{s_r^{\max}} - 1$. The same argument may be made for the case where $s_j < s_r^{\max} - 1$, with the same conclusion.

We may therefore conclude that each point of each node in R_{r-1} is separated by $2^{s_r^{\max}-1}$. Note that $R'_{r-1} \subseteq R_{r-1}$ and that $R \setminus R_{r-1} \subseteq R$ in order to see that this condition holds for all nodes in R'_{r-1}.

Because we have shown that the condition holds for the initial reference set and for any reference set produced by a reference recursion (which will be R at some other level of recursion), we have shown that the statement of the lemma is true.

Note that in this proof, we have considered the child reference set R_{r-1}, not the original reference set R, and shown that with respect to $s_r^{\max}$ as defined by R (not R_{r-1}), all nodes are separated by $2^{s_r^{\max}-1}$. Then, in the frame of the next recursion where $R \leftarrow R_{r-1}$, the lemma will hold, as $s_r^{\max}$ will then be the maximum scale present in R. ■

This observation means that the set of points P held by all nodes in R is always a subset of $C_{s_r^{\max}}$. This fact will be useful in our later runtime proofs.

Next, we develop notions with which to understand the behavior of the cover tree dual-tree traversal when the datasets are of significantly different scale distributions.

If the datasets are similar in scale distribution (that is, inter-point distances tend to follow the same distribution), then the recursion will alternate between query recursions and reference recursions. But if the query set contains points which are, in general, much farther apart than the reference set, then the recursion will start with many query recursions before reaching a reference recursion. The converse case also holds. We are interested in formalizing this notion of scale distribution; therefore, define the following dataset-dependent constants for the query set S_q and the reference set S_r:

- η_q: the largest pairwise distance in S_q
- δ_q: the smallest nonzero pairwise distance in S_q
- η_r: the largest pairwise distance in S_r
- δ_r: the smallest nonzero pairwise distance in S_r

These constants are directly related to the aspect ratio of the datasets; indeed, η_q/δ_q is exactly the aspect ratio of S_q. Further, let us define and bound the top and bottom levels of each tree:

- The *top scale* s_q^T of the query tree $\mathscr{T}_q$ is such that as $\lceil \log_2(\eta_q) \rceil - 1 \leq s_q^T \leq \lceil \log_2(\eta_q) \rceil$.
- The *minimum scale* of the query tree $\mathscr{T}_q$ is defined as $s_q^{\min} = \lceil \log_2(\delta_q) \rceil$.
- The top scale s_r^T of the reference tree $\mathscr{T}_r$ is such that as $\lceil \log_2(\eta_r) \rceil - 1 \leq s_r^T \leq \lceil \log_2(\eta_r) \rceil$.
- The minimum scale of the reference tree $\mathscr{T}_r$ is defined as $s_r^{\min} = \lceil \log_2(\delta_r) \rceil$.

Note that the minimum scale is not the minimum scale of *any* cover tree node (that would be $-\infty$), but the minimum scale of any non-leaf node in the tree.

Suppose that our datasets are of a similar scale distribution: $s_q^T = s_r^T$, and $s_q^{\min} = s_r^{\min}$. In this setting we will have alternating query and reference recursions. But if this is not the case, then we have extra reference recursions before the first query recursion or after the last query recursion (situations where both these cases happen are possible). Motivated by this observation, let us quantify these extra reference recursions:

Lemma 3 *For a dual-tree algorithm with $|S_q| \sim |S_r| \sim O(N)$ using cover trees and the traversal given in Algorithm 1, the number of extra reference recursions that happen before the first query recursion is bounded by*

$$\min\left(O(N), \log_2(\eta_r/\eta_q) - 1\right). \tag{7}$$

Proof The first query recursion happens once $s_q \geq s_r^{\max}$. The number of reference recursions before the first query recursion is then bounded as the number of levels in the reference tree between s_r^T and s_q^T that have at least one explicit node. Because there are $O(N)$ nodes in the reference tree, the number of levels cannot be greater than $O(N)$ and thus the result holds.

The second bound holds by applying the definitions of s_r^T and s_q^T to the expression $s_r^T - s_q^T - 1$:

$$\begin{aligned} s_r^T - s_q^T - 1 &\leq \lceil \log_2(\eta_r) \rceil - (\lceil \log_2(\eta_q) \rceil - 1) - 1 & (8) \\ &\leq \log_2(\eta_r) + 1 - \log_2(\eta_q) & (9) \end{aligned}$$

which gives the statement of the lemma after applying logarithmic identities. ■

Note that the $O(N)$ bound may be somewhat loose, but it suffices for our later purposes. Now let us consider the other case:

Lemma 4 *For a dual-tree algorithm with $|S_q| \sim |S_r| \sim O(N)$ using cover trees and the traversal given in Algorithm 1, the number of extra reference recursions that happen after the last query recursion is bounded by*

$$\max\left(\min\left(O(N \log_2(\delta_q/\delta_r)), O(N^2)\right),\ 0\right). \tag{10}$$

For convenience, we define a term that encapsulates this bound.

Definition 4 *Define θ as a bound on the number of extra reference recursions that happen after the last query recursion. Then,*

$$\theta = \max\left\{\min\left(O(N \log_2(\delta_q/\delta_r)), O(N^2)\right),\ 0\right\}. \tag{11}$$

Proof Our goal here is to count the number of reference recursions after the final query recursion at level $s_q^{\min}$; the first of these reference recursions is at scale $s_r^{\max} = s_q^{\min}$. Because query nodes are not pruned in this traversal, each reference recursion we are counting will be duplicated over the whole set of $O(N)$ query nodes. The first part of the bound follows by observing that $s_q^{\min} - s_r^{\min} \leq \lceil \log_2(\delta_q) \rceil - \lceil \log_2(\delta_r) \rceil - 1 \leq \log_2(\delta_q / \delta_r)$.

The second part follows by simply observing that there are $O(N)$ reference nodes. ■

These two previous lemmas allow us a better understanding of what happens as the reference set and query set become different. Lemma 3 shows that the number of extra recursions caused by a reference set with larger pairwise distances than the query set (η_r larger than η_q) is modest; on the other hand, Lemma 4 shows that for each extra level in the reference tree below $s_q^{\min}$, $O(N)$ extra recursions are required. Using these lemmas and this intuition, we will prove general runtime bounds for the cover tree traversal.

Theorem 1 *Given a reference set S_r of size $O(N)$ with an expansion constant c_r and a set of queries S_q of size $O(N)$, a standard cover tree based dual-tree algorithm (Algorithm 1) takes*

$$O\left(c_r^4 |R^*| \chi \psi (N + I_t(\mathscr{T}_q) + \theta)\right) \tag{12}$$

time, where $|R^*|$ is the maximum size of the reference set R (line 1) during the dual-tree recursion, χ is the maximum possible runtime of `BaseCase()`*, ψ is the maximum possible runtime of* `Score()`*, and θ is defined as in Lemma 4.*

Proof First, split the algorithm into two parts: reference recursions (lines 4–12) and query recursions (lines 13–18). The runtime of the algorithm is the runtime of a reference recursion times the total number of reference recursions plus the total runtime of all query recursions.

Consider a reference recursion (lines 4–12). Define R^* to be the largest set R for any scale $s_r^{\max}$ and any query node $\mathscr{N}_q$ during the course of the algorithm; then, it is true that $|R| \leq |R^*|$. The work done in the base case loop from lines 6–8 is thus $O(\chi |R|) \leq O(\chi |R^*|)$. Then, lines 10 and 11 take $O(c_r^4 \psi |R|) \leq O(c_r^4 \psi |R^*|)$ time, because each reference node has up to c_r^4 children. So, one full reference recursion takes $O(c_r^4 \psi \chi |R^*|)$ time.

Now, note that there are $O(N)$ nodes in $\mathscr{T}_q$. Thus, line 17 is visited $O(N)$ times. The amount of work in line 16, like in the reference recursion, is bounded as $O(c_r^4 \psi |R^*|)$. Therefore, the total runtime of all query recursions is $O(c_r^4 \psi |R^*| N)$.

Lastly, we must bound the total number of reference recursions. Reference recursions happen in three cases: *(1)* $s_r^{\max}$ is greater than the scale of the root of the query tree (no query recursions have happened yet); *(2)* $s_r^{\max}$ is less than or equal to the scale of the root of the query tree, but is greater than the minimum scale of the query tree that is not $-\infty$; *(3)* $s_r^{\max}$ is less than the minimum scale of the query tree that is not $-\infty$.

First, consider case *(1)*. Lemma 3 shows that the number of reference recursions of this type is bounded by $O(N)$. Although there is also a bound that depends on the sizes of the datasets, we only aim to show a linear runtime bound, so the $O(N)$ bound is sufficient here.

Next, consider case *(2)*. In this situation, each query recursion implies at least one reference recursion before another query recursion. For some query node $\mathscr{N}_q$, the exact number of reference recursions before the children of $\mathscr{N}_q$ are recursed into is bounded above

by $I_n(\mathscr{N}_q) + 1$: if $\mathscr{N}_q$ has imbalance 0, then it is exactly one level below its parent, and thus there is only one reference recursion. On the other hand, if $\mathscr{N}_q$ is many levels below its parent, then it is possible that a reference recursion may occur for each level in between; this is a maximum of $I_n(\mathscr{N}_q) + 1$.

Because each query node in $\mathscr{T}_q$ is recursed into once, the total number of reference recursions before each query recursion is

$$\sum_{\mathscr{N}_q \in \mathscr{T}_q} I_n(\mathscr{N}_q) + 1 = I_t(\mathscr{T}_q) + O(N) \tag{13}$$

since there are $O(N)$ nodes in the query tree.

Lastly, for case *(3)*, we may refer to Lemma 4, giving a bound of θ reference recursions in this case.

We may now combine these results for the runtime of a query recursions with the total number of reference recursions in order to give the result of the theorem:

$$O\left(c_r^4 |R^*| \psi\chi \left(N + I_t(\mathscr{T}_q) + \theta\right)\right) + O\left(c_r^4 |R^*| \psi N\right) \sim O\left(c_r^4 |R^*| \psi\chi \left(N + I_t(\mathscr{T}_q) + \theta\right)\right). \tag{14}$$

■

When we consider the monochromatic case (where $S_q = S_r$), the results trivially simplify.

Corollary 1 *Given the situation of Theorem 1 but with $S_q = S_r = S$ so that $c_q = c_r = c$ and $\mathscr{T}_q = \mathscr{T}_r = \mathscr{T}$, a dual-tree algorithm using the standard cover tree traversal (Algorithm 1) takes*

$$O\left(c^4 |R^*| \chi\psi \left(N + I_t(\mathscr{T})\right)\right) \tag{15}$$

time, where $|R^*|$ is the maximum size of the reference set R (line 1) during the dual-tree recursion, χ is the maximum possible runtime of `BaseCase()`*, and ψ is the maximum possible runtime of* `Score()`*.*

An intuitive understanding of these bounds is best achieved by first considering the monochromatic case (this case arises, for instance, in all-nearest-neighbor search). The linear dependence on N arises from the fact that all query nodes must be visited. The dependence on the reference tree, however, is encapsulated by the term $c^4|R^*|$, with $|R^*|$ being the maximum size of the reference set R; this value must be derived for each specific problem. The poor performance of trees on datasets with large c (or, in the worst case, $c \sim N$) is then captured in both of those terms. These datasets for which trees perform poorly may also have a high cover tree imbalance $I_t(\mathscr{T})$; the linear dependence of runtime on imbalance is thus sensible for datasets where trees perform well.

The bichromatic case ($S_q \neq S_r$) is a slightly more complex result which deserves a bit more attention. The intuition for all terms except θ remain virtually the same.

The term θ captures the effect of query and reference datasets with different widths, and has one unfortunate corner case: when $\delta_q > \eta_r$, then the query tree must be entirely descended before any reference recursion. This results in a bound of the form $O(N \log(\eta_r/\delta_r))$,

or $O(N^2)$ (see Lemma 4). This is because the reference tree must be descended separately for each query point.

The quantity $|R^*|$ bounds the amount of work that needs to be done for each recursion. In the worst case, $|R^*|$ can be N. However, dual-tree algorithms rely on branch-and-bound techniques to prune away work (lines 11 and 16 in Algorithm 1). A small value of $|R^*|$ will imply that the algorithm is extremely successful in pruning away work. An (upper) bound on $|R^*|$ (and the algorithm's success in pruning work) will depend on the problem and the data. As we will show, bounding $|R^*|$ is often possible. For many dual-tree algorithms, $\chi \sim \psi \sim O(1)$; often, cached sufficient statistics (Moore, 2000) can enable $O(1)$ runtime implementations of `BaseCase()` and `Score()`.

These results hold for any dual-tree algorithm regardless of the problem. Hence, the runtime of any dual-tree algorithm can be bounded no more tightly than $O(N)$ with our bound, which matches the intuition that answering $O(N)$ queries will take at least $O(N)$ time. For a particular problem and data, if c_r, $|R^*|$, χ, and ψ are bounded by constants independent of N and θ is no more than linear in N (for large enough N), then the dual-tree algorithm for that problem has a runtime linear in N. Our theoretical result separates out the problem-dependent and the problem-independent elements of the runtime bound, which allows us to simply plug in the problem-dependent bounds in order to get runtime bounds for any dual-tree algorithm without requiring an analysis from scratch.

Our results are similar to that of Ram et al. (2009a), but those results depend on a quantity called the *constant of bichromaticity*, denoted κ, which has unclear relation to cover tree imbalance. The dependence on κ is given as $c_q^{4\kappa}$, which is not a good bound, especially because κ may be much greater than 1 in the bichromatic case (where $S_q \neq S_r$).

The more recent results of Curtin and Ram (2014) are more related to these results, but they depend on the *inverse constant of bichromaticity* ν which suffers from the same problem as κ. Although the dependence on ν is linear (that is, $O(\nu N)$), bounding ν is difficult and it is not true that $\nu = 1$ in the monochromatic case.

The quantity ν corresponds to the maximum number of reference recursions between a single query recursion, and κ corresponds to the maximum number of query recursions between a single reference recursion. The respective proofs that use these constants then apply them as a worst-case measure for the whole algorithm: when using κ, Ram et al. (2009a) assume that *every* reference recursion may be followed by κ query recursions; similarly, Curtin and Ram (2014) assume that *every* query recursion may be followed by ν reference recursions. Here, we have simply used $I_t(\mathscr{T}_q)$ and θ as an exact summation of the total extra reference recursions, which gives us a much tighter bound than ν or κ on the running time of the whole algorithm.

Further, both ν and κ are difficult to empirically calculate and require an entire run of the dual-tree algorithm. On the other hand, bounding $I_t(\mathscr{T}_q)$ (and θ) can be done in one pass of the tree (assuming the tree is already built). Thus, not only is our bound tighter when the cover tree imbalance is sublinear in N, it more closely reflects the actual behavior of dual-tree algorithms, and the constants which it depends upon are straightforward to calculate.

In the following sections, we will apply our results to specific problems and show the utility of our bound in simplifying runtime proofs for dual-tree algorithms.

Algorithm 2 Nearest neighbor search `BaseCase()`

Input: query point p_q, reference point p_r, list of candidate neighbors N and distances D

Output: distance d between p_q and p_r

if $d(p_q, p_r) < D[p_q]$ **and** `BaseCase(`p_q`,` p_r`)` not yet called **then**
  $D[p_q] \leftarrow d(p_q, p_r)$, and $N[p_q] \leftarrow p_r$
end if
return $d(p_q, p_r)$

Algorithm 3 Nearest neighbor search `Score()`

Input: query node $\mathscr{N}_q$, reference node $\mathscr{N}_r$
Output: a score for the node combination $(\mathscr{N}_q, \mathscr{N}_r)$, or ∞ if the combination should be pruned

if $d_{\min}(\mathscr{N}_q, \mathscr{N}_r) < B(\mathscr{N}_q)$ **then**
  return $d_{\min}(\mathscr{N}_q, \mathscr{N}_r)$
end if
return ∞

5. Nearest Neighbor Search

The standard task of nearest neighbor search can be simply described: given a query set S_q and a reference set S_r, for each query point $p_q \in S_q$, find the nearest neighbor p_r in the reference set S_r. The task is well-studied and well-known, and there exist numerous approaches for both exact and approximate nearest neighbor search, including the cover tree nearest neighbor search algorithm due to Beygelzimer et al. (2006). We will consider that algorithm, but in a tree-independent sense as given by Curtin et al. (2013b); this means that to describe the algorithm, we require only a `BaseCase()` and `Score()` function; these are given in Algorithms 2 and 3, respectively. The point-to-point `BaseCase()` function compares a query point p_q and a reference point p_r, updating the list of candidate neighbors for p_q if necessary.

The node-to-node `Score()` function determines if the entire subtree of nodes under the reference node $\mathscr{N}_r$ can improve the candidate neighbors for all descendant points of the query node $\mathscr{N}_q$; if not, the node combination is pruned. The `Score()` function depends on the function $d_{\min}(\cdot, \cdot)$, which represents the minimum possible distance between any two descendants of two nodes. Its definition for cover tree nodes is

$$d_{\min}(\mathscr{N}_q, \mathscr{N}_r) = d(p_q, p_r) - 2^{s_q+1} - 2^{s_r+1}. \tag{16}$$

Given a type of tree and traversal, these two functions store the current nearest neighbor candidates in the array N and their distances in the array D. (See Curtin et al., 2013b, for a more complete discussion of how this algorithm works and a proof of correctness.) The `Score()` function depends on a bound function $B(\mathscr{N}_q)$ which represents the maximum distance that could possibly improve a nearest neighbor candidate for any descendant point

of the query node $\mathscr{N}_q$. The standard bound function $B(\mathscr{N}_q)$ used for cover trees is adapted from Beygelzimer et al. (2006):

$$B(\mathscr{N}_q) := D[p_q] + 2^{s_q+1} \tag{17}$$

In this formulation, the query node $\mathscr{N}_q$ holds the the query point p_q, the quantity $D[p_q]$ is the current nearest neighbor candidate distance for the query point p_q, and 2^{s_q+1} corresponds to the furthest descendant distance of $\mathscr{N}_q$. For notational convenience in the following proof, take $c_{qr} = \max((\max_{p_q \in S_q} c'_r), c_r)$, where c'_r is the expansion constant of the set $S_r \cup \{p_q\}$.

Theorem 2 *Using cover trees, the standard cover tree pruning dual-tree traversal, and the nearest neighbor search* `BaseCase()` *and* `Score()` *as given in Algorithms 2 and 3, respectively, and also given a reference set S_r of size $O(N)$ with expansion constant c_r, and a query set S_q of size $O(N)$, the running time of the algorithm is bounded by $O(c_r^4 c_{qr}^5 (N + I_t(\mathscr{T}_q) + \theta))$ with $I_t(\mathscr{T}_q)$ and θ defined as in Definition 3 and Lemma 4, respectively.*

Proof The running time of `BaseCase()` and `Score()` are clearly $O(1)$. Due to Theorem 1, we therefore know that the runtime of the algorithm is bounded by $O(c_r^4 |R^*| (N + I_t(\mathscr{T}_q) + \theta))$. Thus, the only thing that remains is to bound the maximum size of the reference set, $|R^*|$.

Assume that when R^* is encountered, the maximum reference scale is $s_r^{\max}$ and the query node is $\mathscr{N}_q$. Every node $\mathscr{N}_r \in R^*$ satisfies the property enforced in line 11 that $d_{\min}(\mathscr{N}_q, \mathscr{N}_r) \leq B(\mathscr{N}_q)$. Using the definition of $d_{\min}(\cdot, \cdot)$ and $B(\cdot)$, we expand the equation. Note that p_q is the point held in $\mathscr{N}_q$ and p_r is the point held in $\mathscr{N}_r$. Also, take $\hat{p}_r$ to be the current nearest neighbor candidate for p_q; that is, $D[p_q] = d(p_q, \hat{p}_r)$ and $N[p_q] = \hat{p}_r$. Then,

$$\begin{aligned} d_{\min}(\mathscr{N}_q, \mathscr{N}_r) &\leq B(\mathscr{N}_q) && (18) \\ d(p_q, p_r) &\leq d(p_q, \hat{p}_r) + 2^{s_q+1} + 2^{s_r+1} + 2^{s_q+1} && (19) \\ &\leq d(p_q, \hat{p}_r) + 2(2^{s_r^{\max}+1}) && (20) \end{aligned}$$

where the last step follows because $s_q + 1 \leq s_r^{\max}$ and $s_r \leq s_r^{\max}$. Define the set of points P as the points held in each node in R^* (that is, $P = \{p_r \in \mathscr{P}(\mathscr{N}_r) : \mathscr{N}_r \in R^*\}$). Then, we can write

$$P \subseteq B_{S_r}(p_q, d(p_q, \hat{p}_r) + 2(2^{s_r^{\max}+1})). \tag{21}$$

Suppose that the true nearest neighbor is p_r^* and $d(p_q, p_r^*) > 2^{s_r^{\max}+1}$. Then, p_r^* must be held as a descendant point of some node in R^* which holds some point $\tilde{p}_r$. Using the triangle inequality,

$$d(p_q, \hat{p}_r) \leq d(p_q, \tilde{p}_r) \leq d(p_q, p_r^*) + d(\tilde{p}_r, p_r^*) \leq d(p_q, p_r^*) + 2^{s_r^{\max}+1}. \tag{22}$$

This gives that $P \subseteq B_{S_r \cup \{p_q\}}(p_q, d(p_q, p_r^*) + 3(2^{s_r^{\max}+1}))$. The previous step is necessary: to apply the definition of the expansion constant, the ball must be centered at a point in the set; now, the center (p_q) is part of the set.

$$|B_{S_r \cup \{p_q\}}(p_q, d(p_q, p_r^*) + 3(2^{s_r^{\max}+1}))| \leq |B_{S_r \cup \{p_q\}}(p_q, 4d(p_q, p_r^*))| \quad (23)$$
$$\leq c_{qr}^3 |B_{S_r \cup \{p_q\}}(p_q, d(p_q, p_r^*)/2)| \quad (24)$$

which follows because the expansion constant of the set $S_r \cup \{p_q\}$ is bounded above by c_{qr}. Next, we know that p_r^* is the closest point to p_q in $S_r \cup \{p_q\}$; thus, there cannot exist a point $p_r' \neq p_q \in S_r \cup \{p_q\}$ such that $p_r' \in B_{S_{qr}}(p_q, d(p_q, p_r^*)/2)$ because that would imply that $d(p_q, p_r') < d(p_q, p_r^*)$, which is a contradiction. Thus, the only point in the ball is p_q, and we have that $|B_{S_r \cup \{p_q\}}(p_q, d(p_q, p_r^*)/2)| = 1$, giving the result that $|R| \leq c_{qr}^3$ in this case.

The other case is when $d(p_q, p_r^*) \leq 2^{s_r^{\max}+1}$, which means that $d(p_q, \hat{p}_r) \leq 2^{s_r^{\max}+2}$. Note that $P \in C_{s_r^{\max}}$, and therefore

$$P \subseteq B_{S_r}(p_q, d(p_q, p_r^*) + 3(2^{s_r^{\max}+1})) \cap C_{s_r^{\max}} \quad (25)$$
$$\subseteq B_{S_r}(p_q, 4(2^{s_r^{\max}+1})) \cap C_{s_r^{\max}}. \quad (26)$$

Every point in $C_{s_r^{\max}}$ is separated by at least $2^{s_r^{\max}}$. Using Lemma 1 with $\delta = 2^{s_r^{\max}}$ and $\rho = 8$ yields that $|P| \leq c_r^5$. This gives the result, because $c_r^5 \leq c_{qr}^5$. ■

In the monochromatic case where $S_q = S_r$[6], the bound is $O(c^9(N + I_t(\mathscr{T}))$ because $c = c_r = c_{qr}$ and $\theta = 0$. For well-behaved trees where $I_t(\mathscr{T}_q)$ is linear or sublinear in N, this represents the current tightest worst-case runtime bound for nearest neighbor search.

6. Approximate Kernel Density Estimation

Ram et al. (2009a) present a clever technique for bounding the running time of approximate kernel density estimation based on the properties of the kernel, when the kernel is shift-invariant and satisfies a few assumptions. We will restate these assumptions and provide an adapted proof using Theorem 1, which gives a tighter bound.

Approximate kernel density estimation is a common application of dual-tree algorithms (Gray and Moore, 2003, 2001). Given a query set S_q, a reference set S_r of size N, and a kernel function $\mathcal{K}(\cdot, \cdot)$, the true kernel density estimate for a query point p_q is given as

$$f^*(p_q) = \sum_{p_r \in S_r} \mathcal{K}(p_q, p_r). \quad (27)$$

In the case of an infinite-tailed kernel $\mathcal{K}(\cdot, \cdot)$, the exact computation cannot be accelerated; thus, attention has turned towards tractable approximation schemes. Two simple schemes for the approximation of $f^*(p_q)$ are well-known: *absolute value approximation* and *relative value approximation*. Absolute value approximation requires that each density estimate $f(p_q)$ is within ϵ of the true estimate $f^*(p_q)$:

$$|f(p_q) - f^*(p_q)| < \epsilon \ \ \forall p_q \in S_q. \quad (28)$$

6. In the monochromatic case, we do not take a point as its own nearest neighbor, so slight modification of `BaseCase()` is necessary. The runtime bound result remains unchanged.

Relative value approximation is a more flexible approximation scheme; given some parameter ϵ, the requirement is that each density estimate is within a relative tolerance of $f^*(p_q)$:

$$\frac{|f(p_q) - f^*(p_q)|}{|f^*(p_q)|} < \epsilon \ \ \forall p_q \in S_q. \tag{29}$$

Kernel density estimation is related to the well-studied problem of kernel summation, which can also be solved with dual-tree algorithms (Lee and Gray, 2006, 2009). In both of those problems, regardless of the approximation scheme, simple geometric observations can be made to accelerate computation: when $\mathcal{K}(\cdot, \cdot)$ is shift-invariant, faraway points have very small kernel evaluations. Thus, trees can be built on S_q and S_r, and node combinations can be pruned when the nodes are far apart while still obeying the error bounds.

In the following two subsections, we will separately consider both the absolute value approximation scheme and the relative value approximation scheme, under the assumption of a shift-invariant kernel $\mathcal{K}(p_q, p_r) = \mathcal{K}(\|p_q - p_r\|)$ which is monotonically decreasing and non-negative. In addition, we assume that there exists some bandwidth h such that $\mathcal{K}(d)$ must be concave for $d \in [0, h]$ and convex for $d \in [h, \infty)$. This assumption implies that the magnitude of the derivative $|\mathcal{K}'(d)|$ is maximized at $d = h$. These are not restrictive assumptions; most standard kernels fall into this class, including the Gaussian, exponential, and Epanechnikov kernels.

6.1 Absolute Value Approximation

A tree-independent algorithm for solving approximate kernel density estimation with absolute value approximation under the previous assumptions on the kernel is given as a `BaseCase()` function in Algorithm 4 and a `Score()` function in Algorithm 5 (a correctness proof can be found in Curtin et al., 2013b). The list f_p holds partial kernel density estimates for each query point, and the list f_n holds partial kernel density estimates for each query node. At the beginning of the dual-tree traversal, the lists f_p and f_n, which are both of size $O(N)$, are each initialized to 0. As the traversal proceeds, node combinations are pruned if the difference between the maximum kernel value $\mathcal{K}(d_{\min}(\mathcal{N}_q, \mathcal{N}_r))$ and the minimum kernel value $\mathcal{K}(d_{\max}(\mathcal{N}_q, \mathcal{N}_r))$ is sufficiently small (line 3). If the node combination can be pruned, then the partial node estimate is updated (line 4). When node combinations cannot be pruned, `BaseCase()` may be called, which simply updates the partial point estimate with the exact kernel evaluation (line 3).

After the dual-tree traversal, the actual kernel density estimates f must be extracted. This can be done by traversing the query tree and calculating $f(p_q) = f_p(p_q) + \sum_{\mathcal{N}_i \in T} f_n(\mathcal{N}_i)$, where T is the set of nodes in $\mathcal{T}_q$ that have p_q as a descendant. Each query node needs to be visited only once to perform this calculation; it may therefore be accomplished in $O(N)$ time.

Note that this version is far simpler than other dual-tree algorithms that have been proposed for approximate kernel density estimation (see, for instance, Gray and Moore, 2003); however, this version is sufficient for our runtime analysis. Real-world implementations, such as the one found in **mlpack** (Curtin et al., 2013a), tend to be far more complex.

Algorithm 4 Approximate kernel density estimation `BaseCase()`

Input: query point p_q, reference point p_r, list of kernel point estimates $\hat{f}_p$
Output: kernel value $\mathcal{K}(p_q, p_r)$
$f_p(p_q) \leftarrow f_p(p_q) + \mathcal{K}(p_q, p_r)$
return $\mathcal{K}(p_q, p_r)$

Algorithm 5 Absolute-value approximate kernel density estimation `Score()`

Input: query node $\mathscr{N}_q$, reference node $\mathscr{N}_r$, list of node kernel estimates $\hat{f}_n$
Output: a score for the node combination $(\mathscr{N}_q, \mathscr{N}_r)$, or ∞ if the combination should be pruned
if $\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) - \mathcal{K}(d_{\max}(\mathscr{N}_q, \mathscr{N}_r)) < \epsilon$ **then**
  $f_n(\mathscr{N}_q) \leftarrow f_n(\mathscr{N}_q) + |\mathscr{D}^p(\mathscr{N}_r)| \left(\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) + \mathcal{K}(d_{\max}(\mathscr{N}_q, \mathscr{N}_r))\right) / 2$
  return ∞
end if
return $\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) - \mathcal{K}(d_{\max}(\mathscr{N}_q, \mathscr{N}_r))$

Theorem 3 *Assume that $\mathcal{K}(\cdot, \cdot)$ is a kernel with bandwidth h satisfying the assumptions of the previous subsection. Then, given a query set S_q of size $O(N)$ and a reference set S_r of size $O(N)$ with expansion constant c_r, and using the approximate kernel density estimation `BaseCase()` and `Score()` as given in Algorithms 4 and 5, respectively, with the traversal given in Algorithm 1, the running time of approximate kernel density estimation for some error parameter ϵ is bounded by $O(c_r^{8+\lceil \log_2 \zeta \rceil}(N + I_t(\mathscr{T}_q) + \theta))$ with $\zeta = -\mathcal{K}'(h)\mathcal{K}^{-1}(\epsilon)\epsilon^{-1}$, $I_t(\mathscr{T}_q)$ defined as in Definition 3, and θ defined as in Lemma 4.*

Proof It is clear that `BaseCase()` and `Score()` both take $O(1)$ time, so Theorem 1 implies the total runtime of the dual-tree algorithm is bounded by $O(c_r^4 |R^*|(N + I_t(\mathscr{T}_q) + \theta))$. Thus, we will bound $|R^*|$ using techniques related to those used by Ram et al. (2009a). The bounding of $|R^*|$ is split into two sections: first, we show that when the scale $s_r^{\max}$ is small enough, R^* is empty. Second, we bound R^* when $s_r^{\max}$ is larger.

The `Score()` function is such that any node in R^* for a given query node $\mathscr{N}_q$ obeys

$$\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) - \mathcal{K}(d_{\max}(\mathscr{N}_q, \mathscr{N}_r)) \geq \epsilon. \tag{30}$$

Thus, we are interested in the maximum possible value $\mathcal{K}(a) - \mathcal{K}(b)$ for a fixed value of $b - a > 0$. Due to our assumptions, the maximum value of $\mathcal{K}'(\cdot)$ is $\mathcal{K}'(h)$; therefore, the maximum possible value of $\mathcal{K}(a) - \mathcal{K}(b)$ is when the interval $[a, b]$ is centered on h. This allows us to say that $\mathcal{K}(a) - \mathcal{K}(b) \leq \epsilon$ when $(b - a) \leq (-\epsilon / \mathcal{K}'(h))$. Note that

$$\begin{aligned} d_{\max}(\mathscr{N}_q, \mathscr{N}_r) - d_{\min}(\mathscr{N}_q, \mathscr{N}_r) &\leq d(p_q, p_r) + 2^{s_r^{\max}+1} - d(p_q, p_r) + 2^{s_r^{\max}+1} && (31) \\ &\leq 2^{s_r^{\max}+2}. && (32) \end{aligned}$$

Therefore, $R^* = \emptyset$ when $2^{s_r^{\max}+2} \leq -\epsilon / \mathcal{K}'(h)$, or when $s_r^{\max} \leq \log_2(-\epsilon / \mathcal{K}'(h)) - 2$. Consider, then, the case when $s_r^{\max} > \log_2(-\epsilon / \mathcal{K}'(h)) - 2$. Because of the pruning rule,

for any $\mathscr{N}_r \in R^*$, $\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) > \epsilon$; we may refactor this by applying definitions to show $d(p_q, p_r) < \mathcal{K}^{-1}(\epsilon) + 2^{s_r^{\max}+1}$. Therefore, bounding the number of points in the set $B_{S_r}(p_q, \mathcal{K}^{-1}(\epsilon) + 2^{s_r^{\max}+1}) \cap C_{s_r^{\max}}$ is sufficient to bound $|R^*|$. For notational convenience, define $\omega = (\mathcal{K}^{-1}(\epsilon)/2^{s_r^{\max}+1}) + 1$, and the statement may be more concisely written as $B_{S_r}(p_q, \omega 2^{s_r^{\max}+1}) \cap C_{s_r^{\max}}$.

Using Lemma 1 with $\delta = 2^{s_r^{\max}}$ and $\rho = 2\omega$ gives $|R^*| = c_r^{3+\lceil \log_2 \omega \rceil}$.

The value ω is maximized when $s_r^{\max}$ is minimized. Using the lower bound on $s_r^{\max}$, ω is bounded as $\omega = -2\mathcal{K}'(h)\mathcal{K}^{-1}(\epsilon)\epsilon^{-1}$. Finally, with $\zeta = -\mathcal{K}'(h)\mathcal{K}^{-1}(\epsilon)\epsilon^{-1}$, we are able to conclude that $|R^*| \leq c_r^{3+\lceil \log_2 (2\zeta) \rceil} = c_r^{4+\lceil \log_2 \zeta \rceil}$. Therefore, the entire dual-tree traversal takes $O(c_r^{8+\lceil \log_2 \zeta \rceil}(N + \theta))$ time.

The postprocessing step to extract the estimates $f(\cdot)$ requires one traversal of the tree $\mathscr{T}_r$; the tree has $O(N)$ nodes, so this takes only $O(N)$ time. This is less than the runtime of the dual-tree traversal, so the runtime of the dual-tree traversal dominates the algorithm's runtime, and the theorem holds. ■

The dependence on ϵ (through ζ) is expected: as $\epsilon \to 0$ and the search becomes exact, ζ diverges both because ϵ^{-1} diverges and also because $\mathcal{K}^{-1}(\epsilon)$ diverges, and the runtime goes to the worst-case $O(N^2)$; exact kernel density estimation means no nodes can be pruned at all.

For the Gaussian kernel with bandwidth σ defined by $\mathcal{K}_g(d) = \exp(-d^2/(2\sigma^2))$, ζ does not depend on the kernel bandwidth; only the approximation parameter ϵ. For this kernel, $h = \sigma$ and therefore $-\mathcal{K}_g'(h) = \sigma^{-1}e^{-1/2}$. Additionally, $\mathcal{K}_g^{-1}(\epsilon) = \sigma\sqrt{2\ln(1/\epsilon)}$. This means that for the Gaussian kernel, $\zeta = \sqrt{(-2\ln\epsilon)/(e\epsilon^2)}$. Again, as $\epsilon \to 0$, the runtime diverges; however, note that there is no dependence on the kernel bandwidth σ. To demonstrate the relationship of runtime to ϵ, see that for a reasonably chosen $\epsilon = 0.05$, the runtime is approximately $O(c_r^{8.89}(N+\theta))$; for $\epsilon = 0.01$, the runtime is approximately $O(c_r^{11.52}(N+\theta))$. For very small $\epsilon = 0.00001$, the runtime is approximately $O(c_r^{22.15}(N+\theta))$.

Next, consider the exponential kernel: $\mathcal{K}_l(d) = \exp(-d/\sigma)$. For this kernel, $h = 0$ (that is, the kernel is always convex), so then $\mathcal{K}_l'(h) = \sigma^{-1}$. Simple algebraic manipulation gives $\mathcal{K}_l^{-1}(\epsilon) = -\sigma\ln\epsilon$, resulting in $\zeta = -\mathcal{K}_l'(h)\mathcal{K}_l^{-1}(\epsilon)\epsilon^{-1} = \epsilon^{-1}\ln\epsilon$. So both the exponential and Gaussian kernels do not exhibit dependence on the bandwidth.

To understand the lack of dependence on kernel bandwidth more intuitively, consider that as the kernel bandwidth increases, two things happen: *(a)* the reference set R becomes empty at larger scales, and *(b)* $\mathcal{K}^{-1}(\epsilon)$ grows, allowing less pruning at higher levels. These effects are opposite, and for the Gaussian and exponential kernels they cancel each other out, giving the same bound regardless of bandwidth.

6.2 Relative Value Approximation

Approximate kernel density estimation using relative-value approximation may be bounded by reducing the absolute-value approximation algorithm (in linear time or less) to relative-value approximation. This is the same strategy as performed by Ram et al. (2009a).

Algorithm 6 Relative-value approximate kernel density estimation `Score()`

1: **Input:** query node $\mathscr{N}_q$, reference node $\mathscr{N}_r$, list of node kernel estimates $\hat{f}_n$
2: **Output:** a score for the node combination $(\mathscr{N}_q, \mathscr{N}_r)$, or ∞ if the combination should be pruned
3: **if** $\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) - \mathcal{K}(d_{\max}(\mathscr{N}_q, \mathscr{N}_r)) < \epsilon \mathcal{K}^{\max}$ **then**
4: $\quad f_n(\mathscr{N}_q) \leftarrow f_n(\mathscr{N}_q) + |\mathscr{D}^p(\mathscr{N}_r)| \left(\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) + \mathcal{K}(d_{\max}(\mathscr{N}_q, \mathscr{N}_r))\right) / 2$
5: $\quad$ **return** ∞
6: **end if**
7: **return** $\mathcal{K}(d_{\min}(\mathscr{N}_q, \mathscr{N}_r)) - \mathcal{K}(d_{\max}(\mathscr{N}_q, \mathscr{N}_r))$

First, we must establish a `Score()` function for relative value approximation. The difference between Equations 28 and 29 is the division by the term $|f^*(p_q)|$. But we can quickly bound $|f^*(p_q)|$:

$$|f^*(p_q)| \geq N\mathcal{K}\left(\max_{p_r \in S_r} d(p_q, p_r)\right). \tag{33}$$

This is clearly true: each point in S_r must contribute more than $\mathcal{K}(\max_{p_r \in S_r} d(p_q, p_r))$ to $f^*(p_q)$. Now, we may revise the relative approximation condition in Equation 29:

$$|f(p_q) - f^*(p_q)| \leq \epsilon \mathcal{K}^{\max} \tag{34}$$

where $\mathcal{K}^{\max}$ is lower bounded by $\mathcal{K}(\max_{p_r \in S_r} d(p_q, p_r))$. Assuming we have some estimate $\mathcal{K}^{\max}$, this allows us to create a `Score()` algorithm, given in Algorithm 6.

Using this, we may prove linear runtime bounds for relative value approximate kernel density estimation.

Theorem 4 *Assume that $\mathcal{K}(\cdot, \cdot)$ is a kernel satisfying the same assumptions as Theorem 3. Then, given a query set S_q and a reference set S_r both of size $O(N)$, it is possible to perform relative value approximate kernel density estimation (satisfying the condition of Equation 29) in $O(N)$ time, assuming that the expansion constant c_r of S_r is not dependent on N.*

Proof It is easy to see that Theorem 3 may be adapted to the very slightly different `Score()` rule of Algorithm 6 while still providing an $O(N)$ bound. With that `Score()` function, the dual-tree algorithm will return relative-value approximate kernel density estimates satisfying Equation 29.

We now turn to the calculation of $\mathcal{K}^{\max}$. Given the cover trees $\mathscr{T}_q$ and $\mathscr{T}_r$ with root nodes $\mathscr{N}_r^R$ and $\mathscr{N}_r^R$, respectively, we may calculate a suitable $\mathcal{K}^{\max}$ value in constant time:

$$\mathcal{K}^{\max} = d_{\max}(\mathscr{N}_q^R, \mathscr{N}_r^R) = d(p_q^R, p_r^R) + 2^{s_q^{\max}+1} + 2^{s_r^{\max}+1}. \tag{35}$$

This proves the statement of the theorem. ■

In this case, we have not shown tighter bounds because the algorithm we have proposed is not useful in practice. For an example of a better relative-value approximate kernel density estimation dual-tree algorithm, see the work of Gray and Moore (2003).

Algorithm 7 Range search `BaseCase()`

Input: query point p_q, reference point p_r, range sets $N[p_q]$ and range $[l, u]$
Output: distance d between p_q and p_r
if $d(p_q, p_r) \in [r_{\min}, r_{\max}]$ **and** `BaseCase(`p_q`,` p_r`)` not yet called **then**
  $S[p_q] \leftarrow S[p_q] \cup \{p_r\}$
end if
return d

Algorithm 8 Range search `Score()`

Input: query node $\mathscr{N}_q$, reference node $\mathscr{N}_r$
Output: a score for the node combination $(\mathscr{N}_q, \mathscr{N}_r)$, or ∞ if the combination should be pruned
if $d_{\min}(\mathscr{N}_q, \mathscr{N}_r) \in [l, u]$ or $d_{\max}(\mathscr{N}_q, \mathscr{N}_r) \in [l, u]$ **then**
  return $d_{\min}(\mathscr{N}_q, \mathscr{N}_r)$
end if
return ∞

7. Range Search and Range Count

In the range search problem, the task is to find the set of reference points

$$S[p_q] = \{p_r \in S_r : d(p_q, p_r) \in [l, u]\} \tag{36}$$

for each query point p_q, where $[l, u]$ is the given range. The range count problem is practically identical, but only the size of the set, $|S[p_q]|$, is desired. Our proof works for both of these algorithms similarly, but we will focus on range search. A `BaseCase()` and `Score()` function are given in Algorithms 7 and 8, respectively (a correctness proof can be found in Curtin et al., 2013b). The sets $N[p_q]$ (for each p_q) are initialized to $\emptyset$ at the beginning of the traversal.

In order to bound the running time of dual-tree range search, we require better notions for understanding the difficulty of the problem. Observe that if the range is sufficiently large, then for every query point p_q, $S[p_q] = S_r$. Clearly, for $S_q \sim S_r \sim O(N)$, this cannot be solved in anything less than quadratic time simply due to the time required to fill each output array $S[p_q]$. Define the maximum result size for a given query set S_q, reference set S_r, and range $[l, u]$ as

$$|S_{\max}| = \max_{p_q \in S_q} |S[p_q]|. \tag{37}$$

Small $|S_{\max}|$ implies an easy problem; large $|S_{\max}|$ implies a difficult problem. For bounding the running time of range search, we require one more notion of difficulty, related to how $|S_{\max}|$ changes due to changes in the range $[l, u]$.

Definition 5 *For a range search problem with query set S_q, reference set S_r, range $[l, u]$, and results $S[p_q]$ for each query point p_q given as*

$$S[p_q] = \{p_r : p_r \in S_r, l \le d(p_q, p_r) \le u\}, \tag{38}$$

define the α-expansion of the range set $S[p_q]$ as the slightly larger set

$$S^{\alpha}[p_q] = \{p_r : p_r \in S_r, (1-\alpha)l \leq d(p_q, p_r) \leq (1+\alpha)u\}. \tag{39}$$

When the α-expansion of the set $S_{\max}$ is approximately the same size as $S_{\max}$, then the problem would not be significantly more difficult if the range $[l, u]$ was increased slightly. Using these notions, then, we may now bound the running time of range search.

Theorem 5 *Given a reference set S_r of size $O(N)$ with expansion constant c_r, and a query set S_q of size $O(N)$, a search range of $[l, u]$, and using the range search* `BaseCase()` *and* `Score()` *as given in Algorithms 7 and 8, respectively, with the standard cover tree pruning dual-tree traversal as given in Algorithm 1, and also assuming that for some $\alpha > 0$,*

$$|S^{\alpha}[p_q] \setminus S[p_q]| \leq C \ \ \forall \, p_q \in S_q, \tag{40}$$

the running time of range search or range count is bounded by

$$O\left(c_r^4 \max\left(c_r^{4+\beta}, |S_{\max}| + C\right)(N + I_t(\mathscr{N}_q) + \theta)\right) \tag{41}$$

with θ defined as in Lemma 4, $\beta = \lceil \log_2(1 + \alpha^{-1}) \rceil$, and $S_{\max}$ as defined in Equation 37.

Proof Both `BaseCase()` (Algorithm 7) and `Score()` (Algorithm 8) take $O(1)$ time. Therefore, using Lemma 1, we know that the runtime of the algorithm is bounded by $O(c_r^4 |R^*|(N + I_t(\mathscr{N}_q) + \theta))$. As with the previous proofs, then, our only task is to bound the maximum size of the reference set, $|R^*|$.

By the pruning rule, for a query node $\mathscr{N}_q$, the reference set R^* is made up of reference nodes $\mathscr{N}_r$ that are within a margin of $2^{s_q+1} + 2^{s_r+1} \leq 2^{s_r^{\max}+2}$ of the range $[l, u]$. Given that p_r is the point in $\mathscr{N}_r$,

$$p_r \in \left(B_{S_r}(p_q, u + 2^{s_r^{\max}+2}) \cap C_{s_r^{\max}}\right) \setminus \left(B_{S_r}(p_q, l - 2^{s_r^{\max}+2}) \cap C_{s_r^{\max}}\right). \tag{42}$$

A bound on the number of elements in this set is a bound on $|R^*|$. First, consider the case where $u \leq \alpha^{-1} 2^{s_r^{\max}+2}$. Ignoring the smaller ball, take $\delta = 2^{s_r^{\max}}$ and $\rho = 4(1 + \alpha^{-1})$ and apply Lemma 1 to produce the bound

$$|R^*| \leq c_r^{4+\lceil \log_2(1+\alpha^{-1}) \rceil}. \tag{43}$$

Now, consider the other case: $u > \alpha^{-1} 2^{s_r^{\max}+1}$. This means

$$B_{S_r}(p_q, u + 2^{s_r^{\max}+1}) \setminus B_{S_r}(p_q, l - 2^{s_r^{\max}+1}) \subseteq B_{S_r}(p_q, (1+\alpha)u) \setminus B_{S_r}(p_q, (1-\alpha)l). \tag{44}$$

This set is necessarily a subset of $S^{\alpha}[p_q]$; by assumption, the number of points in this set is bounded above by $|S_{\max}| + C$. We may then conclude that $|R^*| \leq |S_{\max}| + C$. By taking the maximum of the sizes of $|R^*|$ in both cases above, we obtain the statement of the theorem. ■

This bound displays both the expected dependence on c_r and $|S_{\max}|$. As the largest range set $S_{\max}$ increases in size (with the worst case being $S_{\max} \sim N$), the runtime degenerates to quadratic. But for adequately small $S_{\max}$ the runtime is instead dependent on c_r and the parameter C of the α-expansion of $S_{\max}$. This situation leads to a simplification.

Corollary 2 *For sufficiently small $|S_{\max}|$ and sufficiently small C, the runtime of range search under the conditions of Theorem 5 simplifies to*

$$O(c_r^{8+\beta}(N + I_t(\mathscr{N}_q) + \theta)). \tag{45}$$

In this setting we can more easily consider the relation of the running time to α. Consider $\alpha = (1/3)$; this yields a running time of $O(c^8(N+\theta))$. $\alpha = (1/7)$ yields $O(c^9(N+I_t(\mathscr{N}_q+\theta))$, $\alpha = (1/15)$ yields $O(c^{10}(N + I_t(\mathscr{N}_q) + \theta))$, and so forth. As α gets smaller, the exponent on c gets larger, and diverges as $\alpha \to 0$.

For reasonable runtime it is necessary that the α-expansion of $S_{\max}$ be bounded. This is because the dual-tree recursion must retain reference nodes which may contain descendants in the range set $S[p_q]$ for some query p_q. The parameter C of the α-expansion allows us to bound the number of reference nodes of this type, and if α increases but C remains small enough that Corollary 2 applies, then we are able to obtain tighter running bounds.

8. Conclusion

We have presented a unified framework for bounding the runtimes of dual-tree algorithms that use cover trees and the standard cover tree pruning dual-tree traversal (Algorithm 1). In order to produce an understandable bound, we have introduced the notion of cover tree imbalance; one possible interesting direction of future work is to empirically and theoretically minimize this quantity by way of modified tree construction algorithms; this is likely to provide both tighter runtime bounds and also accelerated empirical results.

Our main result, Theorem 1, allows plug-and-play runtime bounding of these algorithms. We have shown that Theorem 1 is useful for bounding the runtime of nearest neighbor search (Theorem 2), approximate kernel density estimation (Theorem 3), exact range count, and exact range search (Theorem 5). With our contribution, bounding a cover tree dual-tree algorithm is streamlined and only involves bounding the maximum size of the reference set, $|R^*|$.

Acknowledgements

The authors gratefully acknowledge the helpful and insightful comments of the anonymous reviewers.

References

J.K. Adelman-McCarthy, M.A. Agüeros, S.S. Allam, C.A. Prieto, K.S.J. Anderson, S.F. Anderson, J. Annis, N.A. Bahcall, C.A.L. Bailer-Jones, I.K. Baldry, et al. The sixth data

release of the Sloan Digital Sky Survey. *The Astrophysical Journal Supplement Series*, 175(2):297, 2008.

S. Amizadeh, B. Thiesson, and M. Hauskrecht. Variational dual-tree framework for large-scale transition matrix approximation. In *Proceedings of the Twenty-Eighth Annual Conference on Uncertainty in Artificial Intelligence (UAI-12)*, pages 64–73, Catalina Island, 2012.

K. Bache and M. Lichman. UCI Machine Learning Repository, 2013. `http://archive.ics.uci.edu/ml`.

A. Beygelzimer, S.M. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, pages 97–104, Pittsburgh, 2006.

D.H. Colless. Review of 'Phylogenetics: The Theory and Practice of Phylogenetic Systematics', by E.O. Wiley. *Systematic Zoology*, 31:100–104, 1982.

R.R. Curtin and P. Ram. Dual-tree fast exact max-kernel search. *Statistical Analysis and Data Mining*, 7(4):229–253, 2014.

R.R. Curtin, J.R. Cline, N.P. Slagle, W.B. March, P. Ram, N.A. Mehta, and A.G. Gray. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013a.

R.R. Curtin, W.B. March, P. Ram, D.V. Anderson, A.G. Gray, and C.L. Isbell Jr. Tree-independent dual-tree algorithms. In *Proceedings of The 30th International Conference on Machine Learning (ICML '13)*, pages 1435–1443, Atlanta, 2013b.

R.R. Curtin, P. Ram, and A.G. Gray. Fast exact max-kernel search. In *Proceedings of the 13th SIAM International Conference on Data Mining (SDM '13)*, pages 1–9, Philadelphia, 2013c.

R.A. Finkel and J.L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

A.G. Gray and A.W. Moore. N-body problems in statistical learning. In *Advances in Neural Information Processing Systems 13 (NIPS 2000)*, pages 521–527, Vancouver, 2001.

A.G. Gray and A.W. Moore. Nonparametric density estimation: Toward computational tractability. In *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM '03)*, pages 203–211, San Francisco, 2003.

D.R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing (STOC 2002)*, pages 741–750, Montréal, 2002.

M. Klaas, M. Briers, N. De Freitas, A. Doucet, S. Maskell, and D. Lang. Fast particle smoothing: if I had a million particles. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '06)*, pages 25–29, Pittsburgh, 2006.

R. Krauthgamer and J.R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA04)*, pages 798–807, New Orleans, 2004.

Y. LeCun, C. Cortes, and C.J.C. Burges. MNist dataset, 2000. `http://yann.lecun.com/exdb/mnist/`.

D. Lee and A.G. Gray. Faster Gaussian summation: Theory and Experiment. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 281–288, Arlington, 2006.

D. Lee and A.G. Gray. Fast high-dimensional kernel summations using the monte carlo multipole method. *Advances in Neural Information Processing Systems 21 (NIPS 2008)*, pages 929–936, 2009.

W.B. March. *Multi-tree algorithms for computational statistics and physics.* PhD thesis, Georgia Institute of Technology, 2013.

W.B. March, P. Ram, and A.G. Gray. Fast Euclidean minimum spanning tree: algorithm, analysis, and applications. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '10)*, pages 603–612, Washington, D.C., 2010.

W.B. March, A.J. Connolly, and A.G. Gray. Fast algorithms for comprehensive n-point correlation estimates. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*, pages 1478–1486, Beijing, 2012.

A.W. Moore. The Anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 397–405, Stanford, 2000.

D.A. Moore and S.J. Russell. Fast Gaussian process posteriors with product trees. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI-14)*, Quebec City, July 2014.

P. Ram, D. Lee, W.B. March, and A.G. Gray. Linear-time algorithms for pairwise statistical problems. *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 1527–1535, 2009a.

P. Ram, D. Lee, H. Ouyang, and A.G. Gray. Rank-approximate nearest neighbor search: Retaining meaning and speed in high dimensions. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, pages 1536–1544, Vancouver, 2009b.

M.J. Sackin. "Good" and "bad" phenograms. *Systematic Biology*, 21(2):225–226, 1972.

L. Van Der Maaten. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

M. Vladymyrov and M.A. Carreira-Perpinán. Linear-time training of nonlinear low-dimensional embeddings. In *Proceedings of The Seventeenth International Conference on Artificial Intelligence and Statistics, JMLR W&CP (AISTATS 2014)*, volume 33, pages 968–977, 2014.

P. Wang, D. Lee, A.G. Gray, and J.M. Rehg. Fast mean shift with accurate and stable convergence. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS 2007)*, pages 604–611, San Juan, 2007.