Dual-Tree Fast Exact Max-Kernel Search

Ryan R. Curtin and Parikshit Ram

December 11, 2013

Abstract

The problem of max-kernel search arises everywhere: given a query point p_q, a set of reference objects S_r and some kernel $\mathcal{K}$, find $\arg\max_{p_r \in S_r} \mathcal{K}(p_q, p_r)$. Max-kernel search is ubiquitous and appears in countless domains of science, thanks to the wide applicability of kernels. A few domains include image matching, information retrieval, bio-informatics, similarity search, and collaborative filtering (to name just a few). However, there are no generalized techniques for efficiently solving max-kernel search. This paper presents a single-tree algorithm called *single-tree FastMKS* which returns the max-kernel solution for a single query point in provably $O(\log N)$ time (where N is the number of reference objects), and also a dual-tree algorithm (*dual-tree FastMKS*) which is useful for max-kernel search with many query points. If the set of query points is of size $O(N)$, this algorithm returns a solution in provably $O(N)$ time, which is significantly better than the $O(N^2)$ linear scan solution; these bounds are dependent on the expansion constant of the data. These algorithms work for abstract objects, as they *do not* require explicit representation of the points in kernel space. Empirical results for a variety of datasets show up to 5 orders of magnitude speedup in some cases. In addition, we present approximate extensions of the FastMKS algorithms that can achieve further speedups.

1 Max-kernel search

One particularly ubiquitous problem in computer science is that of *max-kernel search*: for a given set S_r of N objects (the *reference set*), a similarity function $\mathcal{K}(\cdot, \cdot)$, and a query object p_q, find the object $p_r \in R$ such that

$$p_r = \arg\max_{p \in S_r} \mathcal{K}(p_q, p). \tag{1}$$

Often, max-kernel search is performed for a large set of query objects S_q.

The most simple approach to this general problem is a linear scan over all the objects in S_r. However, the computational cost of this approach scales linearly with the size of the reference set for a single query, making it computationally prohibitive for large datasets. If $|S_q| = |S_r| = O(N)$, then this approach scales as $O(N^2)$; thus, the approach quickly becomes infeasible for large N.

In our setting we restrict the similarity function $\mathcal{K}(\cdot, \cdot)$ to be a Mercer kernel. As we will see, this is not very restrictive. A Mercer kernel is a positive semidefinite kernel function; these can be expressed as an inner product in some Hilbert space $\mathcal{H}$:

$$\mathcal{K}(x, y) = \langle \varphi(x), \varphi(y) \rangle_{\mathcal{H}}. \tag{2}$$

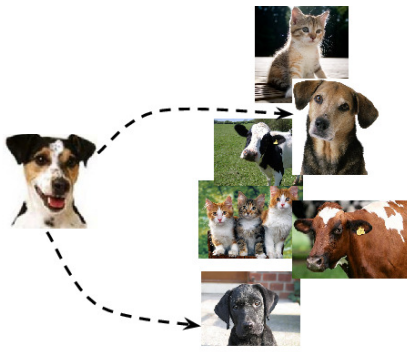


Figure 1: Matching images: an example of max-kernel search.

Often, in practice, the space $\mathcal{H}$ is unknown; thus, the mapping of x to $\mathcal{H}$ ($\varphi(x)$) for any object x is not known. Fortunately, we do not need to know φ because of the renowned "kernel trick"—the ability to evaluate inner products between any pair of objects in the space $\mathcal{H}$ without requiring the explicit representations of those objects.

Because Mercer kernels do not require explicit representations in $\mathcal{H}$, they are ubiquitous and can be devised for any new class of objects, such as images and documents (which can be considered as points in $\mathbb{R}^{\mathcal{D}}$), to more abstract objects like strings (protein sequences [42], text), graphs (molecules [10], brain

neuron activation paths), and time series (music, financial data) [51].

As we mentioned, the max-kernel search problem appears everywhere in computer science and related applications. The widely studied problem of image matching in computer vision is an instance of max-kernel search (Figure 1 presents a simple example). The size of the image sets is continually growing, rendering linear scan computationally prohibitive. Max-kernel search also appears in maximum a posteriori inference [33] as well as collaborative filtering via the widely successful matrix factorization framework [35]. This matrix factorization obtains an accurate representation of the data in terms of user vectors and item vectors, and the desired result—the user preference of an item—is the inner product between the two respective vectors (this is a Mercer kernel). With ever-scaling item sets and millions of users [20], efficient retrieval of recommendations (which is also max-kernel search) is critical for real-world systems.

Finding similar protein/DNA sequences for a query sequence from a large set of biological sequences is also an instance of max-kernel search with biological sequences as the objects and various domain-specific kernels (for example, the p-spectrum kernel [42], the maximal segment match score [1] and the Smith-Waterman alignment score [60][1]).

An efficient max-kernel search algorithm can be directly applied to biological sequence matching. The field of document retrieval—and information retrieval in general—can be easily seen to be an instance of max-kernel search: for some given similarity function, return the document that is most similar to the query. Spell checking systems are an interesting corollary of information retrieval and also an instance of max-kernel search [27].

A special case of max-kernel search is the problem of nearest neighbor search in metric spaces. In this problem, the closest object to the query with respect to a distance metric is sought. The requirement of a distance metric allows numerous efficient methods for exact and approximate nearest neighbor search, including searches based on space partitioning trees [23, 5, 62, 43, 58, 50] and other types of data structures [2, 14, 36, 15]. However, none of these numerous algorithms are suitable for solving generalized max-kernel search, which is the problem we are considering.

Given the wide applicability of kernels, it is desirable to have a *general* method for efficient max-kernel search that is applicable to any class of objects and corresponding Mercer kernels. To this end, we present a method to accelerate exact max-kernel search for any general class of objects and corresponding Mercer kernels. The specific contributions of this document (which is an extension of a previous work [18]) are listed below.

- The first concept for characterizing the hardness of max-kernel search in terms of the concentration of the kernel values in any direction: the *directional concentration*.
- An $O(N \log N)$ algorithm to index any set of N objects *directly in the Hilbert space defined by the kernel* without requiring explicit representations of the objects in this space.
- Novel single-tree and dual-tree branch-and-bound algorithms on the Hilbert space indexing, which can achieve orders of magnitude speedups over linear search.
- Value-approximate, order-approximate, and rank-approximate extensions to the exact max-kernel search algorithms.
- An $O(\log N)$ runtime bound for *exact* max-kernel search for one query with our proposed single-tree algorithm for *any* Mercer kernel.
- An $O(N)$ runtime bound for *exact* max-kernel search for $O(N)$ queries with our proposed dual-tree algorithm for *any* Mercer kernel.

1.1 Related work

Although there are existing techniques for max-kernel search, almost all of them solve the approximate search problem under restricted settings. The most common assumption is that the objects are in some metric space and the kernel function is *shift-invariant*—a monotonic function of the distance between the two objects ($\mathcal{K}(p, q) = f(\|p - q\|)$). One example is the Gaussian radial basis function (RBF) kernel.

For shift-invariant kernels, a tree-based recursive algorithm has been shown to scale to large datasets for maximum a posteriori inference [33]. However, a shift-invariant kernel is only applicable to objects already represented in some metric space. In fact, max-kernel search with a shift-invariant kernel is equivalent to nearest neighbor search in the input space itself, and can be solved directly using existing methods for nearest neighbor search—an easier and better-studied problem. For shift-invariant kernels, the points can be explicitly embedded in some low-dimensional metric space such that the inner product between these representations of any two points approximates their corresponding kernel value [55]. This step takes $O(N\mathcal{D}^2)$ time for $S_r \subset \mathbb{R}^{\mathcal{D}}$ and can

[1] These functions provide matching scores for pairs of sequences and can easily be shown to be Mercer kernels.

be followed by nearest neighbor search on these representations to obtain results for max-kernel search in the setting of a shift-invariant kernel.

This technique of obtaining the explicit embedding of objects in some low-dimensional metric space while approximating the kernel function can also be applied to dot-product kernels [30]. Dot-product kernels produce kernel values between any pair of points by operating a monotonic non-decreasing function on their mutual dot-product ($\mathcal{K}(x, y) = f(\langle x, y \rangle)$). Linear and polynomial kernels are simple examples of dot-product kernels. However, this is only applicable to objects which already are represented in some vector space which allows the computation of the dot-products.

Locality-sensitive hashing (LSH) [24] is widely used for image matching, but only with explicitly representable kernel functions that admit a locality sensitive hashing function [13][2]. Kulis and Grauman [37] apply LSH to solve max-kernel search *approximately* for normalized kernels without any explicit representation. Normalized kernels restrict the self-similarity value to a constant ($\mathcal{K}(x, x) = \mathcal{K}(y, y)\ \forall\ x, y \in S$). The preprocessing time for this locality sensitive hashing is $O(p^3)$ and a single query requires $O(p)$ kernel evaluations. Here p controls the accuracy of the algorithm—larger p implies better approximation; the suggested value for p is $O(\sqrt{N})$ with no rigorous approximation guarantees.

A recent work [56] proposed the first technique for exact max-kernel search using a tree-based branch-and-bound algorithm, but is restricted only to linear kernels and the algorithm does not have any runtime guarantees. The authors suggest a method for extending their algorithm to non-representable spaces with general Mercer kernels, but this requires $O(N^2)$ preprocessing time.

There has also been recent interest in similarity search with Bregman divergences [11], which are non-metrics. Bregman divergences are not directly comparable to kernels, though; they are harder to work with because they are not symmetric like kernels, and are also not as generally applicable to any class of objects as kernel functions. In this paper, we do not address this form of similarity search; Bregman divergences are not Mercer kernels.

1.2 Unnormalized kernels

Some kernels used in machine learning are normalized ($\mathcal{K}(x, x) = \mathcal{K}(y, y)\ \forall\ x, y$); examples include the Gaussian and the cosine kernel. As we have discussed, there already exist techniques to solve the max-kernel search problem with normalized kernels.

However, many common kernels like the linear kernel ($\mathcal{K}(x, y) = x^T y$) and the polynomial kernels ($\mathcal{K}(x, y) = (\alpha + x^T y)^d$) for some offset α and degree d) are not normalized. Many applications require unnormalized kernels:

- In the retrieval of recommendations, the normalized linear kernel will result in inaccurate user-item preference scores.
- In biological sequence matching with domain-specific matching functions, $\mathcal{K}(x, x)$ implicitly corresponds to the presence of genetically valuable letters (such as W, H, or P) or not valuable letters (such as X)[3] in the sequence x. This crucial information is lost in kernel normalization.

Therefore, this paper considers unnormalized kernels. No existing techniques consider un-normalized kernels, and thus no existing techniques can be directly applied to every instance of max-kernel search with general Mercer kernels and any class of objects (Equation 1). Moreover, almost all existing techniques resort to approximate solutions. Not only do our algorithms work for general Mercer kernels instead of just normalized or shift-invariant kernels, but they also provide exact solutions; in addition, extensions to our algorithms for approximation are trivial, and for both the exact and approximate algorithms, we can give asymptotic preprocessing and runtime bounds, as well as rigorous accuracy guarantees for approximate max-kernel searches.

2 Speedups via trees

The introduction of the quad tree in 1974 [21] and *kd*-tree in 1975 [5] for use in nearest neighbor search [22], range search [7], and minimum spanning tree calculation [6] paved the way for numerous algorithms that took advantage of the triangle inequality to eliminate unnecessary calculations.

A tree (or space tree) is a hierarchical structure where each node in the tree corresponds to a certain subset of the dataset it is built on [17]. For a *kd*-tree, this subset is a hyperrectangle. In the context of a problem such as nearest neighbor search, the triangle inequality can be used to place bounds on the minimum and maximum distances between a point p_q and a node $\mathscr{N}_r$: $d_{\min}(p_q, \mathscr{N}_r)$ and $d_{\max}(p_q, \mathscr{N}_r)$,

[2] The Gaussian and cosine kernels admit locality sensitive hashing functions with some modifications.

[3] See the score matrix for letter pairs in protein sequences at http://www.ncbi.nlm.nih.gov/Class/FieldGuide/BLOSUM62.txt.

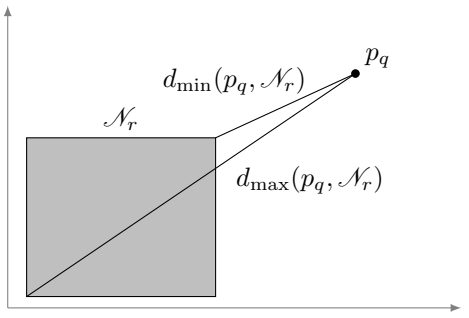


Figure 2: $d_{\min}(p_q, \mathscr{N}_r)$ and $d_{\max}(p_q, \mathscr{N}_r)$ in $\mathbb{R}^2$.

respectively. An example of this bounding can be seen in Figure 2. These bounds, then, can be used to *prune* nodes in the tree, reducing the number of distance computations necessary to find the solution. For instance, in the case of range searching, a node can be pruned if the range $[d_{\min}(q, \mathscr{N}_r), d_{\max}(q, \mathscr{N}_r)]$ does not overlap with the desired range—in that case, there can be no points in $\mathscr{N}_r$ that are in the desired distance range to the query point q. For a better review of this type of approach, see [6] and [17].

Later years saw the introduction of numerous other types of trees: oct-trees [29], metric trees [62], vantage-point trees [64], random projection trees [19], spill trees [43], cover trees [8], cone trees [56]—to name just a few.

Trees have been applied to a variety of problems, in addition to nearest neighbor search, range search, and minimum spanning tree calculation. These problems include approximate nearest neighbor search [2], Gaussian summation [39], particle smoothing [33], Gaussian process regression [59], clustering [34], feature subset selection [52], and mixture model training [48]. More recently, Gray and Moore proposed using a second tree for problems with large query sets [25], such as all-nearest-neighbors and density estimation [26]. This dual-tree approach was then applied to numerous problems: singular value decomposition [28], n-point correlation estimates in astronomy [45], mean shift [63], kernel summation [39, 41, 40], rank-approximate nearest neighbor search [58], and minimum spanning tree calculation [46], as well as numerous others.

The use of a hierarchical space-partitioning approach such as the *kd*-tree gives large speedups. For instance, nearest neighbor search for a single query point with a *kd*-tree runs in expected $O(\log N)$ time (where N is the number of points in the dataset), as opposed to $O(N)$ time for linear scan. Similar results can be shown for other trees and other tasks. The cover tree [8], a more recent tree structure, can be shown to have a worst-time bound of $O(\log N)$ for single-query nearest neighbor search [8] and a total all-nearest-neighbors runtime that scales as $O(N)$ [57]. This is a huge improvement over the linear scan all-nearest-neighbors runtime of $O(N^2)$. Similar runtime bound results have been shown for some dual-tree algorithms when cover trees are chosen as the tree type [58, 45, 46].

These results from the literature make the use of trees an attractive option for solving max-kernel search. Importantly, trees only require a single construction. Thus, we can re-use trees for multiple tasks and amortize the cost of construction over many runs of an algorithm. In addition, once a tree has been constructed, point insertions and deletions are generally cheap. However, as we mentioned earlier, the numerous existing nearest-neighbor search approaches using trees [50, 5, 8] all require a distance metric. In general, a Mercer kernel $K(\cdot, \cdot)$ is not a distance metric—so we must find a novel approach.

3 Analysis of the problem

Remember that a Mercer kernel implies that the kernel value for a pair of objects (x, y) corresponds to an inner product between the vector representation of the objects $(\varphi(x), \varphi(y))$ in some Hilbert space $\mathcal{H}$ (see Equation 2). Hence, every Mercer kernel induces the following metric in $\mathcal{H}$:

$$\begin{aligned} d_{\mathcal{K}}(x, y) &= \|\varphi(x) - \varphi(y)\|_{\mathcal{H}} \\ &= \sqrt{\mathcal{K}(x, x) + \mathcal{K}(y, y) - 2\mathcal{K}(x, y)}. \end{aligned} \quad (3)$$

3.1 Reduction to nearest neighbor search

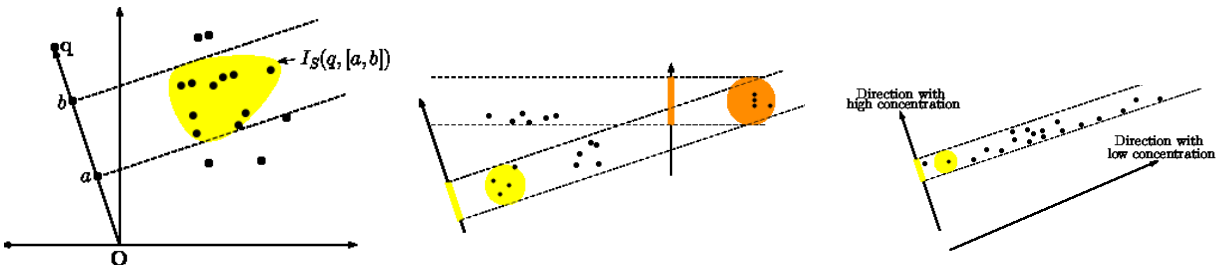
In situations where max-kernel search can be reduced to nearest neighbor search in $\mathcal{H}$, nearest neighbor search methods for general metrics [15] can be used for efficient max-kernel search. This reduction is possible for normalized kernels. The nearest neighbor for a query p_q in $\mathcal{H}$,

$$\arg\min_{p_r \in S_r} d_{\mathcal{K}}(p_q, p_r), \quad (4)$$

is the max-kernel candidate (Equation 1) if $\mathcal{K}(\cdot, \cdot)$ is a normalized kernel. To see this, note that for normalized kernels, $\mathcal{K}(p_q, p_q) = \mathcal{K}(p_r, p_r)$ and thus,

$$d_{\mathcal{K}}(p_q, p_r) = \sqrt{2c - 2\mathcal{K}(p_q, p_r)} \quad (5)$$

where the normalization constant $c = \mathcal{K}(p_q, p_q) = \mathcal{K}(p_r, p_r)$ and is a constant not dependent on p_q or



(a) Projection interval set. (b) Low value of γ. (c) High value of γ.

Figure 3: Concentration of projections.

p_r. Therefore, $d_{\mathcal{K}}(p_q, p_r)$ decreases as $\mathcal{K}(p_q, p_r)$ increases, and so $d_{\mathcal{K}}(\cdot,\cdot)$ is minimized when $\mathcal{K}(\cdot,\cdot)$ is maximized. However, the two problems can have very different answers with unnormalized kernels, because $d_{\mathcal{K}}(p_q, p_r)$ is not guaranteed to decrease as $\mathcal{K}(p_q, p_r)$ increases. As we discussed earlier in Section 1.2, unnormalized kernels are an important class of kernels that we wish to consider. Thus, although a reduction to nearest neighbor search is sometimes possible, it is only under the strict condition of a normalized kernel.

3.2 Hardness of max-kernel search

Even if max-kernel search can be reduced to nearest neighbor search, the problem is still hard ($\Omega(N)$ for a single query) without any assumption on the structure of the metric or the dataset S_r. Here we present one notion of characterizing the hardness in terms of the structure of the metric [31]:

Definition 1. *Let $B_S(p, \Delta)$ be the set of points in S within a ball of closed 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*

$$|B_S(p, 2\Delta)| \leq c|B_S(p, \Delta)| \ \forall\ p \in S,\ \forall\ \Delta > 0.$$

The expansion constant effectively bounds the number of points that could be sitting on the surface of a hyper-sphere of any radius around any point. If c is high, nearest neighbor search could be expensive. A value of $c \sim \Omega(N)$ implies that the search cannot be better than linear scan asymptotically. Under the assumption of a bounded expansion constant, though, nearest-neighbor search methods with sublinear or logarithmic theoretical runtime guarantees have been presented [31, 8, 57].

Now, we extend the concept of the expansion concept in order to characterize the difficulty of max-kernel search.

For a given query p_q and Mercer kernel $\mathcal{K}(\cdot,\cdot)$, the kernel values are proportional to the length of the projections in the direction of $\varphi(p_q)$ in $\mathcal{H}$. While the hardness of nearest-neighbor search depends on how concentrated the surface of spheres are (as characterized by the expansion constant), the hardness of max-kernel search should depend on the distribution of the projections in the direction of the query. This distribution can be characterized using the distribution of points in terms of distances:

> *If two points are close in distance, then their projections in any direction are close as well. However, if two points have close projections in a direction, it is not necessary that the points themselves are close to each other.*

It is to characterize this reverse relationship between points (closeness in projections to closeness in distances) that we present a new notion of concentration in any direction:

Definition 2. *Let $\mathcal{K}(x, y) = \langle \varphi(x), \varphi(y) \rangle_{\mathcal{H}}$ be a Mercer kernel, $d_{\mathcal{K}}(x, y)$ be the induced metric from $\mathcal{K}$ (Equation 3), and let $B_S(p, \Delta)$ denote the closed ball of radius Δ around a point p in $\mathcal{H}$. Also, let*

$$I_S(v, [a, b]) = \{r \in S : \langle v, \varphi(r) \rangle_{\mathcal{H}} \in [a, b]\} \quad (6)$$

be the set of objects in S projected onto a direction v in $\mathcal{H}$ lying in the interval $[a, b]$ along v. Then, the **directional concentration constant** *of S with respect to the Mercer kernel $\mathcal{K}(\cdot,\cdot)$ is defined as the smallest $\gamma \geq 1$ such that $\forall u \in \mathcal{H}$ such that $\|u\|_{\mathcal{H}} = 1$, $\forall p \in S$ and $\forall \Delta > 0$, the set*

$$I_S(u, [\langle u, \varphi(p) \rangle_{\mathcal{H}} - \Delta, \langle u, \varphi(p) \rangle_{\mathcal{H}} + \Delta])$$

can be covered by at most γ balls of radius Δ.

The directional concentration constant is not a measure of the number of points projected into a small interval, but rather a measure of the number of "patches" of the data in a particular direction. For a set of points to be close in projections, there are at most γ subsets of points that are close in distances as well. Consider the set of points $B = I_S(q, [a, b])$ projected onto an interval in some direction (Figure 3(a)). A high value of γ implies that the number of points in B is high but the points are spread out and the number of balls (with diameter $|b - a|$) required to cover all these points is high as well—with each point possibly requiring an individual ball. Figure 3(c) provides one such example. A low value of γ implies that either B has a small size or the size of B is high and B can be covered with a small number of balls (of diameter $|b - a|$). Figure 3(b) is an example of a set with low γ. The directional concentration constant bounds the number of balls of a particular radius required to index points that project onto an interval of size twice the radius.

4 Indexing points in $\mathcal{H}$

Earlier, we discussed the use of space trees for max-kernel search. The first problem, which is the lack of distance metric, is addressed by the induced metric $d_{\mathcal{K}}(\cdot, \cdot)$ in the space $\mathcal{H}$. However, we now have another problem. The standard procedure for constructing *kd*-trees depends on axis-aligned splits along the mean (or median) of a subset of the data in a particular dimension. In $\mathcal{H}$ this does not make sense because we do not have access to the mapping $\varphi(\cdot)$. Thus, *kd*-trees—and any tree that requires knowledge of $\varphi(\cdot)$—cannot be used to index points in $\mathcal{H}$. This includes random projection trees [19] and principal-axis trees [47][4].

Metric trees [54] are a type of space tree that does not require axis-aligned splits. Instead, during construction, metric trees calculate a mean μ for each node [49]. In general, μ is not a point in the dataset the tree is built on. In our situation, we cannot calculate μ because it lies in $\mathcal{H}$ and we do not have access to $\varphi(\cdot)$. However, we can use the kernel trick to avoid calculating μ and evaluate kernels involving μ (assume μ is the mean of node $\mathscr{N}$, and $\mathscr{D}^p(\mathscr{N})$ refers to the set of descendant points of $\mathscr{N}$):

$$\mathcal{K}(q, \mu) = \frac{\sum_{r \in \mathscr{D}^p(\mathscr{N})} \mathcal{K}(q, r)}{|\mathscr{D}^p(\mathscr{N})|}. \tag{7}$$

This type of trick can also be applied to ball trees and some other similar tree structures. However, it is clear that a single kernel evaluation against the mean is now numerous kernel evaluations, making the use of metric or ball trees computationally prohibitive in our setting, for both tree construction in $\mathcal{H}$ and max-kernel search.

Therefore, we consider the cover tree [8], a recently formulated tree that bears some similarity to the ball tree. The tree itself will not be detailed here due to its complexity; consult [8] and [9] for details. In addition, the **mlpack** machine learning library [16] presents a documented reference implementation of cover trees.

The cover tree has the interesting property that explicit object representations are unnecessary for tree construction: the tree can be built entirely with only knowledge of the metric function $d_{\mathcal{K}}(\cdot, \cdot)$ evaluated on points in the dataset. Each node $\mathscr{N}_i$ in the cover tree represents a ball in $\mathcal{H}$ with a known radius λ_i and its center μ_i is a point in the dataset. Thus, we can evaluate the minimum distance between two nodes $\mathscr{N}_q$ and $\mathscr{N}_r$ quickly:

$$d_{\min}(\mathscr{N}_q, \mathscr{N}_r) = d_{\mathcal{K}}(\mu_q, \mu_r) - \lambda_q - \lambda_r. \tag{8}$$

Our knowledge of $\mathcal{K}(\cdot, \cdot)$ and its induced metric $d_{\mathcal{K}}(\cdot, \cdot)$ in $\mathcal{H}$, then, is entirely sufficient to construct a cover tree with no computational penalty. In addition to this clear advantage, the time complexities of building and querying a cover tree have been analyzed extensively [8, 57], whereas *kd*-trees, metric trees, and other similar structures have been analyzed only under very limited settings [22].

Although we have presented the cover tree as the best tree option, it is not the only option for a choice of tree. What we require of a tree structure is that it can be built only with kernel evaluations between points in the dataset (or distance evaluations)[5]. Therefore, we introduce some definitions and notation from [17] in order to present our algorithm in a tree-independent manner. The following notation will be used throughout the paper, and a reference table is given in Table 1.

- A node in a tree is denoted with the letter $\mathscr{N}$.
- The set of child nodes of a node $\mathscr{N}_i$ is denoted $\mathscr{C}(\mathscr{N}_i)$ or $\mathscr{C}_i$.

[4] The explicit embedding techniques mentioned earlier [55, 30] could be used to approximate the mapping $\varphi(\cdot)$ and allow *kd*-trees (and other types of trees) to be used. However, we do not consider that approach in this work.

[5] Earlier, we mentioned kernels that work between abstract objects. For our purposes, it does not make a difference if the kernel works on abstract objects or points, so for simplicity we use the term 'points' instead of 'objects' although the two are essentially interchangeable.

- The set of points held in a node $\mathscr{N}_i$ is denoted $\mathscr{P}(\mathscr{N}_i)$ or $\mathscr{P}_i$. Each cover tree node only holds one point.

- The set of descendant nodes of a node $\mathscr{N}_i$, denoted $\mathscr{D}^n(\mathscr{N}_i)$ or $\mathscr{D}^n_i$, is the set of nodes $\mathscr{C}(\mathscr{N}_i) \cup \mathscr{C}(\mathscr{C}(\mathscr{N}_i)) \cup \ldots$[6].

- The set of descendant points of a node $\mathscr{N}_i$, denoted $\mathscr{D}^p(\mathscr{N}_i)$ or $\mathscr{D}^p_i$, is the set of points $\{p : p \in \mathscr{P}(\mathscr{D}^n(\mathscr{N}_i)) \cup \mathscr{P}(\mathscr{N}_i)$[7].

- The center of a node $\mathscr{N}_i$ is denoted μ_i. For cover trees, μ_i is the single point that $\mathscr{N}_i$ holds in $\mathcal{H}$. Therefore we also denote p_i as the point such that $\varphi(p_i) = \mu_i$.

- The furthest descendant distance for a node $\mathscr{N}_i$ and a metric $d(\cdot, \cdot)$ is defined as

$$\lambda(\mathscr{N}_i) = \max_{p \in \mathscr{D}^p(\mathscr{N}_i)} d_{\mathcal{K}}(p_i, p). \tag{9}$$

  For cover trees, $\lambda(\mathscr{N}_i)$ (or λ_i) is computed at tree construction time and cached.

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 $\mathscr{N}_i$
$\mathscr{D}^n_i$	Set of descendant nodes of $\mathscr{N}_i$
$\mathscr{D}^p_i$	Set of points contained in $\mathscr{N}_i$ and $\mathscr{D}^n_i$
μ_i	Center of $\mathscr{N}_i$ (for cover trees, $\mu_i = p_i$)
λ_i	Furthest descendant distance

Table 1: Notation for trees. See text for details.

5 Bounding the kernel value

To construct a tree-based algorithm that prunes certain subtrees, we must be able to determine the maximum kernel value possible between a point and any descendant point of a node.

Theorem 1. *Given a space tree node $\mathscr{N}_i$ with center $\varphi(p_i) = \mu_i$ and furthest descendant distance λ_i, the maximum kernel function value between some point p_q and any point in $\mathscr{D}^p_i$ is bounded by the function*

$$\mathcal{K}_{\max}(p_q, \mathscr{N}_i) = \mathcal{K}(p_q, p_i) + \lambda_i \sqrt{\mathcal{K}(p_q, p_q)}. \tag{10}$$

[6] By $\mathscr{C}(\mathscr{C}(\mathscr{N}_i))$, we mean all the children of the children of node $\mathscr{N}_i$: $\{\mathscr{C}(\mathscr{N}_c) : \mathscr{N}_c \in \mathscr{C}(\mathscr{N}_i)\}$.

[7] The meaning of $\mathscr{P}(\mathscr{D}^n(\mathscr{N}_i))$ is similar to $\mathscr{C}(\mathscr{C}(\mathscr{N}_i))$.

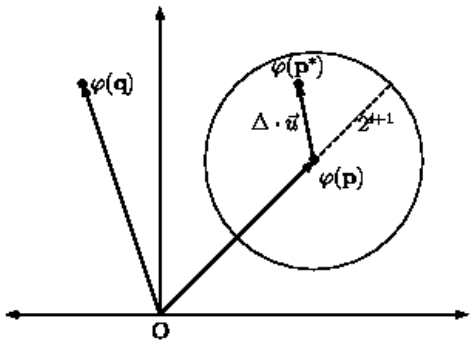


Figure 4: Point-to-node max-kernel upper bound.

Proof. Suppose that p^* is the best possible match in $\mathscr{D}^p_i$ for p_q, and let $\vec{u}$ be a unit vector in the direction of the line joining $\varphi(p_i)$ to $\varphi(p^*)$ in $\mathcal{H}$. Then,

$$\varphi(p^*) = \varphi(p_i) + \Delta\vec{u} \tag{11}$$

where $\Delta = d_{\mathcal{K}}(\mu_i, p^*)$ is the distance between $\varphi(p_i)$ and the best possible match $\varphi(p^*)$ (see Figure 4). Then, we have the following:

$$\begin{aligned} \mathcal{K}(p_q, p^*) &= \langle \varphi(p_q), \varphi(p^*) \rangle_{\mathcal{H}} \\ &= \langle \varphi(p_q), \varphi(p_i) + \Delta\vec{u} \rangle_{\mathcal{H}} \\ &= \langle \varphi(p_q), \varphi(p_i) \rangle_{\mathcal{H}} + \langle \varphi(p_q), \Delta\vec{u} \rangle_{\mathcal{H}} \\ &\leq \langle \varphi(p_q), \varphi(p_i) \rangle_{\mathcal{H}} + \Delta \left\| \varphi(p_q) \right\|_{\mathcal{H}} \end{aligned} \tag{12}$$

where the inequality step follows from the Cauchy-Schwartz inequality ($\langle x, y \rangle \leq \|x\| \cdot \|y\|$) and the fact that $\|\vec{u}\|_{\mathcal{H}} = 1$. From the definition of the kernel function, Equation 12 gives

$$\mathcal{K}(p_q, p^*) \leq \mathcal{K}(p_q, p_i) + \Delta \sqrt{\mathcal{K}(p_q, p_q)}. \tag{13}$$

We can bound Δ by noting that the distance $d_{\mathcal{K}}(\cdot, \cdot)$ between the center of $\mathscr{N}_i$ and any point in $\mathscr{D}^p_i$ is less than or equal to λ_i. We call our bound $\mathcal{K}_{\max}(p_q, \mathscr{N}_i)$, and the statement of the theorem follows. □

In addition, to construct a dual-tree algorithm, it is useful to extend the maximum point-to-node kernel value of Theorem 1 to the node-to-node setting.

Theorem 2. *Given two space tree nodes $\mathscr{N}_q$ and $\mathscr{N}_r$ with centers $\mu_q = \varphi(p_q)$ and $\mu_r = \varphi(p_r)$, respectively, the maximum kernel function value between any point in $\mathscr{D}^p_q$ and $\mathscr{D}^p_r$ is bounded by the function*

$$\begin{aligned} \mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) = \mathcal{K}(p_q, p_r) + \lambda_q \sqrt{\mathcal{K}(p_r, p_r)} \\ + \lambda_r \sqrt{\mathcal{K}(p_q, p_q)} + \lambda_q \lambda_r. \end{aligned} \tag{14}$$

Proof. Suppose that $p_q^* \in \mathscr{D}_q^p$ and $p_r^* \in \mathscr{D}_r^p$ are the best possible matches between $\mathscr{N}_q$ and $\mathscr{N}_r$; that is,

$$\mathcal{K}(p_q^*, p_r^*) = \max_{p_q \in \mathscr{D}_q^p, p_r \in \mathscr{D}_r^p} \mathcal{K}(p_q, p_r). \tag{15}$$

Now, let $\vec{u_q}$ be a vector in the direction of the line joining $\varphi(p_q)$ to $\varphi(p_q^*)$ in $\mathcal{H}$, and let $\vec{u_r}$ be a vector in the direction of the line joining $\varphi(p_r)$ to $\varphi(p_r^*)$ in $\mathcal{H}$. Then let $\Delta_q = d_{\mathcal{K}}(p_q, p_q^*)$ and $\Delta_r = d_{\mathcal{K}}(p_r, p_r^*)$. We can use similar reasoning as in the proof for Theorem 1 to show the following:

$$\begin{aligned}
\mathcal{K}(p_q^*, p_r^*) &= \langle \varphi(p_q^*), \varphi(p_r^*) \rangle_{\mathcal{H}} \\
&= \langle \varphi(p_q) + \Delta_q \vec{u_q}, \varphi(p_r) + \Delta_r \vec{u_r} \rangle_{\mathcal{H}} \\
&= \langle \varphi(p_q) + \Delta_q \vec{u_q}, \varphi(p_r) \rangle_{\mathcal{H}} \\
&\quad + \langle \varphi(p_q) + \Delta_q \vec{u_q}, \Delta_r \vec{u_r} \rangle_{\mathcal{H}} \\
&= \langle \varphi(p_q), \varphi(p_r) \rangle_{\mathcal{H}} + \langle \Delta_q \vec{u_q}, \varphi(p_r) \rangle_{\mathcal{H}} \\
&\quad + \langle \varphi(p_q), \Delta_r \vec{u_r} \rangle_{\mathcal{H}} + \langle \Delta_q \vec{u_q}, \Delta_r \vec{u_r} \rangle_{\mathcal{H}} \\
&\leq \langle \varphi(p_q), \varphi(p_r) \rangle_{\mathcal{H}} + \Delta_q \left\| \varphi(p_r) \right\|_{\mathcal{H}} \\
&\quad + \Delta_r \left\| \varphi(p_q) \right\|_{\mathcal{H}} + \Delta_q \Delta_r,
\end{aligned} \tag{16}$$

where again the inequality steps follow from the Cauchy-Schwarz inequality. We can then substitute in the kernel functions to obtain

$$\begin{aligned}
\mathcal{K}(p_q^*, p_r^*) \leq \mathcal{K}(p_q, p_r) + \Delta_q \sqrt{\mathcal{K}(p_r, p_r)} \\
+ \Delta_r \sqrt{\mathcal{K}(p_q, p_q)} + \Delta_q \Delta_r.
\end{aligned} \tag{17}$$

Then, as with the point-to-node case, we can bound Δ_q by λ_q and Δ_r can be bounded by λ_r. Call the bound $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r)$, and the statement of the theorem follows. □

For normalized kernels ($\mathcal{K}(x, x) = 1 \ \forall x$)[8], all the points are on the surface of a hypersphere in $\mathcal{H}$. In this case, the above bounds in Theorems 1 and 2 are both correct but possibly loose. Therefore, we can present tighter bounds specifically for this condition.

Theorem 3. *Consider a kernel $\mathcal{K}$ such that $\mathcal{K}(x, x) = 1 \ \forall \ x$, and space tree node $\mathscr{N}_i$ with center $\mu_i = \varphi(p_i)$ and furthest descendant distance λ_i. Define the following quantities:*

$$\alpha_i = \left(1 - \frac{1}{2}\lambda_i^2\right), \tag{18}$$

$$\beta_i = \lambda_i \sqrt{1 - \frac{1}{4}\lambda_i^2}. \tag{19}$$

[8]Earlier we defined normalized kernels as $\mathcal{K}(x, x) = c$ for some constant c, but here for simplicity we consider only $c = 1$. Adapting the proof and bounds for $c \neq 1$ is straightforward.

Then, the maximum kernel function value between some point p_q and any point in $\mathscr{D}_i^p$ is bounded from above by the function

$$\mathcal{K}_{\max}^n(p_q, \mathscr{N}_i) = \begin{cases} \mathcal{K}(p_q, p_i)\alpha_i + \beta_i \sqrt{(1 - \mathcal{K}(p_q, p_i)^2)} \\ \quad \textit{if } \mathcal{K}(p_q, p_i) \leq \alpha_i \\ 1 \ \textit{otherwise} \end{cases} \tag{20}$$

Proof. Since all the points p_q and $\mathscr{D}_i^p$ are sitting on the surface of a hypersphere in $\mathcal{H}$, $\mathcal{K}(p_q, p)$ denotes the cosine of the angle made by $\varphi(p_q)$ and $\varphi(p)$ at the origin. If we first consider the case where p_q lies within the ball bounding space tree node $\mathscr{N}_i$ (that is, if $d_{\mathcal{K}}(p_q, p_i) < \lambda_i$), it is clear that the maximum possible kernel evaluation should be 1, because there could exist a point in $\mathscr{D}_i^p$ whose angle to p_q is 0. We can restate our condition as a condition on $\mathcal{K}(p_q, p_i)$ instead of $d_{\mathcal{K}}(p_q, p_i)$:

$$\begin{aligned}
d_{\mathcal{K}}(p_q, p_i) &< \lambda_i, \\
\sqrt{\mathcal{K}(p_q, p_q) + \mathcal{K}(p_i, p_i) - 2\mathcal{K}(p_q, p_i)} &< \lambda_i, \\
\mathcal{K}(p_q, p_i) &> 1 - \frac{1}{2}\lambda_i^2, \\
\mathcal{K}(p_q, p_i) &> \alpha_i.
\end{aligned}$$

Now, for the other case, let $\cos\theta_{p_q p_i} = \mathcal{K}(p_q, p_i)$ and $p^* = \arg\max_{p \in \mathscr{D}_i^p} \mathcal{K}(p_q, p)$. Let $\theta_{p_i p^*}$ be the angle between $\varphi(p_i)$ and $\varphi(p^*)$ at the origin, let $\theta_{p_q p^*}$ be the angle between $\varphi(p_q)$ and $\varphi(p^*)$ at the origin, and let $\theta_{p_q p_i}$ be the angle between $\varphi(p_q)$ and $\varphi(p_i)$ at the origin. Then,

$$\begin{aligned}
\mathcal{K}(p_q, p^*) &= \cos\theta_{p_q p^*} \\
&\leq \cos(\{\theta_{p_q p_i} - \theta_{p_i p^*}\}_+).
\end{aligned}$$

We know that $d_{\mathcal{K}}(p_i, p^*) \leq \lambda_i$, and also that $d_{\mathcal{K}}(p_i, p^*) = \sqrt{2 - 2\cos\theta_{p_i p^*}}$. Therefore, $\cos\theta_{p_i p^*} \geq 1 - \frac{1}{2}\lambda_i^2$. This means

$$\theta_{p_i p^*} \leq |\cos^{-1}(1 - \frac{1}{2}\lambda_i^2)|. \tag{21}$$

Combining this with Equation 21, we get:

$$\mathcal{K}(p_q, p^*) \leq \cos\left([\theta_{p_q p_i} - \theta_{p_i p^*}]_+\right) \tag{22}$$

Now, if we substitute $|\cos^{-1}(1 - \frac{1}{2}\lambda_i^2)|$, the largest possible value for $\theta_{p_i p^*}$, we obtain the following:

$$\mathcal{K}_{\max}(p_q, \mathscr{N}_i) \leq \cos\left(\left[\theta_{qp_i} - \left|\cos^{-1}(1 - \frac{1}{2}\lambda_i^2)\right|\right]_+\right)$$

which can be reduced to the statement of the theorem by the use of trigonometric identities. Combine with the case where $\mathcal{K}(p_q, p_i) > \alpha_i$, and call that bound $\mathcal{K}^n_{\max}(p_q, \mathscr{N}_i)$. Then, the theorem holds. □

We can show a similar tighter bound for the dual-tree case.

Theorem 4. *Consider a kernel $\mathcal{K}$ such that $\mathcal{K}(x, x) = 1 \ \forall \ x$, and two space tree nodes $\mathscr{N}_q$ and $\mathscr{N}_r$ with centers $\varphi(p_q) = \mu_q$ and $\varphi(p_r) = \mu_r$, respectively, and furthest descendant distances λ_q and λ_r, respectively. Define the following four quantities:*

$$\begin{aligned} \alpha_q &= \left(1 - \frac{1}{2}\lambda_q^2\right), \\ \alpha_r &= \left(1 - \frac{1}{2}\lambda_r^2\right), \\ \beta_q &= \lambda_q\sqrt{1 - \frac{1}{4}\lambda_q^2}, \\ \beta_r &= \lambda_r\sqrt{1 - \frac{1}{4}\lambda_r^2}. \end{aligned}$$

Then, the maximum kernel function value between any point in $\mathscr{D}^p_q$ and $\mathscr{D}^p_r$ is bounded from above by the function

$$\mathcal{K}^n_{\max}(\mathscr{N}_q, \mathscr{N}_r) = \begin{cases} \mathcal{K}(p_q, p_r)(\alpha_q\alpha_r - \beta_q\beta_r) \\ \quad + \left(\sqrt{1 - \mathcal{K}(p_q, p_r)^2}\right)(\gamma_q\delta_r + \delta_r\gamma_q) \\ \quad \textit{if } \mathcal{K}(p_q, p_r) \leq 1 - \frac{1}{2}(\lambda_q + \lambda_r)^2 \\ 1 \ \textit{otherwise.} \end{cases} \quad (23)$$

Proof. All of the points in $\mathscr{D}^p_q$ and $\mathscr{D}^p_r$ are sitting on the surface of a hypersphere in $\mathcal{H}$. This means that $\mathcal{K}(p_q, p_r)$ denotes the cosine of the angle made by $\varphi(p_q)$ and $\varphi(p_r)$ at the origin. Similar to the previous proof, we first consider the case where the balls in $\mathcal{H}$ centered at $\varphi(p_q)$ and $\varphi(p_r)$ with radii λ_q and λ_r, respectively, overlap. This situation happens when $d_{\mathcal{K}}(p_q, p_r) < \lambda_q + \lambda_r$. In this case, it is clear that the maximum possible kernel evaluation should be 1, because there could exist a point in $\mathscr{D}^p_q$ whose angle to a point in $\mathscr{D}^p_r$ is 0. We can restate the condition as a condition on $\mathcal{K}(p_q, p_r)$:

$$\mathcal{K}(p_q, p_r) > 1 - \frac{1}{2}(\lambda_q + \lambda_r)^2. \quad (24)$$

Now, for the other case, assume that p^*_q and p^*_r are the best matches between points in $\mathscr{D}^p_q$ and $\mathscr{D}^p_r$. Let $\cos\theta_{p_qp_r} = \mathcal{K}(p_q, p_r)$; let $\theta_{p_qp^*_q}$ be the angle between $\varphi(p_q)$ and $\varphi(p^*_q)$ at the origin; similarly, let $\theta_{p_rp^*_r}$ be the angle between $\varphi(p_r)$ and $\varphi(p^*_r)$ at the origin. Lastly, let $\theta_{p^*_qp^*_r}$ be the angle between $\varphi(p^*_q)$ and $\varphi(p^*_r)$ at the origin. Then,

$$\begin{aligned} \mathcal{K}(p^*_q, p^*_r) &= \cos\theta_{p^*_qp^*_r} \\ &\leq \cos\left(\left[\theta_{p_qp_r} - \theta_{p_qp^*_q} - \theta_{p_rp^*_r}\right]_+\right) \quad (25) \end{aligned}$$

Using reasoning similar to the last proof, we obtain the following bounds:

$$\theta_{p_qp^*_q} \leq \left|\cos^{-1}\left(1 - \frac{1}{2}\lambda_q^2\right)\right| \quad (26)$$

$$\theta_{p_rp^*_r} \leq \left|\cos^{-1}\left(1 - \frac{1}{2}\lambda_r^2\right)\right|. \quad (27)$$

We can substitute these two values into Equation 25 to obtain

$$\mathcal{K}(p^*_q, p^*_r) \leq \cos\left(\left[\theta_{p_qp_r} - \left|\cos^{-1}\left(1 - \frac{1}{2}\lambda_q^2\right)\right| - \left|\cos^{-1}\left(1 - \frac{1}{2}\lambda_r^2\right)\right|\right]_+\right). \quad (28)$$

This can be reduced to the statement of the theorem by the use of trigonometric identities. Combine with the conditional from earlier and call the combined bound $\mathcal{K}^n_{\max}(\mathscr{N}_q, \mathscr{N}_r)$. Then, the theorem holds. □

In the upcoming algorithms, we will not use the tighter bounds for normalized kernels given in Theorems 3 and 4; however, it is easy to re-derive the algorithm with the tighter bounds, if a normalized kernel is being used. Simply replace instances of $\mathcal{K}_{\max}(\cdot, \cdot)$ with $\mathcal{K}^n_{\max}(\cdot, \cdot)$.

6 Single-tree algorithm

First, we will present a single-tree algorithm called **single-tree FastMKS** that works on a single query p_q and a reference set S_r. Following the tree-independent algorithmic framework of [17], we will present our algorithm as two functions: a `BaseCase(`p_q`,` p_r`)` function that runs on two points, and a `Score(`p_q`,` $\mathscr{N}_r$`)` function that runs on the query point p_q and a node $\mathscr{N}_r$.[9]

[9] In the original version of this paper, the algorithm was presented specifically for cover trees. This formulation is much more general and intuitive, and reduces exactly to the cover tree formulation given in the original paper [18].

Algorithm 1 `BaseCase`$(p_q,\ p_r)$ for FastMKS.

Input: query point p_q, reference point p_r
Output: none

if $\mathcal{K}(p_q, p_r) > k^*$ **then**
  $k^* \leftarrow \mathcal{K}(p_q, p_r)$
  $p^* \leftarrow p_r$
end if

Algorithm 2 `Score`$(p_q,\ \mathscr{N}_r)$ for FastMKS.

Input: query point p_q, reference space tree node $\mathscr{N}_r$, max-kernel candidate p^* for p_q and corresponding max-kernel value k^*
Output: a score for the node, or ∞ if the node can be pruned

if $\mathcal{K}_{\max}(p_q, \mathscr{N}_r) < k^*$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(p_q, \mathscr{N}_r)$
end if

Given those two functions, a single-tree algorithm can be assembled using any space tree (with additional constraints as given in Section 4) and any valid pruning single-tree traversal [17]. In short, a pruning single-tree traversal visits nodes in the tree, and calls the `Score()` function to determine if the given node $\mathscr{N}_r$ can be *pruned*. If the node can be pruned, `Score()` will return ∞, and no descendants of $\mathscr{N}_r$ will be visited. Otherwise, `BaseCase()` will be called with query point p_q and each point $p_r \in \mathscr{P}_r$.

In our problem setting, we can prune a node $\mathscr{N}_r$ if no points in $\mathscr{D}_r^p$ can possibly contain a better max-kernel candidate than what has already been found as a max-kernel candidate for p_q. Thus, any descendants of $\mathscr{N}_r$ do not need to be visited, as they cannot improve the solution.

The `BaseCase()` function can be seen in Algorithm 1. It assumes p^* is a global variable representing the current max-kernel candidate and k^* is a global variable representing the current best max-kernel value. The method itself is very simple: calculate $\mathcal{K}(p_q, p_r)$, and if that kernel evaluation is larger than the current best max-kernel value candidate k^*, then store that kernel and p_r as the new best max-kernel candidate and $\mathcal{K}(p_q, p_r)$ as the new best max-kernel value candidate.

The `Score()` function for single-tree FastMKS is given in Algorithm 2. The intuition is clear: if the maximum possible kernel value between p_q and any point in $\mathscr{D}_i^p$ is less than the current max-kernel candidate value, then $\mathscr{N}_i$ cannot possibly hold a better candidate and it can be pruned (return ∞). Otherwise, the kernel value itself is returned. This return value is chosen because pruning single-tree traversals may use the value returned by `Score()` to determine the order in which to visit subsequent nodes [17].

The actual single-tree FastMKS algorithm is constructed by selecting a type of space tree and selecting a pruning single-tree traversal with the `BaseCase()` function as in Algorithm 1 and the `Score()` function as in Algorithm 2. The algorithm is run by building a space tree $\mathscr{T}_r$ on the set of reference points S_r, then using the pruning single-tree traversal with point p_q and tree $\mathscr{T}_q$. At the beginning of the traversal, p^* is initialized to an invalid value and k^* is initialized to $-\infty$.

Proving the correctness of the single-tree FastMKS algorithm is trivial, but first, we will explicitly define a pruning single-tree traversal as in [17].

Definition 3. *A* **pruning single-tree traversal** *is a process that, given a space tree, will visit nodes in the tree and perform a computation to assign a score to that node. If the score is above some bound (or ∞), the node is "pruned" and none of its descendants will be visited; otherwise, a computation is performed on any points contained within that node. If no nodes are pruned, then the traversal will visit each node in the tree once.*

Theorem 5. *At the termination of the single-tree FastMKS algorithm for a given space tree and pruning single-tree traversal,*

$$p^* = \arg\max_{p_r \in S_r} \mathcal{K}(p_q, p_r). \tag{29}$$

Proof. First, assume that `Score()` does not prune any nodes during the traversal of the tree $\mathscr{T}_r$. Then, by the definition of pruning single-tree traversal, `BaseCase()` is called with p_q and every $p_r \in S_r$. This is equivalent to linear scan and will give the correct result.

Then, by Theorem 1 (or Theorem 3 if $\mathcal{K}(\cdot,\cdot)$ is normalized and $\mathcal{K}_{\max}^n(\cdot,\cdot)$ is being used), a node is only pruned if it does *not* contain a point p_r where $\mathcal{K}(p_q, p_r) > k^*$. Thus, `BaseCase()` is only *not* called in situations where p^* and k^* would not be modified. This, combined with the previous observation, means that p^* and k^* are equivalent to the linear scan results at the end of the traversal—and we know the linear scan results are correct. Thus, the theorem holds. □

7 Single-tree runtime analysis

For the runtime analysis of single-tree FastMKS, we will restrict the type of space tree to the cover tree,

Algorithm 3 The standard pruning single-tree traversal for cover trees.

1: **Input:** query point p_q, reference cover tree $\mathscr{T}_r$
2: **Output:** none
3: $R \leftarrow \{$ root$(\mathscr{T}_r)$ $\}$
4: $s \leftarrow \infty$
5: **if** `Score`$(p_q,$ root$(\mathscr{T}_r)) = \infty$ **then**
6: **return**
7: **end if**
8: **do**
9: $s \leftarrow$ largest scale of all non-visited nodes in R
10: **for each** $\mathscr{N}_i \in R$ **where** $s_i = s$ **do**
11: `BaseCase`$(p_q,\ p_i)$
12: **for each** $\mathscr{N}_c \in \mathscr{C}(\mathscr{N}_i)$ **do**
13: **if** `Score`$(p_q,\ \mathscr{N}_c) \neq \infty$ **then**
14: add $\mathscr{N}_c$ to R
15: **end if**
16: **end for**
17: **end for**
18: **while** $s \neq -\infty$

due to the desirable theoretical properties of the cover tree. First, we will detail the cover tree datastructure more comprehensively. For readers familiar with the cover tree as described in [8], we are focusing only on the explicit representation.

We already know that the cover tree is a space tree; it is also a leveled tree: each node $\mathscr{N}_i$ holds one point p_i and has a scale s_i that represents its level in the tree. A large s_i represents a node closer to the root of the tree; the root has the largest scale of all nodes in the tree. Each child (if any) of $\mathscr{N}_i$ has scale less than s_i. If $\mathscr{N}_i$ has no children, then its scale is $-\infty$. In addition, for a node $\mathscr{N}_i$, $\lambda_i \leq 2^{s_i+1}$. Therefore, we can bound the furthest descendant distance at scale s_i from above with 2^{s_i+1}.

The last important property of cover trees is the *separation invariant*, which is integral to our proofs. There cannot exist two nodes $\mathscr{N}_i$ and $\mathscr{N}_j$ at scale s_i such that $d(p_i, p_j) \leq 2^{s_i}$. Alternately stated, for any $\mathscr{N}_i$ and $\mathscr{N}_j$ both at scale s_i, $d(p_i, p_j) > 2^{s_i}$.

Algorithm 3 describes the standard pruning single-tree traversal used for cover trees, adapted to a tree-independent form from the original formulation in [8]. Note that this traversal will not work on arbitrary types of space trees because it depends on the scale, which is specific to the cover tree. The traversal itself is breadth-first; it maintains a set R of nodes that have not been pruned, and iteratively reduces the maximum scale present in R to $-\infty$.

Now, we introduce a few useful results from [8]. Proofs of each lemma can be found in that paper.

Lemma 1. *The number of children of any cover tree node $\mathscr{N}_i$ is bounded by c^4, where c is the expansion constant of the dataset the cover tree is built on, as defined in Definition 1.*

Lemma 2. *The maximum depth of any point p_r in a cover tree $\mathscr{T}_r$ is $O(c^2 \log N)$, where N is the number of points in the dataset that $\mathscr{T}_r$ is built on.*

The main result of this section is the search time complexity of single-tree FastMKS in terms of the number of points in the reference set S_r and the properties of the kernel.

Theorem 6. *Given a Mercer kernel $\mathcal{K}(\cdot,\cdot)$, a query point p_q, and a dataset S_r of size N with expansion constant c (Definition 1) with respect to the induced metric $d_{\mathcal{K}}$ (Equation 3) and directional concentration constant γ (Definition 2), the single-tree FastMKS algorithm using cover trees and the standard single-tree cover tree traversal on p_q and S_r requires $O(c^{12}\gamma^2 \log N)$ time.*

Proof. The first part of the proof is similar to the runtime analysis of nearest-neighbor search with cover trees [8]. Call R_i the set of nodes in R with scale s_i. Now, let s^* be the scale that has the greatest number of elements in R:

$$s^* = \underset{s \in [-\infty, \infty)}{\arg\max} \left| \{\mathscr{N}_i \in R : s_i = s\} \right|. \tag{30}$$

Define the set R^* as the set of nodes in R with scale s^*.

By Lemma 2, the depth of any node in the tree is at most $k = O(c^2 \log N)$. Because $|R_{-\infty}| \leq |R^*|$, we can conclude that the maximum number of outer iterations on s required (lines 8—18) is $O(k|R^*|)$. Each inner iteration (lines 10—17) considers a maximum of $|R^*|$ points, and the innermost loop that considers the children of each element in $\mathscr{N}_i$ (lines 12—16) considers a maximum of c^4 points for each $\mathscr{N}_i$ (because of Lemma 1). Combining all of these things, we obtain a runtime bound of $O(kc^4|R^*|^2) = O(c^6|R^*|^2 \log N)$. Thus, the theorem will hold if we can show that $|R^*| \leq c^3\gamma$.

To bound $|R^*|$, let $u = \varphi(p_q)/\|\varphi(p_q)\|_{\mathcal{H}}$. Then,

$$I_{S_r}(\varphi(p_q), [a, b]) = I_{S_r}\left(u, \left[\frac{a}{\|\varphi(p_q)\|_{\mathcal{H}}}, \frac{b}{\|\varphi(p_q)\|_{\mathcal{H}}}\right]\right).$$

For any scale s_i, let R_i be the set of all nodes in R with scale s_i when $s = s_i$ (that is, R_i is the set of all nodes considered by line 10 when $s = s_i$):

$$R_i = \{\mathscr{N}_i : \mathscr{N}_i \in R, s_i = s\}.$$

But, for any node $\mathscr{N}_i$ to be in R, then we know that $\mathcal{K}_{\max}(p_q, \mathscr{N}_i) \geq k^*$. Thus, we can express R_i differently:

$$\begin{aligned} R_i &= \{\mathscr{N}_i : \mathscr{N}_i \in R, s_i = s\} \\ &\subseteq \{\mathscr{N}_i : \mathscr{N}_i \in \mathscr{T}_r, \mathcal{K}_{\max}(p_q, \mathscr{N}_i) \geq k^*, s_i = s\}. \end{aligned}$$

Now, by Equation 6, any $\mathscr{N}_i$ in the set R_i satisfies

$$\varphi(p_i) \in I_{S_r}\left(\varphi(p_q), \left[k^* - \lambda_i \|\varphi(p_q)\|_{\mathcal{H}}, \hat{k}\right]\right)$$

where $\hat{k}$ is the true max-kernel value (that is, $\hat{k} = \max_{p_r \in S_r} \mathcal{K}(p_q, p_r)$). Because $\lambda_i \leq 2^{s_i+1}$,

$$\begin{aligned} \varphi(p_i) &\in I_{S_r}\left(\varphi(p_q), \left[k^* - 2^{s_i+1}\|\varphi(p_q)\|_{\mathcal{H}}, \hat{k}\right]\right) \\ &\subseteq I_{S_r}\left(\varphi(p_q), \left[\hat{k} - 2^{s_i+2}\|\varphi(p_q)\|_{\mathcal{H}}, \hat{k}\right]\right) \end{aligned}$$

because $\hat{k} \leq k^* + 2^{s_i+1}\|\varphi(p_q)\|_{\mathcal{H}}$. Further,

$$\begin{aligned} \varphi(p_i) &\in I_{S_r}\left(\varphi(p_q), \left[\mathcal{K}(p_q, p_i) - 2^{s_i+2}\|\varphi(p_q)\|_{\mathcal{H}}, \right.\right. \\ &\qquad \left.\left. \mathcal{K}(p_q, p_i) + 2^{s_i+2}\|\varphi(p_q)\|_{\mathcal{H}}\right]\right) \\ &= I_{S_r}\left(u, \left[\langle u, \varphi(p_i)\rangle_{\mathcal{H}} - 2^{s_i+2}, \right.\right. \\ &\qquad \left.\left. \langle u, \varphi(p_i)\rangle_{\mathcal{H}} + 2^{s_i+2}\right]\right). \end{aligned}$$

Remember that this inclusion applies for all p_i of nodes in R_i. By the definition of directional concentration constant (Definition 2), there exist γ points $p_j \in S_r$ such that

$$\begin{aligned} I_{S_r}\left(u, \left[\langle u, \varphi(p_r)\rangle_{\mathcal{H}} - 2^{s_i+2}, \langle u, \varphi(p_r)\rangle_{\mathcal{H}} + 2^{s_i+2}\right]\right) \\ \subseteq \bigcup_{j=1}^{\gamma} B_{S_r}(p_j, 2^{s_i+2}). \end{aligned}$$

Due to the separation invariant, R_i only has points p_i that are separated by at least 2^{s_i}. Thus, $|R_i|$ is less than or equal to the number of balls of radius 2^{s_i-1} that can be packed into the set

$$\bigcup_{j=1}^{\gamma} B_{S_r}(p_j, 2^{s_i+2}).$$

Consider each $B_{S_r}(p_j, 2^{s_i+2})$ individually. Using the definition of expansion constant (Definition 1), we have

$$|B_{S_r}(p_j, 2^{s_i+2})| \leq c^3 |B_{S_r}(p_j, 2^{s_i-1})|$$

and $|B_{S_r}(p_j, 2^{s_i-1})|$ can only contain one point at scale s_i. Hence, for all s_i, $|R_i| \leq \gamma c^3$, and thus $|R^*| \leq \gamma c^3$, giving the statement of the theorem. □

Comparing to the query time $O(c^{12} \log n)$ for single-tree cover tree nearest neighbor search [8], it is clear that FastMKS has similar $O(\log n)$ scaling, but also has an extra price of γ^2 to solve the more general problem of max-kernel search.

It is also worth noting that the tighter bound given in Theorem 3 for normalized kernels could be used to produce a tighter runtime bound.

8 Dual-tree algorithm

Now, we present a dual-tree algorithm for max-kernel search, called **dual-tree FastMKS**. This algorithm, as with the single-tree algorithm in Section 6, is presented in the tree-independent framework of [17]. The `BaseCase(`p_q`,` p_r`)` function is the same as the single-tree function (Algorithm 1), and we present a dual-tree pruning rule with a `Score(`$\mathscr{N}_q$`,` $\mathscr{N}_r$`)` function that runs on a query node $\mathscr{N}_q$ and a reference node $\mathscr{N}_r$.

Because the dual-tree algorithm solves max-kernel search for an entire set of query points S_q, we must store a kernel candidate p^* and value k^* for each query point p_q; call these $p^*(p_q)$ and $k^*(p_q)$, respectively. At the initialization of the algorithm $k^*(p_q) = \infty$ for each $p_q \in S_q$ and $p^*(p_q)$ is set to some invalid point.

The pruning rule is slightly more complex. In the dual-tree setting, we can only prune a node combination $(\mathscr{N}_q, \mathscr{N}_r)$ if and only if $\mathscr{D}_r^p$ contains *no* points that can improve $p^*(p_q)$ and $k^*(p_q)$ for any $p_q \in \mathscr{D}_q^p$. There are multiple ways to express this concept, and we will use two of them to construct a bound function to determine when we can prune. This section is heavily based on the reasoning used to derive the nearest-neighbor search bound in [17].

First, consider the smallest max-kernel value $k^*(p_q)$ for all points $p_q \in \mathscr{D}_q^p$; call this $B_1(\mathscr{N}_q)$:

$$\begin{aligned} B_1(\mathscr{N}_q) &= \min_{p_q \in \mathscr{D}_q^p} k^*(p_q) \\ &= \min\left\{\min_{p_q \in \mathscr{P}_q} k^*(p_q), \min_{\mathscr{N}_c \in \mathscr{C}_q} B_1(\mathscr{N}_q)\right\} \end{aligned}$$

where the simplification is a result of expressing $B_1(\mathscr{N}_q)$ recursively. Now, note also that for any point $p_q \in \mathscr{D}_q^p$ with max-kernel candidate value $k^*(p_q)$, we can place a lower bound on the true max-kernel value $\hat{k}(p'_q)$ for any $p'_q \in \mathscr{D}_r^p$ by bounding $\mathcal{K}(p'_q, p^*(p_q))$. This gives

$$\hat{k}(p'_q) \geq k^*(p_q) - (\rho_q + \lambda_q)\sqrt{\mathcal{K}\left(p^*(p_q), p^*(p_q)\right)}$$

Algorithm 4 `Score`$(\mathscr{N}_q, \mathscr{N}_r)$ for FastMKS.

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 can be pruned
if $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) < B(\mathscr{N}_q)$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r)$
end if

where ρ_q is the maximum distance from any $p \in \mathscr{P}_q$ to the centroid of $\mathscr{N}_q$ (for cover trees, this value is always 0). This inequality follows using similar reasoning as Theorem 1, except for that we are finding a lower bound instead of an upper bound.

Considering all the points $p_q \in \mathscr{D}_q^p$, we find that the minimum possible max-kernel value for any point p_q can be expressed as

$$\max_{p_q \in \mathscr{D}_q^p} k^*(p_q) - (\rho_q + \lambda_q)\sqrt{\mathcal{K}\left(p^*(p_q), p^*(p_q)\right)}.$$

However, this is difficult to calculate in practice; thus, we introduce a second bounding function that can be quickly calculated by only considering points in $\mathscr{P}_q$ and not $\mathscr{D}_q^p$:

$$B_2(\mathscr{N}_q) = \max_{p_q \in \mathscr{P}_q} k^*(p_q) - (\rho_q + \lambda_q)\sqrt{\mathcal{K}\left(p^*(p_q), p^*(p_q)\right)}.$$

Now, we can take the better of $B_1(\mathscr{N}_q)$ and $B_2(\mathscr{N}_q)$ as our pruning bound:

$$B(\mathscr{N}_q) = \max\left\{B_1(\mathscr{N}_q), B_2(\mathscr{N}_q)\right\}. \tag{31}$$

This means that we can prune a node combination $(\mathscr{N}_q, \mathscr{N}_r)$ if

$$\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) < B(\mathscr{N}_q),$$

and therefore we introduce a `Score()` function in Algorithm 4 that uses $B(\mathscr{N}_q)$ to determine if a node combination should be pruned.

As with the single-tree algorithm, we will explicitly define a pruning dual-tree traversal as in [17] before proving correctness.

Definition 4. *A* **pruning dual-tree traversal** *is a process that, given two space trees $\mathscr{T}_q$ (query tree) and $\mathscr{T}_r$ (reference tree), will visit combinations of nodes $(\mathscr{N}_q, \mathscr{N}_r)$ such that $\mathscr{N}_q \in \mathscr{T}_q$ and $\mathscr{N}_r \in \mathscr{T}_r$ no more than once, and perform an computation to assign a score to that combination. If the score is above some bound (or ∞), the combination is pruned and no combinations $(\mathscr{N}_{qc}, \mathscr{N}_{rc})$ such that $\mathscr{N}_{qc} \in \mathscr{D}_q^n \cup \mathscr{N}_q$ and $\mathscr{N}_{rc} \in \mathscr{D}_r^n \cup \mathscr{N}_r$ will be visited; otherwise, a computation is performed between each point in $\mathscr{N}_q$ and each point in $\mathscr{N}_r$. If no nodes are pruned, a computation is performed between each point in the query tree and the reference tree.*

Theorem 7. *At the termination of the dual-tree FastMKS algorithm for a given space tree and pruning dual-tree traversal,*

$$p^*(p_q) = \arg\max_{p_r \in S_r} \mathcal{K}(p_q, p_r) \;\forall\; p_q \in S_q. \tag{32}$$

Proof. First, assume that `Score()` does not prune any node combinations during the dual traversal of the trees $\mathscr{T}_q$ and $\mathscr{T}_r$. Then, by the definition of pruning dual-tree traversal, `BaseCase()` will be called with each $p_q \in S_q$ and each $p_r \in S_r$; this is equivalent to linear scan and will give the correct results.

We have already stated the validity of $B(\mathscr{N}_q)$ (Equation 31). Because of that, and also by Theorem 2 (or Theorem 4 if $\mathcal{K}(\cdot,\cdot)$ is normalized and $\mathcal{K}_{\max}^n(\cdot,\cdot)$ is being used), a node combination is only pruned is it does *not* contain a point p_r that would modify $p^*(p_q)$ or $k^*(p_q)$ for any $p_q \in \mathscr{D}_q^p$. This, combined with the previous observation, means that p^* and k^* are equivalent to the linear scan results for each $p_q \in S_r$, and thus, the theorem holds. □

9 Dual-tree runtime analysis

As in Section 7, we will restrict the type of space tree to the cover tree for the runtime analysis of dual-tree FastMKS. For this analysis, we must introduce a few quantities and useful lemmas.

Lemma 3. *Consider a set R of cover tree nodes from the cover tree $\mathscr{T}$. If each node $\mathscr{N}_r$ has a parent $\text{par}(\mathscr{N}_r)$ with scale at least s^*, then for any two nodes $\mathscr{N}_x \in R, \mathscr{N}_y \in R$ with points p_x and p_y, respectively,*

$$d(p_x, p_y) > 2^{(s^*-1)}. \tag{33}$$

Proof. For this proof we will use the implicit representation of the cover tree $\mathscr{T}$ (see [8] for more details). Any explicit cover tree node $\mathscr{N}_x$ will have an implicit parent node (call this $\mathscr{N}_i$) where $s_i = s_x + 1$. Given $\mathscr{N}_i$, either $p_i = p_x$ or $p_i \neq p_x$.

If $p_i \neq p_x$, then $\mathscr{N}_i$ is also an explicit cover tree node; that is, $\mathscr{N}_i = \text{par}(\mathscr{N}_x)$, in which case we know that the only possibility is that $s_i = s^*$ and therefore $\mathscr{N}_x$ has scale $s^* - 1$.

If $p_i = p_x$, then $\mathscr{N}_x$ has a series of implicit parent nodes which each have point p_x. The last implicit node in this series will have implicit parent $\text{par}(\mathscr{N}_x)$, which is also an explicit node with scale at least s^*. Thus, an implicit node with point p_i and scale $s^* - 1$ exists.

Consequently, for every node in R, there exists an implicit node with the same point and scale $s^* - 1$. Because the separation invariant also applies to implicit nodes, each pair of points is separated by greater than 2^{s^*-1}, and the lemma holds. □

Now, we define the maximum norms and minimum norms of the query set S_q and reference set S_r:

$$\eta_q = \max_{p_q \in S_q} \|\varphi(p_q)\|_{\mathcal{H}}, \tag{34}$$

$$\eta_r = \max_{p_r \in S_r} \|\varphi(p_r)\|_{\mathcal{H}}, \tag{35}$$

$$\tau_q = \min_{p_q \in S_q} \|\varphi(p_q)\|_{\mathcal{H}}, \tag{36}$$

$$\tau_r = \min_{p_r \in S_r} \|\varphi(p_r)\|_{\mathcal{H}}. \tag{37}$$

Next, we use these quantities to place bounds on the maximum distances $d_{\mathcal{H}}(\cdot, \cdot)$ between points in the dataset, and place an upper bound on the maximum scale of cover tree nodes.

Lemma 4. *For the query set S_q, the maximum distance between any points in S_q,*

$$d_{\mathcal{H}}^{\max}(S_q) \leq 2\eta_q. \tag{38}$$

Proof. We can alternately write $d_{\mathcal{H}}^{\max}(S_q)$ as

$$\begin{aligned} d_{\mathcal{H}}^{\max}(S_q) &= \max_{p_i \in S_q, p_j \in S_q} d_{\mathcal{H}}(p_i, p_j) \\ (d_{\mathcal{H}}^{\max}(S_q))^2 &= \max_{p_i \in S_q, p_j \in S_q} \|\varphi(p_i)\|_{\mathcal{H}}^2 + \|\varphi(p_j)\|_{\mathcal{H}}^2 \\ &\quad - 2\langle \varphi(p_i), \varphi(p_j) \rangle_{\mathcal{H}}. \end{aligned}$$

Note that $\langle \varphi(p_i), \varphi(p_j) \rangle_{\mathcal{H}}$ is minimized when $\varphi(p_i)$ and $\varphi(p_j)$ point opposite ways in $\mathcal{H}$: $\varphi(p_i)/\|\varphi(p_i)\|_{\mathcal{H}} = -(\varphi(p_j)/\|\varphi(p_j)\|_{\mathcal{H}})$. Thus,

$$\begin{aligned} (d_{\mathcal{H}}^{\max}(S_q))^2 &\leq \max_{p_i \in S_q, p_j \in S_q} \|\varphi(p_i)\|_{\mathcal{H}}^2 + \|\varphi(p_j)\|_{\mathcal{H}}^2 \\ &\quad - 2\max\{\langle \varphi(p_i), -\varphi(p_i) \rangle_{\mathcal{H}}, \\ &\qquad \langle \varphi(p_j), -\varphi(p_j) \rangle_{\mathcal{H}}\} \\ &\leq \max_{p_i \in S_q, p_j \in S_q} \|\varphi(p_i)\|_{\mathcal{H}}^2 + \|\varphi(p_j)\|_{\mathcal{H}}^2 \\ &\quad + 2\max\{\|\varphi(p_i)\|_{\mathcal{H}}^2, \|\varphi(p_j)\|_{\mathcal{H}}^2\} \\ &\leq 4\eta_q^2. \end{aligned}$$

This trivially reduces to the result. □

Corollary 1. *The maximum distance between any points in S_r is*

$$d_{\mathcal{H}}^{\max}(S_r) \leq 2\eta_r. \tag{39}$$

Lemma 5. *The top scale s_r^T (maximum/largest scale) in the cover tree $\mathscr{T}_r$ built on S_r is bounded as*

$$s_r^T \leq \log_2(\eta_r). \tag{40}$$

Proof. The root of the tree $\mathscr{T}_r$ is the node with the largest scale, and it is the only node of that scale (call this scale s_r^T). The furthest descendant distance of the root node is bounded by $2^{s_r^T+1}$; however, this is not necessarily the distance between the two furthest points in the dataset (consider a tree where the root node is near the centroid of the data). This, with Corollary 1, yields $2^{s_r^T+1} \leq 2\eta_r$ which is trivially reduced to the result. □

Algorithm 5 details the standard dual-tree traversal for cover trees, adapted from [8]. The traversal is begun on trees $\mathscr{T}_q$ and $\mathscr{T}_r$ by calling Algorithm 5 with root($\mathscr{T}_q$) and { root($\mathscr{T}_r$) }.

Algorithm 5 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 $(s_q < s_r^{\max})$ **or** $(s_r^{\max} = -\infty)$ **then**
  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
  for each $\mathscr{N}_{qc} \in \mathscr{C}(\mathscr{N}_q)$ **do**
    $R' \leftarrow \{\mathscr{N}_r \in R : \texttt{Score}(\mathscr{N}_q, \mathscr{N}_r) \neq \infty\}$
    recurse with $\mathscr{N}_{qc}$ and R'
  end for
end if

Note that the traversal given in Algorithm 5 attempts to descend the two trees in such a way that the scales s_q and $s_r^{\max}$ remain close to equal. Thus, the traversal's running time will depend on the differences in scales of each tree. To quantify this difference, we introduce a definition based on the degree of bichromaticity defined in [57].

Definition 5. *Let $\mathscr{T}_q$ and $\mathscr{T}_r$ be two cover trees built on query set S_q and reference set S_r, respectively.*

Now consider a pruning dual-tree traversal (such as Algorithm 5 with the property that the scales of nodes in $\mathscr{T}_q$ and $\mathscr{T}_r$ are kept as close as possible—that is, the tree with the larger scale is always descended. Then, the **inverse degree of bichromaticity** ν *of the tree pair $(\mathscr{T}_q, \mathscr{T}_r)$ is the maximum number of recursions in $\mathscr{T}_r$ following a recursion in $\mathscr{T}_q$ before another recursion in $\mathscr{T}_q$ or the termination of the algorithm (whichever happens first).*

This quantity is related to the **degree of bichromaticity** *[57], which is the maximum number of recursions in $\mathscr{T}_q$ between any two recursions in $\mathscr{T}_r$.*

Using these definitions and lemmas, we can show the main result of this section.

Theorem 8. *Given a Mercer kernel $\mathcal{K}(\cdot,\cdot)$, a reference set S_r of size N with expansion constant c_r and directional concentration constant γ_r, a query set S_q of size $O(N)$, and with α defined as*

$$\alpha = 1 + \frac{2\eta_r}{\tau_q}, \tag{41}$$

the dual-tree FastMKS algorithm using cover trees and the standard dual-tree cover tree traversal on $\mathscr{T}_q$ (a cover tree built on S_q) and $\mathscr{T}_r$ (a cover tree built on S_r) with inverse degree of bichromaticity ν requires $O(\gamma_r c_r^{(7\log_2\alpha)}\nu N)$ time.

Proof. Consider a reference recursion (lines 4—11). The work done in the base case loop from lines 5–7 is $O(|R|)$. This is bounded as $|R| \leq |R^*|$, where $|R^*|$ is the largest set $|R|$ for any scale $s_r^{\max}$ and any query node $\mathscr{N}_q$ during the course of the dual-tree recursion.

Then, lines 9 and 10 take $O(c_r^4|R_r|) \leq O(c_r^4|R^*|)$ time; this is due to the width bound (Lemma 1). So, one full reference recursion takes $O(c_r^4|R^*|)$ time.

Now, note that there are $O(N)$ nodes in $\mathscr{T}_q$. Thus, line 15 is visited $O(N)$ times (remember, query nodes cannot be pruned, so every one is visited). Each of these $O(N)$ visits to line 15 implies a recursion, in which the reference set is descended up to ν times 4—11) before the query node is descended or the algorithm terminates. In addition, each $O(N)$ recursion implies an $O(|R|) \leq O(|R^*|)$ operation for the calculation of R' (line 14. Thus, the full runtime of the algorithm is bounded as $O(c_r^4|R^*|\nu N + |R^*|N) = O(c_r^4|R^*|\nu N)$.

The next step is to produce a bound on $|R^*|$. Consider some reference set R encountered with maximum reference scale $s_r^{\max}$ and query node $\mathscr{N}_q$. Every node $\mathscr{N}_r \in R$ satisfies the property enforced in line 10 that

$$\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) \geq B(\mathscr{N}_q). \tag{42}$$

Remembering that $\sqrt{\mathcal{K}(p,p)} = \|\varphi(p)\|_{\mathcal{H}}$, we can relax $B(\mathscr{N}_q)$ (Equation 31) for the cover tree (where $\rho_i = 0$ for all $\mathscr{N}_i$) to show

$$\begin{aligned} B(\mathscr{N}_q) &\geq \max_{p \in \mathscr{P}_q} \left(k^*(p) + \lambda_q \|\varphi(p^*(p))\|_{\mathcal{H}}\right) \\ &= k^*(p_q) - \lambda_q \|\varphi(p^*(p_q))\|_{\mathcal{H}} \end{aligned} \tag{43}$$

which we can combine with Equation 42 to obtain

$$\begin{aligned} \mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) &\geq k^*(p_q) + \lambda_q \|\varphi(p_q)\|_{\mathcal{H}} \\ \mathcal{K}(p_q, p_r) &\geq k^*(p_q) - \lambda_q\big(\|\varphi(p_r)\|_{\mathcal{H}} + \\ &\quad \|\varphi(p^*(p_q))\|_{\mathcal{H}}\big) - \lambda_r \|\varphi(p_q)\|_{\mathcal{H}} - \lambda_q\lambda_r \end{aligned} \tag{44}$$

and, remembering that the scale of $\mathscr{N}_q$ is s_q and the scale of $\mathscr{N}_r$ is bounded above by $s_r^{\max}$, we simplify further to

$$\begin{aligned} \mathcal{K}(p_q, p_r) \geq k^*(p_q) - 2^{s_q+1}\big(\|\varphi(p_r)\|_{\mathcal{H}} + \|\varphi(p^*(p_q))\|_{\mathcal{H}}\big) \\ - 2^{s_r^{\max}+1}\|\varphi(p_q)\|_{\mathcal{H}} - 2^{s_q+s_r^{\max}+2}. \end{aligned} \tag{45}$$

We can express this conditional as membership in a set I_{S_r} by first defining the true maximum kernel value for p_q as

$$\hat{k}(p_q) = \max_{p_r \in S_r} \mathcal{K}(p_q, p_r). \tag{46}$$

The condition (Equation 45) can be stated as membership in a set:

$$\varphi(p_r) \in I_{S_r}\Big(\varphi(p_q), \big[b_l, \hat{k}(p_q)\big]\Big) \tag{47}$$

where

$$\begin{aligned} b_l = k^*(p_q) - 2^{s_q+1}\big(\|\varphi(p_r)\|_{\mathcal{H}} + \|\varphi(p^*(p_q))\|_{\mathcal{H}}\big) \\ - 2^{s_r^{\max}+1}\|\varphi(p_q)\|_{\mathcal{H}} - 2^{s_q+s_r^{\max}+2}. \end{aligned} \tag{48}$$

Now, we produce a lower bound for b_l. Note that $\hat{k}(p_q) \leq k^*(p_q) + 2^{s_r^{\max}+1}\|\varphi(p_q)\|_{\mathcal{H}}$, and see

$$\begin{aligned} b_l &\geq \hat{k}(p_q) - 2^{s_q+1}\big(\|\varphi(p_r)\|_{\mathcal{H}} + \|\varphi(p^*(p_q))\|_{\mathcal{H}}\big) \\ &\quad -2^{s_r^{\max}+2}\|\varphi(p_q)\|_{\mathcal{H}} - 2^{s_q+s_r^{\max}+2} \\ &\geq \hat{k}(p_q) - 2^{s_r^{\max}+1}\big(\|\varphi(p_r)\|_{\mathcal{H}} + \|\varphi(p^*(p_q))\|_{\mathcal{H}}\big) \\ &\quad -2^{s_r^{\max}+2}\|\varphi(p_q)\|_{\mathcal{H}} - 2^{2s_r^{\max}+2} \end{aligned} \tag{49}$$

which follows because $s_q < s_r^{\max}$ during a reference recursion (see line 4). Using the maximum and minimum norms defined earlier, we can bound b_l further:

$$
\begin{aligned}
b_l &\geq \hat{k}(p_q) - 2^{s_r^{\max}+1}(\eta_r + \eta_r) - 2^{s_r^{\max}+2}\|\varphi(p_q)\|_{\mathcal{H}} \\
&\quad - 2^{2s_r^{\max}+2} \\
&= \hat{k}(p_q) - 2^{s_r^{\max}+2}\left(\|\varphi(p_q)\|_{\mathcal{H}} + \eta_r + 2^{s_r^{\max}}\right) \\
&\geq \mathcal{K}(p_q, p_r) - 2^{s_r^{\max}+2}\left(\|\varphi(p_q)\|_{\mathcal{H}} + \eta_r + 2^{s_r^{\max}}\right) \\
&\geq \mathcal{K}(p_q, p_r) - 2^{s_r^{\max}+2}\left(\|\varphi(p_q)\|_{\mathcal{H}} + \eta_r + 2^{s_r^{T}}\right) \\
&\geq \mathcal{K}(p_q, p_r) - 2^{s_r^{\max}+2}\left(\|\varphi(p_q)\|_{\mathcal{H}} + 2\eta_r\right)
\end{aligned}
$$

where the last two bounding steps result from Lemma 5. Now, note that

$$
\frac{b_l}{\|\varphi(p_q)\|_{\mathcal{H}}} \geq \langle u, \varphi(p_r)\rangle_{\mathcal{H}} - 2^{s_r^{\max}+2}\left(1 + \frac{\eta_r + 2^{s_r^T}}{\tau_q}\right) \tag{50}
$$

then set $\alpha = 1 + (2\eta_r/\tau_q)$ (α is not dependent on the scale $s_r^{\max}$; this is important) and use the conditional from Equation 47 to get

$$
\begin{aligned}
\varphi(p_r) &\in I_{S_r}(\varphi(p_q), [b_l, \hat{k}(p_q)]) \\
&\subseteq I_{S_r}(\varphi(p_q), [b_l, \mathcal{K}(p_q, p_r) + 2^{s_r^{\max}+1}\|\varphi(p_q)\|_{\mathcal{H}}]) \\
&\subseteq I_{S_r}(u, [\langle u, \varphi(p_r)\rangle_{\mathcal{H}} - 2^{s_r^{\max}+2}\alpha, \\
&\qquad \langle u, \varphi(p_r)\rangle_{\mathcal{H}} + 2^{s_r^{\max}+1}]) \\
&\subseteq I_{S_r}(u, [\langle u, \varphi(p_r)\rangle_{\mathcal{H}} - 2^{s_r^{\max}+2}\alpha, \\
&\qquad \langle u, \varphi(p_r)\rangle_{\mathcal{H}} + 2^{s_r^{\max}+2}\alpha]).
\end{aligned} \tag{51}
$$

This is true for each point p_i of each node $\mathcal{N}_i$ in R_i. Thus, if we can place a bound on the number of points in the set given in Equation 51, then we are placing a bound on $|R_i|$ for any scale s_i. To this end, we can use the definition of directional concentration constant, to show that there exist γ_r points $p_j \in S_r$ such that

$$
\begin{aligned}
&I_{S_r}(u, [\langle u, \varphi(p_r)\rangle_{\mathcal{H}} - 2^{s_r+2}\alpha, \\
&\qquad \langle u, \varphi\rangle(p_r)_{\mathcal{H}} + 2^{s_r+2}\alpha]) \\
&\qquad\qquad \subseteq \bigcup_{j=1}^{\gamma_r} B_{S_r}(p_j, 2^{s_r+2}\alpha).
\end{aligned} \tag{52}
$$

By Lemma 3, each point p_r of each node $\mathcal{N}_r \in R$ must be separated by at least $2^{s_r^{\max}}$, because each point in R must have a parent with scale at least $s_r^{\max}+1$. Thus, we must bound the number of balls of radius $2^{s_r^{\max}-1}$ that can be packed into the set defined by Equation 52. For each p_j, we have

$$
\begin{aligned}
|B_{S_r}(p_j, 2^{s_r^{\max}+2}\alpha)| &\leq c_r^2 |B_{S_r}(p_j, 2^{s_r^{\max}-1}\alpha)| \\
&\leq c_r^{3\log_2\alpha} |B_{S_r}(p_j, 2^{s_r^{\max}-1})|.
\end{aligned}
$$

This allows us to conclude that $|R^*| \leq \gamma_r c_r^{(3\log_2\alpha)}$ and therefore the total running time of the algorithm is $O(\gamma_r c_r^{(7\log_2\alpha)}\nu N)$, and the theorem holds. □

Note that if dual-tree FastMKS is being run with the same set as the query set and reference set, $\nu = 1$, yielding a tighter bound.

10 Empirical evaluation

We evaluate single-tree and dual-tree FastMKS with different kernels and datasets. For each experiment, we query the top $\{1, 2, 5, 10\}$ max-kernel candidates and report the speedup over linear search (in terms of the number of kernel evaluations performed during the search). The cover tree and the algorithms are implemented in C++ in the **mlpack** machine learning library [16].

10.1 Datasets

We use two different classes of datasets. First, we use datasets with fixed-length objects. These include the MNIST dataset [38], the Isomap "Images" dataset, several datasets from the UCI machine learning repository [3], three collaborative filtering datasets (MovieLens, Netflix [4], Yahoo! Music [20]), the LCDM astronomy dataset [44], the LiveJournal blog moods text dataset [32] and a subset of the 80 Million Tiny Images dataset [61]. The sizes of the datasets are presented in Table 2.

The second class of dataset we use are those without fixed length representation. We use protein sequences from GenBank[10].

10.2 Kernels

We consider the following kernels for the vector datasets:

- linear: $\mathcal{K}(x, y) = x^T y$
- polynomial: $\mathcal{K}(x, y) = (x^T y)^2$
- cosine: $\mathcal{K}(x, y) = (x^T y)/(\|x^T\|\|y\|)$
- polynomial, deg. 10: $\mathcal{K}(x, y) = (x^T y)^{10}$
- Epanechnikov: $\mathcal{K}(x, y) = \max(0, 1 - \|x - y\|^2/b^2)$

While the Epanechnikov kernel is normalized and thus reduces to nearest neighbor search, we choose it regardless to show the applicability of FastMKS to a variety of kernels. It is important to remember

[10] See ftp://ftp.ncbi.nih.gov/refseq/release/complete.

Datasets	$\lvert S_q \rvert$	$\lvert S_r \rvert$	*dims*
Y! Music	10000	624961	51
MovieLens	6040	3706	11
Opt-digits	450	1347	64
Physics	37500	112500	78
Homology	75000	210409	74
Covertype	100000	481012	55
LiveJournal	10000	10000	25327
MNIST	10000	60000	784
Netflix	17770	480189	51
Corel	10000	27749	32
LCDM	6000000	10777216	3
TinyImages	1000	1000000	384

Table 2: **Details of the vector datasets.** $|S_q|$ and $|S_r|$ denote the number of objects in the query and reference sets respectively and *dims* denotes the dimensionality of the sets.

that standard techniques for nearest neighbor search should be able to perform the task faster—we do not compare with those techniques in these experiments.

For the protein sequences, we use the p-spectrum string kernel [42], which is a measure of string similarity. The kernel value for two given strings is the number of length-p substrings that appear in both strings.

10.3 Implementation

For maximum performance, the implementation in **mlpack** does not precisely follow the algorithms we have given. By default, the cover tree is designed to use a base of 2 during construction, but following the authors' observations, we find that a base of 1.3 seems to give better performance results [8]. In addition, for both the single-tree algorithms, we attempt to first score nodes (and node combinations) whose kernel values $\mathcal{K}(p_q, p_r)$ are higher, in hopes of tightening the bounds $B(\mathcal{N}_q)$ and $k^*(p_q)$ more quickly.

Lastly, the `Score()` method as implemented in **mlpack** is somewhat more complex: it attempts to prune the node combination $(\mathcal{N}_q, \mathcal{N}_r)$ with a looser bound that does not evaluate $\mathcal{K}(p_q, p_r)$. If that is not successful, `Score()` proceeds as in Algorithm 4 (or 2 in the single-tree case). This type of prune seems to give 10–30% reductions in the number of kernel evaluations (or more, depending on the dataset).

The **mlpack** implementation can be downloaded from `http://www.mlpack.org/` and its FastMKS implementation includes both C++ library bindings for FastMKS and each kernel we have discussed as well as a `fastmks` executable that can be used to run FastMKS easily from the command line. In addition, a tutorial can be found on the website, and the source code is extensively documented.

10.4 Results

The results for the vector datasets are summarized in Figure 5 and detailed for $k = 1$ in Tables 3 and 4. The tables also provide the number of kernel evaluations calculated during the search for linear search, single-tree FastMKS, and dual-tree FastMKS. Speedups over a factor of 100 are highlighted in bold. While the speedups range from anywhere between 1 (which indicates no speedup) to 50000, many datasets give speedups of an order of magnitude or more. As would be expected with the $O(\log N)$ bounds for single-tree FastMKS and the $O(N)$ bounds for dual-tree FastMKS, larger datasets (such as LCDM) tend to provide larger speedups. In the cases where large datasets are used but small speedup values are obtained, the conclusion must be that the expansion constant c_r and the directional concentration constant γ_r for that dataset and kernel are large. In addition, the Epanechnikov kernel is parameterized by a bandwidth b; this bandwidth will seriously affect the runtime if it is too small (all kernel evaluations are 0) or too large (all kernel evaluations are 1). We have arbitrarily chosen 10 as our bandwidth for simplicity in simulations, but for each dataset, it is certain that a better bandwidth value that will provide additional speedup exists.

Another observation is that the single-tree algorithm tends to perform better than the dual-tree algorithm, in spite of the better scaling of the dual-tree algorithm. There are multiple potential explanations for this phenomenon:

- The single-tree bounds given in Theorem 1 (Equation 10) and Theorem 3 (Equation 20) are tighter than the dual-tree bounds of Theorem 2 (Equation 14) and Theorem 4 (Equation 23).
- The dual-tree algorithm's runtime is also bounded by the parameters ν, η_r, and τ_q, whereas the single-tree algorithm is not. This could mean that N would need to be very large before the dual-tree algorithm became faster, despite the fact that the dual-tree algorithm scales with c_r^7 and the single-tree algorithm scales with c_r^{12}.
- The single-tree algorithm scales considers each element in the set $|S_q|$ linearly, but the dual-tree algorithm is able to obtain max-kernel bounds

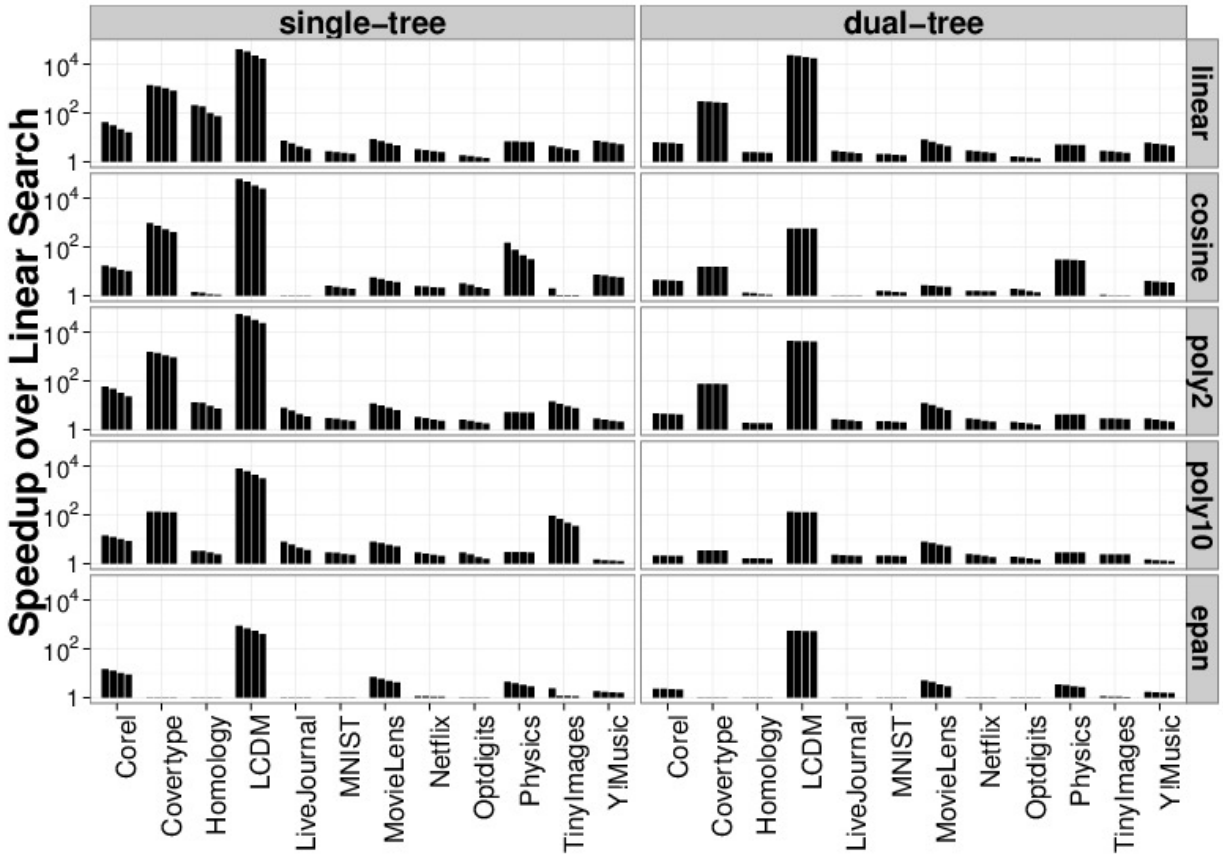


Figure 5: Speedups of single-tree and dual-tree FastMKS over linear scan with $k = \{1, 2, 5, 10\}$.

for many query points at once thanks to the use of the second tree. Thus, the dual-tree algorithm may require a much larger S_q before it outperforms the single-tree algorithm.

The results for the protein sequence data are shown in Figure 6 and Table 5. The table shows that for constant reference set size (649), the dual-tree algorithm provides better scaling as the query set grows. This agrees with the better scaling of dual-tree FastMKS as exhibited in Theorem 8.

However, in every case in Table 5, the single-tree algorithm provides better performance than the dual-tree algorithm. This implies that the query sets and reference sets would have to be possibly several orders of magnitude larger for the dual-tree algorithm to provide better speedups. With larger datasets, the single-tree algorithm showed more than 3000x speedup over linear scan. Other datasets may exhibit better or worse scaling depending on the expansion constant and directional concentration constant.

11 Approximate Extensions

For further scalability, we can develop an extension of FastMKS that does not return the exact max-kernel value but instead an approximation thereof. Even though we are focusing on exact max-kernel search, we wish to demonstrate that the tree based method can be very easily extended to perform the approximate max-kernel search. For any query p_q, we are seeking $\hat{p}(p_q) = \arg\max_{p_r \in S_r} \mathcal{K}(p_q, p_r)$. Let $\mathcal{K}(p_q, \hat{p}(p_q)) = \hat{k}(p_q)$ (as before). Then approximation can be achieved in the following ways:

1. Absolute value approximation: for all queries $p_q \in S_q$, find $p_r \in S_r$ such that $\mathcal{K}(p_q, p_r) \geq \hat{k}(p_q) - \epsilon$ for some $\epsilon > 0$.

2. Relative value approximation: for all queries $p_q \in S_q$, find $p_r \in S_r$ such that $\mathcal{K}(p_q, p_r) \geq (1 - \epsilon)\hat{k}(p_q)$ for some $\epsilon > 0$[11].

[11] Here we are assuming that $\hat{k}(p_q) > 0$. In the case where $\hat{k}(p_q) < 0$, we seek a $p_r \in S_r$ such that $\mathcal{K}(p_q, p_r) > \hat{k}(p_q) - \epsilon|\hat{k}(p_q)|$

		Kernel evaluations			Speedup	
Kernel	Dataset	Linear scan	Single-tree	Dual-tree	Single-tree	Dual-tree
linear	Y! Music	6.249B	859.1M	1.056B	7.27	5.91
	MovieLens	22.38M	2.635M	2.790M	8.49	8.02
	Optdigits	606.1k	333.2k	366.6k	1.82	1.65
	Physics	4.219B	628.8M	852.9M	6.71	4.95
	Bio	20.36B	100.2M	8.174B	**203.2**	2.49
	Covertype	48.10B	35.06M	160.9M	**1372**	**299.0**
	LiveJournal	100.0M	13.88M	36.09M	7.21	2.77
	MNIST	600.0M	229.6M	288.2M	2.62	2.08
	Netflix	8.532B	2.632B	2.979B	3.12	2.86
	Corel	277.5M	6.626M	44.02M	41.88	6.30
	LCDM	64.66T	1.566B	2.778B	**41282**	**23269**
	TinyImages	100.0M	22.30M	35.70M	4.48	2.80
polynomial	Y! Music	6.249B	2.187B	2.221B	2.86	2.81
	MovieLens	22.38M	1.865M	1.833M	12.00	12.21
	Optdigits	606.1k	235.1k	296.5k	2.58	2.04
	Physics	4.219B	823.9M	1.017B	5.12	4.15
	Bio	20.36B	1.538B	10.87B	13.23	1.87
	Covertype	48.10B	30.65M	629.7M	**1569**	76.39
	LiveJournal	100.0M	12.91M	38.16M	7.75	2.62
	MNIST	600.0M	202.8M	266.8M	2.96	2.25
	Netflix	8.532B	2.528B	2.953B	3.37	2.89
	Corel	277.5M	4.687M	60.30M	59.20	4.60
	LCDM	64.66T	1.171B	14.98B	**55204**	**4316**
	TinyImages	100.0M	6.957M	34.32M	14.37	2.91
polynomial-deg10	Y! Music	6.249B	4.296B	4.310B	1.45	1.45
	MovieLens	22.38M	2.814M	2.826M	7.96	7.92
	Optdigits	606.1k	212.3k	318.2k	2.86	1.91
	Physics	4.219B	1.441B	1.481B	2.93	2.91
	Bio	20.36B	6.018B	12.45B	3.38	1.63
	Covertype	48.10B	361.1M	13.78B	**133.2**	3.49
	LiveJournal	100.0M	12.75M	43.25M	7.84	2.31
	MNIST	600.0M	205.4M	277.1M	2.92	2.17
	Netflix	8.532B	2.977B	3.470B	2.87	2.46
	Corel	277.5M	19.68M	131.1M	14.10	2.12
	LCDM	64.66T	8.124B	485.2B	**7959**	**133.3**
	TinyImages	100.0M	1.076M	42.23M	92.96	2.37
cosine	Y! Music	6.249B	849.6M	1.586B	7.36	3.94
	MovieLens	22.38M	4.044M	8.322M	5.54	2.69
	Optdigits	606.1k	190.0k	319.8k	3.19	1.90
	Physics	4.219B	28.82M	140.0M	**146.3**	30.14
	Bio	20.36B	14.40B	15.54B	1.41	1.31
	Covertype	48.10B	50.15M	3.119B	**959.2**	15.42
	LiveJournal	100.0M	99.23M	98.78M	1.01	1.01
	MNIST	600.0M	237.0M	376.7M	2.53	1.59
	Netflix	8.532B	3.426B	5.344B	2.49	1.60
	Corel	277.5M	16.22M	61.95M	17.10	4.48
	LCDM	64.66T	1.058B	112.9B	**61063**	**572.6**
	TinyImages	100.0M	50.49M	92.36M	1.98	1.02

Table 3: Single-tree and dual-tree FastMKS on vector datasets with $k = 1$.

		Kernel evaluations			Speedup	
Kernel	Dataset	Linear scan	Single-tree	Dual-tree	Single-tree	Dual-tree
Epanechnikov	Y! Music	6.249B	3.439B	3.630B	1.82	1.72
	MovieLens	22.38M	3.243M	4.471M	6.90	5.01
	Optdigits	606.1k	606.1k	606.1k	1.00	1.00
	Physics	4.219B	957.6M	1.213B	4.40	3.48
	Bio	20.36B	20.25B	20.25B	1.01	1.01
	Covertype	48.10B	48.10B	48.10B	1.00	1.00
	LiveJournal	100.0M	99.57M	99.15M	1.00	1.01
	MNIST	600.0M	600.0M	600.0M	1.00	1.00
	Netflix	8.532B	7.602B	8.293B	1.12	1.03
	Corel	277.5M	18.53M	119.9M	14.98	2.31
	LCDM	64.66T	72.32B	119.0B	**894.1**	**543.3**
	TinyImages	100.0M	42.49M	87.99M	2.35	1.14

Table 4: Single-tree and dual-tree FastMKS on vector datasets with the Epanechnikov kernel with $k = 1$.

		Kernel evaluations			Speedup	
$\|S_q\|$	$\|S_r\|$	Linear scan	Single-tree	Dual-tree	Single-tree	Dual-tree
391	649	253.8k	5.255k	43.27k	48.29	5.87
1091	649	708.1k	14.99k	122.7k	47.25	5.77
2635	649	1.710M	36.04k	327.7k	47.45	5.22
8604	649	5.584M	115.3k	832.9k	48.43	6.70
37606	649	24.41M	512.9k	3.763M	47.58	6.49
63180	649	41.00M	848.1k	4.999M	48.35	8.20
63180	391	24.70M	484.3k	3.511M	51.01	7.04
63180	1091	68.93M	834.9k	7.529M	82.56	9.16
63180	2635	166.5M	927.8k	22.95M	**179.4**	7.25
63180	8604	543.6M	692.5k	32.26M	**785.1**	16.85
63180	37606	2.376B	743.2k	65.09M	**3197**	36.50
63180	63180	3.992B	1.140M	150.2M	**3500**	26.59
391	391	152.8k	2.973k	30.68k	51.42	4.98
1091	1091	1.190M	14.96k	183.2k	79.56	6.50
2635	2635	6.943M	43.95k	1.689M	**158.0**	4.11
8604	8604	323.6M	104.4k	13.76M	**783.2**	12.79
37606	37606	1.414B	470.2k	39.70M	**3007**	35.62
63180	63180	3.992B	1.141M	150.2M	**3500**	26.59

Table 5: Single-tree and dual-tree FastMKS on protein sequences with $k = 1$.

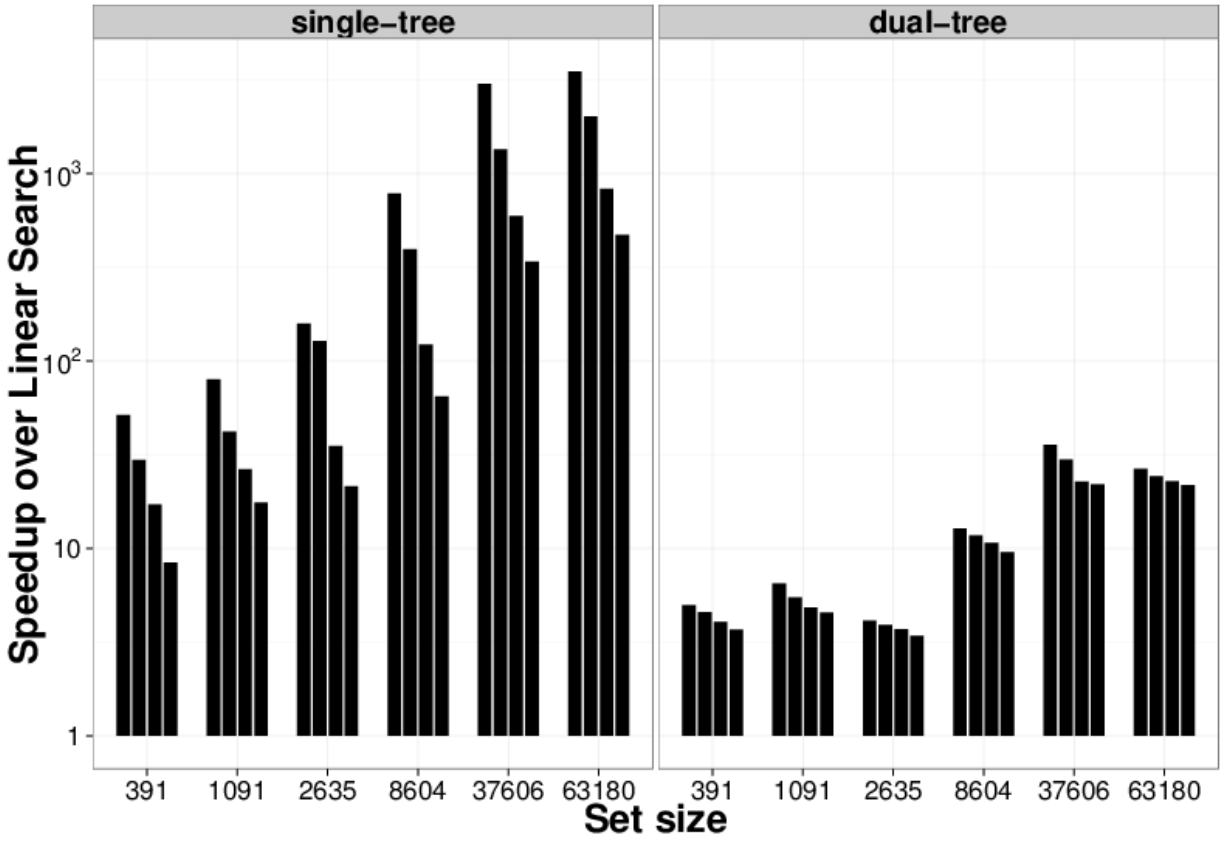


Figure 6: Speedups of single-tree and dual-tree FastMKS over linear scan for protein sequences with $k = \{1, 2, 5, 10\}$.

3. Rank approximation: return $p_r \in S_r$ such that $|\{p'_r \in S_r : \mathcal{K}(p_q, p'_r) > \mathcal{K}(p_q, p_r)\}| \leq \tau$.

The following three subsections present how single-tree FastMKS can be easily extended for approximate max-kernel search.

11.1 Absolute value approximation

From Theorem 1 and Algorithm 2, at any point in the single-tree algorithm with query point p_q and node $\mathscr{N}_i$ and best candidate kernel value $k^*(p_q)$, we know that we must descend $\mathscr{N}_i$ if

$$\mathcal{K}_{\max}(p_q, \mathscr{N}_i) \geq k^*(p_q) \tag{53}$$

but with absolute value approximation for some ϵ, we can loosen the condition to

$$\mathcal{K}_{\max}(p_q, \mathscr{N}_i) \geq k^*(p_q) + \epsilon \tag{54}$$

which can be simplified:

$$\begin{aligned} \mathcal{K}(p_q, p_i) + \lambda_i \sqrt{\mathcal{K}(p_q, p_q)} &\geq k^*(p_q) + \epsilon \\ \mathcal{K}(p_q, p_i) + \lambda_i \sqrt{\mathcal{K}(p_q, p_q)} &\geq \mathcal{K}(p_q, p_i) + \epsilon \\ \lambda_i \sqrt{\mathcal{K}(p_q, p_q)} &\geq \epsilon. \end{aligned} \tag{55}$$

This yields that we can prune if $\epsilon > \lambda_i \sqrt{\mathcal{K}(p_q, p_q)}$. While this is looser than possible, it has the advantage that $\mathcal{K}(p_q, p_i)$ does not need to be calculated to prune $\mathscr{N}_i$. This yields a modified `Score()` algorithm, given in Algorithm 6.

In the dual-tree case, we must descend $(\mathscr{N}_q, \mathscr{N}_r)$ if

$$\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) \geq B(\mathscr{N}_q). \tag{56}$$

Using absolute value approximation this condition loosens to

$$\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) \geq B(\mathscr{N}_q) + \epsilon \tag{57}$$

but we cannot easily simplify this to eliminate the evaluation of $\mathcal{K}(p_q, p_r)$ due to the complexity of $B(\mathscr{N}_q)$. A modified `Score()` function for dual-tree

Algorithm 6 `Score(`p_q`,` $\mathscr{N}_r$`)` for absolute value approximation of FastMKS.

Input: query point p_q, reference space tree node $\mathscr{N}_r$, max-kernel candidate p^* for p_q and corresponding max-kernel value k^*, absolute value approximation ϵ
Output: a score for the node, or ∞ if the node can be pruned
if $\epsilon > \lambda_r \sqrt{\mathcal{K}(p_q, p_q)}$ **then**
  return ∞
else if $\mathcal{K}_{\max}(p_q, \mathscr{N}_r) < k^*$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(p_q, \mathscr{N}_r)$
end if

Algorithm 7 `Score(`$\mathscr{N}_q$`,` $\mathscr{N}_r$`)` for absolute value approximation of FastMKS.

Input: query node $\mathscr{N}_q$, reference node $\mathscr{N}_r$, absolute value approximation ϵ
Output: a score for the node combination $(\mathscr{N}_q, \mathscr{N}_r)$ or ∞ if the combination can be pruned
if $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) < B(\mathscr{N}_q) + \epsilon$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r)$
end if

Algorithm 8 `Score(`p_q`,` $\mathscr{N}_r$`)` for relative value approximation of FastMKS.

Input: query point p_q, reference space tree node $\mathscr{N}_r$, max-kernel candidate p^* for p_q and corresponding max-kernel value k^*, relative value approximation ϵ
Output: a score for the node, or ∞ if the node can be pruned
if $k^* > (\lambda_r / \epsilon) \sqrt{\mathcal{K}(p_q, p_q)}$ **then**
  return ∞
else if $\mathcal{K}_{\max}(p_q, \mathscr{N}_r) < k^*$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(p_q, \mathscr{N}_r)$
end if

Algorithm 9 `Score(`$\mathscr{N}_q$`,` $\mathscr{N}_r$`)` for relative value approximation of FastMKS.

Input: query node $\mathscr{N}_q$, reference node $\mathscr{N}_r$, relative value approximation ϵ
Output: a score for the node combination $(\mathscr{N}_q, \mathscr{N}_r)$ or ∞ if the combination can be pruned
if $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) < (1 + \epsilon) B(\mathscr{N}_q)$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r)$
end if

absolute value approximate FastMKS is given in Algorithm 7.

11.2 Relative value approximation

Relative value approximation is a more useful form of approximation, because the user does not need knowledge of $\hat{k}(p_q)$ to set ϵ reasonably. However, care has to be taken for relative value approximation because there is no guarantee that $\hat{k}(p_q) > 0$.

We can take Equation 53 and modify it for ϵ-relative-value-approximate pruning. In this case, we must descend $\mathscr{N}_i$ if

$$\mathcal{K}_{\max}(p_q, \mathscr{N}_i) \geq (1 + \epsilon) k^*(p_q) \tag{58}$$

and similar algebraic manipulations yield

$$\begin{aligned} \mathcal{K}(p_q, p_i) + \lambda_i \sqrt{\mathcal{K}(p_q, p_q)} &\geq (1 + \epsilon) k^*(p_q) \\ \mathcal{K}(p_q, p_i) + \lambda_i \sqrt{\mathcal{K}(p_q, p_q)} &\geq \mathcal{K}(p_q, p_i) + \epsilon k^*(p_q) \\ \lambda_i \sqrt{\mathcal{K}(p_q, p_q)} &\geq \epsilon k^*(p_q) \end{aligned}$$

meaning we can prune a node $\mathscr{N}_i$ when $k^*(p_q) > (\lambda_i / \epsilon) \sqrt{\mathcal{K}(p_q, p_q)}$. This is looser than possible (like the absolute-value approximation bound) but has the advantage that $\mathcal{K}(p_q, p_i)$ does not need to be calculated to prune $\mathscr{N}_i$. This yields a modified `Score()` algorithm, given in Algorithm 8.

Similar to absolute value approximation, we can loosen the condition for recursion given in Equation 56 to obtain the rule

$$\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) \geq (1 + \epsilon) B(\mathscr{N}_q). \tag{59}$$

This rule does not easily simplify, as in the case of the single-tree relative value approximation rule; this is due to the complexity of $B(\mathscr{N}_q)$. A modified `Score()` function is given in Algorithm 9.

11.3 Rank Approximation

Rank approximation is a relatively new approximation paradigm introduced by Ram et al. [58]. The idea is to return a max-kernel candidate p'_r for query p_q, reference set S_r, and parameter τ such that p'_r is in the top τ max-kernel results with high probability.

That is, for p_q, S_r, and τ, find an object $p_r \in S_r$ such that

$$|\{p'_r \in S_r : \mathcal{K}(p_q, p'_r) > \mathcal{K}(q, p_r)\}| < \tau. \quad (60)$$

This is often a better technique than absolute-value-approximate search, which requires a tuned parameter ϵ for each dataset, and relative-value-approximate search, which may return useless results when the values of $\mathcal{K}(p_q, p_r)$ are very close for all $p_r \in S_r$.

The idea presented in [58] is to draw a set of samples S'_r large enough that the maximum kernel value between p_q and any point in S'_r (call this k^*) is such that

$$\Pr\left(|\{p'_r \in S_r : \mathcal{K}(p_q, p'_r) > k^*\}| < \tau\right) \geq 1 - \delta. \quad (61)$$

Simplifying the formulation presented in [58], the probability of always missing the top τ values for a given query p_q after k samples with replacement is given by $(1 - (\tau/n))^k$, where $|S_r| = n$. If we want a $(1 - \delta)$ success rate of sampling, then we want k to be such that

$$\left(1 - \frac{\tau}{n}\right)^k < \delta, \quad \text{and} \quad \left(1 - \frac{\tau}{n}\right)^{k-1} > \delta,$$

which gives

$$k = \left\lceil \frac{\log \delta}{\log\left(1 - \frac{\tau}{n}\right)} \right\rceil.$$

Following the logic of [58], if a node $\mathscr{N}_i$ contains more than (n/k) points ($|\mathscr{D}_i^p| > (n/k)$), then we can prune the node *after* we randomly sample $\lceil (k/n)|\mathscr{D}_i^p| \rceil$ points from $|\mathscr{D}_i^p|$. In addition to that, the standard FastMKS pruning rules still apply. An updated single-tree `Score()` function is given in Algorithm 10.

An extension of this for a dual-tree algorithm is straightforward; for a reference node $\mathscr{N}_r$, if $|\mathscr{D}_r^p| > (n/k)$, then we can sample it for each query point $p_q \in \mathscr{D}_q^p$ and prune the node combination. A `Score()` function is given in Algorithm 11.

Algorithm 10 `Score(`p_q`,` $\mathscr{N}_r$`)` for rank approximation of FastMKS.

Input: query point p_q, reference space tree node $\mathscr{N}_r$, max-kernel candidate p^* for p_q and corresponding max-kernel value k^*, required number of samples k for τ-rank approximation in a reference set of size n
Output: a score for the node, or ∞ if the node can be pruned
if $|\mathscr{D}_r^p| \leq (n/k)$ **then**
  $S'_r \leftarrow \lceil (k/n)|\mathscr{D}_r^p| \rceil$ random samples from $|\mathscr{D}_r^p|$
  for each $p'_r \in S'_r$ **do**
    `BaseCase(`p_q`,` p'_r`)`
  end for
  return ∞
else if $\mathcal{K}_{\max}(p_q, \mathscr{N}_r) < k^*$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(p_q, \mathscr{N}_r)$
end if

12 Conclusion

In this manuscript we have described two general-purpose algorithms for solving the max-kernel problem (Equation 1) when the kernel satisfies the non-restrictive condition that it is positive semidefinite (a Mercer kernel). With the exception of the publication this expanded work is based on [18], there exists no technique as general as ours for max-kernel search other than linear scan, which, for a query set S_q and a reference set S_r both of size N, scales quadratically $O(N^2)$.

We have detailed a generic tree-independent algorithm called FastMKS based on the tree-independent framework of [17] for both single-tree max-kernel search (Algorithms 1 and 2) and dual-tree max-kernel search (Algorithms 1 and 4). When the algorithm is using cover trees with a query set S_q and a reference set S_r both of size N, we have shown scaling of $O(N \log N)$ for the single-tree algorithm (Theorem 6) and $O(N)$ for the dual-tree algorithm (Theorem 8) though it should be remembered that these bounds depend on both the expansion constant and the directional concentration constant of the dataset.

Our tree-independent algorithms can be applied to any type of space tree that can be built using only distance evaluations between points in the dataset. Thus, even space trees designed after the publication of this paper can be easily paired with a pruning single-tree traversal or pruning dual-tree traversal and the base case and scoring functions given in this paper to produce a provably correct implementation of single-tree or dual-tree FastMKS (Theorems 5 and 7).

The empirical performance of our algorithms were evaluated after their implementation in **mlpack** [16]. Both the single-tree and dual-tree algorithms can produce speedups of over 50000, but despite the better

Algorithm 11 `Score(`$\mathscr{N}_q$`,` $\mathscr{N}_r$`)` for rank approximation of FastMKS.

Input: query node $\mathscr{N}_q$, reference node $\mathscr{N}_r$, required number of samples k for τ rank approximation in a reference set of size n
Output: a score for the node combination $(\mathscr{N}_q, \mathscr{N}_r)$ or ∞ if the combination can be pruned
if $|\mathscr{D}_r^p| \leq (n/k)$ **then**
  for each $p_q \in \mathscr{D}_q^p$ **do**
    $S'_r \leftarrow \lceil (k/n)|\mathscr{D}_r^p| \rceil$ random samples from $|\mathscr{D}_r^p|$
    for each $p'_r \in S'_r$ **do**
      `BaseCase(`p_q`,` p'_r`)`
    end for
  end for
  return ∞
else if $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r) > B(\mathscr{N}_q)$ **then**
  return ∞
else
  return $\mathcal{K}_{\max}(\mathscr{N}_q, \mathscr{N}_r)$
end if

asymptotic scaling of the dual-tree algorithm, it tends to be outperformed by the single-tree algorithm. We suspect that larger datasets are necessary to show the better scaling characteristics of the dual-tree algorithm.

Our reference implementation of single-tree and dual-tree FastMKS is open-source and available as part of the **mlpack** machine learning library, starting with version 1.0.6. The implementation is extensively documented and there is also a tutorial on the website (`http://www.mlpack.org/`).

12.1 Future improvements/extensions

Though we have shown significant speedups for the single-tree and dual-tree FastMKS algorithms, we believe that there is room for further improvements and extensions. Below we list two possible paths that are interesting and warrant further investigation.

12.1.1 Tighter bounds for specific kernels

In Theorems 3 and 4 we described a tighter bound for normalized kernels ($\mathcal{K}(x, x) = 1\ \forall x$). It is our intuition that similar tighter bounds can be developed for other specific types of Mercer kernels.

This may be especially applicable in domain-specific kernels such as string kernels or graph kernels. Any kernel that has some known structure on how points are mapped to $\mathcal{H}$ may be bounded more tightly than the general Mercer kernel bounds given in Equations 10 and 14.

12.1.2 Domain-specific applications

In the introduction, we mentioned the wide applicability of max-kernel search, discussing its use in image retrieval, document retrieval, collaborative filtering, and even finding similar protein/DNA sequences. That list only contains a few of the numerous max-kernel search problems that arise ubiquitiously in countless fields (not just computing-related fields).

In many of these fields, there are existing domain-specific solutions. One example in genomics is BLAST (Basic Local Alignment Search Tool) [1], a utility that searches for similarity between biological sequences. Another tool of this sort is the older FASTA algorithm [53]. Both of these algorithms are improvements over linear scan with the Smith-Waterman alignment score [60]. However, in contrast with its large speedups, BLAST cannot guarantee exact results.

The Smith-Waterman alignment score can easily be shown to be a Mercer kernel; therefore, FastMKS could be used to give speedups over linear scan and it would also return provably exact results. Furthermore, approximation extensions to FastMKS could provide additional speedups by relaxing the exact result constraint, potentially making FastMKS competitive with BLAST.

12.1.3 Massive parallelism

The implementation of FastMKS that we have provided requires that the datasets and trees both fit into memory. However, with today's datasets becoming larger and larger, this is often not feasible. Thus, a massively parallel implementation of FastMKS is desirable.

In the context of tree-independent dual-tree algorithms (of which FastMKS is one), one parallelization strategy is to parallelize the traversal [17]. Then, the parallel traversal can be applied with a type of space tree and the FastMKS `BaseCase()` and `Score()` functions; there is no modification necessary to the base case and pruning rule. One recent work that may be generalizable to a massively parallel pruning dual-tree traversal is that of Lee et al. [41], who proposed a distributed framework in the context of kernel summations. Other related parallelism schemes could likely be adapted to dual-tree FastMKS [17, 12] to provide significant speedup and enable FastMKS to be performed on huge datasets.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[2] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 1998.

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

[4] J. Bennett and S. Lanning. The Netflix Prize. In *Proceedings of the KDD Cup and Workshop*, pages 3–6, 2007.

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

[6] J.L. Bentley and J.H. Friedman. Fast algorithms for constructing minimal spanning trees in coordinate spaces. *IEEE Transactions on Computers*, 100(2):97–105, 1978.

[7] J.L. Bentley and J.H. Friedman. Data structures for range searching. *ACM Computing Surveys (CSUR)*, 11(4):397–409, 1979.

[8] 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, 2006.

[9] A. Beygelzimer, S.M. Kakade, and J.C. Langford. Cover Trees for Nearest Neighbor (longer version). *Paper URL*.

[10] K.M. Borgwardt, C.S. Ong, S. Schönauer, S. V. N. Vishwanathan, A.J. Smola, and H.P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl. 1):i47–i56, 2005.

[11] L. Cayton. Fast nearest neighbor retrieval for Bregman divergences. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*, pages 112–119, 2008.

[12] L. Cayton. Accelerating nearest neighbor search on manycore systems. In *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS '12)*, pages 402–413, 2012.

[13] M.S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC '02)*, pages 380–388, 2002.

[14] K.L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

[15] K.L. Clarkson. Nearest-neighbor searching and metric space dimensions. *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59, 2006.

[16] 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, 2013.

[17] 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)*, 2013.

[18] R.R. Curtin, P. Ram, and A.G. Gray. Fast exact max-kernel search. In *SIAM International Conference on Data Mining (SDM '13)*, pages 1–9, 2013.

[19] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th Annual ACM Symposium on Theory Of Computing (STOC '08)*, pages 537–546, 2008.

[20] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music Dataset and KDD-Cup'11. *Journal of Machine Learning Research (Proceedings Track)*, 18:8–18, 2012.

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

[22] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[23] K. Fukunaga and P. M. Nagendra. A branch-and-bound algorithm for computing k-nearest-neighbors. *IEEE Transactions on Computers*, 100(7):750–753, 1975.

[24] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases (VLDB '99)*, volume 99, pages 518–529, 1999.

[25] A.G. Gray and A.W. Moore. 'N-Body' problems in statistical learning. In *Advances in Neural Information Processing Systems 14 (NIPS '01)*, volume 4, pages 521–527, 2002.

[26] A.G. Gray and A.W. Moore. Nonparametric density estimation: Toward computational tractability. In *SIAM International Conference on Data Mining (SDM '03)*, pages 203–211, 2003.

[27] V.J. Hodge and J. Austin. A comparison of standard spell checking algorithms and a novel binary neural approach. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1073–1081, 2003.

[28] M.P. Holmes, A.G. Gray, and C.L. Isbell Jr. QUIC-SVD: Fast SVD using cosine trees. In *Advances in Neural Information Processing Systems (NIPS '08)*, volume 21, pages 673–680, 2009.

[29] C.L. Jackins and S.L. Tanimoto. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, 14(3):249–270, 1980.

[30] P. Kar and H. Karnick. Random feature maps for dot product kernels. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS '12)*, volume 22, pages 583–591, 2012.

[31] 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 '02)*, pages 741–750. ACM, 2002.

[32] S. Kim, F. Li, G. Lebanon, and I. Essa. Beyond sentiment: The manifold of human emotions. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS '13)*, pages 360–369, 2013.

[33] 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 481–488. ACM, 2006.

[34] W.L.G. Koontz, P.M. Narendra, and K. Fukunaga. A branch and bound clustering algorithm. *IEEE Transactions on Computers*, 100(9):908–915, 1975.

[35] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[36] 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 (SODA '04)*, pages 798–807, 2004.

[37] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the 12th IEEE International Conference on Computer Vision (ICCV '09)*, 2009.

[38] Y. LeCun, C. Cortes, and C.J.C. Burges. MNIST dataset, 2000. http://yann.lecun.com/exdb/mnist/.

[39] 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)*, 2006.

[40] 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 '08)*, 21, 2009.

[41] D. Lee, R.W. Vuduc, and A.G. Gray. A distributed kernel summation framework for general-dimension machine learning. In *SIAM International Conference on Data Mining (SDM '12)*, pages 391–402, 2012.

[42] C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, 2002.

[43] T. Liu, A.W. Moore, K. Yang, and A.G. Gray. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems 18 (NIPS '04)*, pages 825–832, 2004.

[44] R. Lupton, J.E. Gunn, Z. Ivezic, G.R. Knapp, S. Kent, and N. Yasuda. The SDSS imaging pipelines. *Astronomical Data Analysis Software and Systems X*, 238:269–278, 2001.

[45] 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, 2012.

[46] 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, 2010.

[47] J. McNames. A fast nearest-neighbor algorithm based on a principal axis search tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):964–976, 2001.

[48] A.W. Moore. Very fast EM-based mixture model clustering using multiresolution kd-trees. *Advances in Neural Information Processing Systems 11 (NIPS '98)*, pages 543–549, 1999.

[49] 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. Morgan Kaufmann Publishers Inc., 2000.

[50] M. Muja and D.G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications (VISAPP)*, 2009.

[51] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with Support Vector Machines. *Proceedings of the 7th International Conference on Artificial Neural Networks (ICANN '97)*, pages 999–1004, 1997.

[52] P.M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 100(9):917–922, 1977.

[53] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[54] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer, 1985.

[55] A. Rahimi and B. Recht. Random Features for Large-scale Kernel Machines. *Advances in Neural Information Processing Systems 20 (NIPS '07)*, pages 1177–1184, 2008.

[56] P. Ram and A.G. Gray. Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*, pages 931–939, 2012.

[57] 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 '09)*, 23:1527–1535, 2010.

[58] P. Ram, D. Lee, H. Ouyang, and A.G. Gray. Rank-approximate nearest neighbor search: Retaining meaning and speed in high dimensions. *Advances in Neural Information Processing Systems 22 (NIPS '09)*, 22:1536–1544, 2010.

[59] Y. Shen, A.Y. Ng, and M. Seeger. Fast Gaussian process regression using kd-trees. *Advances in Neural Information Processing Systems 18 (NIPS '05)*, pages 1225–1232, 2006.

[60] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[61] A. Torralba, R. Fergus, and W.T. Freeman. 80 Million Tiny Images: A large data set for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[62] J.K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

[63] P. Wang, D. Lee, A.G. Gray, and J.M. Rehg. Fast mean shift with accurate and stable convergence. In *Workshop on Artificial Intelligence and Statistics (AISTATS '07)*, 2007.

[64] P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '93)*, pages 311–321. Society for Industrial and Applied Mathematics, 1993.