Fast k-Nearest Neighbour Search via Prioritized DCI

Ke Li [1] **Jitendra Malik** [1]

Abstract

Most exact methods for k-nearest neighbour search suffer from the curse of dimensionality; that is, their query times exhibit exponential dependence on either the ambient or the intrinsic dimensionality. Dynamic Continuous Indexing (DCI) (Li & Malik, 2016) offers a promising way of circumventing the curse and successfully reduces the dependence of query time on intrinsic dimensionality from exponential to sublinear. In this paper, we propose a variant of DCI, which we call Prioritized DCI, and show a remarkable improvement in the dependence of query time on intrinsic dimensionality. In particular, a *linear* increase in intrinsic dimensionality, or equivalently, an *exponential* increase in the number of points near a query, can be mostly counteracted with just a *linear* increase in space. We also demonstrate empirically that Prioritized DCI significantly outperforms prior methods. In particular, relative to Locality-Sensitive Hashing (LSH), Prioritized DCI reduces the number of distance evaluations by a factor of 14 to 116 and the memory consumption by a factor of 21.

1. Introduction

The method of k-nearest neighbours is a fundamental building block of many machine learning algorithms and also has broad applications beyond artificial intelligence, including in statistics, bioinformatics and database systems, e.g. (Biau et al., 2011; Behnam et al., 2013; Eldawy & Mokbel, 2015). Consequently, since the problem of nearest neighbour search was first posed by Minsky & Papert (1969), it has for decades intrigued the artificial intelligence and theoretical computer science communities alike. Unfortunately, the myriad efforts at devising efficient algorithms have encountered a recurring obstacle: *the curse of dimensionality*, which describes the phenomenon of query time complexity depending exponentially on dimensionality. As a result, even on datasets with moderately high dimensionality, practitioners often have resort to naïve exhaustive search.

Two notions of dimensionality are commonly considered. The more familiar notion, ambient dimensionality, refers to the dimensionality of the space data points are embedded in. On the other hand, intrinsic dimensionality[1] characterizes the intrinsic properties of the data and measures the rate at which the number of points inside a ball grows as a function of its radius. More precisely, for a dataset with intrinsic dimension d', any ball of radius r contains at most $O(r^{d'})$ points. Intuitively, if the data points are uniformly distributed on a manifold, then the intrinsic dimensionality is roughly the dimensionality of the manifold.

Most existing methods suffer from some form of curse of dimensionality. Early methods like k-d trees (Bentley, 1975) and R-trees (Guttman, 1984) have query times that grow exponentially in ambient dimensionality. Later methods (Krauthgamer & Lee, 2004; Beygelzimer et al., 2006; Dasgupta & Freund, 2008) overcame the exponential dependence on ambient dimensionality, but have not been able to escape from an exponential dependence on intrinsic dimensionality. Indeed, since a linear increase in the intrinsic dimensionality results in an exponential increase in the number of points near a query, the problem seems fundamentally hard when intrinsic dimensionality is high.

Recently, Li & Malik (2016) proposed an approach known as Dynamic Continuous Indexing (DCI) that successfully reduces the dependence on intrinsic dimensionality from exponential to sublinear, thereby making high-dimensional nearest neighbour search more practical. The key observation is that the difficulties encountered by many existing methods, including k-d trees and Locality-Sensitive Hashing (LSH) (Indyk & Motwani, 1998), may arise from their reliance on space partitioning, which is a popular divide-and-conquer strategy. It works by partitioning the vector space into discrete cells and maintaining a data structure

[1]University of California, Berkeley, CA 94720, United States. Correspondence to: Ke Li <ke.li@eecs.berkeley.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

[1]The measure of intrinsic dimensionality used throughout this paper is the expansion dimension, also known as the KR-dimension, which is defined as $\log_2 c$, where c is the expansion rate introduced in (Karger & Ruhl, 2002).

that keeps track of the points lying in each cell. At query time, these methods simply look up of the contents of the cell containing the query and possibly adjacent cells and perform brute-force search over points lying in these cells. While this works well in low-dimensional settings, would it work in high dimensions?

Several limitations of this approach in high-dimensional space are identified in (Li & Malik, 2016). First, because the volume of space grows exponentially in dimensionality, either the number or the volumes of cells must grow exponentially. Second, the discretization of the space essentially limits the "field of view" of the algorithm, as it is unaware of points that lie in adjacent cells. This is especially problematic when the query lies near a cell boundary, as there could be points in adjacent cells that are much closer to the query. Third, as dimensionality increases, surface area grows faster than volume; as a result, points are increasingly likely to lie near cell boundaries. Fourth, when the dataset exhibits varying density across space, choosing a good partitioning is non-trivial. Furthermore, once chosen, the partitioning is fixed and cannot adapt to changes in density arising from updates to the dataset.

In light of these observations, DCI is built on the idea of avoiding partitioning the vector space. Instead, it constructs a number of indices, each of which imposes an ordering of all data points. Each index is constructed so that two points with similar ranks in the associated ordering are nearby along a certain random direction. These indices are then combined to allow for retrieval of points that are close to the query along multiple random directions.

In this paper, we propose a variant of DCI, which assigns a priority to each index that is used to determine which index to process in the upcoming iteration. For this reason, we will refer to this algorithm as Prioritized DCI. This simple change results in a significant improvement in the dependence of query time on intrinsic dimensionality. Specifically, we show a remarkable result: a *linear* increase in intrinsic dimensionality, which could mean an *exponential* increase in the number of points near a query, can be mostly counteracted with a corresponding *linear* increase in the number of indices. In other words, Prioritized DCI can make a dataset with high intrinsic dimensionality seem almost as easy as a dataset with low intrinsic dimensionality, with just a linear increase in space. To our knowledge, there had been no exact method that can cope with high intrinsic dimensionality; Prioritized DCI represents the first method that can do so.

We also demonstrate empirically that Prioritized DCI significantly outperforms prior methods. In particular, compared to LSH, it achieves a 14- to 116-fold reduction in the number of distance evaluations and a 21-fold reduction in the memory usage.

2. Related Work

There is a vast literature on algorithms for nearest neighbour search. They can be divided into two categories: exact algorithms and approximate algorithms. Early exact algorithms are deterministic and store points in tree-based data structures. Examples include k-d trees (Bentley, 1975), R-trees (Guttman, 1984) and X-trees (Berchtold et al., 1996; 1998), which divide the vector space into a hierarchy of half-spaces, hyper-rectangles or Voronoi polygons and keep track of the points that lie in each cell. While their query times are logarithmic in the size of the dataset, they exhibit exponential dependence on the ambient dimensionality. A different method (Meiser, 1993) partitions the space by intersecting multiple hyperplanes. It effectively trades off space for time and achieves polynomial query time in ambient dimensionality at the cost of exponential space complexity in ambient dimensionality.

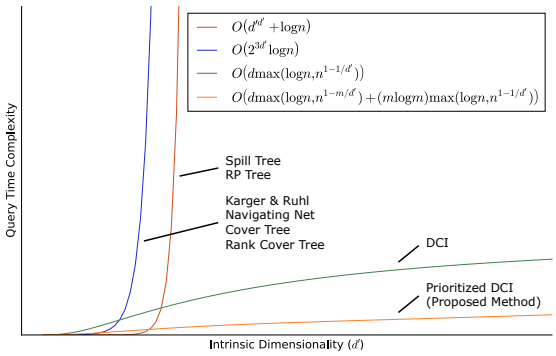


Figure 1. Visualization of the query time complexities of various exact algorithms as a function of the intrinsic dimensionality d'. Each curve represents an example from a class of similar query time complexities. Algorithms that fall into each particular class are shown next to the corresponding curve.

To avoid poor performance on worst-case configurations of the data, exact randomized algorithms have been proposed. Spill trees (Liu et al., 2004), RP trees (Dasgupta & Freund, 2008) and virtual spill trees (Dasgupta & Sinha, 2015) extend the ideas behind k-d trees by randomizing the orientations of hyperplanes that partition the space into half-spaces at each node of the tree. While randomization enables them to avoid exponential dependence on the ambient dimensionality, their query times still scale exponentially in the intrinsic dimensionality. Whereas these methods rely on space partitioning, other algorithms (Orchard, 1991; Clarkson, 1999; Karger & Ruhl, 2002) have been proposed that utilize local search strategies. These methods start with a random point and look in the neighbourhood of the current point to find a new point that is closer to the query than the original in each iteration. Like space partitioning-based approaches, the query time of (Karger & Ruhl, 2002) scales exponentially in the intrinsic dimensionality. While

Method	Query Time Complexity
Exact Algorithms:	
RP Tree	$O((d' \log d')^{d'} + \log n)$
Spill Tree	$O(d'^{d'} + \log n)$
Karger & Ruhl (2002)	$O(2^{3d'} \log n)$
Navigating Net	$2^{O(d')} \log n$
Cover Tree	$O(2^{12d'} \log n)$
Rank Cover Tree	$O(2^{O(d' \log h)} n^{2/h})$ for $h \geq 3$
DCI	$O(d \max(\log n, n^{1-1/d'}))$
Prioritized DCI	$O(d \max(\log n, n^{1-m/d'})$
(Proposed Method)	$+(m \log m) \max(\log n, n^{1-1/d'}))$ for $m \geq 1$
Approximate Algorithms:	
k-d Tree	$O((1/\epsilon)^d \log n)$
BBD Tree	$O((6/\epsilon)^d \log n)$
LSH	$\approx O(dn^{1/(1+\epsilon)^2})$

Table 1. Query time complexities of various algorithms for 1-NN search. Ambient dimensionality, intrinsic dimensionality, dataset size and approximation ratio are denoted as d, d', n and $1 + \epsilon$. A visualization of the growth of various time complexities as a function of the intrinsic dimensionality is shown in Figure 1.

the query times of (Orchard, 1991; Clarkson, 1999) do not exhibit such undesirable dependence, their space complexities are quadratic in the size of the dataset, making them impractical for large datasets. A different class of algorithms performs search in a coarse-to-fine manner. Examples include navigating nets (Krauthgamer & Lee, 2004), cover trees (Beygelzimer et al., 2006) and rank cover trees (Houle & Nett, 2015), which maintain sets of subsampled data points at different levels of granularity and descend through the hierarchy of neighbourhoods of decreasing radii around the query. Unfortunately, the query times of these methods again scale exponentially in the intrinsic dimensionality.

Due to the difficulties of devising efficient algorithms for the exact version of the problem, there has been extensive work on approximate algorithms. Under the approximate setting, returning any point whose distance to the query is within a factor of $1 + \epsilon$ of the distance between the query and the true nearest neighbour is acceptable. Many of the same strategies are employed by approximate algorithms. Methods based on tree-based space partitioning (Arya et al., 1998) and local search (Arya & Mount, 1993) have been developed; like many exact algorithms, their query times also scale exponentially in the ambient dimensionality. Locality-Sensitive Hashing (LSH) (Indyk & Motwani, 1998; Datar et al., 2004; Andoni & Indyk, 2006) partitions the space into regular cells, whose shapes are implicitly defined by the choice of the hash function. It achieves a query time of $O(dn^\rho)$ using $O(dn^{1+\rho})$ space, where d is the ambient dimensionality, n is the dataset size and $\rho \approx 1/(1 + \epsilon)^2$ for large n in Euclidean space, though the dependence on intrinsic dimensionality is not made explicit. In practice, the performance of LSH degrades on datasets with large variations in density, due to the uneven distribution of points across cells. Consequently, various data-dependent hashing schemes have been proposed (Paulevé et al., 2010; Weiss et al., 2009; Andoni & Razenshteyn, 2015); unlike data-independent hashing schemes, however, they do not allow dynamic updates to the dataset. A related approach (Jégou et al., 2011) decomposes the space into mutually orthogonal axis-aligned subspaces and independently partitions each subspace. It has a query time linear in the dataset size and no known guarantee on the probability of correctness under the exact or approximate setting. A different approach (Anagnostopoulos et al., 2015) projects the data to a lower dimensional space that approximately preserves approximate nearest neighbour relationships and applies other approximate algorithms like BBD trees (Arya et al., 1998) to the projected data. Its query time is also linear in ambient dimensionality and sublinear in the dataset size. Unlike LSH, it uses space linear in the dataset size, at the cost of longer query time than LSH. Unfortunately, its query time is exponential in intrinsic dimensionality.

Our work is most closely related to Dynamic Continuous Indexing (DCI) (Li & Malik, 2016), which is an exact randomized algorithm for Euclidean space whose query time is linear in ambient dimensionality, sublinear in dataset size and sublinear in intrinsic dimensionality and uses space linear in the dataset size. Rather than partitioning the vector space, it uses multiple global one-dimensional indices, each of which orders data points along a certain random direction and combines these indices to find points that are near the query along multiple random directions. The proposed algorithm builds on the ideas introduced by DCI and achieves a significant improvement in the dependence on intrinsic dimensionality.

A summary of the query times of various prior algorithms and the proposed algorithm is presented in Table 1 and their growth as a function of intrinsic dimensionality is illustrated in Figure 1.

3. Prioritized DCI

DCI constructs a data structure consisting of multiple *composite indices* of data points, each of which in turn consists of a number of *simple indices*. Each simple index orders data points according to their projections along a particular random direction. Given a query, for every composite index, the algorithm finds points that are near the query in every constituent simple index, which are known as *candidate points*, and adds them to a set known as the *candidate set*. The true distances from the query to every candidate point are evaluated and the ones that are among the k clos-

est to the query are returned.

More concretely, each simple index is associated with a random direction and stores the projections of every data point along the direction. They are implemented using standard data structures that maintain one-dimensional ordered sequences of elements, like self-balancing binary search trees (Bayer, 1972; Guibas & Sedgewick, 1978) or skip lists (Pugh, 1990). At query time, the algorithm projects the query along the projection directions associated with each simple index and finds the position where the query would have been inserted in each simple index, which takes logarithmic time. It then iterates over, or *visits*, data points in each simple index in the order of their distances to the query under projection, which takes constant time for each iteration. As it iterates, it keeps track of how many times each data point has been visited across all simple indices of each composite index. If a data point has been visited in every constituent simple index, it is added to the candidate set and is said to have been *retrieved* from the composite index.

Algorithm 1 Data structure construction procedure

Require: A dataset D of n points $p^1, \ldots, p^n$, the number of simple indices m that constitute a composite index and the number of composite indices L
  function CONSTRUCT(D, m, L)
    $\{u_{jl}\}_{j\in[m],l\in[L]} \leftarrow mL$ random unit vectors in $\mathbb{R}^d$
    $\{T_{jl}\}_{j\in[m],l\in[L]} \leftarrow mL$ empty binary search trees or skip lists
    for $j = 1$ **to** m **do**
      for $l = 1$ **to** L **do**
        for $i = 1$ **to** n **do**
          $\overline{p}^i_{jl} \leftarrow \langle p^i, u_{jl}\rangle$
          Insert $(\overline{p}^i_{jl}, i)$ into T_{jl} with $\overline{p}^i_{jl}$ being the key and i being the value
        end for
      end for
    end for
    return $\{(T_{jl}, u_{jl})\}_{j\in[m],l\in[L]}$
  end function

DCI has a number of appealing properties compared to methods based on space partitioning. Because points are visited by rank rather than location in space, DCI performs well on datasets with large variations in data density. It naturally skips over sparse regions of the space and concentrates more on dense regions of the space. Since construction of the data structure does not depend on the dataset, the algorithm supports dynamic updates to the dataset, while being able to automatically adapt to changes in data density. Furthermore, because data points are represented in the indices as continuous values without being discretized, the granularity of discretization does not need to be chosen at construction time. Consequently, the same data structure can support queries at varying desired levels of accuracy, which allows a different speed-vs-accuracy trade-off to be made for each individual query.

Prioritized DCI differs from standard DCI in the order in which points from different simple indices are visited. In standard DCI, the algorithm cycles through all constituent simple indices of a composite index at regular intervals and visits exactly one point from each simple index in each pass. In Prioritized DCI, the algorithm assigns a priority to each constituent simple index; in each iteration, it visits the upcoming point from the simple index with the highest priority and updates the priority at the end of the iteration. The priority of a simple index is set to the negative absolute difference between the query projection and the next data point projection in the index.

Algorithm 2 k-nearest neighbour querying procedure

Require: Query point q in $\mathbb{R}^d$, binary search trees/skip lists and their associated projection vectors $\{(T_{jl}, u_{jl})\}_{j\in[m],l\in[L]}$, the number of points to retrieve k_0 and the number of points to visit k_1 in each composite index
  function QUERY($q, \{(T_{jl}, u_{jl})\}_{j,l}, k_0, k_1$)
    $C_l \leftarrow$ array of size n with entries initialized to 0 $\forall l \in [L]$
    $\overline{q}_{jl} \leftarrow \langle q, u_{jl}\rangle \; \forall j \in [m], l \in [L]$
    $S_l \leftarrow \emptyset \; \forall l \in [L]$
    $P_l \leftarrow$ empty priority queue $\forall l \in [L]$
    for $l = 1$ **to** L **do**
      for $j = 1$ **to** m **do**
        $(\overline{p}^{(1)}_{jl}, h^{(1)}_{jl}) \leftarrow$ the node in T_{jl} whose key is the closest to $\overline{q}_{jl}$
        Insert $(\overline{p}^{(1)}_{jl}, h^{(1)}_{jl})$ with priority $-|\overline{p}^{(1)}_{jl} - \overline{q}_{jl}|$ into P_l
      end for
    end for
    for $i' = 1$ **to** $k_1 - 1$ **do**
      for $l = 1$ **to** L **do**
        if $|S_l| < k_0$ **then**
          $(\overline{p}^{(i)}_{jl}, h^{(i)}_{jl}) \leftarrow$ the node with the highest priority in P_l
          Remove $(\overline{p}^{(i)}_{jl}, h^{(i)}_{jl})$ from P_l and insert the node in T_{jl} whose key is the next closest to $\overline{q}_{jl}$, which is denoted as $(\overline{p}^{(i+1)}_{jl}, h^{(i+1)}_{jl})$, with priority $-|\overline{p}^{(i+1)}_{jl} - \overline{q}_{jl}|$ into P_l
          $C_l[h^{(i)}_{jl}] \leftarrow C_l[h^{(i)}_{jl}] + 1$
          if $C_l[h^{(i)}_{jl}] = m$ **then**
            $S_l \leftarrow S_l \cup \{h^{(i)}_{jl}\}$
          end if
        end if
      end for
    end for
    return k points in $\bigcup_{l\in[L]} S_l$ that are the closest in Euclidean distance in $\mathbb{R}^d$ to q
  end function

Intuitively, this ensures data points are visited in the order of their distances to the query under projection. Because data points are only retrieved from a composite index when they have been visited in all constituent simple indices, data

Property	Complexity
Construction	$O(m(dn + n\log n))$
Query	$O\left(dk\max(\log(n/k), (n/k)^{1-m/d'}) + mk\log m\left(\max(\log(n/k), (n/k)^{1-1/d'})\right)\right)$
Insertion	$O(m(d + \log n))$
Deletion	$O(m\log n)$
Space	$O(mn)$

Table 2. Time and space complexities of Prioritized DCI.

points are retrieved in the order of the maximum of their distances to the query along multiple projection directions. Since distance under projection forms a lower bound on the true distance, the maximum projected distance approaches the true distance as the number of projection directions increases. Hence, in the limit as the number of simple indices approaches infinity, data points are retrieved in the ideal order, that is, the order of their true distances to the query.

The construction and querying procedures of Prioritized DCI are presented formally in Algorithms 1 and 2. To ensure the algorithm retrieves the exact k-nearest neighbours with high probability, the analysis in the next section shows that one should choose $k_0 \in \Omega(k\max(\log(n/k), (n/k)^{1-m/d'}))$ and $k_1 \in \Omega(mk\max(\log(n/k), (n/k)^{1-1/d'}))$, where d' denotes the intrinsic dimensionality. Though because this assumes worst-case configuration of data points, it may be overly conservative in practice; so, these parameters may be chosen by cross-validation.

We summarize the time and space complexities of Prioritized DCI in Table 2. Notably, the first term of the query complexity, which dominates when the ambient dimensionality d is large, has a more favourable dependence on the intrinsic dimensionality d' than the query complexity of standard DCI. In particular, a linear increase in the intrinsic dimensionality, which corresponds to an exponential increase in the expansion rate, can be mitigated by just a linear increase in the number of simple indices m. This suggests that Prioritized DCI can better handle datasets with high intrinsic dimensionality than standard DCI, which is confirmed by empirical evidence later in this paper.

4. Analysis

We analyze the time and space complexities of Prioritized DCI below and derive the stopping condition of the algorithm. Because the algorithm uses standard data structures, analysis of the construction time, insertion time, deletion time and space complexity is straightforward. Hence, this section focuses mostly on analyzing the query time.

In high-dimensional space, query time is dominated by the time spent on evaluating true distances between candidate points and the query. Therefore, we need to find the number of candidate points that must be retrieved to ensure the algorithm succeeds with high probability. To this end, we derive an upper bound on the failure probability for any given number of candidate points. The algorithm fails if sufficiently many distant points are retrieved from each composite index before some of the true k-nearest neighbours. We decompose this event into multiple (dependent) events, each of which is the event that a particular distant point is retrieved before some true k-nearest neighbours. Since points are retrieved in the order of their maximum projected distance, this event happens when the maximum projected distance of the distant point is less than that of a true k-nearest neighbour. We start by finding an upper bound on the probability of this event. To simplify notation, we initially consider displacement vectors from the query to each data point, and so relationships between projected distances of triplets of points translate relationships between projected lengths of pairs of displacement vectors.

We start by examining the event that a vector under random one-dimensional projection satisfies some geometric constraint. We then find an upper bound on the probability that some combinations of these events occur, which is related to the failure probability of the algorithm.

Lemma 1. *Let $v^l, v^s \in \mathbb{R}^d$ be such that $\left\|v^l\right\|_2 > \left\|v^s\right\|_2$, $\left\{u'_j\right\}_{j=1}^M$ be i.i.d. unit vectors in $\mathbb{R}^d$ drawn uniformly at random. Then $\Pr\left(\max_j\left\{\left|\langle v^l, u'_j\rangle\right|\right\} \le \left\|v^s\right\|_2\right) = \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v^l\right\|_2\right)\right)^M$.*

Proof. The event $\left\{\max_j\left\{\left|\langle v^l, u'_j\rangle\right|\right\} \le \|v^s\|_2\right\}$ is equivalent to the event that $\left\{\left|\langle v^l, u'_j\rangle\right| \le \|v^s\|_2 \ \forall j\right\}$, which is the intersection of the events $\left\{\left|\langle v^l, u'_j\rangle\right| \le \|v^s\|_2\right\}$. Because u'_j's are drawn independently, these events are independent.

Let θ_j be the angle between v^l and u'_j, so that $\langle v^l, u'_j\rangle = \left\|v^l\right\|_2\cos\theta_j$. Since u'_j is drawn uniformly, θ_j is uniformly distributed on $[0, 2\pi]$. Hence,

$$
\begin{aligned}
&\Pr\left(\max_j\left\{\left|\langle v^l, u'_j\rangle\right|\right\} \le \|v^s\|_2\right)\\
&= \prod_{j=1}^M \Pr\left(\left|\langle v^l, u'_j\rangle\right| \le \|v^s\|_2\right)\\
&= \prod_{j=1}^M \Pr\left(|\cos\theta_j| \le \frac{\|v^s\|_2}{\|v^l\|_2}\right)\\
&= \prod_{j=1}^M \left(2\Pr\left(\theta_j \in \left[\cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right), \pi - \cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right)\right]\right)\right)\\
&= \left(1 - \frac{2}{\pi}\cos^{-1}\left(\frac{\|v^s\|_2}{\|v^l\|_2}\right)\right)^M
\end{aligned}
$$

□

Lemma 2. *For any set of events $\{E_i\}_{i=1}^N$, the probability that at least k' of them happen is at most $\frac{1}{k'}\sum_{i=1}^N \Pr(E_i)$.*

Proof. For any set $T \subseteq [N]$, define $\tilde{E}_T$ to be the intersection of events indexed by T and complements of events not indexed by T, i.e. $\tilde{E}_T = \left(\bigcap_{i\in T} E_i\right) \cap \left(\bigcap_{i\notin T} \overline{E}_i\right)$. Observe that $\left\{\tilde{E}_T\right\}_{T\subseteq[N]}$ are disjoint and that for any $I \subseteq [N]$, $\bigcap_{i\in I} E_i = \bigcup_{T\supseteq I} \tilde{E}_T$. The event that at least k' of E_i's happen is $\bigcup_{I\subseteq[N]:|I|=k'} \bigcap_{i\in I} E_i$, which is equivalent to $\bigcup_{I\subseteq[N]:|I|=k'} \bigcup_{T\supseteq I} \tilde{E}_T = \bigcup_{T\subseteq[N]:|T|\geq k'} \tilde{E}_T$. We will henceforth use $\mathcal{T}$ to denote $\{T \subseteq [N] : |T| \geq k'\}$. Since $\mathcal{T}$ is a finite set, we can impose an ordering on its elements and denote the l^{th} element as T_l. The event can therefore be rewritten as $\bigcup_{l=1}^{|\mathcal{T}|} \tilde{E}_{T_l}$.

Define $E'_{i,j}$ to be $E_i \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right)$. We claim that $\sum_{i=1}^N \Pr\left(E'_{i,j}\right) \geq k' \sum_{l=1}^j \Pr\left(\tilde{E}_{T_l}\right)$ for all $j \in \{0, \ldots, |\mathcal{T}|\}$. We will show this by induction on j.

For $j = 0$, the claim is vacuously true because probabilities are non-negative. For $j > 0$, we observe that $E'_{i,j} = \left(E'_{i,j} \setminus \tilde{E}_{T_j}\right) \cup \left(E'_{i,j} \cap \tilde{E}_{T_j}\right) = E'_{i,j-1} \cup \left(E'_{i,j} \cap \tilde{E}_{T_j}\right)$ for all i. Since $E'_{i,j} \setminus \tilde{E}_{T_j}$ and $E'_{i,j} \cap \tilde{E}_{T_j}$ are disjoint, $\Pr\left(E'_{i,j}\right) = \Pr\left(E'_{i,j-1}\right) + \Pr\left(E'_{i,j} \cap \tilde{E}_{T_j}\right)$.

Consider the quantity $\sum_{i\in T_j} \Pr\left(E'_{i,j}\right)$, which is $\sum_{i\in T_j} \left(\Pr\left(E'_{i,j-1}\right) + \Pr\left(E'_{i,j} \cap \tilde{E}_{T_j}\right)\right)$ by the above observation. For each $i \in T_j, \tilde{E}_{T_j} \subseteq E_i$, and so $\tilde{E}_{T_j} \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right) \subseteq E_i \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right) = E'_{i,j}$. Because $\left\{\tilde{E}_{T_l}\right\}_{l=j}^{|\mathcal{T}|}$ are disjoint, $\tilde{E}_{T_j} \setminus \left(\bigcup_{l=j+1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right) = \tilde{E}_{T_j}$. Hence, $\tilde{E}_{T_j} \subseteq E'_{i,j}$ and so $E'_{i,j} \cap \tilde{E}_{T_j} = \tilde{E}_{T_j}$. Thus, $\sum_{i\in T_j} \Pr\left(E'_{i,j}\right) = |T_j| \Pr\left(\tilde{E}_{T_j}\right) + \sum_{i\in T_j} \Pr\left(E'_{i,j-1}\right)$.

It follows that $\sum_{i=1}^N \Pr\left(E'_{i,j}\right) = |T_j| \Pr\left(\tilde{E}_{T_j}\right) + \sum_{i\in T_j} \Pr\left(E'_{i,j-1}\right) + \sum_{i\notin T_j} \Pr\left(E'_{i,j}\right)$. Because $\Pr\left(E'_{i,j}\right) = \Pr\left(E'_{i,j-1}\right) + \Pr\left(E'_{i,j} \cap \tilde{E}_{T_j}\right) \geq \Pr\left(E'_{i,j-1}\right)$ and $|T_j| \geq k'$, $\sum_{i=1}^N \Pr\left(E'_{i,j}\right) \geq k' \Pr\left(\tilde{E}_{T_j}\right) + \sum_{i=1}^N \Pr\left(E'_{i,j-1}\right)$. By the inductive hypothesis, $\sum_{i=1}^N \Pr\left(E'_{i,j-1}\right) \geq k' \sum_{l=1}^{j-1} \Pr\left(\tilde{E}_{T_l}\right)$. Therefore, $\sum_{i=1}^N \Pr\left(E'_{i,j}\right) \geq k' \sum_{l=1}^j \Pr\left(\tilde{E}_{T_l}\right)$, which concludes the induction argument.

The lemma is a special case of this claim when $j = |\mathcal{T}|$, since $E'_{i,|\mathcal{T}|} = E_i$ and $\sum_{l=1}^{|\mathcal{T}|} \Pr\left(\tilde{E}_{T_l}\right) = \Pr\left(\bigcup_{l=1}^{|\mathcal{T}|} \tilde{E}_{T_l}\right)$. □

Combining the above yields the following theorem, the proof of which is found in the supplementary material.

Theorem 1. *Let $\left\{v_i^l\right\}_{i=1}^N$ and $\left\{v_{i'}^s\right\}_{i'=1}^{N'}$ be sets of vectors such that $\left\|v_i^l\right\|_2 > \left\|v_{i'}^s\right\|_2 \quad \forall i \in [N], i' \in [N']$. Furthermore, let $\left\{u'_{ij}\right\}_{i\in[N],j\in[M]}$ be random uniformly distributed unit vectors such that $u'_{i1}, \ldots, u'_{iM}$ are independent for any given i. Consider the events $\left\{\exists v_{i'}^s \text{ s.t. } \max_j \left\{\left|\langle v_i^l, u'_{ij}\rangle\right|\right\} \leq \left\|v_{i'}^s\right\|_2\right\}_{i=1}^N$. The probability that at least k' of these events occur is at most $\frac{1}{k'}\sum_{i=1}^N \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M$, where $\left\|v_{\max}^s\right\|_2 = \max_{i'}\left\{\left\|v_{i'}^s\right\|_2\right\}$. Furthermore, if $k' = N$, it is at most $\min_{i\in[N]}\left\{\left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M\right\}$.*

We now apply the results above to analyze specific properties of the algorithm. For convenience, instead of working directly with intrinsic dimensionality, we will analyze the query time in terms of a related quantity, global relative sparsity, as defined in (Li & Malik, 2016). We reproduce its definition below for completeness.

Definition 1. *Given a dataset $D \subseteq \mathbb{R}^d$, let $B_p(r)$ be the set of points in D that are within a ball of radius r around a point p. A dataset D has global relative sparsity of (τ, γ) if for all r and $p \in \mathbb{R}^d$ such that $|B_p(r)| \geq \tau$, $|B_p(\gamma r)| \leq 2|B_p(r)|$, where $\gamma \geq 1$.*

Global relative sparsity is related to the expansion rate (Karger & Ruhl, 2002) and intrinsic dimensionality in the following way: a dataset with global relative sparsity of (τ, γ) has $(\tau, 2^{(1/\log_2\gamma)})$-expansion and intrinsic dimensionality of $1/\log_2\gamma$.

Below we derive two upper bounds on the probability that some of the true k-nearest neighbours are missing from the set of candidate points retrieved from a given composite index, which are in expressed in terms of k_0 and k_1 respectively. These results inform us how k_0 and k_1 should be chosen to ensure the querying procedure returns the correct results with high probability. In the results that follow, we use $\{p^{(i)}\}_{i=1}^n$ to denote a re-ordering of the points $\{p^i\}_{i=1}^n$ so that $p^{(i)}$ is the i^{th} closest point to the query q. Proofs are found in the supplementary material.

Lemma 3. *Consider points in the order they are retrieved from a composite index that consists of m simple indices. The probability that there are at least n_0 points that are not the true k-nearest neighbours but are retrieved before some of them is at most $\frac{1}{n_0-k}\sum_{i=2k+1}^n \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)} - q\right\|_2 / \left\|p^{(i)} - q\right\|_2\right)\right)^m$.*

Lemma 4. *Consider point projections in a composite index that consists of m simple indices in the order they are visited. The probability that n_0 point projections that are not of the true k-nearest neighbours are visited before all true k-nearest neighbours have been retrieved is at most $\frac{m}{n_0-mk}\sum_{i=2k+1}^n \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)} - q\right\|_2 / \left\|p^{(i)} - q\right\|_2\right)\right)$.*

Lemma 5. *On a dataset with global relative sparsity (k, γ), the quantity $\sum_{i=2k+1}^{n} \left(1 - \frac{2}{\pi} \cos^{-1}\left(\left\|p^{(k)} - q\right\|_2 / \left\|p^{(i)} - q\right\|_2\right)\right)^m$ is at most $O\left(k \max(\log(n/k), (n/k)^{1-m \log_2 \gamma})\right)$.*

Lemma 6. *For a dataset with global relative sparsity (k, γ) and a given composite index consisting of m simple indices, there is some $k_0 \in \Omega(k \max(\log(n/k), (n/k)^{1-m \log_2 \gamma}))$ such that the probability that the candidate points retrieved from the composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha_0 < 1$.*

Lemma 7. *For a dataset with global relative sparsity (k, γ) and a given composite index consisting of m simple indices, there is some $k_1 \in \Omega(mk \max(\log(n/k), (n/k)^{1-\log_2 \gamma}))$ such that the probability that the candidate points retrieved from the composite index do not include some of the true k-nearest neighbours is at most some constant $\alpha_1 < 1$.*

Theorem 2. *For a dataset with global relative sparsity (k, γ), for any $\epsilon > 0$, there is some L, $k_0 \in \Omega(k \max(\log(n/k), (n/k)^{1-m \log_2 \gamma}))$ and $k_1 \in \Omega(mk \max(\log(n/k), (n/k)^{1-\log_2 \gamma}))$ such that the algorithm returns the correct set of k-nearest neighbours with probability of at least $1 - \epsilon$.*

Now that we have found a choice of k_0 and k_1 that suffices to ensure correctness with high probability, we can derive a bound on the query time that guarantees correctness. We then analyze the time complexity for construction, insertion and deletion and the space complexity. Proofs of the following are found in the supplementary material.

Theorem 3. *For a given number of simple indices m, the algorithm takes $O\left(dk \max(\log(n/k), (n/k)^{1-m/d'}) + mk \log m \left(\max(\log(n/k), (n/k)^{1-1/d'})\right)\right)$ time to retrieve the k-nearest neighbours at query time, where d' denotes the intrinsic dimensionality.*

Theorem 4. *For a given number of simple indices m, the algorithm takes $O(m(dn + n \log n))$ time to preprocess the data points in D at construction time.*

Theorem 5. *The algorithm requires $O(m(d + \log n))$ time to insert a new data point and $O(m \log n)$ time to delete a data point.*

Theorem 6. *The algorithm requires $O(mn)$ space in addition to the space used to store the data.*

5. Experiments

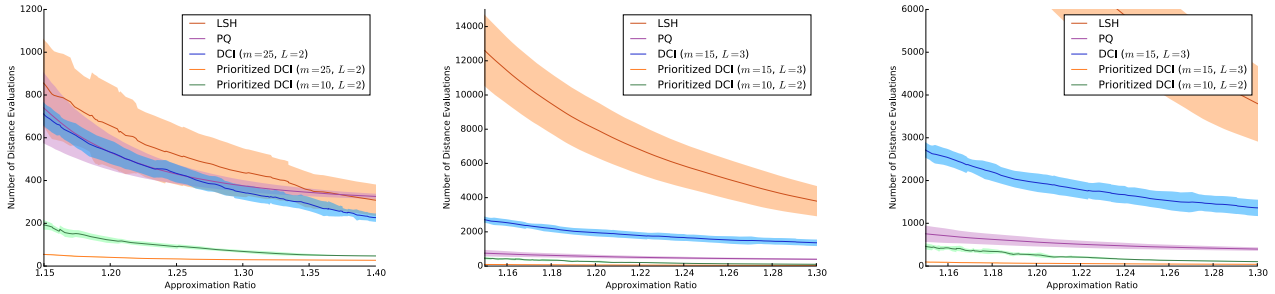
We compare the performance of Prioritized DCI to that of standard DCI (Li & Malik, 2016), product quantization (Jégou et al., 2011) and LSH (Datar et al., 2004), which is perhaps the algorithm that is most widely used in high-dimensional settings. Because LSH operates under the approximate setting, in which the performance metric of interest is how close the returned points are to the query rather than whether they are the true k-nearest neighbours. All algorithms are evaluated in terms of the time they would need to achieve varying levels of approximation quality.

Evaluation is performed on two datasets, CIFAR-100 (Krizhevsky & Hinton, 2009) and MNIST (LeCun et al., 1998). CIFAR-100 consists of $60,000$ colour images of 100 types of objects in natural scenes and MNIST consists of $70,000$ grayscale images of handwritten digits. The images in CIFAR-100 have a size of 32×32 and three colour channels, and the images in MNIST have a size of 28×28 and a single colour channel. We reshape each image into a vector whose entries represent pixel intensities at different locations and colour channels in the image. So, each vector has a dimensionality of $32 \times 32 \times 3 = 3072$ for CIFAR-100 and $28 \times 28 = 784$ for MNIST. Note that the dimensionalities under consideration are much higher than those typically used to evaluate prior methods.

For the purposes of nearest neighbour search, MNIST is a more challenging dataset than CIFAR-100. This is because images in MNIST are concentrated around a few modes; consequently, data points form dense clusters, leading to higher intrinsic dimensionality. On the other hand, images in CIFAR-100 are more diverse, and so data points are more dispersed in space. Intuitively, it is much harder to find the closest digit to a query among 6999 other digits of the same category that are all plausible near neighbours than to find the most similar natural image among a few other natural images with similar appearance. Later results show that all algorithms need fewer distance evaluations to achieve the same level of approximation quality on CIFAR-100 than on MNIST.

We evaluate performance of all algorithms using cross-validation, where we randomly choose ten different splits of query vs. data points. Each split consists of 100 points from the dataset that serve as queries, with the remainder designated as data points. We use each algorithm to retrieve the 25 nearest neighbours at varying levels of approximation quality and report mean performance and standard deviation over all splits.

Approximation quality is measured using the approximation ratio, which is defined to be the ratio of the radius of the ball containing the set of true k-nearest neighbours to the radius of the ball containing the set of approximate k-nearest neighbours returned by the algorithm. The closer the approximation ratio is to 1, the higher the approximation quality. In high dimensions, the time taken to compute true distances between the query and the candidate points dominates query time, so the number of distance evalua-



(a) (b) (c)

Figure 2. Comparison of the number of distance evaluations needed by different algorithms to achieve varying levels of approximation quality on (a) CIFAR-100 and (b,c) MNIST. Each curve represents the mean over ten folds and the shaded area represents ± 1 standard deviation. Lower values are better. (c) Close-up view of the figure in (b).

tions can be used as an implementation-independent proxy for the query time.

For LSH, we used 24 hashes per table and 100 tables, which we found to achieve the best approximation quality given the memory constraints. For product quantization, we used a data-independent codebook with 256 entries so that the algorithm supports dynamic updates. For standard DCI, we used the same hyparameter settings used in (Li & Malik, 2016) ($m = 25$ and $L = 2$ on CIFAR-100 and $m = 15$ and $L = 3$ on MNIST). For Prioritized DCI, we used two different settings: one that matches the hyperparameter settings of standard DCI, and another that uses less space ($m = 10$ and $L = 2$ on both CIFAR-100 and MNIST).

We plot the number of distance evaluations that each algorithm requires to achieve each desired level of approximation ratio in Figure 2. As shown, on CIFAR-100, under the same hyperparameter setting used by standard DCI, Prioritized DCI requires 87.2% to 92.5% fewer distance evaluations than standard DCI, 91.7% to 92.8% fewer distance evaluations than product quantization, and 90.9% to 93.8% fewer distance evaluations than LSH to achieve same levels approximation quality, which represents a 14-fold reduction in the number of distance evaluations relative to LSH on average. Under the more space-efficient hyperparameter setting, Prioritized DCI achieves a 6-fold reduction compared to LSH. On MNIST, under the same hyperparameter setting used by standard DCI, Prioritized DCI requires 96.4% to 97.0% fewer distance evaluations than standard DCI, 87.1% to 89.8% fewer distance evaluations than product quantization, and 98.8% to 99.3% fewer distance evaluations than LSH, which represents a 116-fold reduction relative to LSH on average. Under the more space-efficient hyperparameter setting, Prioritized DCI achieves a 32-fold reduction compared to LSH.

We compare the space efficiency of Prioritized DCI to that of standard DCI and LSH. As shown in Figure 3 in the supplementary material, compared to LSH, Prioritized DCI uses 95.5% less space on CIFAR-100 and 95.3% less space on MNIST under the same hyperparameter settings used by standard DCI. This represents a 22-fold reduction in memory consumption on CIFAR-100 and a 21-fold reduction on MNIST. Under the more space-efficient hyperparameter setting, Prioritized DCI uses 98.2% less space on CIFAR-100 and 97.9% less space on MNIST relative to LSH, which represents a 55-fold reduction on CIFAR-100 and a 48-fold reduction on MNIST.

In terms of wall-clock time, our implementation of Prioritized DCI takes 1.18 seconds to construct the data structure and execute 100 queries on MNIST, compared to 104.71 seconds taken by LSH.

6. Conclusion

In this paper, we presented a new exact randomized algorithm for k-nearest neighbour search, which we refer to as Prioritized DCI. We showed that Prioritized DCI achieves a significant improvement in terms of the dependence of query time complexity on intrinsic dimensionality compared to standard DCI. Specifically, Prioritized DCI can to a large extent counteract a *linear* increase in the intrinsic dimensionality, or equivalently, an *exponential* increase in the number of points near a query, using just a *linear* increase in the number of simple indices. Empirical results validated the effectiveness of Prioritized DCI in practice, demonstrating the advantages of Prioritized DCI over prior methods in terms of speed and memory usage.

Acknowledgements. This work was supported by DARPA W911NF-16-1-0552. Ke Li thanks the Natural Sciences and Engineering Research Council of Canada (NSERC) for fellowship support.

References

Anagnostopoulos, Evangelos, Emiris, Ioannis Z, and Psarros, Ioannis. Low-quality dimension reduction and high-dimensional approximate nearest neighbor. In *31st International Symposium on Computational Geometry (SoCG 2015)*, pp. 436–450, 2015.

Andoni, Alexandr and Indyk, Piotr. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pp. 459–468. IEEE, 2006.

Andoni, Alexandr and Razenshteyn, Ilya. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pp. 793–801. ACM, 2015.

Arya, Sunil and Mount, David M. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pp. 271–280, 1993.

Arya, Sunil, Mount, David M, Netanyahu, Nathan S, Silverman, Ruth, and Wu, Angela Y. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

Bayer, Rudolf. Symmetric binary b-trees: Data structure and maintenance algorithms. *Acta informatica*, 1(4): 290–306, 1972.

Behnam, Ehsan, Waterman, Michael S, and Smith, Andrew D. A geometric interpretation for local alignment-free sequence comparison. *Journal of Computational Biology*, 20(7):471–485, 2013.

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

Berchtold, Stefan, Keim, Daniel A., and peter Kriegel, Hans. The X-tree: An index structure for high-dimensional data. In *Very Large Data Bases*, pp. 28–39, 1996.

Berchtold, Stefan, Ertl, Bernhard, Keim, Daniel A, Kriegel, H-P, and Seidl, Thomas. Fast nearest neighbor search in high-dimensional space. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pp. 209–218. IEEE, 1998.

Beygelzimer, Alina, Kakade, Sham, and Langford, John. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 97–104. ACM, 2006.

Biau, Gérard, Chazal, Frédéric, Cohen-Steiner, David, Devroye, Luc, Rodriguez, Carlos, et al. A weighted k-nearest neighbor density estimate for geometric inference. *Electronic Journal of Statistics*, 5:204–237, 2011.

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

Dasgupta, Sanjoy and Freund, Yoav. Random projection trees and low dimensional manifolds. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, pp. 537–546. ACM, 2008.

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

Datar, Mayur, Immorlica, Nicole, Indyk, Piotr, and Mirrokni, Vahab S. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253–262. ACM, 2004.

Eldawy, Ahmed and Mokbel, Mohamed F. SpatialHadoop: A MapReduce framework for spatial data. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pp. 1352–1363. IEEE, 2015.

Guibas, Leo J and Sedgewick, Robert. A dichromatic framework for balanced trees. In *Foundations of Computer Science, 1978., 19th Annual Symposium on*, pp. 8–21. IEEE, 1978.

Guttman, Antonin. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pp. 47–57, 1984.

Houle, Michael E and Nett, Michael. Rank-based similarity search: Reducing the dimensional dependence. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(1):136–150, 2015.

Indyk, Piotr and Motwani, Rajeev. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pp. 604–613. ACM, 1998.

Jégou, Hervé, Douze, Matthijs, and Schmid, Cordelia. Product quantization for nearest neighbor search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(1):117–128, 2011.

Karger, David R and Ruhl, Matthias. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, pp. 741–750. ACM, 2002.

Krauthgamer, Robert and Lee, James R. Navigating nets: simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 798–807. Society for Industrial and Applied Mathematics, 2004.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 2009.

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

Li, Ke and Malik, Jitendra. Fast k-nearest neighbour search via Dynamic Continuous Indexing. In *International Conference on Machine Learning*, pp. 671–679, 2016.

Liu, Ting, Moore, Andrew W, Yang, Ke, and Gray, Alexander G. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems*, pp. 825–832, 2004.

Meiser, Stefan. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

Minsky, Marvin and Papert, Seymour. Perceptrons: an introduction to computational geometry. pp. 222, 1969.

Orchard, Michael T. A fast nearest-neighbor search algorithm. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pp. 2297–2300. IEEE, 1991.

Paulevé, Loïc, Jégou, Hervé, and Amsaleg, Laurent. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010.

Pugh, William. Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

Weiss, Yair, Torralba, Antonio, and Fergus, Rob. Spectral hashing. In *Advances in Neural Information Processing Systems*, pp. 1753–1760, 2009.