Fast k-Nearest Neighbour Search via Prioritized DCI

Ke Li **Jitendra Malik**
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720
United States
{ke.li,malik}@eecs.berkeley.edu

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. Consequently, 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

[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 by Karger & Ruhl (2002).

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

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.

Li & Malik (2016) proposed a method known as Dynamic Continuous Indexing (DCI), which successfully reduces the dependence on intrinsic dimensionality from exponential to sublinear, thereby making high-dimensional nearest neighbour search more practical. In this paper, we propose a variant of DCI, known as Prioritized DCI, which achieves 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). While their query times are logarithmic in the size of the dataset, they exhibit exponential dependence on the ambient dimensionality. 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 dividing hyperplanes. While randomization enables them to avoid exponential dependence on the ambient dimensionality, their query times still scale exponentially in the intrinsic dimensionality. Other algorithms (Karger & Ruhl, 2002; Krauthgamer & Lee, 2004; Beygelzimer et al., 2006; Houle & Nett, 2015) use local search or coarse-to-fine strategies. Unfortunately, the query times of these methods again scale exponentially in the intrinsic dimensionality. Figure 1 shows the query time complexities of various exact algorithms as a function of intrinsic dimensionality.

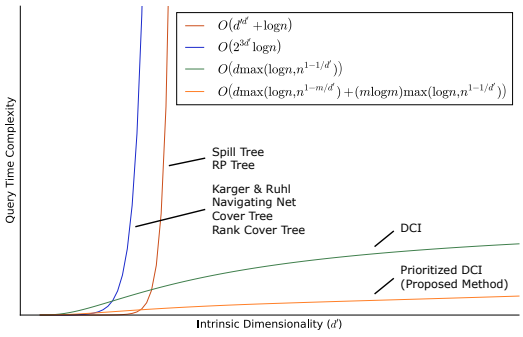


Figure 1: Visualization of the query time complexities of various exact algorithms as a function of the intrinsic dimensionality d'. Each curve represents an example from a class of similar query time complexities. Algorithms that fall into each particular class are shown next to the corresponding curve.

There has also 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. 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.

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 1: Time and space complexities of Prioritized DCI.

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.

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 closest 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. 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. 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.

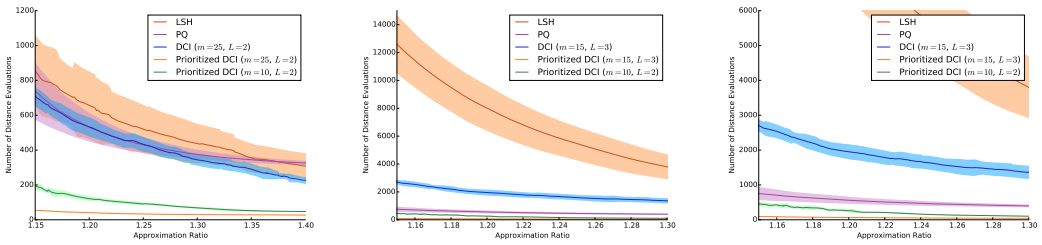
Prioritized DCI differs from standard DCI in the order in which points from different simple indices are visited. In standard DCI, the algorithm cycles through all constituent simple indices of a composite index at regular intervals and visits exactly one point from each simple index in each pass. In 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.

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 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 in the appendix. The time and space complexities of Prioritized DCI are summarized in Table 1. The analysis can be found in the appendix.

4 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



(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).

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.

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

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.

In terms of space efficiency, Prioritized DCI uses 95.5% less space on CIFAR-100 and 95.3% less space on MNIST compared to LSH. This represents a 22-fold reduction in memory consumption on CIFAR-100 and a 21-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.

5 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.

References

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.

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.

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.

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.

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.

Li, Ke and Malik, Jitendra. Fast k-nearest neighbour search via Dynamic Continuous Indexing. In *Proceedings of The 33rd 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.

6 Appendix I: Algorithm

Below we present the pseudocode of the proposed algorithm.

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

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($q, \{(T_{jl}, u_{jl})\}_{j,l}$)
  $C_l \leftarrow$ array of size n with entries initialized to 0 $\forall l \in [L]$
  $\overline{q}_{jl} \leftarrow \langle q, u_{jl}\rangle$ $\forall j \in [m], l \in [L]$
  $S \leftarrow \emptyset$
  $P_l \leftarrow$ 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 $-\left|\overline{p}^{(1)}_{jl} - \overline{q}_{jl}\right|$ 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

7 Appendix II: 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* $\Pr\left(\max_j \left\{\left|\langle v^l, u'_j\rangle\right|\right\} \leq \left\|v^s\right\|_2\right) = \left(1 - \frac{2}{\pi}\cos^{-1}\left(\left\|v^s\right\|_2 / \left\|v^l\right\|_2\right)\right)^M$.

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

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

□

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 $\left\|v_i^l\right\|_2 > \left\|v_{i'}^s\right\|_2 \ \forall i \in [N], i' \in [N']$. 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 $\min_{i \in [N]} \left\{\left(1 - \frac{2}{\pi} \cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M\right\}$.*

Proof. The event that $\exists v_{i'}^s$ s.t. $\max_j \left\{\left|\langle v_i^l, u'_{ij}\rangle\right|\right\} \leq \left\|v_{i'}^s\right\|_2$ is equivalent to the event that $\max_j \left\{\left|\langle v_i^l, u'_{ij}\rangle\right|\right\} \leq \max_{i'} \left\{\left\|v_{i'}^s\right\|_2\right\} = \left\|v_{\max}^s\right\|_2$. Take E_i to be the event that $\max_j \left\{\left|\langle v_i^l, u'_{ij}\rangle\right|\right\} \leq \left\|v_{\max}^s\right\|_2$. By Lemma 1, $\Pr(E_i) \leq \left(1 - \frac{2}{\pi} \cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M$. It follows from Lemma 2 that the probability that k' of E_i's occur is at most $\frac{1}{k'} \sum_{i=1}^{N} \Pr\left(E_i\right) \leq \frac{1}{k'} \sum_{i=1}^{N} \left(1 - \frac{2}{\pi} \cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M$. If $k' = N$, we use the fact that $\bigcap_{i'=1}^{N} E_{i'} \subseteq E_i \ \forall i$, which implies that $\Pr\left(\bigcap_{i'=1}^{N} E_{i'}\right) \leq \min_{i \in [N]} \Pr\left(E_i\right) \leq \min_{i \in [N]} \left\{\left(1 - \frac{2}{\pi} \cos^{-1}\left(\left\|v_{\max}^s\right\|_2 / \left\|v_i^l\right\|_2\right)\right)^M\right\}$. □

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 $\frac{1}{n_0-k}\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2 / \left\|p^{(i)}-q\right\|_2\right)\right)^m$.*

Proof. Points that are not the true k-nearest neighbours but are retrieved before some of them will be referred to as *extraneous points* and are divided into two categories: *reasonable* and *silly*. An extraneous point is reasonable if it is one of the $2k$-nearest neighbours, and is silly otherwise. For there to be n_0 extraneous points, there must be $n_0 - k$ silly extraneous points. Therefore, the probability that there are n_0 extraneous points is upper bounded by the probability that there are $n_0 - k$ silly extraneous points.

Since points are retrieved from the composite index in the order of increasing maximum projected distance to the query, for any pair of points p and p', if p is retrieved before p', then $\max_j\{|\langle p-q, u_{jl}\rangle|\} \leq \max_j\{|\langle p'-q, u_{jl}\rangle|\}$, where $\{u_{jl}\}_{j=1}^m$ are the projection directions associated with the constituent simple indices of the composite index.

By Theorem 1, if we take $\left\{v_i^l\right\}_{i=1}^N$ to be $\left\{p^{(i)}-q\right\}_{i=2k+1}^n$, $\left\{v_{i'}^s\right\}_{i'=1}^{N'}$ to be $\left\{p^{(i)}-q\right\}_{i=1}^k$, M to be m, $\left\{u_{ij}'\right\}_{j\in[M]}$ to be $\{u_{jl}\}_{j\in[m]}$ for all $i \in [N]$ and k' to be $n_0 - k$, we obtain an upper bound for the probability of there being a subset of $\left\{p^{(i)}\right\}_{i=2k+1}^n$ of size $n_0 - k$ such that for all points p in the subset, $\max_j\{|\langle p-q, u_{jl}\rangle|\} \leq \|p'-q\|_2$ for some $p' \in \left\{p^{(i)}-q\right\}_{i=1}^k$. In other words, this is the probability of there being $n_0 - k$ points that are not the $2k$-nearest neighbours whose maximum projected distances are no greater than the distance from some k-nearest neighbours to the query, which is at most $\frac{1}{n_0-k}\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\left\|p^{(k)}-q\right\|_2 / \left\|p^{(i)}-q\right\|_2\right)\right)^m$.

Since the event that $\max_j\{|\langle p-q, u_{jl}\rangle|\} \leq \max_j\{|\langle p'-q, u_{jl}\rangle|\}$ is contained in the event that $\max_j\{|\langle p-q, u_{jl}\rangle|\} \leq \|p'-q\|_2$ for any p, p', this is also an upper bound for the probability of there being $n_0 - k$ points that are not the $2k$-nearest neighbours whose maximum projected distances do not exceed those of some of the k-nearest neighbours, which by definition is the probability that there are $n_0 - k$ silly extraneous points. Since this probability is no less than the probability that there are n_0 extraneous points, the upper bound also applies to this probability. □

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

Proof. Projections of points that are not the true k-nearest neighbours but are visited before the k-nearest neighbours have all been retrieved will be referred to as *extraneous projections* and are divided into two categories: *reasonable* and *silly*. An extraneous projection is reasonable if it is of one of the $2k$-nearest neighbours, and is silly otherwise. For there to be n_0 extraneous projections, there must be $n_0 - mk$ silly extraneous projections, since there could be at most mk reasonable extraneous projections. Therefore, the probability that there are n_0 extraneous projections is upper bounded by the probability that there are $n_0 - mk$ silly extraneous projections.

Since point projections are visited in the order of increasing projected distance to the query, each extraneous silly projection must be closer to the query projection than the maximum projection of some k-nearest neighbour.

By Theorem 1, if we take $\left\{v_i^l\right\}_{i=1}^N$ to be $\left\{p^{(2k+\lfloor (i-1)/m \rfloor+1)}-q\right\}_{i=1}^{m(n-2k)}$, $\left\{v_{i'}^s\right\}_{i'=1}^{N'}$ to be $\left\{p^{(\lfloor (i-1)/m \rfloor+1)}-q\right\}_{i=1}^{mk}$, M to be 1, $\left\{u'_{i1}\right\}_{i=1}^N$ to be $\left\{u_{(i \mod m),l}\right\}_{i=1}^{m(n-2k)}$ and k' to be n_0-mk, we obtain an upper bound for the probability of there being n_0-mk point projections that are not of the $2k$-nearest neighbours whose distances to their respective query projections are no greater than the true distance between the query and some k-nearest neighbour, which is $\frac{1}{n_0-mk}\sum_{i=2k+1}^{n} m\left(1-\frac{2}{\pi}\cos^{-1}\left(\frac{\left\|p^{(k)}-q\right\|_2}{\left\|p^{(i)}-q\right\|_2}\right)\right)$.

Because maximum projected distances are no more than true distances, this is also an upper bound for the probability of there being n_0-mk silly extraneous projections. Since this probability is no less than the probability that there are n_0 extraneous projections, the upper bound also applies to this probability. □

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 $O\left(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma})\right)$.*

Proof. By definition of global relative sparsity, for all $i\geq 2k+1$, $\left\|p^{(i)}-q\right\|_2>\gamma\left\|p^{(k)}-q\right\|_2$. A recursive application shows that for all $i\geq 2^{i'}k+1$, $\left\|p^{(i)}-q\right\|_2>\gamma^{i'}\left\|p^{(k)}-q\right\|_2$.

Applying the fact that $1-(2/\pi)\cos^{-1}(x)\leq x\ \forall x\in[0,1]$ and the above observation yields:

$$\begin{aligned}
&\sum_{i=2k+1}^{n}\left(1-\frac{2}{\pi}\cos^{-1}\left(\frac{\left\|p^{(k)}-q\right\|_2}{\left\|p^{(i)}-q\right\|_2}\right)\right)^m\\
&\leq \sum_{i=2k+1}^{n}\left(\frac{\left\|p^{(k)}-q\right\|_2}{\left\|p^{(i)}-q\right\|_2}\right)^m\\
&< \sum_{i'=1}^{\lceil\log_2(n/k)\rceil-1}2^{i'}k\gamma^{-i'm}
\end{aligned}$$

If $\gamma\geq\sqrt[m]{2}$, this quantity is at most $k\log_2(n/k)$. On the other hand, if $1\leq\gamma<\sqrt[m]{2}$, this quantity can be simplified to:

$$\begin{aligned}
&k\left(\frac{2}{\gamma^m}\right)\left(\left(\frac{2}{\gamma^m}\right)^{\lceil\log_2(n/k)\rceil-1}-1\right)/\left(\frac{2}{\gamma^m}-1\right)\\
&=O\left(k\left(\frac{2}{\gamma^m}\right)^{\lceil\log_2(n/k)\rceil-1}\right)\\
&=O\left(k\left(\frac{n}{k}\right)^{1-m\log_2\gamma}\right)
\end{aligned}$$

Therefore, $\sum_{i=2k+1}^{n}\left(\left\|p^{(k)}-q\right\|_2/\left\|p^{(i)}-q\right\|_2\right)^m\leq O\left(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma})\right)$. □

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 $\alpha_0<1$.*

Proof. We will refer to the true k-nearest neighbours that are among first k_0 points retrieved from the composite index as *true positives* and those that are not as *false negatives*. Additionally, we will refer to points that are not true k-nearest neighbours but are among the first k_0 points retrieved as *false positives*.

When not all the true k-nearest neighbours are among the first k_0 candidate points, there must be at least one false negative and so there can be at most $k-1$ true positives. Consequently, there must be at least $k_0-(k-1)$ false positives. To find an upper bound on the probability of the existence of $k_0-(k-1)$ false positives in terms of global relative sparsity, we apply Lemma 3 with n_0 set to $k_0-(k-1)$, followed by Lemma 5. We conclude that this probability is at most $\frac{1}{k_0-2k+1}O\left(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma})\right)$. Because the event that not all the true k-nearest neighbours are among the first k_0 candidate points is contained in the event that there are $k_0-(k-1)$ false positives, the former is upper bounded by the same quantity. So, we can choose some $k_0\in\Omega(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$ to make it strictly less than 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 $\alpha_1<1$.*

Proof. We will refer to the projections of true k-nearest neighbours that are among first k_1 visited point projections as *true positives* and those that are not as *false negatives*. Additionally, we will refer to projections of points that are not of the true k-nearest neighbours but are among the first k_1 visited point projections as *false positives*.

When a k-nearest neighbour is not among the candidate points that have been retrieved, some of its projections must not be among the first k_1 visited point projections. So, there must be at least one false negative, implying that there can be at most $mk-1$ true positives. Consequently, there must be at least $k_1-(mk-1)$ false positives. To find an upper bound on the probability of the existence of $k_1-(mk-1)$ false positives in terms of global relative sparsity, we apply Lemma 4 with n_0 set to $k_1-(mk-1)$, followed by Lemma 5. We conclude that this probability is at most $\frac{m}{k_1-2mk+1}O\left(k\max(\log(n/k),(n/k)^{1-\log_2\gamma})\right)$. Because the event that some true k-nearest neighbour is missing from the candidate points is contained in the event that there are $k_1-(mk-1)$ false positives, the former is upper bounded by the same quantity. So, we can choose some $k_1\in\Omega(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ to make it strictly less than 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 $1-\epsilon$.*

Proof. For a given composite index, by Lemma 6, there is some $k_0\in\Omega(k\max(\log(n/k),(n/k)^{1-m\log_2\gamma}))$ such that the probability that some of the true k-nearest neighbours are missed is at most some constant $\alpha_0<1$. Likewise, by Lemma 7, there is some $k_1\in\Omega(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$ such that this probability is at most some constant $\alpha_1<1$. By choosing such k_0 and k_1, this probability is therefore at most $\min\{\alpha_0,\alpha_1\}<1$. For the algorithm to fail, all composite indices must miss some k-nearest neighbours. Since each composite index is constructed independently, the algorithm fails with probability of at most $(\min\{\alpha_0,\alpha_1\})^L$, and so must succeed with probability of at least $1-(\min\{\alpha_0,\alpha_1\})^L$. Since $\min\{\alpha_0,\alpha_1\}<1$, there is some L that makes $1-(\min\{\alpha_0,\alpha_1\})^L\geq 1-\epsilon$. □

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.*

Proof. Computing projections of the query point along all u_{jl}'s takes $O(dm)$ time, since L is a constant. Searching in the binary search trees/skip lists T_{jl}'s takes $O(m\log n)$ time. The total number of point projections that are visited is at most $\Theta(mk\max(\log(n/k),(n/k)^{1-\log_2\gamma}))$. Because determining the next point to visit requires popping and pushing a priority queue, which takes

$O(\log m)$ time, the total time spent on visiting points is $O(mk \log m \max(\log(n/k), (n/k)^{1-\log_2 \gamma}))$. The total number of candidate points retrieved is at most $\Theta(k \max(\log(n/k), (n/k)^{1-m \log_2 \gamma}))$. Because true distances are computed for every candidate point, the total time spent on distance computation is $O(dk \max(\log(n/k), (n/k)^{1-m \log_2 \gamma}))$. We can find the k closest points to the query among the candidate points using a selection algorithm like quickselect, which takes $O(k \max(\log(n/k), (n/k)^{1-m \log_2 \gamma}))$ time on average. Since the time for visiting points and for computing distances dominates, the entire algorithm takes $O(dk \max(\log(n/k), (n/k)^{1-m \log_2 \gamma}) + mk \log m \max(\log(n/k), (n/k)^{1-\log_2 \gamma}))$ time. Substituting $1/d'$ for $\log_2 \gamma$ yields the desired expression. □

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.*

Proof. Computing projections of all n points along all u_{jl}'s takes $O(dmn)$ time, since L is a constant. Inserting all n points into mL self-balancing binary search trees/skip lists takes $O(mn \log n)$ 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.*

Proof. In order to insert a data point, we need to compute its projection along all u_{jl}'s and insert it into each binary search tree or skip list. Computing the projections takes $O(md)$ time and inserting them into the corresponding self-balancing binary search trees or skip lists takes $O(m \log n)$ time. In order to delete a data point, we simply remove its projections from each of the binary search trees or skip lists, which takes $O(m \log n)$ time. □

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

Proof. The only additional information that needs to be stored are the mL binary search trees or skip lists. Since n entries are stored in each binary search tree/skip list, the total additional space required is $O(mn)$. □