Sub-linear Memory Sketches for Near Neighbor Search on Streaming Data

Benjamin Coleman [1] **Richard G Baraniuk** [1 2] **Anshumali Shrivastava** [1 2]

Abstract

We present the first sublinear memory sketch that can be queried to find the nearest neighbors in a dataset. Our online sketching algorithm compresses an N element dataset to a sketch of size $O(N^b \log^3 N)$ in $O(N^{(b+1)} \log^3 N)$ time, where $b < 1$. This sketch can correctly report the nearest neighbors of any query that satisfies a stability condition parameterized by b. We achieve sublinear memory performance on stable queries by combining recent advances in locality sensitive hash (LSH)-based estimators, online kernel density estimation, and compressed sensing. Our theoretical results shed new light on the memory-accuracy tradeoff for nearest neighbor search, and our sketch, which consists entirely of short integer arrays, has a variety of attractive features in practice. We evaluate the memory-recall tradeoff of our method on a friend recommendation task in the Google Plus social media network. We obtain orders of magnitude better compression than the random projection based alternative while retaining the ability to report the nearest neighbors of practical queries.

1. Introduction

Approximate near-neighbor search (ANNS) is a fundamental problem with widespread applications in databases, learning, computer vision, and much more (Gionis et al., 1999). Furthermore, ANNS is the first stage of several data processing and machine learning pipelines and is a popular baseline data analysis method. Informally, the problem is as follows. Given a dataset $\mathcal{D} = x_1,\ x_2, ...,\ x_N$, observed in a one pass sequence, build a data structure $\mathcal{S}$ that can efficiently identify a small number of data points $x_i \in \mathcal{D}$ that have high similarity to any dynamically generated query q.

In this paper, we focus on low-memory ANNS in settings where it is prohibitive to store the complete data in any form. Such restrictions naturally arise in extremely large databases, computer networks, and internet-of-things systems (Johnson et al., 2019). We want to compress the dataset $\mathcal{D}$ into a sketch $\mathcal{S}$ that is as small as possible while still retaining the ability to find near-neighbors for any query. Furthermore, the algorithm should be one pass as the second pass is prohibitive when we cannot store the full data in any form. It is common wisdom that the size of $\mathcal{S}$ must scale linearly ($\geq \Omega(N)$), even if we allow algorithms that only identify the locations of the nearest neighbors. In this work, we challenge that wisdom by constructing a sketch of size $O(N^b \log^3 N)$ bits in $O(N^{b+1} \log^3 N)$ time. Our sketch can identify near-neighbors for *stable* queries with high probability in $O(N^{b+1} \log^3 N)$ time. The value of b depends on the dataset, but b can be significantly less than 1 for many applications of practical importance. It should be noted that our sketch does not return the near neighbors themselves, since we do not store the vectors in any form. Instead, we output the identity or the index of the nearest neighbor, which is sufficient for most applications and does not fundamentally change the problem. Our sketch also does not attempt to correctly answer every possible near-neighbor query in sublinear memory, as this would violate information theoretic lower bounds. Instead, we provide a graceful tradeoff between the stability of a near neighbor search query and the memory required to obtain a correct answer.

1.1. Our Contribution

Our main contribution is a one-pass algorithm that produces a sketch $\mathcal{S}$ that solves the exact v-nearest neighbor problem in sub-linear memory with high probability. A formal problem statement is available in Section 2.3 and our theoretical results are formally stated in Section 4. Our algorithm requires $O(N^{b+1} \log^3 N)$ time to construct $\mathcal{S}$ and the same time to return the v nearest neighbors for a query. Here, b is a query-dependent value that describes the stability or difficulty of the query. Our guarantees are general and work for any query, but the sketch is only sub-linear when $b < 1$. In practice, one commits to a given b value and obtains the guarantees for all queries satisfying our condi-

[1] Department of Electrical and Computer Engineering, Rice University, Houston, Texas, USA [2] Department of Computer Science, Rice University, Houston, Texas, USA. Correspondence to: Benjamin Coleman <ben.coleman@rice.edu>, Anshumali Shrivastava <anshumali@rice.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

tions.

We obtain our sketch by merging compressed sensing techniques with recently-developed sketching algorithms. Surprisingly, we find that the hardness of a near-neighbor query is directly related to the notion of sparsity, or signal-to-noise ratio (SNR), in compressed sensing (Donoho, 2006; Tropp & Gilbert, 2007). This connection allows us to analyze geometric structure in the dataset using the very well-studied compressed sensing framework. The idea of exploiting structure to improve theoretical guarantees has recently gained traction because it can lead to stronger guarantees. For instance, the first improvements over the seminal near-neighbor search results of (Indyk & Motwani, 1998) were obtained using *data-dependent* hashing (Andoni et al., 2014). These methods use information about the data distribution to generate an optimal hash for a given dataset. In this work, we assume that the dataset has a set of general properties that are common in practice and we construct a data structure that exploits these properties. In general, the communication complexity of the near neighbor problem is $O(N)$. Our method requires sub-linear memory because our data assumptions limit the set of valid queries.

We support our theoretical findings with real experiments on large social-network datasets. Our theoretical techniques are sufficiently general to accommodate a variety of compressed sensing methods and KDE approximation algorithms. However, in practice we apply our theory using the Count-Min Sketch (CMS) as the compressed sensing method and the recently-proposed RACE sketch for KDE (Coleman & Shrivastava, 2020). Our RACE-CMS sketch inherits a variety of desirable practical properties from the RACE and CMS sketches that are used in its construction. When implemented this way, our near neighbor sketch consists entirely of a set of integer arrays. Furthermore, RACE sketches are linear, parallel and mergeable, allowing us to realize many practical gains using RACE-CMS. For instance, despite a query time complexity that is theoretically worse than linear search, RACE-CMS can be implemented in such a way that it is fast and practical to construct and query, processing thousands of vectors each second. As a result, we believe that our method will enable a variety of practical applications that need to perform near neighbor search in the distributed streaming setting with limited memory.

2. Applications

Here, we describe several applications for low-memory near neighbor sketches.

Graph Compression for Recommendation: In recommendation systems, we represent relationships, such as friendship or co-purchases, as graphs. Given N users, we represent each user as an N dimensional sparse vector, where non-zero entries correspond to edges or connections. To perform recommendations, we often wish to find pairs of users that are mutually connected to a similar set of other users. The process of identifying these users is a similarity search problem over the N dimensional sparse vector representation of the graph (Hsu et al., 2006). Online graphs can be very large, with billions of nodes and trillions of edges (Ching et al., 2015). Since graphs at this scale are prohibitively expensive to store and transmit, methods capable of compressing the network into a small and informative sketch could be invaluable for large-scale recommendations.

Robust Caching: The process of caching previously-seen data is a central component of many latency-critical applications including search engines, computer networks, web browsers and databases. While there are many well-established methods, such as Bloom filters, to detect exact matches, caching systems cannot currently report the distance between a query element and the contents of the cache. Our sketches can be used to implement caching mechanisms that are robust to minor perturbations in the query. Such a capability naturally provides better anomaly detection, robust estimation and retrieval. Since similar data structures can fit into the cache of modern processors (Luo & Shrivastava, 2018), our sketches could be an effective practical tool for online caching algorithms.

Distributed Data Streaming: In application domains such as the internet-of-things (IoT) and computer networks, we often with to build classifiers and other machine learning systems in the streaming setting (Ma et al., 2009). In practice, sketching is a critical component of distributed data collection pipelines. For instance, Apple uses a wide variety of sketches to enable mobile users to transmit valuable information that can be used to train machine learning models while minimizing the data transmission cost (Apple Differential Privacy Team, 2017). Similar challenges occur with distributed databases and IoT settings, where data generators can be scattered across a network of connected devices. Such applications require sketching methods to minimize the data communication cost while preserving utility for downstream learning applications. Since our sketches consist of integer arrays, they can easily be serialized and sent over a network.

2.1. Related Work

The problem of finding near-neighbors in *sub-linear time* is a very well-studied problem with several solutions (Indyk & Motwani, 1998). However, the *memory requirement* for near-neighbor search has only recently started receiving attention (Indyk & Wagner, 2018; 2017). Although hueristic methods for sample compression are employed in

Table 1. Summary of related work. Results are shown for a d-dimensional dataset of N points. Existing methods (Johnson & Lindenstrauss, 1984; Indyk & Wagner, 2018; Agarwal et al., 2005) can estimate distances to all points in the dataset with a $1 \pm \epsilon$ multiplicative error (full ϵ dependence not shown). Our method estimates the similarity with all points having a $\pm\epsilon$ additive error, where b depends on the properties of the dataset.

Method	Sketch Size (bits)	Sketch Time	Comments
No compression	$dN \log N$	N/A	-
Random projections (Johnson & Lindenstrauss, 1984)	$N \log^2 N$	$N \log N$	Widely used in practice
Compressed clustering tree (Indyk & Wagner, 2018)	$N \log N$	$Nd \log^{O(1)} N$	Multiple passes
Coresets (Agarwal et al., 2005)	$d\epsilon^{-(d-1)}$	$N + \epsilon^{-(d-1)}$	Multiple Passes
This work	$N^b \log^3 N$	$N^{b+1} \log^3 N$	$b < 1$ for stable queries

practice, the best theoretical result in this direction requires $O(N \log N)$ memory and therefore does not break the linear memory bound (Indyk & Wagner, 2018). Table 1 contains a summary of existing work in the area. To the best of our knowledge, the algorithm described in this paper is the first to perform near-neighbor search using asymptotically sub-linear memory.

Coresets or Clustering Based Approaches: A reasonable compression approach is to construct a coreset or represent the dataset as a set of clusters. For instance, the widely-used FAISS system compresses vectors using product quantization (Jegou et al., 2010). There are also sampling procedures to construct a subset P of $\mathcal{D}$ and guarantee the existence of a point $p \in P$ such that $d(p, x) < \epsilon$ for $\epsilon > 0$. The cluster-based approach from (Har-Peled & Kumar, 2014) uses similar ideas to reduce the space for v-nearest-neighbor by a constant factor of $\frac{1}{v}$. However, our procedure is superior in the following two regards. First, coresets and sample-based compression methods require parallel access to the entire dataset at once to determine which points to retain in the sketch. As an example, the sketch in (Har-Peled & Kumar, 2014) requires an offline clustering step. Therefore, it is impossible to stream queries to the sketch efficiently using existing methods. Second, cluster approximations of the data cannot solve the exact v-nearest neighbor problem because the sketching process removes points from the dataset. Despite the guarantees that can be obtained using ϵ coverings of the dataset, there may be any number of near-neighbors within ϵ of the query that have been discarded during sketching.

Perhaps most importantly, our method requires weaker assumptions about the dataset. Cluster-based methods assume that the dataset has a clustered structure that can be approximated by a small collection of centroids. To achieve high compression ratios, coreset methods require similar assumptions. However, our method is valid even when there is no efficient cluster representation. Our weak assumptions are particularly applicable to recent problems in recommendation systems, graph compression and neural embedding models. In this context, we are given a dataset where each embedding or object representation is close to a relatively small number of other elements. Furthermore, we expect most of our queries to be issued in regions that contain only a few elements from the dataset. Although there may be no large-scale hierchical clustering structure, our method can exploit the weaker structure in the dataset to provide good compression without the need for complex clustering and sample compression algorithms.

Finally, we note that our approach is much simpler to understand and analyze than existing methods. While clustering methods can achieve good performance, they usually require complex distance-approximation methods at query time. Sketch construction consists of computationally-intensive clustering steps or coreset sampling routines that have many moving parts. In contrast, our data structure is a simple array of integer counters with a fixed size. Therefore, we expect that our method will be attractive to practitioners and system designers.

2.2. Background

Our algorithm uses recent advances in locality-sensitive hashing (LSH)-based sketching with standard compressed sensing techniques. Before covering our method in detail and presenting theoretical results, we briefly review some useful results in sketching and compressed sensing.

2.3. Problem Statement

In this paper, we solve the exact v-nearest neighbor problem. The v-nearest neighbor problem is to identify all of the v closest points to a query with high probability. The difficulty of the v-nearest neighbor problem is data-dependent. To capture the difficulty of a query, we use the notion of near-neighbor stability from the seminal paper (Beyer et al., 1999).

Definition 1 *Exact v-nearest neighbor*
Given a set $\mathcal{D}$ of points in a d-dimensional space and a parameter v, construct a data structure which, given any

query point q, reports a set of v points in $\mathcal{D}$ with the following property: Each of the v nearest neighbors to q is in the set with probability $1-\delta$.

Definition 2 *Unstable near-neighbor search*
A nearest neighbor query is unstable for a given ϵ if the distance from the query point to most data points is $\leq (1+\epsilon)$ times the distance from the query point to its nearest neighbor.

2.4. Compressed Sensing and the Count Min Sketch

Compressed sensing is the area in signal processing that deals with the recovery of compressible signals from a sub-linear number of measurements. The task is to recover an N-length vector $\mathbf{x}$ from a vector $\mathbf{y}$ of M linear combinations, or measurements, of the N components of $\mathbf{x}$. The problem is tractable when $\mathbf{x}$ is v-sparse and has only v nonzero elements. For a more detailed description of the compressed sensing problem, see (Baraniuk, 2007). The fundamental result in compressed sensing is that we can exactly recover $\mathbf{x}$ from $\mathbf{y}$ using only $M = O(v \log N/v)$ measurements.

In the streaming literature, the v nonzero elements are often called *heavy hitters*. The Count-Min Sketch (CMS) is a classical data summary to identify heavy hitters in a data stream. The CMS is a $d \times w$ array of counts that are indexed and incremented in a randomized fashion. Given a vector $\mathbf{s}$, for every element s_i in $\mathbf{s}$, we apply d universal hash functions $h_1(\cdot), ...h_d(\cdot)$ to i to obtain a set of d indices. Then, we increment the CMS cells at these indices. When all elements of $\mathbf{s}$ are non-negative, we have a point-wise bound on the estimated s_i values returned by the CMS (Cormode & Muthukrishnan, 2005). For the sake of simplicity, we only consider the CMS when presenting our results. Finding heavy hitters is equivalent to compressed sensing (Indyk, 2013), and there are an enormous number of valid measurement matrices in the literature (Candes & Plan, 2011). Other compressed sensing methods can improve our bounds, but we defer this discussion the supplementary materials.

Theorem 1 *Given a CMS sketch of the non-negative vector $\mathbf{s} \in \mathbb{R}_+^N$ with $d = O\left(\log\left(\frac{N}{\delta}\right)\right)$ rows and $w = O\left(\frac{1}{\epsilon}\right)$ columns, we can recover a vector $\mathbf{s}^{\text{CMS}}$ such that we have the following point-wise recovery guarantee with probability $1-\delta$ for each recovered element s_i^{CMS}:*

$$s_i \leq s_i^{CMS} \leq s_i + \epsilon|\mathbf{s}|_1 \tag{1}$$

2.5. Locality-Sensitive Hashing

LSH (Indyk & Motwani, 1998) is a popular technique for efficient approximate nearest-neighbor search. An LSH family is a family of functions with the following property: Under the hash mapping, similar points have a high probability of having the same hash value. We say that a collision occurs whenever the hash values for two points are equal, i.e. $h(p) = h(q)$. The probability $\Pr_{\mathcal{H}}[h(p) = h(q)]$ is known as the collision probability of p and q. In this paper we will use the notation $p(p,q)$ to denote the collision probability of p and q. For our arguments, we will assume a slightly stronger notion of LSH than the one given by (Indyk & Motwani, 1998). We will suppose that the collision probability is a monotonic function of the similarity between p and q. That is

$$p(p,q) \propto f(\text{sim}(p,q)) \tag{2}$$

where $\text{sim}(p,q)$ is a similarity function and $f(\cdot)$ is monotone increasing. LSH is a very well-studied topic with a number of well-known LSH families in the literature (Gionis et al., 1999). Most LSH families satisfy this assumption.

2.6. Repeated Array-of-Counts Estimator (RACE)

Recent work has shown that LSH can be used for efficient unbiased statistical estimation (Spring & Shrivastava, 2017; Charikar & Siminelakis, 2017; Luo & Shrivastava, 2018). The RACE algorithm (Coleman & Shrivastava, 2020) replaces the universal hash function in the CMS with an LSH function. The result is a sketch that approximates the kernel density estimate (KDE) of a query. Here, we re-state the main theorem from (Luo & Shrivastava, 2018) using simpler notation.

Theorem 2 *ACE Estimator (Luo & Shrivastava, 2018)*
Given a dataset $\mathcal{D}$, an LSH function $l(\cdot) \mapsto [1,R]$ and a parameter K, construct an LSH function $h(\cdot) \mapsto [1,R^K]$ by concatenating K independent $l(\cdot)$ hashes. Let $A \in \mathbb{R}^{R^K}$ be an array of $O(R^K \log N)$ bits where the i^{th} component is

$$A[i] = \sum_{x \in \mathcal{D}} 1_{\{h(x)=i\}}$$

Then for any query q,

$$\mathbb{E}[A[h(q)]] = \sum_{x \in \mathcal{D}} p(x,q)^K$$

We will heavily leverage the observation that $A[h(q)]$ is an unbiased estimator of the summation of collision probabilities. This sum is a kernel density estimate over the dataset (Coleman & Shrivastava, 2020), where the kernel is defined by the LSH function.

Algorithm 1 One-Pass Online Sketching Algorithm

```
Require: $\mathcal{D}$
Ensure: $d \times w$ RACE arrays indexed as $A_{i,j,o}$
  Initialize: $k \times d \times w \times R$ independent LSH family (denoted by $L(\cdot)$) and $d$ independent 2-universal hash functions $h_i(\cdot)$, $i \in [1-d]$, each taking values in range $[1-w]$.
  while not at end of data $\mathcal{D}$ do
    read current $x_j$;
    for $o$ in 1 to $r$ do
      for $i$ in 1 to $d$ do
        $A_{i,h_i(j),o}[L[x_j]]$++;
      end for
    end for
  end while
```

Algorithm 2 Querying Algorithm

```
Require: Sketch from Algorithm 1, query $q$
Ensure: Identities of Top-$v$ neighbors of $q$
  We already have: $k \times d \times w \times R$ independent LSH family (denoted by $L(\cdot)$) and $d$ independent 2-universal hash functions $h_i(\cdot)$, $i \in [1-d]$, each taking values in range $[1-w]$ from Algorithm 1.
  for $i$ in 1 to $d$ do
    for $j$ in 1 to $w$ do
      $\text{CMS}_{(i,j)} = \text{MoM}(A_{i,j,o}[L(q)])$
    end for
  end for
  for $j$ in 1 to $n$ do
    $\mathbf{s}_j = p(q, x_j)^K = \min_i \ \text{CMS}_{(i,h_i(j))}$
  end for
  Report top-$v$ indices of $\mathbf{s}$ as the neighbors.
```

3. Intuition

We propose Algorithm 1 as an online near-neighbor sketching method and Algorithm 2 to query the sketch. The intuition behind our algorithm is as follows. Consider the naive method to perform near-neighbor search. We begin by finding the pairwise distances between the query and each point in the dataset. Given a query q, this procedure results in a vector of N distances, where the i^{th} position in the vector contains the distance $d(x_i, q)$. If j is the index of the smallest element in the vector, then x_j is the nearest neighbor to the query. Now suppose that we are given a vector $\mathbf{s}$ of N kernel evaluations rather than explicit distances. Here, the i^{th} component of $\mathbf{s}$ is $s_i = k(x_i, q)$, where $k(\cdot,\cdot)$ is a radial kernel. Radial kernels are nearly 1 when $d(x_i, q)$ is small and decrease to 0 as $d(x_i, q)$ increases. Since $k(x_i, q)$ is a monotone decreasing function with respect to $d(x_i, q)$, the vector of kernel values is also sufficient to perform near neighbor search. If s_j is the largest component of $\mathbf{s}$, then x_j is the nearest neighbor to the query. The main idea of our algorithm is to apply compressed sensing techniques to $\mathbf{s}$.

The main result from compressed sensing is that a sparse vector $\mathbf{s}$ can be recovered from a sub-linear memory sketch of its components. If we assume that $\mathbf{s}$ is v-sparse (contains only v elements that are large), then we can recover $\mathbf{s}$ from $O(v \log N/v)$ random linear combinations of the entries of $\mathbf{s}$. The key insight is that each measurement is a weighted kernel density estimate (KDE) over the dataset. Using a small collection of KDE sums, we can identify the near neighbors of the query. If we choose the coefficients to be $\{1, 0\}$, then each measurement is an unweighted KDE over a partition of the dataset. While it requires N memory to compute the exact KDE, recent results (Coleman & Shrivastava, 2020) show that the KDE may be approximated by an online sketch in space that is constant with respect to N. While larger sketches improve the quality of the approximation, the memory does not grow when elements are added to the dataset. Thus, each of the $O(v \log N/v)$ measurements can be approximated using constant memory in the streaming setting.

4. Theory

Due to space constraints, we omit proofs and corner cases. For a thorough presentation that includes proofs, see the supplementary material.

4.1. Estimation of Compressed Sensing Measurements

To bound the error of the approximation for our compressed sensing measurements, we bound the variance of the RACE estimator using standard inequalities.

Theorem 3 *Given a dataset $\mathcal{D}$, K independent LSH functions $l(\cdot)$ and any choice of constants $r_i \in \mathbb{R}$, RACE can estimate a linear combination of $\mathbf{s}_i(q) = p(x_i, q)^K$ with the following variance bound.*

$$\mathbb{E}[A[L(q)]] = \sum_{x_i \in \mathcal{D}} r_i p(x_i, q)^K \tag{3}$$

$$\text{var}(A[l(q)]) \le |\tilde{\mathbf{s}}(q)|_1^2 \tag{4}$$

where $L(\cdot)$ is formed by concatenating the K copies of $l(\cdot)$ and $\tilde{\mathbf{s}}_i(q) = \sqrt{\mathbf{s}_i(q)}$.

Let $\mathbf{y} \in \mathbb{R}^M$ be the M compressed sensing measurements of the KDE vector $\mathbf{s}(q)$. A direct corollary of Theorem 3 is that by setting the coefficients correctly, we can obtain unbiased estimators of each measurement with bounded variance. Using the median-of-means (MoM) technique, we can obtain an arbitrarily close estimate of each compressed sensing measurement. To ensure that all M measurements obey this bound with probability $1-\delta$, we also apply the probability union bound. Note that the multiplicative M factor comes from the fact that we are using ACE to estimate M different measurements.

Theorem 4 *Given any* $\epsilon > 0$ *and* $O\left(M\frac{|\tilde{\mathbf{s}}(q)|_1^2}{\epsilon^2}\log\left(\frac{M}{\delta}\right)\right)$ *independent ACE repetitions, for any query q, we have the following bound for each of the M measurements with probability* $1-\delta$

$$y_i(q) - \epsilon \leq \hat{y}_i(q) \leq y_i(q) + \epsilon \tag{5}$$

Therefore, by repeating ACE estimators (RACE), we can obtain low-variance estimates of the compressed sensing measurements of $\mathbf{s}(q)$. The exact number of measurements M depends on both Φ and the dataset, but $M < O(N)$.

4.2. Query-Dependent Sparsity Conditions

For our compressed sensing measurements to be useful, $\mathbf{s}(q)$ needs to be *sparse* with a bound on $|\mathbf{s}(q)|_1$(Donoho, 2006). We also require a bound on $|\tilde{\mathbf{s}}(q)|_1$ to avoid a memory blow-up in Theorem 4. If we simply assume a bound on $|\tilde{\mathbf{s}}(q)|_1$, it is straightforward to show that the sketch requires sub-linear memory. See the supplementary materials for details. To characterize the type of queries that are appropriate for our algorithm, we connect sparsity with the idea of near-neighbor stability (Beyer et al., 1999), a well-established notion of query difficulty.

Given any vector $\mathbf{s}(q)$ with elements between 0 and 1, we can tune K to make $\mathbf{s}(q)$ sparse and obtain the required bounds. However, increasing K also increases the memory because we require increasingly more precise estimates to differentiate between $\mathbf{s}_v$ and $\mathbf{s}_{v+1}$. Therefore, we want K to be just large enough. The largest value of K is required when all points in the dataset other than the v nearest neighbors are equidistant to the query ($|\mathbf{s}|_1 = O(N)$). To choose K appropriately, we begin by defining two data-dependent values Δ and B to characterize this situation. Suppose that x_v and x_{v+1} are the v^{th} and $(v+1)^{\text{th}}$ nearest neighbors, respectively. Let Δ be defined as $\Delta = \frac{p(x_{v+1},q)}{p(x_v,q)}$ and $B = \sum_{i=v+1}^{N}\frac{\tilde{s}_i}{\tilde{s}_{v+1}}$. Δ measures the stability (Definition 2) of the query and is a measure of the gap between the near-neighbors and the rest of the dataset. If $\Delta \approx 1$, then x_v and x_{v+1} are very difficult to separate and the query is *unstable*. B measures the sparsity of $\mathbf{s}$. If B is $O(N)$, then every element of $\mathbf{s}$ is nonzero (Figure 1). We are now ready to present our results for K in terms of B and Δ.

Theorem 5 *Given a query q and query-dependent parameters B and* Δ*, if* $K = \left\lceil 2\frac{\log B}{\log\frac{1}{\Delta}}\right\rceil$ *then* $p(x_v,q)^K \geq \sum_{i=v+1}^{N} p(x_i,q)^K$ *and we have the bounds* $|\mathbf{s}(q)|_1 \leq v+1$ *and* $|\tilde{\mathbf{s}}(q)|_1 \leq v+1$

In practice, this assumption is unrealistically pessimistic because $\mathbf{s}(q)$ is often sufficiently sparse without any intervention using K. However, Theorem 5 always allows us to choose K so that $|\mathbf{s}(q)|_1$ is bounded by a constant.

4.3. Reduce Near-Neighbor to Compressed Recovery

We can apply Theorem 4 to estimate each of the M CMS measurements, which we call $\widehat{\text{CMS}}$. We want to recover an estimate $\hat{\mathbf{s}}$ of $\mathbf{s}$ from our approximate compressed sensing measurements $\widehat{\text{CMS}}$. Since the error ϵ_E in our approximation simply adds to the CMS recovery error ϵ_C from Theorem 1, we can recover the values of $\mathbf{s}(q)$ by choosing appropriate values for ϵ_C and ϵ_E.

Theorem 6 *We require*

$$O\left(\frac{|\tilde{\mathbf{s}}(q)|_1^2|\mathbf{s}(q)|_1}{\epsilon^3}\log\left(\frac{|\mathbf{s}(q)|_1}{\epsilon\delta}\log\left(\frac{N}{\delta}\right)\right)\log\left(\frac{N}{\delta}\right)\right)$$

ACE estimates to recover $\hat{\mathbf{s}}(q)$ *with probability* $1-\delta$ *such that*

$$s_i(q) - \frac{\epsilon}{2} \leq \hat{s_i}(q) \leq s_i(q) + \frac{\epsilon}{2} \tag{6}$$

If $\mathbf{s}$ is sparse, then this result can be used to identify the top v elements of $\mathbf{s}$ by setting $\epsilon = s_v - s_{v+1} = p_v^K - p_{v+1}^K$. These elements correspond to the largest kernel evaluations and therefore the nearest neighbors. For the equidistant case, we substitute the value of K from Theorem 5 into the expression in Theorem 6 to obtain our final results. Our main theorem is a simplified result that relates the size of the RACE sketch with the query-dependent parameters Δ and p_v. The full derivation, including the dependence on δ, is available in the supplementary materials.

Theorem 7 *It is possible to construct a sketch that solves the exact v-nearest neighbor problem with probability* $1-\delta$ *using* $O\left(N^b\log^3(N)\right)$ *bits, where*

$$b = \frac{6|\log p_v| + 2\log r}{\log\frac{1}{\Delta}}$$

Here, r is the range of the LSH function, and p_v *is the collision probability of the* v^{th} *nearest neighbor with the query.*

5. Experiments

In this section, we rigorously evaluate our RACE-CMS sketch on friend recommendation tasks on social network graphs, similar to the ones described in (Sharma et al., 2017). Our goal is to compare and contrast the practical compression-accuracy tradeoff of RACE with streaming baselines. We use the Google Plus social network dataset, obtained from (Leskovec & Mcauley, 2012), and the Twitter and Slashdot graphs from (Leskovec & Krevl, 2014) to

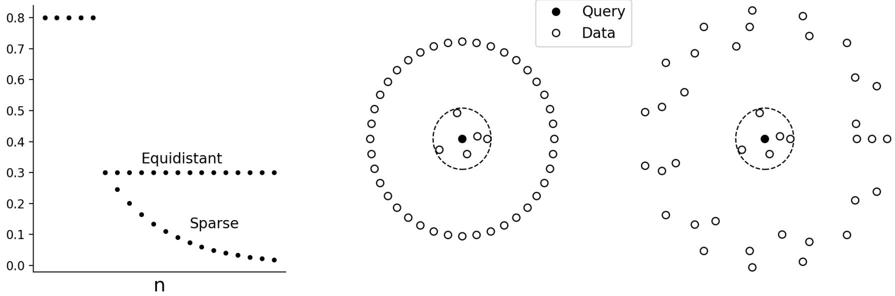

Figure 1. Geometric interpretation of B and Δ. Δ characterizes the gap between the v nearest neighbors, while B characterizes whether $\mathbf{s}$ is sparse. The worst-case situation occurs when all points are equidistant to the query (center). However, if $\mathbf{s}$ is already sparse, then far fewer points in the dataset are near the query (right).

Table 2. Dataset Statistics

Dataset	Nodes	Nonzeros	Mean Edges	Mean Similarity
Google+	108k	13.6M	127	0.002
Twitter	81.4k	1.8M	22	2.2e-4
Slashdot	82.2k	1.1M	13	1.4e-5

evaluate our algorithm. Google Plus is a directed graph of 107,614 Google Plus users, where each element in the dataset is an adjacency list of connections to other users. The uncompressed dataset size is 121 MB when stored in a sparse format as the smallest possible unsigned integral type. The other datasets are structured the same way, with similar sizes. Additional statistics are displayed in Table 2. These characteristics are typical for large scale graphs, where the data is high dimensional and sparse. Note that the low mean similarity between elements indirectly implies that $\mathbf{s}(q)$ is sparse.

5.1. Implementation

We use the RACE-CMS sketch that was presented in Section 4. However, we slightly deviate from the algorithm described in Algorithm 1 in our implementation by rehashing the K LSH hash values to a range r using a universal hash function. Our algorithm is characterized by the hyperparameters K, d, w, R and r and by the hash functions $l(\cdot)$ and $h(\cdot)$. Here, $l(\cdot)$ is MinHash, an LSH function for the Jaccard distance. We use MurmurHash for $h(\cdot)$, the universal hash function in the CMS. For all experiments, we vary K, d, w and R to trade off memory for performance. We present the operating points on the Pareto frontier for all algorithms. Typical values of d are between 2 and 5, w between 100 and 1000, and R between 2 and 8. We varied the range r between 100 and 1000 and used $K \in \{1, 2\}$.

We implemented RACE-CMS in C++ with the following considerations. First, we do not store the RACE counters as full 32-bit integers. The count values tend to be small because the CMS only assigns each data point to d cells out of dw total cells, and each each cell further divides the counts into the RACE arrays. In our evaluation, we used 16-bit short integers, although more aggressive memory optimizations are likely possible. For example, we found that all counts were less than 32 in our Google Plus experiments, suggesting that 8-bit integer arrays are sufficient. The second optimization comes from our observation that many count values are zero. By storing the RACE sketches as sparse arrays or maps, we do not have to store the zero counts. We present results for the situation where we store dense arrays of counts (Array-RACE) and where we store RACE as a sparse array (Map-RACE). An implementation diagram is shown in Figure 2.

5.2. Baselines

We compare our method with dimensionality reduction and random sampling followed by exact near-neighbor search. We reduce the size of the dataset until a given compression ratio is achieved and then find the nearest neighbors with the Euclidean distance. We compare against all methods that can operate in the strict one-pass streaming environment (Fiat, 1998), which is required in many high-speed applications. We considered a comparison with product quantization using FAISS (Johnson et al., 2019) but we encountered issues due to the dimensionality ($> 100k$) of our graph data, which agrees with previous evaluations of FAISS on high-dimensional data (Wang et al., 2018). Samples are represented using 32-bit integer node IDs and are stored in sparse format, since the graph vectors tend to have many zeros. Projections are stored as dense arrays of single-precision (32-bit) floating point numbers.

Random Projections: We use sparse random projections (Achlioptas, 2003) and the Johnson-Lindenstrauss lemma to reduce the dimensionality of the dataset. This is the best known streaming method that is also practical.

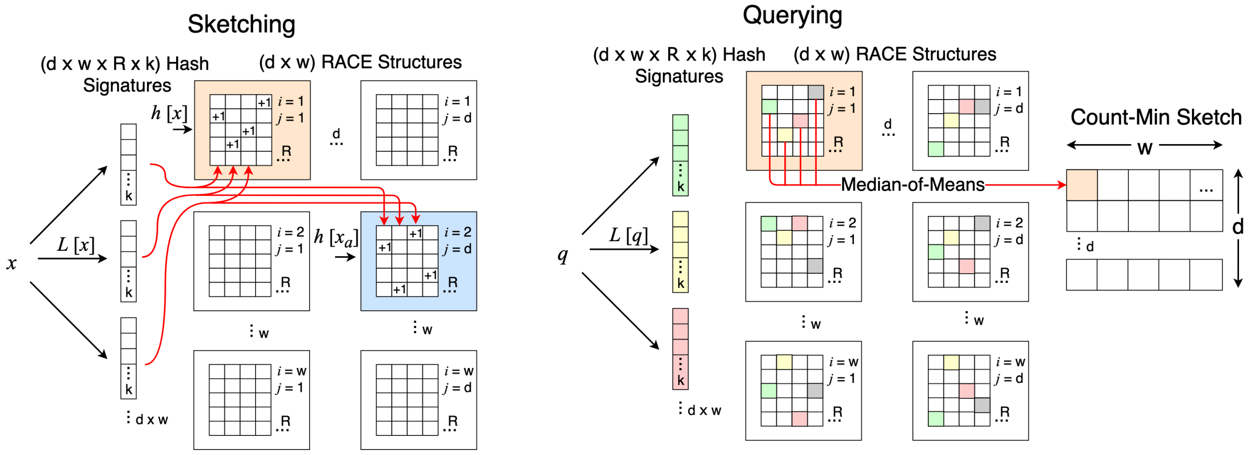

Figure 2. Implementation of sketching (Algorithm 1) and querying (Algorithm 2) using RACE data structures. During sketching, we compute $d \times w \times R \times k$ hash values for each $x \in \mathcal{D}$ and update the RACEs selected using $h(\cdot)$. During querying, we compute the hash values of the query q and estimate the CMS measurements.

Random Sampling: With random sampling, we reduce the original dataset to the desired size by selecting a random subset of elements of the dataset. Given a query, we perform exact nearest neighbor search on the random samples.

5.3. Experimental Setup

We computed the ground truth Jaccard similarities and nearest neighbors for each vector in the dataset. We are primarily interested in queries for which high similarity neighbors exist in the dataset due to the constraints of the friend recommendation problem. This is also consistent with the near-neighbor problem statements in Section 2.3, which assume the existence of a near-neighbor. We return the 20 nearest neighbors and report the recall of points with similarity greater than 0.8 and 0.9 to the query. To confirm that the sketch is not simply memorizing our queries, we remove the query from the dataset before creating the sketch.

For random projections, we performed a sweep of the number of random projections from 5 to 500. Random sampling was performed by decimating the dataset (without replacement) so that the sampled dataset had the desired size.

5.4. Results

Figure 3 shows the mean recall of ground-truth neighbors for the RACE-CMS sketch. Array-RACE and Map-RACE are both implementations of our method, but with a different underlying data structure used to represent the RACE sketch. We obtained good recall (> 0.85) on the set of queries with high-quality neighbors ($\text{sim}(x, q) > 0.9$) even for an extreme 20x compression ratio on Google Plus and 5x compression ratios on the other datasets. Since many entries in the array are zero, we find that Map-RACE outperforms Array-RACE by a sizeable margin.

It is evident that RACE performs best for high similarity search. This is due to increased sparsity of $\mathbf{s}(q)$ (any two random users are unlikely to share a friend and hence have similarity zero) and higher $p(x_v, q)$. In the recommender system setting, we usually wish to recommend nodes with very high similarity. If we require the algorithm to recover neighbors on the Google Plus graph with similarity measure greater than 0.9 with an expected recall of 80% or higher, our algorithm requires only 5% of the space of the original dataset (6 MB) while random projections require 60 MB (50%) and random sampling requires nearly the entire dataset. For neighbors with lower similarity (0.8), our method requires roughly one quarter of the memory needed by random projections.

6. Conclusion

We have presented RACE-CMS, the first sub-linear memory algorithm for near-neighbor search. Our analysis connects the stability of a near-neighbor search problem with the memory required to provide an accurate solution. Additionally, our core idea of using LSH to estimate compressed sensing measurements creates a sketch that can encode structural information and can process data not seen during the sketching process.

We supported our theoretical findings with experimental results. In practical test settings, RACE-CMS outperformed existing methods for low-memory near-neighbor search by a factor of 10. We expect that RACE-CMS will enable large-scale similarity search for a variety of applications and will find utility in situations where memory and communication are limiting factors.

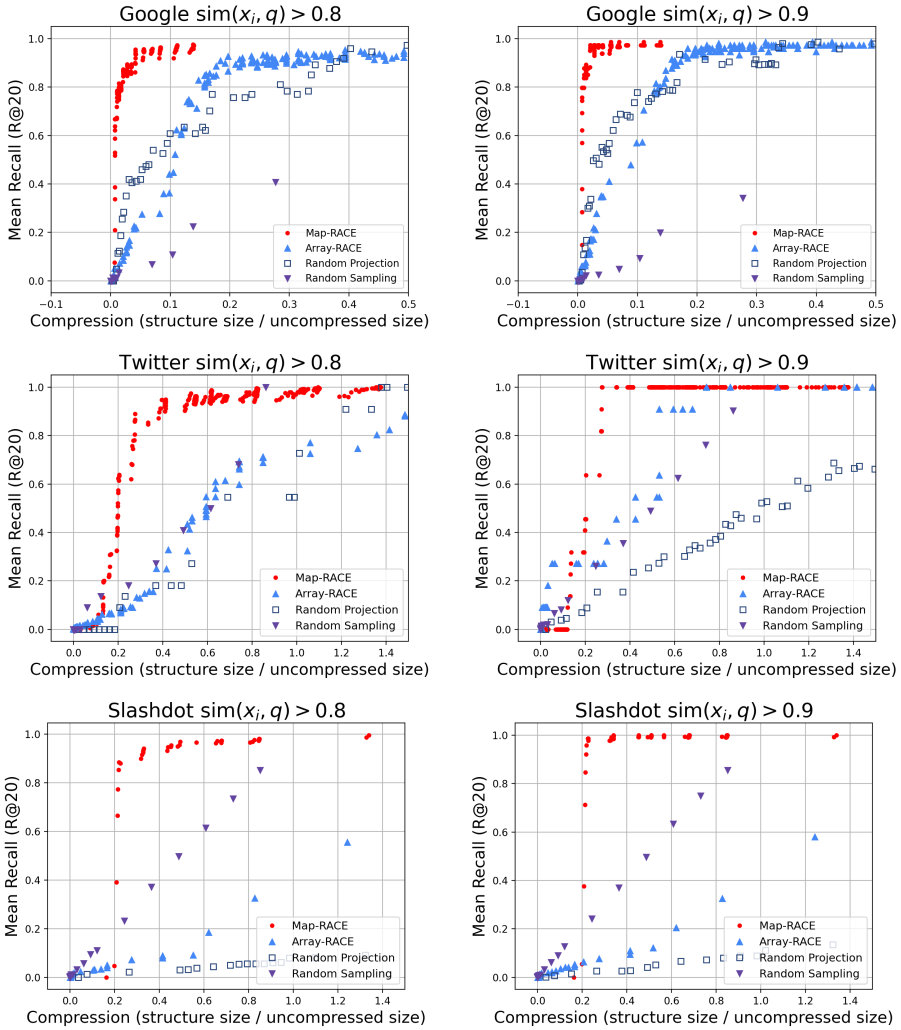

Figure 3. Average recall vs compressed dataset size. The dataset size is expressed as the inverse compression ratio, or the ratio of the compressed size to the uncompressed size. Recall is reported as the average recall of neighbors with Jaccard similarity $\mathrm{sim}(x, q) \geq 0.8$ (left) and 0.9 (right) over the set of queries. Higher is better. We report the recall of nodes with similarity greater than or equal to 0.8 and 0.9 for the top 20 search results of the query. Results are averaged over > 500 queries.

Acknowledgements

This work was supported by National Science Foundation IIS-1652131, BIGDATA-1838177, RI-1718478, AFOSR-YIP FA9550-18-1-0152, Amazon Research Award, and the ONR BRC grant on Randomized Numerical Linear Algebra.

References

Achlioptas, D. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

Agarwal, P. K., Har-Peled, S., and Varadarajan, K. R. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

Andoni, A., Indyk, P., Nguyen, H. L., and Razenshteyn, I. Beyond locality-sensitive hashing. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1018–1028. Society for Industrial and Applied Mathematics, 2014.

Apple Differential Privacy Team. Learning with privacy at scale. *Apple Machine Learning Journal*, 1(8), 2017.

Baraniuk, R. G. Compressive sensing. *IEEE signal processing magazine*, 24(4), 2007.

Beyer, K., Goldstein, J., Ramakrishnan, R., and Shaft, U. When is "nearest neighbor" meaningful? In *Inter-*

national conference on database theory, pp. 217–235. Springer, 1999.

Candes, E. J. and Plan, Y. A probabilistic and ripless theory of compressed sensing. *IEEE transactions on information theory*, 57(11):7235–7254, 2011.

Charikar, M. and Siminelakis, P. Hashing-based-estimators for kernel density in high dimensions. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pp. 1032–1043. IEEE, 2017.

Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., and Muthukrishnan, S. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment*, 8(12):1804–1815, 2015.

Coleman, B. and Shrivastava, A. Sub-linear race sketches for approximate kernel density estimation on streaming data. In *Proceedings of the 2020 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 2020.

Cormode, G. and Muthukrishnan, S. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

Donoho, D. L. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

Fiat, A. Online algorithms: The state of the art (lecture notes in computer science). 1998.

Gionis, A., Indyk, P., Motwani, R., et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pp. 518–529, 1999.

Har-Peled, S. and Kumar, N. Down the rabbit hole: Robust proximity search and density estimation in sublinear space. *SIAM Journal on Computing*, 43(4):1486–1511, 2014.

Hsu, W. H., King, A. L., Paradesi, M. S., Pydimarri, T., and Weninger, T. Collaborative and structural recommendation of friends using weblog-based social network analysis. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, volume 6, pp. 55–60, 2006.

Indyk, P. Sketching via hashing: from heavy hitters to compressed sensing to sparse fourier transform. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pp. 87–90. ACM, 2013.

Indyk, P. and Motwani, R. 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.

Indyk, P. and Wagner, T. Near-optimal (euclidean) metric compression. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 710–723. SIAM, 2017.

Indyk, P. and Wagner, T. Approximate nearest neighbors in limited space. *arXiv preprint arXiv:1807.00112*, 2018.

Jegou, H., Douze, M., and Schmid, C. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Johnson, J., Douze, M., and Jégou, H. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

Johnson, W. B. and Lindenstrauss, J. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Leskovec, J. and Krevl, A. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

Leskovec, J. and Mcauley, J. J. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pp. 539–547, 2012.

Luo, C. and Shrivastava, A. Arrays of (locality-sensitive) count estimators (ace): Anomaly detection on the edge. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pp. 1439–1448. International World Wide Web Conferences Steering Committee, 2018.

Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 681–688. ACM, 2009.

Sharma, A., Seshadhri, C., and Goel, A. When hashes met wedges: A distributed algorithm for finding high similarity vectors. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 431–440. International World Wide Web Conferences Steering Committee, 2017.

Spring, R. and Shrivastava, A. A new unbiased and efficient class of lsh-based samplers and estimators for partition function computation in log-linear models. *arXiv preprint arXiv:1703.05160*, 2017.

Tropp, J. A. and Gilbert, A. C. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on information theory*, 53(12):4655–4666, 2007.

Wang, Y., Shrivastava, A., Wang, J., and Ryu, J. Randomized algorithms accelerated over cpu-gpu for ultra-high dimensional similarity search. In *Proceedings of the 2018 International Conference on Management of Data*, pp. 889–903, 2018.