Practical Data-Dependent Metric Compression with Provable Guarantees[*]

Piotr Indyk[†]
MIT
indyk@mit.edu

Ilya Razenshteyn[†‡]
Columbia University
ilyaraz@mit.edu

Tal Wagner[†]
MIT
talw@mit.edu

November 7, 2017

Abstract

We introduce a new distance-preserving compact representation of multi-dimensional point-sets. Given n points in a d-dimensional space where each coordinate is represented using B bits (i.e., dB bits per point), it produces a representation of size $O(d\log(dB/\epsilon)+\log n)$ bits per point from which one can approximate the distances up to a factor of $1\pm\epsilon$. Our algorithm almost matches the recent bound of [IW17] while being much simpler. We compare our algorithm to Product Quantization (PQ) [JDS11], a state of the art heuristic metric compression method. We evaluate both algorithms on several data sets: SIFT (used in [JDS11]), MNIST [LC98], New York City taxi time series [GMRS16] and a synthetic one-dimensional data set embedded in a high-dimensional space. With appropriately tuned parameters, our algorithm produces representations that are comparable to or better than those produced by PQ, while having provable guarantees on its performance.

1 Introduction

Compact distance-preserving representations of high-dimensional objects are very useful tools in data analysis and machine learning. They compress each data point in a data set using a small number of bits while preserving the distances between the points up to a controllable accuracy. This makes it possible to run data analysis algorithms, such as similarity search, machine learning classifiers, etc, on data sets of reduced size. The benefits of this approach include: (a) reduced running time (b) reduced storage and (c) reduced communication cost (between machines, between CPU and RAM, between CPU and GPU, etc). These three factors make the computation more efficient overall, especially on modern architectures where the communication cost is often the dominant factor in the running time, so fitting the data in a single processing unit is highly beneficial. Because of these benefits, various compact representations have been extensively studied over the last decade, for applications such as: speeding up similarity search [Bro97, IM98, KOR00, TFW08, WTF09, JDS11, NFS12, SL14], scalable learning algorithms [WDL$^+$09, LSMK11], streaming algorithms [M$^+$05] and other tasks. For example, a recent paper [JDJ17] describes a similarity search software package based on one such method (Product Quantization (PQ)) that has been used to solve very large similarity search problems over billions of point on GPUs at Facebook.

The methods for designing such representations can be classified into *data-dependent* and *data-oblivious*. The former analyze the whole data set in order to construct the point-set representation, while the latter apply

[*]Code available at https://github.com/talwagner/quadsketch.
[†]Authors ordered alphabetically.
[‡]Supported by Simons Foundation Junior Fellowship.

a fixed procedure individually to each data point. A classic example of the data-oblivious approach is based on randomized dimensionality reduction [JL84], which states that any set of n points in the Euclidean space of arbitrary dimension D can be mapped into a space of dimension $d = O(\epsilon^{-2} \log n)$, such that the distances between all pairs of points are preserved up to a factor of $1 \pm \epsilon$. This allows representing each point using $d(B + \log D)$ bits, where B is the number of bits of precision in the coordinates of the original pointset. [1] More efficient representations are possible if the goal is to preserve only the distances in a certain range. In particular, $O(\epsilon^{-2} \log n)$ *bits* are sufficient to distinguish between distances smaller than 1 and greater than $1 + \epsilon$, independently of the precision parameter [KOR00] (see also [RL09] for kernel generalizations). Even more efficient methods are known if the coordinates are binary [Bro97, LSMK11, SL14].

Data-dependent methods compute the bit representations of points "holistically", typically by solving a global optimization problem. Examples of this approach include Semantic Hashing [SH09], Spectral Hashing [WTF09] or Product Quantization [JDS11] (see also the survey [WLKC16]). Although successful, most of the results in this line of research are empirical in nature, and we are not aware of any worst-case accuracy vs. compression tradeoff bounds for those methods along the lines of the aforementioned data oblivious approaches.

A recent work [IW17] shows that it is possible to combine the two approaches and obtain algorithms that adapt to the data while providing worst-case accuracy/compression tradeoffs. In particular, the latter paper shows how to construct representations of d-dimensional pointsets that preserve all distances up to a factor of $1 \pm \epsilon$ while using only $O((d + \log n) \log(1/\epsilon) + \log(Bn))$ bits per point. Their algorithm uses hierarchical clustering in order to group close points together, and represents each point by a displacement vector from a near by point that has already been stored. The displacement vector is then appropriately rounded to reduce the representation size. Although theoretically interesting, that algorithm is rather complex and (to the best of our knowledge) has not been implemented.

Our results. The main contribution of this paper is QuadSketch (QS), a *simple* data-adaptive algorithm, which is both provable and practical. It represents each point using $O(d \log(dB/\epsilon) + \log n)$ bits, where (as before) we can set $d = O(\epsilon^{-2} \log n)$ using the Johnson-Lindenstrauss lemma. Our bound significantly improves over the "vanilla" $O(dB)$ bound (obtained by storing all d coordinates to full precision), and comes close to bound of [IW17]. At the same time, the algorithm is quite simple and intuitive: it computes a d-dimensional quadtree[2] and appropriately prunes its edges and nodes.[3]

We evaluate QuadSketch experimentally on both real and synthetic data sets: a SIFT feature data set from [JDS11], MNIST [LC98], time series data reflecting taxi ridership in New York City [GMRS16] and a synthetic data set (Diagonal) containing random points from a one-dimensional subspace (i.e., a line) embedded in a high-dimensional space. The data sets are quite diverse: SIFT and MNIST data sets are de-facto "standard" test cases for nearest neighbor search and distance preserving sketches, NYC taxi data was designed to contain anomalies and "irrelevant" dimensions, while Diagonal has extremely low intrinsic dimension. We compare our algorithms to Product Quantization (PQ) [JDS11], a state of the art method for computing distance-preserving sketches, as well as a baseline simple uniform quantization method (Grid). The sketch length/accuracy tradeoffs for QS and PQ are comparable on SIFT and MNIST data, with PQ having higher accuracy for shorter sketches while QS having better accuracy for longer sketches. On NYC taxi data, the accuracy of QS is higher over the whole range of sketch lengths . Finally, Diagonal exemplifies a situation where the low dimensionality of the data set hinders the performance of PQ, while QS naturally adapts to this data set. Overall, QS performs well on "typical" data sets, while its provable guarantees ensure

[1]The bounds can be stated more generally in terms of the *aspect ratio* Φ of the point-set. See Section 2 for the discussion.

[2]Traditionally, the term "quadtree" is used for the case of $d = 2$, while its higher-dimensional variants are called " hyperoctrees" [YS83]. However, for the sake of simplicity, in this paper we use the same term "quadtree" for any value of d.

[3]We note that a similar idea (using kd-trees instead of quadtrees) has been earlier proposed in [AZ14]. However, we are not aware of any provable space/distortion tradeoffs for the latter algorithm.

robust performance in a wide range of scenarios. Both algorithms improve over the baseline quantization method.

2 Formal Statement of Results

Preliminaries. Let $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ be a pointset in Euclidean space. A compression scheme constructs from X a bit representation referred to as a *sketch*. Given the sketch, and without access to the original pointset, one can *decompress* the sketch into an approximate pointset $\tilde{X} = \{\tilde{x}_1, \ldots, \tilde{x}_n\} \subset \mathbb{R}^d$. The goal is to minimize the size of the sketch, while approximately preserving the geometric properties of the pointset, in particular the distances and near neighbors.

In the previous section we parameterized the sketch size in terms of the number of points n, the dimension d, and the bits per coordinate B. In fact, our results are more general, and can be stated in terms of the *aspect ratio* of the pointset, denoted by Φ and defined as the ratio between the largest to smallest distance,

$$\Phi = \frac{\max_{1 \le i < j \le n} \|x_i - x_j\|}{\min_{1 \le i < j \le n} \|x_i - x_j\|}.$$

Note that $\log(\Phi) \le \log d + B$, so our bounds, stated in terms of $\log \Phi$, immediately imply analogous bounds in terms of B.

We will use $[n]$ to denote $\{1, \ldots, n\}$, and $\tilde{O}(f)$ to suppress polylogarithmic factors in f.

QuadSketch. Our compression algorithm, described in detail in Section 3, is based on a randomized variant of a quadtree followed by a pruning step. In its simplest variant, the trade-off between the sketch size and compression quality is governed by a single parameter Λ. Specifically, Λ controls the pruning step, in which the algorithm identifies "non-important" bits among those stored in the quadtree (i.e. bits whose omission would have little effect on the approximation quality), and removes them from the sketch. Higher values of Λ result in sketches that are longer but have better approximation quality.

Approximate nearest neighbors. Our main theorem provides the following guarantees for the basic variant of QuadSketch: for each point, the distances from that point to all other points are preserved up to a factor of $1 \pm \epsilon$ with a constant probability.

Theorem 1. *Given $\epsilon, \delta > 0$, let $\Lambda = O(\log(d \log \Phi / \epsilon\delta))$ and $L = \log \Phi + \Lambda$. QuadSketch runs in time $\tilde{O}(ndL)$ and produces a sketch of size $O(nd\Lambda + n \log n)$ bits, with the following guarantee: For every $i \in [n]$,*

$$\Pr\left[\forall_{j \in [n]} \|\tilde{x}_i - \tilde{x}_j\| = (1 \pm \epsilon)\|x_i - x_j\|\right] \ge 1 - \delta.$$

In particular, with probability $1 - \delta$, if $\tilde{x}_{i^*}$ is the nearest neighbor of $\tilde{x}_i$ in $\tilde{X}$, then x_{i^*} is a $(1+\epsilon)$-approximate nearest neighbor of x_i in X.

Note that the theorem allows us to compress the input point-set into a sketch and then decompress it back into a point-set which can be fed to a black box similarity search algorithm. Alternatively, one can decompress only specific points and approximate the distance between them.

For example, if $d = O(\epsilon^{-2} \log n)$ and Φ is polynomially bounded in n, then Theorem 1 uses $\Lambda = O(\log \log n + \log(1/\epsilon))$ bits per coordinate to preserve $(1 + \epsilon)$-approximate nearest neighbors.

The full version of QuadSketch, described in Section 3, allows extra fine-tuning by exposing additional parameters of the algorithm. The guarantees for the full version are summarized by Theorem 3 in Section 3.

Table 1: Comparison of Euclidean metric sketches with maximum distortion $1 \pm \epsilon$, for $d = O(\epsilon^{-2} \log n)$ and $\log \Phi = O(\log n)$.

REFERENCE	BITS PER POINT	CONSTRUCTION TIME
"Vanilla" bound	$O(\epsilon^{-2} \log^2 n)$	–
Algorithm of [IW17]	$O(\epsilon^{-2} \log n \ \log(1/\epsilon))$	$\tilde{O}(n^{1+\alpha} + \epsilon^{-2} n)$ for $\alpha \in (0, 1]$
Theorem 2	$O(\epsilon^{-2} \log n \ (\log \log n \ + \log(1/\epsilon)))$	$\tilde{O}(\epsilon^{-2} n)$

Maximum distortion. We also show that a recursive application of QuadSketch makes it possible to approximately preserve the distances between *all* pairs of points. This is the setting considered in [IW17]. (In contrast, Theorem 1 preserves the distances from any single point.)

Theorem 2. *Given $\epsilon > 0$, let $\Lambda = O(\log(d \log \Phi/\epsilon))$ and $L = \log \Phi + \Lambda$. There is a randomized algorithm that runs in time $\tilde{O}(ndL)$ and produces a sketch of size $O(nd\Lambda + n \log n)$ bits, such that with high probability, every distance $\|x_i - x_j\|$ can be recovered from the sketch up to distortion $1 \pm \epsilon$.*

Theorem 2 has smaller sketch size than that provided by the "vanilla" bound, and only slightly larger than that in [IW17]. For example, for $d = O(\epsilon^{-2} \log n)$ and $\Phi = \text{poly}(n)$, it improves over the "vanilla" bound by a factor of $O(\log n / \log \log n)$ and is lossier than the bound of [IW17] by a factor of $O(\log \log n)$. However, compared to the latter, our construction time is nearly linear in n. The comparison is summarized in Table 1.

We remark that Theorem 2 does not let us recover an approximate embedding of the pointset, $\tilde{x}_1, \ldots, \tilde{x}_n$, as Theorem 1 does. Instead, the sketch functions as an oracle that accepts queries of the form (i, j) and return an approximation for the distance $\|x_i - x_j\|$.

3 The Compression Scheme

The sketching algorithm takes as input the pointset X, and two parameters L and Λ that control the amount of compression.

Step 1: Randomly shifted grid. The algorithm starts by imposing a randomly shifted axis-parallel grid on the points. We first enclose the whole pointset in an axis-parallel hypercube H. Let $\Delta' = \max_{i \in [n]} \|x_1 - x_i\|$, and $\Delta = 2^{\lceil \log \Delta' \rceil}$. Set up H to be centered at x_1 with side length 4Δ. Now choose $\sigma_1, \ldots, \sigma_d \in [-\Delta, \Delta]$ independently and uniformly at random, and shift H in each coordinate j by σ_j. By the choice of side length 4Δ, one can see that H after the shift still contains the whole pointset. For every integer ℓ such that $-\infty < \ell \leq \log(4\Delta)$, let G_ℓ denote the axis-parallel grid with cell side 2^ℓ which is aligned with H.

Note that this step can be often eliminated in practice without affecting the empirical performance of the algorithm, but it is necessary in order to achieve guarantees for *arbitrary* pointsets.

Step 2: Quadtree construction. The 2^d-ary quadtree on the nested grids G_ℓ is naturally defined by associating every grid cell c in G_ℓ with the tree node at level ℓ, such that its children are the 2^d grid cells in $G_{\ell-1}$ which are contained in c. The edge connecting a node v to a child v' is labeled with a bitstring of length d defined as follows: the j^{th} bit is 0 if v' coincides with the bottom half of v along coordinate j, and 1 if v' coincides with the upper half along that coordinate.

In order to construct the tree, we start with H as the root, and bucket the points contained in it into the 2^d children cells. We only add child nodes for cells that contain at least one point of X. Then we continue by recursing on the child nodes. The quadtree construction is finished after L levels. We denote the resulting edge-labeled tree by T^*. A construction for $L = 2$ is illustrated in Figure 1.

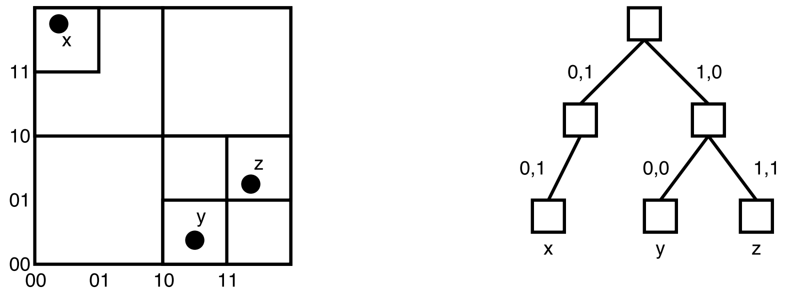

Figure 1: Quadtree construction for points x, y, z. The x and y coordinates are written as binary numbers.

We define the *level* of a tree node with side length 2^ℓ to be ℓ (note that ℓ can be negative). The *degree* of a node in T^* is its number of children. Since all leaves are located at the bottom level, each point $x_i \in X$ is contained in exactly one leaf, which we henceforth denote by v_i.

Step 3: Pruning. Consider a downward path $u_0, u_1, \dots, u_k$ in T^*, such that $u_1, \dots, u_{k-1}$ are nodes with degree 1, and u_0, u_k are nodes with degree other than 1 (u_k may be a leaf). For every such path in T^*, if $k > \Lambda + 1$, we remove the nodes $u_{\Lambda+1}, \dots, u_{k-1}$ from T^* with all their adjacent edges (and edge labels). Instead we connect u_k directly to u_Λ as its child. We refer to that edge as the *long edge*, and label it with the length of the path it replaces ($k - \Lambda$). The original edges from T^* are called *short edges*. At the end of the pruning step, we denote the resulting tree by T.

The sketch. For each point $x_i \in X$ the sketch stores the index of the leaf v_i that contains it. In addition it stores the structure of the tree T, encoded using the Eulerian Tour Technique[4]. Specifically, starting at the root, we traverse T in the Depth First Search (DFS) order. In each step, DFS either explores the child of the current node (downward step), or returns to the parent node (upward step). We encode a downward step by 0 and an upward step by 1. With each downward step we also store the label of the traversed edge (a length-d bitstring for a short edge or the edge length for a long edge, and an additional bit marking if the edge is short or long).

Decompression. Recovering $\tilde{x}_i$ from the sketch is done simply by following the downward path from the root of T to the associated leaf v_i, collecting the edge labels of the short edges, and placing zeros instead of the missing bits of the long edges. The collected bits then correspond to the binary expansion of the coordinates of $\tilde{x}_i$.

More formally, for every node u (not necessarily a leaf) we define $c(u) \in \mathbb{R}^d$ as follows: For $j \in \{1, \dots, d\}$, concatenate the j^{th} bit of every short edge label traversed along the downward path from the root to u. When traversing a long edge labeled with length k, concatenate k zeros.[5] Then, place a binary floating point in the resulting bitstring, after the bit corresponding to level 0. (Recall that the levels in T are defined

[4]See e.g., https://en.wikipedia.org/wiki/Euler_tour_technique.

[5]This is the “lossy” step in our sketching method: the original bits could be arbitrary, but they are replaced with zeros.

by the grid cell side lengths, and T might not have any nodes in level 0; in this case we need to pad with 0's either on the right or on the left until we have a 0 bit in the location corresponding to level 0.) The resulting binary string is the binary expansion of the j^{th} coordinate of $c(u)$. Now $\tilde{x}_i$ is defined to be $c(v_i)$.

Block QuadSketch. We can further modify QuadSketch in a manner similar to Product Quantization [JDS11]. Specifically, we partition the d dimensions into m blocks $B_1 \dots B_m$ of size d/m each, and apply QuadSketch separately to each block. More formally, for each B_i, we apply QuadSketch to the pointset $(x_1)_{B_i} \dots (x_n)_{B_i}$, where x_B denotes the m/d-dimensional vector obtained by projecting x on the dimensions in B.

The following statement is an immediate corollary of Theorem 1.

Theorem 3. *Given $\epsilon, \delta > 0$, and m dividing d, set the pruning parameter Λ to $O(\log(d \log \Phi/\epsilon\delta))$ and the number of levels L to $\log \Phi + \Lambda$. The m-block variant of QuadSketch runs in time $\tilde{O}(ndL)$ and produces a sketch of size $O(nd\Lambda + nm \log n)$ bits, with the following guarantee: For every $i \in [n]$,*

$$\Pr\left[\forall_{j \in [n]} \|\tilde{x}_i - \tilde{x}_j\| = (1 \pm \epsilon)\|x_i - x_j\|\right] \geq 1 - m\delta.$$

It can be seen that increasing the number of blocks m up to a certain threshold ($d\Lambda/\log n$) does not affect the asymptotic bound on the sketch size. Although we cannot prove that varying m allows to *improve* the accuracy of the sketch, this seems to be the case empirically, as demonstrated in the experimental section.

4 Experiments

We evaluate QuadSketch experimentally and compare its performance to Product Quantization (PQ) [JDS11], a state-of-the-art compression scheme for approximate nearest neighbors, and to a baseline of uniform scalar quantization, which we refer to as Grid. For each dimension of the dataset, Grid places k equally spaced landmark scalars on the interval between the minimum and the maximum values along that dimension, and rounds each coordinate to the nearest landmark.

All three algorithms work by partitioning the data dimensions into blocks, and performing a quantization step in each block independently of the other ones. QuadSketch and PQ take the number of blocks as a parameter, and Grid uses blocks of size 1. The quantization step is the basic algorithm described in Section 3 for QuadSketch, k-means for PQ, and uniform scalar quantization for Grid.

We test the algorithms on four datasets: The SIFT data used in [JDS11], MNIST [LC98] (with all vectors normalized to 1), NYC Taxi ridership data [GMRS16], and a synthetic dataset called Diagonal, consisting of random points on a line embedded in a high-dimensional space. The properties of the datasets are summarized in Table 2. Note that we were not able to compute the exact diameters for MNIST and SIFT, hence we only report estimates for Φ for these data sets, obtained via random sampling.

The Diagonal dataset consists of $10,000$ points of the form $(x, x, \dots, x)$, where x is chosen independently and uniformly at random from the interval $[0..40000]$. This yields a dataset with a very large aspect ratio Φ, and on which partitioning into blocks is not expected to be beneficial since all coordinates are maximally correlated.

For SIFT and MNIST we use the standard query set provided with each dataset. For Taxi and Diagonal we use 500 queries chosen at random from each dataset. For the sake of consistency, for all data sets, we apply the same quantization process jointly to both the point set and the query set, for both PQ and QS. We note, however, that both algorithms can be run on "out of sample" queries.

For each dataset, we enumerate the number of blocks over all divisors of the dimension d. For QuadSketch, L ranges in $2, \dots, 20$, and Λ ranges in $1, \dots, L-1$. For PQ, the number of k-means landmarks per block ranges in $2^5, 2^6, \dots, 2^{12}$. For both algorithms we include the results for all combinations of the parameters, and plot the envelope of the best performing combinations.

Table 2: Datasets used in our empirical evaluation. The aspect ratio of SIFT and MNIST is estimated on a random sample.

Dataset	Points	Dimension	Aspect ratio (Φ)
SIFT	$1,000,000$	128	≥ 83.2
MNIST	$60,000$	784	≥ 9.2
NYC Taxi	$8,874$	48	49.5
Diagonal (synthetic)	$10,000$	128	$20,478,740.2$

We report two measures of performance for each dataset: (a) the *accuracy*, defined as the fraction of queries for which the sketch returns the true nearest neighbor, and (b) the *average distortion*, defined as the ratio between the (true) distances from the query to the reported near neighbor and to the true nearest neighbor. The sketch size is measured in bits per coordinate. The results appear in Figures 2 to 5. Note that the vertical coordinate in the distortion plots corresponds to the value of ϵ, not $1+\epsilon$.

For SIFT, we also include a comparison with Cartesian k-Means (CKM) [NF13], in Figure 6.

4.1 QuadSketch Parameter Setting

We plot how the different parameters of QuadSketch effect its performance. Recall that L determines the number of levels in the quadtree prior to the pruning step, and Λ controls the amount of pruning. By construction, the higher we set these parameters, the larger the sketch will be and with better accuracy. The empirical tradeoff for the SIFT dataset is plotted in Figure 7.

The optimal setting for the number of blocks is not monotone, and generally depends on the specific dataset. It was noted in [JDS11] that on SIFT data an intermediate number of blocks gives the best results, and this is confirmed by our experiments. Table 3 lists the performance on the SIFT dataset for a varying number of blocks, for a fixed setting of $L = 6$ and $\Lambda = 5$. It shows that the sketch quality remains essentially the same, while the size varies significantly, with the optimal size attained at 16 blocks.

# Blocks	Bits per coordinate	Accuracy	Average distortion
1	5.17	0.719	1.0077
2	4.523	0.717	1.0076
4	4.02	0.722	1.0079
8	3.272	0.712	1.0079
16	**2.795**	0.712	1.008
32	3.474	0.712	1.0082
64	4.032	0.713	1.0081
128	4.079	0.72	1.0078

Table 3: QuadSketch accuracy on SIFT data by number of blocks, with $L = 6$ and $\Lambda = 5$.

5 Proofs

In this section we prove Theorems 1 and 2. Recall that we have a pointset $x_1, \ldots, x_n \in \mathbb{R}^d$ with aspect ratio Φ, and given error parameters $\epsilon, \delta > 0$. For the remainder of the section we fix the setting

$$\Lambda = \log(16 d^{1.5} \log \Phi / (\epsilon\delta)).$$

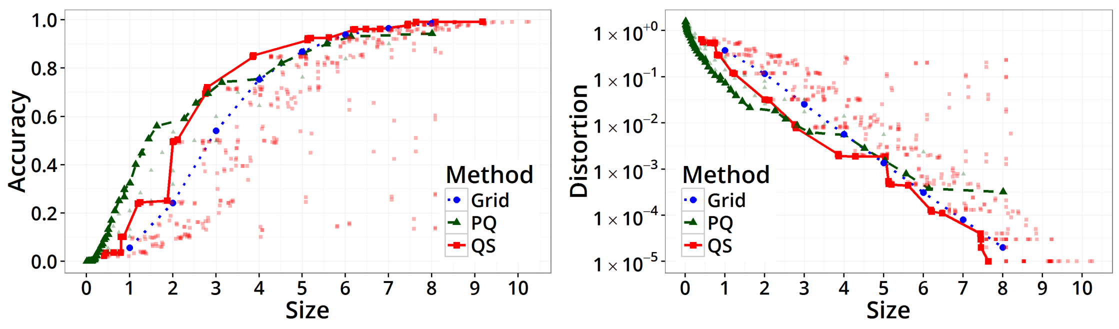

Figure 2: Results for the SIFT dataset.

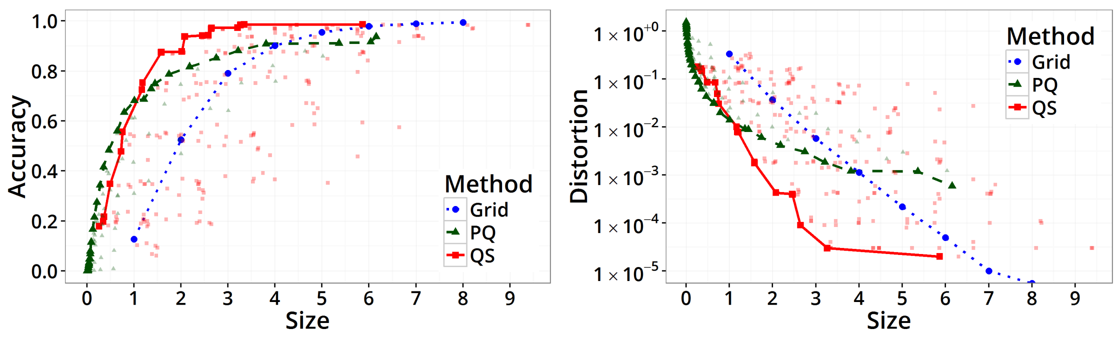

Figure 3: Results for the MNIST dataset.

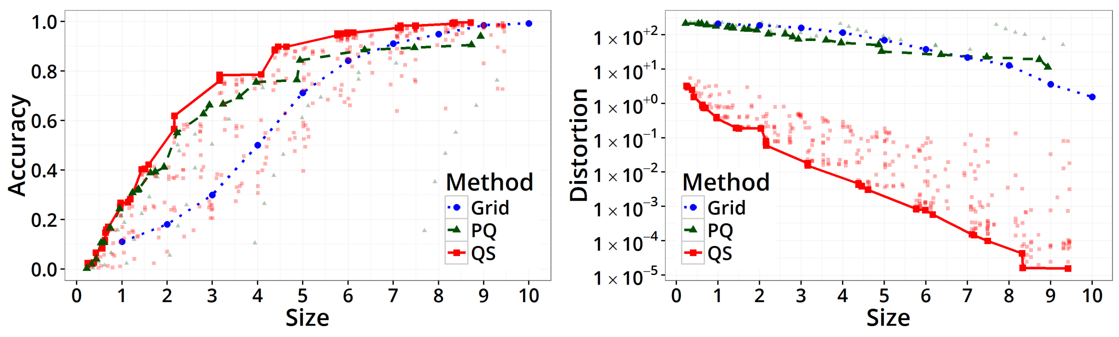

Figure 4: Results for the Taxi dataset.

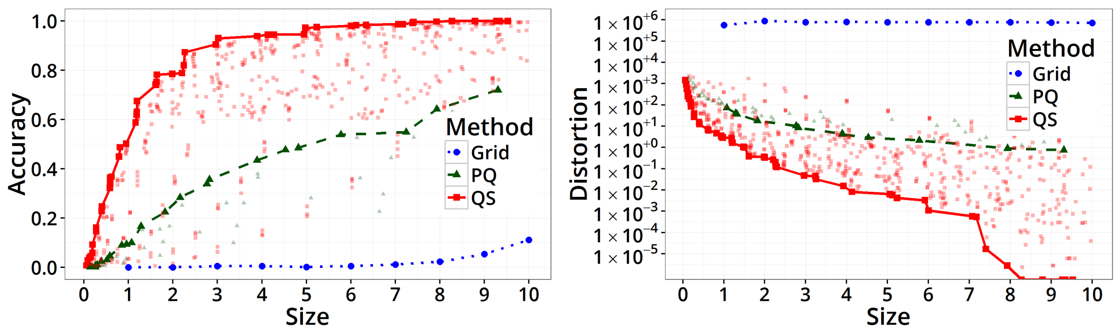

Figure 5: Results for the Diagonal dataset.

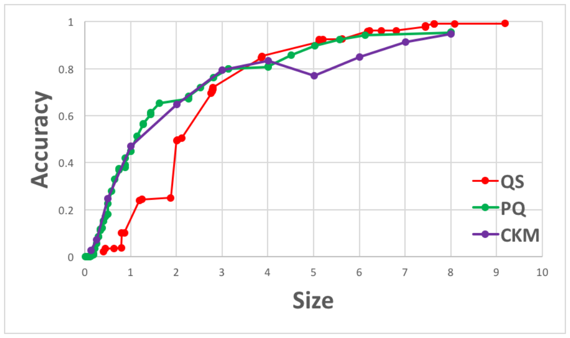

Figure 6: Additional results for the SIFT dataset.

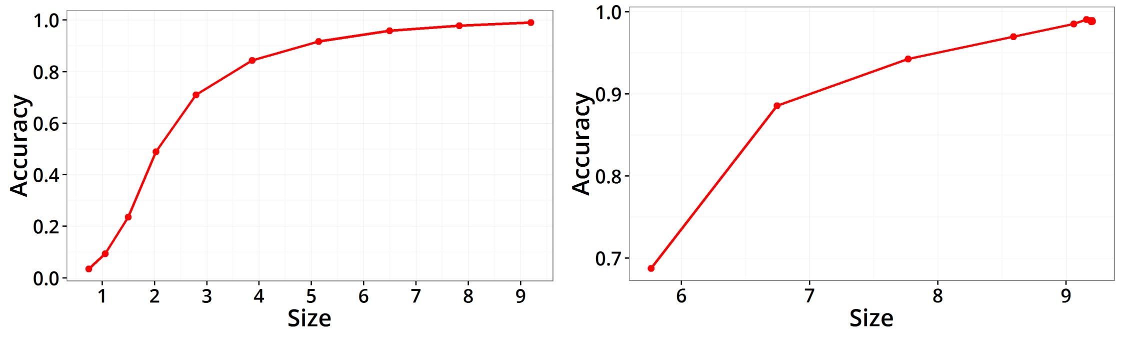

Figure 7: On the left, L varies from 2 to 11 for a fixed setting of 16 blocks and $\Lambda = L - 1$ (no pruning). On the right, Λ varies from 1 to 9 for a fixed setting of 16 blocks and $L = 10$. Increasing Λ beyond 6 does not have further effect on the resulting sketch.

Recall that in Section 3 we let G_ℓ denote the grid with side length 2^ℓ for every integer ℓ. Our analysis is based on the observation that randomly shifting the grids, which is used as the first step of our algorithm, induces a *padded decomposition* [Bar96] of the pointset. We now define this formally.

Definition 1 (padded point)**.** *We say that a point x_i is* $(\epsilon, \Lambda, \ell)$-padded, *if the grid cell in G_ℓ that contains x_i also contains the ball of radius $\rho(\ell)$ centered at x_i, where*

$$\rho(\ell) = 8\epsilon^{-1}2^{\ell-\Lambda}\sqrt{d}.$$

We say that x_i is (ϵ, Λ)-padded *in the quadtree T, if it is $(\epsilon, \Lambda, \ell)$-padded for every level ℓ of T.*

Note that d and Λ are fixed parameters for a given input. We omit their dependence from the notation $\rho(\ell)$ for simplicity.

We now prove Theorem 1. It follows directly from combining the following two lemmas.

Lemma 1. *If the grids are randomly shifted, as in Section 3, then every point x_i is (ϵ, Λ)-padded in T with probability $1 - \delta$.*

Proof. Fix a point x_i, a coordinate $k \in \{1, \ldots, d\}$ and a level ℓ. Let $x_i(k)$ denote the value of x_i in coordinate k. Along this coordinate, we are randomly shifting a 1-dimensional grid partitioned into intervals of length 2^ℓ. Since the shift is uniformly random, the probability for $x_i(k)$ to be at distance at most $\rho(\ell)$ from an endpoint of the interval that contains it equals $2\rho(\ell)/2^\ell$. By plugging our setting of $\rho(\ell)$ and Λ, this probability equals $\delta/(d \log \Phi)$. Taking a union bound over the d coordinates, we have probability at most $\delta/\log\Phi$ for x_i to be at distance at most $\rho(\ell)$ from the boundary of the cell of G_ℓ that contains it. In the complement event x_i is $(\epsilon, \Lambda, \ell)$-padded in G_ℓ. Taking another union bound over the $\log\Phi$ levels in the quadtree, x_i is (ϵ, Λ)-padded with probability at least $1-\delta$. □

Lemma 2. *If a point x_i is (ϵ,Λ)-padded in T, then for every $j \in [n]$,*

$$(1-\epsilon)\|\tilde{x}_i - \tilde{x}_j\| \le \|x_i - x_j\| \le (1+\epsilon)\|\tilde{x}_i - \tilde{x}_j\|,$$

where $\{\tilde{x}_i\}$ are as defined in Section 3.

Proof. We recall that T is a pruned quadtree in which every node v is associated with a grid cell of an axis-parallel grid G_ℓ with side length 2^ℓ, which is aligned with and contained in H. We call ℓ the *level* of v, and denote it henceforth by $\ell(v)$. We will use the term "bottom-left corner" of a grid cell for the corner that minimizes all coordinate values (i.e., the high-dimensional analog of a bottom-left corner in the plane).

Let r be the root of T. We may assume w.l.o.g. that the bottom-left corner of H is the origin in $\mathbb{R}^d$, since translating H together with the entire pointset does not change pairwise distances. Under this assumption, we make the following observation, illustrated in Figure 8.

Observation 1. *Let v be a node in T. If the path from r to v contains only short edges, then $c(v)$ (defined by the decompression algorithm in Section 3) is the bottom-left corner of the grid cell associated with v.*

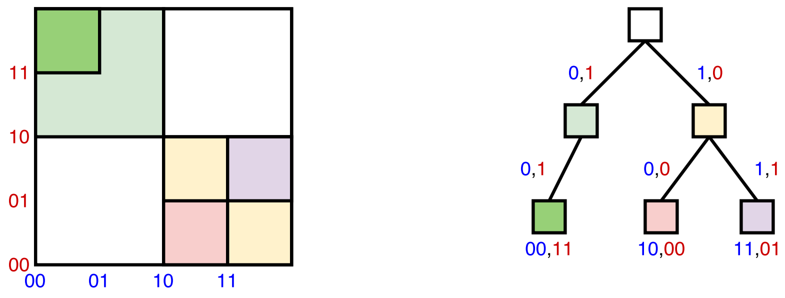

Figure 8: By collecting the edge label bits along every dimension from the root to a node, and padding with zeros as necessary, we obtain the binary expansion of the bottom-left corner of the associated grid cell.

Let x_i be a padded point, and x_j be any point. Recall that we denote by v_i and v_j the leaves corresponding to x_i and x_j respectively (see Section 3). Let w be the lowest common ancestor of v_i and v_j in T. Since x_i and x_j are in separate grid cells of $G_{\ell(w)-1}$, and the cell containing x_i also contains the ball of radius $\rho(\ell(w)-1)$ around x_i, we have

$$\|x_i - x_j\| \ge \rho(\ell(w)-1) = 8\epsilon^{-1}2^{\ell(w)-1-\Lambda}\sqrt{d}. \tag{1}$$

Let u_i be the lowest node on the downward path from w to v_i, that can be reached without traversing a long edge. Similarly define u_j for v_j. See Figure 9 for illustration.

Note that u_i must be either the leaf v_i, or an internal node whose only outgoing edge is a long edge. In both cases, u_i is the bottom of a path of degree-1 nodes of length Λ:

- If u_i is a leaf: Since the pointset has aspect ratio Φ, then after $\log \Phi$ levels the grid becomes sufficiently fine such that each grid cell contains at most one point x_i. Since we generate the quadtree with $L = \log \Phi + \Lambda$ levels, then each point x_i is in its own grid cell for at least the bottom Λ levels of the quadtree.
- If u_i is an internal node which is the head of a long edge: Since the pruning step only places long edges at the bottom of degree-1 paths of length Λ, then u_i must be the bottom node of such path.

On the other hand w is an ancestor of u_i, and it has degree at least 2, since it is also an ancestor of u_j. Hence w is at least Λ levels above u_i, implying $\ell(v_i) \leq \ell(w) - \Lambda$. Applying the same arguments to u_j we get also $\ell(v_j) \leq \ell(w) - \Lambda$.

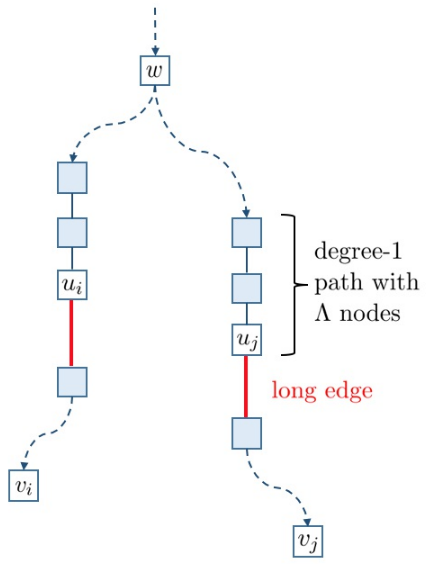

Figure 9: In the proof of Lemma 2, w is the lowest common ancestor of v_i, v_j, the leaves corresponding to x_i, x_j. u_i is the lowest node on the downward path from w to v_i which is achievable without traversing any long edges (marked in red). u_j is defined similarly for v_j.

Let $c^*(u_i), c^*(u_j) \in \mathbb{R}^d$ be the bottom-left corners of the grid cells associated with u_i and u_j. If all edges on the downward paths from the root of T to u_i and u_j were short, then Observation 1 would yield that $c^*(u_i) = c(u_i)$ and $c^*(u_j) = c(u_j)$. In general, there might be some long edges on those paths, but they all must lie on the subpath from the root of T down to w, which is the same for both paths. This is because by the choice of u_i and u_j, all downward edges from w to either of them are short. Therefore $c(u_i)$ and $c(u_j)$ are shifted from the true bottom-left corners by the same shift, which we denote by

$$\eta = c^*(u_i) - c(u_i) = c^*(u_j) - c(u_j).$$

Next, observe that the grid cell associated with u_i has side $2^{\ell(u_i)}$ and it contains both $c^*(u_i)$ and x_j. Therefore $\|x_i - c^*(u_i)\| \leq 2^{\ell(u_i)}\sqrt{d}$.

Furthermore, since u_i is an ancestor of v_i, then by the definition of $c(u_i)$ and $c(v_i)$, in each coordinate, the binary expansions of these two vertices are equal from the location $\ell(u_i)$ and up. In the less significant locations, $c(u_i)$ is zeroed while $c(v_i)$ may have arbitrary bits. This means that the difference between $c(u_i)$ and $c(v_i)$ in each coordinate can be at most $2^{\ell(u_i)}$ in the absolute value, and consequently $\|c(v_i) - c(u_i)\| \le 2^{\ell(u_i)}\sqrt{d}$. Recalling that the decompression algorithm defines $\tilde{x}_i = c(v_i)$, we get $\|\tilde{x}_i - c(u_i)\| \le 2^{\ell(u_i)}\sqrt{d}$.

Collecting the above inequalities, we have

$$\begin{aligned}\|x_i - \eta - \tilde{x}_i\| &= \|x_i - c(u_i) - \eta + c(u_i) - \tilde{x}_i\| \\ &= \|x_i - c^*(u_i) + c(u_i) - \tilde{x}_i\| \\ &\le \|x_i - c^*(u_i)\| + \|c(u_i) - \tilde{x}_i\| \\ &\le 2 \cdot 2^{\ell(u_i)}\sqrt{d} \\ &\le 2 \cdot 2^{\ell(w)-\Lambda}\sqrt{d}.\end{aligned}$$

Similarly for j we have $\|x_i - \eta - \tilde{x}_i\| \le 2 \cdot 2^{\ell(w)-\Lambda}\sqrt{d}$. Together, by the triangle inequality,

$$\begin{aligned}\|\tilde{x}_i - \tilde{x}_j\| &= \|\tilde{x}_i + \eta - x_i + x_i - x_j + x_j - \eta - \tilde{x}_j\| \\ &= \|x_i - x_j\| \pm (\|x_i - \eta - \tilde{x}_i\| + \|x_i - \eta - \tilde{x}_i\|) \\ &= \|x_i - x_j\| \pm 4 \cdot 2^{\ell(w)-\Lambda}\sqrt{d}.\end{aligned}$$

To complete the proof of Lemma 2 it remains to show $4 \cdot 2^{\ell(w)-\Lambda}\sqrt{d} \le \epsilon \cdot \|x_i - x_j\|$, which follows from Equation (1). □

5.1 Sketch Size and Running Time

Lemma 3. *QuadSketch produces a sketch of size $O(nd\Lambda + n\log n)$ bits.*

Proof. The tree T has n leaves, and we have pruned each non-branching path in it to length Λ. Hence its total size is $O(n\Lambda)$, and its structure can be stored with this many bits using (for example) the DFS scan described in Section 3. Each short edge label is d bits long, so together they consume $O(nd\Lambda)$ bits. As for the long edges, there can be at most $O(n)$ of them, since the bottom of each long edge is either a branching node or a leaf. The long edge labels are lengths of downward paths in the non-pruned tree T^*, whose height bounded by is $O(\log\Phi + \Lambda)$. Together the long edge labels consume $O(n\log(\log\Phi + \Lambda))$ bits, which is dominated by $O(n\Lambda)$. Finally for each point x_i we store the index of its corresponding leaf v_i, and since there are n leaves, this requires $O(n\log n)$ additional bits to store. □

Lemma 4. *The QuadSketch construction algorithm runs in time $O(ndL)$.*

Proof. Given a quadtree cell and a point contained in it, in order to bucket the point into a cell in the next level, we need to check for each coordinate whether the point falls in the upper or lower half of the cell. This takes time $O(d)$. Since each point is bucketed once in every level, and we generate T^* for L levels, the quadtree construction time is $O(ndL)$. The pruning step requires just a linear scan of T^*, in time $O(nL)$. □

5.2 Maximum Distortion

We now prove Theorem 2.

Sketching algorithm Given a pointset X, apply QuadSketch to X and let T_1 be the resulting tree. Let $Q \subset X$ be the padded points in T_1 (meaning those for which the condition of Lemma 1 is satisfied for T_1). Continue by recursion on $X \setminus Q$, until all points in X are padded in some tree. The returned sketch contains all trees $T_1, \ldots, T_k$ constructed during the recursion, and in addition, for every point x_i we store the index γ_i of the tree in which it is padded.

Query algorithm Given two point indices i, j, assume w.l.o.g. $\gamma(i) \leq \gamma(j)$, then the tree $T_{\gamma(i)}$ has corresponding leaves for both x_i and x_j. We decompress $\tilde{x}_i$ and $\tilde{x}_j$ from $T_{\gamma(i)}$ and return $\|\tilde{x}_i - \tilde{x}_j\|$.

Analysis The correctness of the estimate up to distortion $1 \pm \epsilon$ follows from Lemma 2. We now bound the sketch size and the running time. Lemma 1 with $\delta = 0.25$ implies that in each of the trees $T_1, \ldots, T_k$, the expected fraction of padded points is 0.75. Hence by Markov's inequality, with probability 0.5 at least half the points are padded. Since the calls to QuadSketch are independent (and its success probability depends only on its internal randomness and not on the input points), with probability $0.5^{\lceil \log_2 n \rceil} \sim 1/n$ this happens in each of the first $k = \lceil \log_2 n \rceil$ iterations. This probability can be amplified to constant by $O(\log n)$ independent repetitions. If this event has happened then the sketching algorithm terminates since Q becomes empty. Therefore the total running time of a successful execution is $O(\log^2 n)$ calls to QuadSketch, which by Lemma 4 is $\tilde{O}(ndL)$.

Furthermore, since the number of padded points decreases by at least half in every iteration, the total size of the sketches $T_1, \ldots, T_k$ is

$$O\left(\sum_{k'=0}^{k-1} \frac{n}{2^{k'}}(d\Lambda + \log \frac{n}{2^{k'}})\right) = O(n(d\Lambda + \log n)),$$

the same as in Theorem 1 up to a constant factor. Finally, since each $\gamma(i)$ is index in $\{1, \ldots, \lceil \log_2 n \rceil\}$, the $\gamma(i)$'s only take additional $O(n \log \log n)$ bits to store. $\square$

Acknowledgements

This research was supported by grants from NSF, Simons Foundation, and Shell (via MIT Energy Initiative).

References

[AZ14] Relja Arandjelović and Andrew Zisserman, *Extremely low bit-rate nearest neighbor search using a set compression tree*, IEEE transactions on pattern analysis and machine intelligence **36** (2014), no. 12, 2396–2406.

[Bar96] Yair Bartal, *Probabilistic approximation of metric spaces and its algorithmic applications*, Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on, IEEE, 1996, pp. 184–193.

[Bro97] Andrei Z Broder, *On the resemblance and containment of documents*, Compression and Complexity of Sequences 1997. Proceedings, IEEE, 1997, pp. 21–29.

[GMRS16] Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers, *Robust random cut forest based anomaly detection on streams*, International Conference on Machine Learning, 2016, pp. 2712–2721.

[IM98] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: towards removing the curse of dimensionality*, Proceedings of the thirtieth annual ACM symposium on Theory of computing, ACM, 1998, pp. 604–613.

[IW17] Piotr Indyk and Tal Wagner, *Near-optimal (euclidean) metric compression*, Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, 2017, pp. 710–723.

[JDJ17] Jeff Johnson, Matthijs Douze, and Hervé Jégou, *Billion-scale similarity search with gpus*, CoRR **abs/1702.08734** (2017).

[JDS11] Herve Jegou, Matthijs Douze, and Cordelia Schmid, *Product quantization for nearest neighbor search*, IEEE transactions on pattern analysis and machine intelligence **33** (2011), no. 1, 117–128.

[JL84] William B Johnson and Joram Lindenstrauss, *Extensions of lipschitz mappings into a hilbert space*, Contemporary mathematics **26** (1984), no. 189-206, 1.

[KOR00] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani, *Efficient search for approximate nearest neighbor in high dimensional spaces*, SIAM Journal on Computing **30** (2000), no. 2, 457–474.

[LC98] Yann LeCun and Corinna Cortes, *The mnist database of handwritten digits*, 1998.

[LSMK11] Ping Li, Anshumali Shrivastava, Joshua L Moore, and Arnd C König, *Hashing algorithms for large-scale learning*, Advances in neural information processing systems, 2011, pp. 2672–2680.

[M$^+$05] Shanmugavelayutham Muthukrishnan et al., *Data streams: Algorithms and applications*, Foundations and Trends® in Theoretical Computer Science **1** (2005), no. 2, 117–236.

[NF13] Mohammad Norouzi and David J Fleet, *Cartesian k-means*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 3017–3024.

[NFS12] Mohammad Norouzi, David J Fleet, and Ruslan R Salakhutdinov, *Hamming distance metric learning*, Advances in neural information processing systems, 2012, pp. 1061–1069.

[RL09] Maxim Raginsky and Svetlana Lazebnik, *Locality-sensitive binary codes from shift-invariant kernels*, Advances in neural information processing systems, 2009, pp. 1509–1517.

[SH09] Ruslan Salakhutdinov and Geoffrey Hinton, *Semantic hashing*, International Journal of Approximate Reasoning **50** (2009), no. 7, 969–978.

[SL14] Anshumali Shrivastava and Ping Li, *Densifying one permutation hashing via rotation for fast near neighbor search.*, ICML, 2014, pp. 557–565.

[TFW08] Antonio Torralba, Rob Fergus, and Yair Weiss, *Small codes and large image databases for recognition*, Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, IEEE, 2008, pp. 1–8.

[WDL$^+$09] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg, *Feature hashing for large scale multitask learning*, Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 1113–1120.

[WLKC16] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang, *Learning to hash for indexing big data: a survey*, Proceedings of the IEEE **104** (2016), no. 1, 34–57.

[WTF09] Yair Weiss, Antonio Torralba, and Rob Fergus, *Spectral hashing*, Advances in neural information processing systems, 2009, pp. 1753–1760.

[YS83] Mann-May Yau and Sargur N Srihari, *A hierarchical data structure for multidimensional digital images*, Communications of the ACM **26** (1983), no. 7, 504–515.