LSH Forest: Practical Algorithms Made Theoretical

Alexandr Andoni
Columbia University

Ilya Razenshteyn
MIT CSAIL

Negev Shekel Nosatzki
Columbia University

February 26, 2017

Abstract

We analyze *LSH Forest* [BCG05]—a popular heuristic for the nearest neighbor search—and show that a careful yet simple modification of it outperforms "vanilla" LSH algorithms. The end result is the first instance of a simple, practical algorithm that provably leverages data-dependent hashing to improve upon data-oblivious LSH.

Here is the entire algorithm for the d-dimensional Hamming space. The LSH Forest, for a given dataset, applies a random permutation to all the d coordinates, and builds a *trie* on the resulting strings. In our modification, we further augment this trie: for each node, we store a *constant* number of points close to the mean of the corresponding subset of the dataset, which are compared to any query point reaching that node. The overall data structure is simply several such tries sampled independently.

While the new algorithm does not *quantitatively* improve upon the best data-dependent hashing algorithms from [AR15] (which are known to be optimal), it is significantly simpler, being based on a practical heuristic, and is provably better than the best LSH algorithm for the Hamming space [IM98, HIM12].

1 Introduction

Nearest Neighbor Search. In the Nearest Neighbor Search problem (NNS), we are given a set P of n points in a d-dimensional space, and the goal is to build a data structure that, given a query point q, reports any point from P within a given distance r to the query q. The problem is of major importance in several areas, including databases, data mining, information retrieval, computer vision, computational geometry, signal processing, etc.

Despite having efficient solutions for the low-dimensional case [Cla88, Mei93], the high-dimensional NNS suffers from the "curse of dimensionality" and is believed to not admit efficient algorithms. For this reason, theoretical research has focused on the *approximate* variant of NNS, making significant progress. In the (c, r)-approximate near neighbor problem (ANN), the data structure may return any data point whose distance from the query is at most cr, for an approximation factor $c > 1$ (provided that there exists a data point within distance r from the query). Many algorithms for the problem are known: e.g., see surveys [AI08, WSSJ14].

To address the approximate version, Indyk and Motwani proposed the Locality-Sensitive Hashing scheme (LSH), which has since proved to be influential in theory and practice [HIM12]. In particular, LSH hits the sweet spot of ANN data structures with subquadratic space for constant approximation factor, which turns out to be the most important regime from the practical perspective. The main idea is to hash the points such that the probability of collision is much higher for points which are close to each other (at distance $\leq r$) than for those which are far apart (at distance $> cr$). Given such hash functions, one can retrieve near neighbors by hashing the query point and retrieving elements stored in buckets containing that point. If the probability of collision is at least p_1 for the close points and at most p_2 for the far points, the algorithm solves the (c, r)-ANN using $n^{1+\rho}$ extra space and dn^ρ query time, where $\rho = \log(1/p_1)/\log(1/p_2)$ [HIM12].

The performance of an LSH family is thus determined by the value of the exponent ρ, for which we now know precise bounds. The original LSH paper [IM98] obtained $\rho = 1/c$ for the Hamming space. For the Euclidean space, subsequent research yielded a better exponent: $\rho = 1/c^2 + o(1)$ [DIIM04, AI06]. These bounds are tight for both Hamming and Euclidean space, as shown by the lower bounds of [MNP07, OWZ14], thus completing this line of research.

Data-dependent hashing: theory and practice. Recent work showed that one can outperform LSH-based algorithms by using *data-dependent* hashing. Quantitatively speaking, the work of [AINR14, AR15] developed algorithms with space overhead $n^{1+\rho}$ and query time dn^ρ, where $\rho = 1/(2c-1) + o(1)$ for the Hamming space and $\rho = 1/(2c^2-1) + o(1)$ for the Euclidean space. Note that, for large c, this is a quadratic improvement of the query time (with better space) over best possible LSH algorithms; the new bounds are known to be optimal for data-dependent hashing [AR16, ALRW16]. In addition to the quantitative improvement, the new space partitions are *qualitatively* different: they adapt to the geometry of a given dataset, opening space for a new type of algorithmic techniques for the ANN problem.

The idea of data-dependent hashing has become quite popular in practice in recent years [WSSJ14, WLKC15], though it is for the following subtly different reason: datasets that arise in practice are hypothesized to not be "worst case", and instead exhibit some structure, such as *low intrinsic dimension.* Algorithms used in practice try to exploit (implicitly or explicitly) precisely this additional structure. Let us remark that most of these algorithms do not have any guarantees, on correctness or performance (but see the related work section for some recent examples that do). In fact, it has become an important challenge to bridge this gap between theoretical and practical algorithms (see, e.g., a National Research Council report [Cou13, Section 5], or the survey [WLKC15, Section 7]).

While data-dependent hashing algorithms come close to bridging the gap—they are empirically better in practice and are now proven to be better in theory—yet, the gap persists: the theoretical and practical algorithms are still remarkably different. In particular, the data-dependent algorithms from [AINR14, AR15] are quite far from the aforementioned practical algorithms, or being practical at all. One step towards practice was undertaken in [AIL$^+$15], who gave an efficient implementation of one of the components of [AR15]. This component is however data-*independent*: it is a better LSH scheme for a dataset that is assumed to already have a nice structure (loosely speaking, it looks like a random set). In particular, the algorithm of [AIL$^+$15] has no data-dependent steps, and hence it remains an open question to develop practical data-dependent algorithms, with theoretical guarantees á la [AR15].

One immediate barrier for making [AR15, AINR14] more practical is the fact that these algorithms use a certain decomposition procedure that partitions a *worst case* dataset into several subsets which are *pseudo-random.* This procedure is precisely the data-dependent component of the algorithm. This decomposition is not particularly efficient, and generously contributes to the $n^{o(1)}$ factors in the bounds for space and query time. Making such reductions for finite subsets of $\mathbb{R}^d$ more efficient would be of independent interest—in particular, for problems where we know algorithms with better performance for the pseudo-random datasets than the worst-case datasets (e.g., [Val15, KKK16, KKKÓ16, ACW16]).

1.1 Our results

In this paper we show new, simple data-dependent algorithms that are *provably* better than data-oblivious LSH algorithms, even for *worst case* instances. While the algorithms do not improve the bounds of [AR15] quantitatively (in fact they are worse), the main advantage is simplicity and relation to algorithms used in practice.

Our improved algorithm is based on *LSH Forest*, a heuristic introduced in [BCG05] and now popular in practice (see, e.g., [Ber13]). It can be applied as a black box to any LSH hash family, but we will need it instantiated to the Bit Sampling LSH of [IM98, HIM12] for the Hamming space. We show that while using LSH Forest directly does not lead to improved algorithms, a simple modification of it does!

Before describing our main algorithm, let us briefly recall the Bit Sampling LSH algorithm from [IM98, HIM12]. We hash an n-point dataset $P \subseteq \{0,1\}^d$ using a hash function $h(x) = x_I$, where x_I is a restriction of the point $x \in \{0,1\}^d$ on a set of indeces $I \subseteq [d]$. The set I is chosen to be a random subset of $[d]$ of fixed, carefully chosen size. Given a query $q \in \{0,1\}^d$, we retrieve all the data points $p \in P$ such that $h(p) = h(x)$ and check them all until we find a point within distance cr from q. In order to boost the probability of success to constant, we repeat the above construction n^ρ times, where $\rho = \frac{1}{c}$. It is not hard to construct an example, where we must take ρ to be at least $\frac{1}{c} - o(1)$ and, moreover, this bound is tight for *any* data-independent LSH for the Hamming distance [OWZ14].

It will be convenient to assume that there is exactly one data point—the near neighbor—within distance cr from the query (and, thus, the data structure must recover the near neighbor). This assumption is adequate for practice: for many real-world datasets there are not too many data points that are not much further from the query than the near neighbor. At the same time, later we will show a modification of our algorithm that can handle the general case of (c, r)-ANN.

Our new, improved algorithm is based on the LSH Forest algorithm, which proceeds as follows when instantiated to the Bit Sampling LSH. We choose a random permutation of the coordinates $\pi : [d] \to [d]$ and apply it to each vector $x \in P$. Then we build a *trie* on the permuted vectors.[1] For a given query

[1]This can be also seen as applying kd-tree on P with for a *random order* of the coordinates, stopping the splitting until we have exactly one point.

q, we descent down the trie until we reach precisely one leaf and compare against the point stored in that leaf. As before, we boost the probability of success by building n^ρ such tries (hence the name of "forest").

While the above algorithm still requires $\rho \geq 1/c$ in the worst case, here is how we modify it to bypass the $1/c$ bound. For a constant parameter $k = O(1)$, and for every tree node v, we choose a fixed set of k *pivots* T_v. Then, when answering a query, while descending down the tree, we check against all the pivots T_v of every node v we visit on the way to the leaf. Overall, we compare the query point with at most dk data points. The following theorem summarizes the main result of this paper assuming that there is only the near neighbor within distance cr from the query.

THEOREM 1.1. (INFORMAL) *For every $k \geq 1$, there exists a procedure of choosing k pivots in each node of the tree of LSH Forest, such that, in order to get a data structure for (c, r)-ANN for the Hamming distance with constant probability of success, it is enough to sample $O(n^\rho)$ tries, where:*

$$\rho \leq \frac{1}{\ln 4} \cdot \frac{1}{c} \cdot \left(1 + O\left(\frac{1}{k}\right)\right) + O\left(\frac{1}{c^2}\right).$$

Choosing k to be large enough, the exponent is $\rho = 0.73/c + O(1/c^2)$. The resulting query time is $O(n^\rho d^2 k)$ and space is $O(n^{1+\rho}k + nd)$.

To get rid of the assumption, we need to augment the set of pivots further. Namely, in addition to k carefully chosen pivots, we need to include $\Theta(\log n)$ *uniformly random pivots.* If we do this, Theorem 1.1 will hold for the original (c, r)-ANN with a logarithmic loss in space and query time.

Discussion of the theorem statement. Our resulting bound of, essentially, $\rho \approx \frac{1}{\ln 4 \cdot c}$ corresponds to the bound that follows from using the Bit Sampling LSH of [IM98, HIM12] on *a random dataset* in $\{0,1\}^d$. Hence the algorithm can be seen as a worst-case–to–random-case reduction in the case of Bit Sampling LSH. While there exist space partitions that are better than Bit Sampling on a random dataset [AINR14, AR15, AIL$^+$15], they are less efficient, and, in particular require a *larger number of parts*, requiring a tree with a large branching factor. This seems required at least in some cases: in particular, for the *Euclidean space*, [AIL$^+$15] prove that, for any space partition where ρ differs from the optimal exponent by ε, the number of parts must be exponential in $1/\varepsilon$. We conjecture a similar phenomenon happens for the Hamming case; indeed, in all the *known* constructions, we need at least 10^6 parts to improve upon the Bit Sampling for random instances, for $c = 2$. Furthermore, in all known space partitions, the decoding time is proportional to the number of parts, and hence it is a very important consideration in practice.

Discussion of the algorithm. We note that LSH Forest by itself is already a data-dependent algorithm, in that it auto-tunes to the dataset. In contrast to Bit Sampling that uses a fixed number of (random) coordinates, LSH Forest uses a variable number of coordinates, which may differ from one part of the tree to the other. Informally speaking, local granularity of a partition depends on the local density of a region being partitioned, which is beneficial for irregular datasets. Nonetheless, despite the vanilla LSH Forest working well in practice, one can still construct a bad example, where one has to sample $n^{\frac{1}{c} - o(1)}$ trees. In particular, in the *worst case* LSH Forest by itself does not offer any advantage over the regular Bit Sampling. It is precisely the carefully selected pivots that allow LSH Forest to achieve the improved bound.

We note that our algorithm can also be seen as the LSH Forest essentially instantiated to the *min-hash* scheme [Bro97, BGMZ97], another very popular (data-independent) hash function for the Hamming space, used extensively in practice. In particular, the min-hash function h chooses a random permutation π and, on point $x \in \{0,1\}^d$, outputs the smallest coordinate i in π where $x_i = 1$. The resulting space partition corresponds to a caterpillar tree with the permutation π. Hence, if we concatenate a few minhashes, and simplify the tree, we obtain precisely the above LSH Forest. Again, pivots are essential to obtain improved bounds.

We also comment on the use of *randomness* in the tree. In the later sections, we use LSH Forest where different parts of the trie use *independent* randomness: i.e., in each node we choose a new random coordinate to split the dataset by, and recurse in each of the two parts corresponding to the dataset conditioned on the value in that coordinate. This does not affect the analysis (due to the linearity of expectation) , but is more general and likely amenable to further improvements.

A final remark regarding the analysis of our algorithm is that this is the first time when the analysis of random space partitions is with respect to the entire dataset. This is in contrast to the standard analysis of LSH [HIM12], data-dependent hashing [AR15], or the analysis of [DS13], where the analysis of collision probability is with respect to *two* or *three* points. In this paper, we perform a global analysis, which turns out to be, technically, the most involved part of the paper.

1.2 Related work As previously mentioned, data-dependent hashing is an ubiquitous idea in practice, albeit with few, if any guarantees on correctness or performance. To list just a few examples, these include PCA-trees [Spr91, McN01, VKS09] and its variants (called randomized kd-tree) [SH08, ML09], spectral hashing [WTF08], semantic hashing [SH09], and WTA hashing [YSRL11], and many more. We point an interested reader to the surveys of [WSSJ14, WLKC15].

There have also been efforts to obtain algorithms that are both data-dependent (in a practical way) and have theoretical guarantees. Most such examples include the algorithms that assume some additional structure in the dataset: such as some notion of *low intrinsic dimension* [KR02, CNBM01, KL04, BKL06, IN07, Cla06, DF08], or low dimensional data-set with high-dimensional noise [AAKK14]. Most relevant to us is the work of [DS13], which, while mostly focusing on the low intrinsic dimensional datasets, give a generic bound for the *worst-case* datasets as well. The bound of [DS13] however depends on the dataset in a non-obvious way, and it is not clear whether it obtains bounds that are better than (data-independent) LSH.

2 Preliminaries

As is the case with most ANN algorithm, we use random partition of a metric space. For a point x and a partition $\mathcal{R}$, we denote $\mathcal{R}(x)$ the part of $\mathcal{R}$ the point x lies in.

Definition 2.1. *The* (c, r)-Approximate Near Neighbor problem *(ANN) is to construct a data structure over an n-point dataset $P \subset X$ lying in a metric space X supporting the following queries: given any fixed query point $q \in X$, if there exists $p^* \in P$ with $d_X(q, p^*) \le r$, then report some $p' \in X$ such that $d_X(q, p') \le cr$, with probability at least* 0.9.

It will be convenient to introduce the following simplification of Definition 2.1.

Definition 2.2. *The* (c, r)-Gap Approximate Near Neighbor problem *is to construct a data structure over an n-point dataset $P \subset X$ lying in a metric space X supporting the following queries: given any fixed query point $q \in X$ with the promise that there is a data point $p^* \in P$ with $d_X(q, p^*) \le r$ and other data points are further than cr from the query q, the goal is to find p^* with probability at least* 0.9.

Definition 2.3. ([IM98, HIM12]) *We call a random partition $\mathcal{R}$ of a metric space X an (r_1, r_2, p_1, p_2)-*sensitive LSH partition, *if for every $x_1, x_2 \in X$ with $d_X(x_1, x_2) \le r_1$, we have $\Pr_{\mathcal{R}}[\mathcal{R}(x_1) = \mathcal{R}(x_2)] \ge p_1$, while for every $x_1, x_2 \in X$ with $d_X(x_1, x_2) > r_2$, we have $\Pr_{\mathcal{R}}[\mathcal{R}(x_1) = \mathcal{R}(x_2)] \le p_2$.*

Theorem 2.1. ([IM98, HIM12]) *Suppose that $\mathcal{R}$ is a (r, cr, p_1, p_2)-sensitive LSH partition of a metric space X. We assume that $0 < p_1, p_2 < 1$ and denote $\rho = \log(1/p_1)/\log(1/p_2)$. Then, there exists a data structure for (c, r)-ANN over a dataset $P \subset X$ that consists of at most n points such that:*

- *The query procedure requires $O(n^\rho/p_1 \cdot \log_{1/p_2} n)$ point locations in partitions sampled according to $\mathcal{R}$ and $O(n^\rho/p_1)$ distance computations and other operations;*
- *The data structure uses at most $O(n^{1+\rho}/p_1)$ words of space, in addition to the space needed to store the dataset P and to the space needed to store $O(\log_{1/p_2} n)$ partitions sampled according to $\mathcal{R}$.*

The failure probability of the data structure is at most 0.1.

In particular, the original LSH paper used the following Bit Sampling partition.

Definition 2.4. *For a Hamming space $\{0, 1\}^d$ the* Bit Sampling LSH *partition [IM98, HIM12] is as follows: one first chooses a random coordinate $i \in [d]$, and the partition is $\{x \mid x_i = 0\} \cup \{x \mid x_i = 1\}$. It is easy to show that for every r and c it is $(r, cr, 1 - r/d, 1 - cr/d)$-sensitive. In particular,*

$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)} = \frac{\log\left(1 - \frac{r}{d}\right)^{-1}}{\log\left(1 - \frac{cr}{d}\right)^{-1}} \le \frac{1}{c}.$$

For the Bit Sampling LSH, the worst-case exponent of $\rho = 1/c$ occurs when $r \ll d$. On the other hand, for $r = d/2c$, one obtains $\rho = \frac{\log 1/2}{\log 1 - 1/2c} \approx \frac{1}{\ln 4} \cdot \frac{1}{c} + O(1/c^2)$ when $c \to \infty$. This case of $r = d/2c$ corresponds to the canonical random instance in the Hamming space: where the dataset is i.i.d. random in $\{0, 1\}^d$ and the query point is planted within distance $d/2c$ from a random dataset point.

3 Generic LSH Forest with pivots

In this section we describe a generic construction and analysis of LSH Forest. This can be seen as a different way to use random partitions than the classical one, as in Theorem 2.1. We instantiate the general LSH Forest algorithm in later sections. LSH Forest was first introduced in [BCG05] as a heuristic.

LSH Forest consists of several decision trees. Each tree is built and queried using Algorithm 1, which works as follows. In the root of the tree, we start with the entire dataset $S = P \subset X$. We sample a random partition $\mathcal{R}$ of X and group the points from S according to the parts of $\mathcal{R}$. We recurse on each non-empty group and add the resulting trees as children of the root node. We stop as soon as a current subset $S \subset P$ becomes of size 1. In each node v we store the pivot set, denoted T_v, the random partition sampled while building v, denoted by $\mathcal{R}_v$, and the list of children. During the query stage, we descend down the tree, where at each node v, we follow the child that captures the query q in the partition $\mathcal{R}_v$. We also compute the distance to the points $p \in T_v$ for each traversed node v. One such tree succeed with a relatively low probability, and hence we build several independent trees by calling BuildTree(P), thus boosting this probability to 0.9. During the query stage, we query every tree.

Algorithm 1 Generic LSH Forest with pivots

```
function BuildTree($S$)    ▷ $S$ is a subset of the dataset
    create a node $v$ with associated set $S$
    compute pivots $\emptyset \neq T_v \subseteq S$, store them in $v$
    if $|S| = 1$ then
        return $v$
    sample a partition $\mathcal{R}_v$ and store it in $v$
    for $U \in \mathcal{R}_v$ do
        if $|S \cap U| \neq 0$ then
            add BuildTree($S \cap U$) as a child of $v$
    return $v$
function QueryTree($v$, $q$)    ▷ $v$ is a tree node, $q$ is a query point
    if $\exists\, p \in T_v$ within distance $cr$ from $q$ then
        return $p$
    if there is a child $v'$ of $v$ associated with $\mathcal{R}_v(q)$ then
        return QueryTree($v'$, $q$)
    else
        return $\perp$
```

To analyze LSH Forest, we need to understand two quantities: how many trees do we need to get probability of success 0.9, say, and how deep the built trees are. The latter is quite easy, while the former is much more delicate and is the main focus of this paper.

Lemma 3.1. *For a dataset P of size n, let $q \in X$ be a query point, and $p^* \in P$ be a near neighbor of q (that is, $d_X(q, p^*) \le r$). Let $0 < \rho \le 1$ be the smallest real number such that, for every subset $S \subseteq P$ where $p^* \in S$, the following holds:*

- *either q is within distance cr from one of the pivots* BuildTree *computes for S,*
- *or one has*

$$\Pr_{\mathcal{R}}[\mathcal{R}(q) = \mathcal{R}(p^*)] \times \tag{3.1}$$
$$\times \operatorname{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right] \ge 1.$$

Then the tree from Algorithm 1 returns a point within distance cr for a query q with probability at least $n^{-\rho}$. Hence, if one samples $O(n^\rho)$ trees, the entire data structure solves (c, r)-ANN for q with probability at least 0.9.

Proof. Consider any tree node we encounter answering the query q for which $p^* \in S$. We would like to prove that the probability of success in this case is at least $|S|^{-\rho}$, which, when applied to the root of the tree, gives the desired claim.

If the first bullet point from the lemma statement holds, the desired probability is equal to one. Similarly, if $|S| = 1$, the probability is also one, since the set of pivots is non-empty, and, by the assumption, $p^* \in S$. Otherwise, let us use the induction on the size of S.

We have:

$$\begin{aligned}
\Pr[\text{success}] &= \Pr_{\mathcal{R}}[\text{success for } S \cap \mathcal{R}(q)] \\
&\ge \Pr_{\mathcal{R}}[\text{success for } S \cap \mathcal{R}(q), \mathcal{R}(q) = \mathcal{R}(p^*)] \\
&= \Pr_{\mathcal{R}}[\mathcal{R}(q) = \mathcal{R}(p^*)] \cdot \\
&\quad \cdot \Pr_{\mathcal{R}}[\text{success for } S \cap \mathcal{R}(q) \mid \mathcal{R}(q) = \mathcal{R}(p^*)] \\
&\ge \Pr_{\mathcal{R}}[\mathcal{R}(q) = \mathcal{R}(p^*)] \cdot \\
&\quad \cdot \operatorname{E}_{\mathcal{R}}[|S \cap \mathcal{R}(q)|^{-\rho} \mid \mathcal{R}(q) = \mathcal{R}(p^*)] \\
&= \Pr_{\mathcal{R}}[\mathcal{R}(q) = \mathcal{R}(p^*)] \cdot \\
&\quad \cdot \operatorname{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right] \cdot \\
&\quad \cdot |S|^{-\rho} \\
&\ge |S|^{-\rho},
\end{aligned}$$

where the fourth step is by the induction assumption (since we condition on $p^* \in S \cap \mathcal{R}(q)$) and the last step is by the lemma assumption. Technically, if $S \cap \mathcal{R}(q) = S$, then we can not quite use the induction assumption, but nevertheless, the required bound follows from the immediate claim "If for $0 < p < 1$ one has $x = px + y$, then $x \ge z$ iff $pz + y \ge z$".

Note that one tree requires $O(nk)$ space: there are at most $O(n)$ nodes in the tree and each stores at most $k + 1$ nodes. The query time depends on the height of tree: for a maximum height h, the query time per tree is $O(h \cdot dk)$. We note that, in our applications, the height will be easy to bound.

4 LSH Forest and the Bit Sampling LSH

In this section we present our main algorithm claimed informally in Theorem 1.1. The algorithm is an instantiation of Algorithm 1 for the Hamming space. In particular, we use the Bit Sampling LSH from Definition 2.4 as the random partition $\mathcal{R}$ as well as a certain procedure for computing pivots, described below.

The formal statement of our main theorem is the following:

THEOREM 4.1. *Fix $r \in \mathbb{N}, k \in \mathbb{N} \geq 2$, and $c > 2$. Then for given $P \subseteq \{0,1\}^d$ of size n, the LSH Forest with k pivots solves (c,r)-ANN with the following guarantees:*

- *The space is $O(n^{1+\rho}d + nd)$;*
- *The worst-case query time is $O(n^\rho \cdot d^2)$;*
- *where*
$$\rho = \frac{1}{\ln 4} \cdot \frac{1}{c} \cdot \left(1 + O\left(\frac{1}{k}\right)\right) + O\left(\frac{1}{c^2}\right).$$

We first show how to prove Theorem 4.1 for the case of (c,r)-Gap ANN (see Definition 2.2). In this case, it is enough to simply choose k pivots as described below. To handle the general case of (c,r)-ANN, we add $\Theta(\log n)$ *uniformly random pivots* to the existing set of pivots. It is not hard to show that if there are $\Omega(n)$ data points closer than cr to the query, then at least one them will end up in a random subset with very high probability, and, in that case, we are done. Otherwise, there are $o(n)$ data points within distance cr, and the analysis for the Gap ANN goes through: namely, the close points do not affect the quantity (3.1) too much. From now on, we focus on solving the (c,r)-Gap ANN problem.

4.1 Construction The algorithm for choosing k pivots is very simple: see Algorithm 2 for the pseudocode. In words, for a subset $S \subseteq P$ of the dataset, we sort all the points according to the distance from the *mean* of S, then we add first k points in this order which are *sufficiently diverse*, namely, we require them to be pairwise $\big((c-1)r\big)$-separated.

4.1.1 Intuition Let us provide some intuition why this procedure helps.

As Lemma 3.1 states, we need to find the smallest ρ such that

$$\Pr_{\mathcal{R}}[\mathcal{R}(q) = \mathcal{R}(p^*)] \cdot$$
$$\cdot \mathrm{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right] \geq 1.$$

The first term is easy: $\Pr_{\mathcal{R}}[\mathcal{R}(q) = \mathcal{R}(p^*)] \geq 1 - \frac{r}{d}$ (remember that $\mathcal{R}$ is the Bit Sampling LSH). For the sake of this discussion, let us ignore the effect of conditioning on the event "$\mathcal{R}(q) = \mathcal{R}(p^*)$" (as we will see later, the effect is provably small provided that c is large enough). Thus, we need to understand the following quantity:

$$\mathrm{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho}\right].$$

For $i \in [d]$, we denote w_i the fraction of points $p \in S$ for which $p_i \neq q_i$. Then,

$$\mathrm{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho}\right] = \mathrm{E}_{i \in [d]}\left[(1 - w_i)^{-\rho}\right].$$

Thus, we need to find the smallest ρ for which

$$\left(1 - \frac{r}{d}\right) \mathrm{E}_{i \in [d]}\left[(1 - w_i)^{-\rho}\right] \geq 1.$$

A naïve argument using Jensen's inequality gives:

$$\left(1 - \frac{r}{d}\right) \mathrm{E}_{i \in [d]}\left[(1 - w_i)^{-\rho}\right] \geq \left(1 - \frac{r}{d}\right) \mathrm{E}_{i \in [d]}\left[1 - w_i\right]^{-\rho}$$
$$\gtrsim \left(1 - \frac{r}{d}\right)\left(1 - \frac{cr}{d}\right)^{-\rho},$$

where the second step follows from the fact that all except one points from S are at distance more than cr from the query point q. This merely allows us to set $\rho \approx \frac{1}{c}$, so we have not obtained any improvement compared to the classical application of the Bit Sampling LSH using Theorem 2.1.

Is the above simple analysis tight? The application of Jensen's inequality is (almost) tight if all w_i's are (almost) the same (around $\frac{cr}{d}$). This is indeed possible, moreover this is precisely a hard example from [OWZ14]. That is exactly where pivots come to the rescue! By using the fact that the distance from q to all the pivots computed by Algorithm 2 is more than cr, we are able to show that w_i *can not be* almost the same, and thus the application of Jensen can be sharpened.

For example, consider the case of $k = 1$ pivot. W.l.o.g. we can assume that $p^* = 0^d$. If p^* was a pivot, we would be done, so w.l.o.g. we can assume that the pivot is $\widetilde{p} \neq p^*$. Since $\|\widetilde{p} - p^*\|_1 > (c-1)r \approx cr$ (q is at least cr apart from $\widetilde{p}$), we can assume w.l.o.g. that $\widetilde{p} = 1^{cr}0^{d-cr}$. Now comes the crucial insight: since $\widetilde{p}$ is closer to the mean of S than p^*, we have that, at least on the first cr coordinates, weights w_i must be at least $1/2$ on average, which is much larger than the average $\frac{cr}{d}$. This allows us to claim that the ℓ_1-mass of w_i's is somewhat concentrated on the first

cr coordinates, and hence the application of Jensen can be significantly improved.

Introducing more pivots allows us to obtain *even better* concentration of the weights w_i. More precisely, for k pivots we can get as many as $\bigl(2 - O(1/k)\bigr)cr$ bits of substantial average weight. Thus, if k is large, almost all the ℓ_1-mass of w_i's is concentrated on $\approx 2cr$ coordinates, which corresponds to a *random instance*, for which the Bit Sampling LSH yields $\rho \approx \frac{0.73}{c}$. We obtain an increased number of large weights by noticing that we have k pivots which are at distance $\approx cr$ from each other, thus there must be a large number of coordinates where at least one pivot is equal to one.

Algorithm 2 Computing pivots for the Hamming space

```
function ComputePivots($v$, $k$)    ▷ $v$ is a tree node, $k$ is the desired number of pivots
    $S \leftarrow$ data points corresponding to $v$
    $T_v \leftarrow \emptyset$
    $p^c \leftarrow$ the mean of $S$
    for $p \in S$ in the order of increasing $\ell_1$ distance from $p^c$ do
        if the $\ell_1$-distance between $p$ to $T_v$ is at least $(c-1)r$ then
            $T_v \leftarrow T_v \cup \{p\}$
            if $|T_v| = k$ then
                return $T_v$
    return $T_v$
```

4.2 Analysis To prove Theorem 4.1, we will use the lemma for the generic LSH Forest, Lemma 3.1. Let us fix some query q and assume that $p^* \in P$ is its near neighbor ($\|q - p^*\|_1 \le r$). We need to establish (3.1) assuming that q is far from all the pivots.

Let us first show the following combinatorial lemmas which will be useful later.

Lemma 4.1. *Fix $s, n \ge 1$, $k \ge 2$ and small $\epsilon > 0$. For $i \in [n]$, let $a_i \in [1, k]$ and $x_i \in [0, 1]$. Suppose we have that:*

- $\sum_i a_i x_i \ge \frac{1}{2} \sum_i a_i (1 - \epsilon)$, *and*
- $\sum a_i(k - a_i) \ge \frac{k(k-1)\cdot s}{2}$.

Then we have that:

$$\prod_i (1 - x_i) \le \exp\Bigl(s \cdot (1 - 1/k) \cdot (1 - \ln(4)) - \sum_i x_i + O(\epsilon s)\Bigr).$$

Proof appears in Appendix A

Lemma 4.2. *Let $x \in [0,1]^d, s \in \mathbb{N}$, and let $\mathcal{U}$ be a family of k subsets of coordinates $[d]$ such that:*

- *For each $U \in \mathcal{U}$, $|U| \ge s$*
- *For each $U \in \mathcal{U}$ one has $\sum_{i \in U} x_i \ge \frac{|U|}{2} \cdot \bigl(1 - O(\frac{1}{c})\bigr)$*
- *For every distinct $U_1, U_2 \in \mathcal{U}$ one has $|U_1 \triangle U_2| \ge s$.*
- $\sum_{i \in [d]} x_i \ge s$.

Then:

$$\prod_i (1 - x_i) \le \exp\left(-s \cdot \frac{\ln(4)k + 1}{k+1} + O(s/c)\right)$$

.

Proof. For $i \in [d]$, let $a_i \in \{0, 1, .., k\}$ denote the number of subsets in $\mathcal{U}$ containing coordinate i. The above constraints imply that:

- $\sum_i a_i \ge ks$.
- $\sum_i a_i x_i \ge \frac{1}{2} \sum_i a_i \cdot \bigl(1 - O(\frac{1}{c})\bigr)$,
- $\sum_i a_i(k - a_i) \ge \frac{k(k-1)s}{2}$.

The last constraint follows from the double counting for the sum $\sum_{\substack{U,U' \in \mathcal{U} \\ U \ne U'}} |U \triangle U'|$, which is, on the one hand, is at least $\binom{k}{2}s$ by the statement of the lemma, and, on the other hand is equal to $\sum_i a_i(k - a_i)$, which follows from the decomposition of the sum over coordinates.

Summing up the first and last constraints gives:

$$\sum_i a_i(k + 1 - a_i) \ge \frac{k(k+1)s}{2}.$$

Let $S = \bigcup_{U \in \mathcal{U}} U$. We decompose objective function as follows:

$$\prod_{i \in [d]} (1 - x_i) = \prod_{i \in S} (1 - x_i) \cdot \prod_{i \in [d] \setminus S} (1 - x_i)$$

We now invoke Lemma 4.1 over the first part:

$$\prod_{i \in S} (1 - x_i) \le \exp\Bigl(s \cdot (1 - 1/(k+1)) \cdot (1 - \ln(4)) - \sum_{i \in S} x_i + O(s/c)\Bigr).$$

The second part can be simplified to:

$$\prod_{i\in[d]\setminus S}(1-x_i) \le \prod_{i\in[d]\setminus S} e^{-x_i} \le \exp\left(\sum_{i\in S}(x_i) - s\right).$$

The last inequality follows the condition that $\sum_{i\in[d]} x_i \ge s$. Finally, we multiply the 2 parts together and get:

$$\begin{aligned}\prod_{i\in[d]}(1-x_i) &\le \exp\left(s\cdot(1-1/(k+1))\cdot(1-\ln(4))\right.\\ &\quad \left. - s + O(s/c)\right)\\ &\le \exp\left(-s\cdot\left(\ln(4)\cdot(1-\frac{1}{k+1}) + \right.\right.\\ &\quad \left.\left.\frac{1}{k+1}\right) + O(s/c)\right)\\ &= \exp\left(-s\cdot\frac{\ln(4)k+1}{k+1} + O(s/c)\right)\end{aligned}$$

as needed.

We use the above lemmas to calculate the probability of success for each hash function as follows:

Lemma 4.3. *Let $P \subset \{0,1\}^d$ be a set of n points, $q \in \{0,1\}^d$ be a selected point and p^* be the near neighbor. The probability to find $p \in P$ such that $\|p-q\|_1 \le cr$ in a tree is at least $n^{-\rho}$, where $\rho \le \frac{1}{\ln(4)\cdot(c-2)}\cdot(1+\frac{2}{7k+5}) + O(\frac{1}{c^2})$.*

Proof. As we have shown earlier in Lemma 3.1, all we need to prove is that for any $S \subseteq P$ with $p^* \in S$ such that q is further than cr from all the pivots, one has:

$$\Pr_{\mathcal{R}}[\mathcal{R}(q)=\mathcal{R}(p^*)]\cdot$$
$$\cdot\mathrm{E}_{\mathcal{R}}\left[\left(\frac{|S\cap\mathcal{R}(q)|}{|S|}\right)^{-\rho}\middle|\,\mathcal{R}(q)=\mathcal{R}(p^*)\right] \ge 1.$$

For $i \in [d]$, let w_i denote the fraction of $p \in S$ for which $p_i \ne p^*_i$, and let p^c denote the mean of S. Since we partition using the Bit Sampling LSH, it is easy to see that:

$$\frac{|S\cap\mathcal{R}_i(p^*)|}{|S|} = 1 - w_i,$$

where $\mathcal{R}_i$ is the partition corresponding to splitting the space according to the bit i.

If there exists a close enough pivot point, we are done. Otherwise, there exists a set of k points $\{p^1,..,p^k\} \subseteq S$ s.t. :

- For each $i \in [k]$: $\|p^i - p^c\|_1 \le \|p^* - p^c\|_1$;
- For every distinct $i,j \in [k]$: $\|p^i - p^j\|_1 \ge (c-1)r$.

For each $i \in [k]$, let $U_i \subseteq [d]$ denote the set of coordinates in which p^i differ from p^*. Since the distance of both p^*, p^i and p^c are equal on all coordinates outside U_i, and since for each $j \in U_i$, $p^*_j - p^c_j = w_j$ and $p^i_j - p^c_j = 1 - w_j$, then the first constraint gives:

- For each U_i: $\sum_{j\in U_i} w_j \ge \sum_{j\in U_i} 1 - w_j \Rightarrow \sum_{j\in U_i} w_j \ge |U_i|/2$;

The second constraint gives:

- For every distinct U_i, U_j : $|U_i \triangle U_j| \ge (c-1)r$.

While the above conditions are enough to invoke Lemma 4.2, we need to condition on the fact that q and p^* collide. Let $D \subseteq [d]$ denote the set of coordinates for which $p^*_i = q_i$, we deduce similar properties when restricted to the set D:

- For each U_i: $|U_i \cap D| \ge (c-1)r$ since $\|p^i - q\|_1 \ge cr$;
- For each U_i: $\sum_{j\in U_i\cap D} w_j \ge \frac{|U_i\cap D|-r}{2} = \frac{|U_i\cap D|}{2}\cdot\left(1-O(\frac{1}{c})\right)$ since $\|p^* - q\|_1 \le r$ and hence $|U_i \setminus D| \le r$;
- For every distinct U_i, U_j : $|(U_i \triangle U_j)\cap D| \ge (c-1)r - r = (c-2)r$.

We can finally apply Lemma 4.2 with vector $\vec{x} = \vec{w}$ over bits in set D and $s = (c-2)r$, obtaining that:

$$\prod_{i\in D}(1-w_i) \le \exp\left(-(c-2)r\cdot\frac{\ln(4)k+1}{k+1} + O(r)\right)$$

Since we are sampling uniformly then:

$$\begin{aligned}&\mathrm{E}_{\mathcal{R}}\left[\left(\frac{|S\cap\mathcal{R}(q)|}{|S|}\right)^{-\rho}\middle|\,\mathcal{R}(q)=\mathcal{R}(p^*)\right]\\ &= \mathrm{E}_{\mathcal{R}}\left[\left(\frac{|S\cap\mathcal{R}(p^*)|}{|S|}\right)^{-\rho}\middle|\,\mathcal{R}(q)=\mathcal{R}(p^*)\right]\\ &= \mathrm{E}_{i\in D}\left[(1-w_i)^{-\rho}\right]\\ &\ge \left(\prod_{i\in D}(1-w_i)^{-\rho}\right)^{1/|D|}\\ &= \left(\prod_{i\in D}(1-w_i)\right)^{-\rho/|D|}\\ &\ge \exp\left(\frac{(c-2)r\cdot\rho\cdot(\ln(4)k+1)}{|D|\cdot(k+1)} - O(\frac{\rho\cdot r}{|D|})\right)\end{aligned}$$

where the third step is by the AM–GM inequality. Thus we can set:

$$\rho = \frac{k+1}{(\ln(4)k+1)\cdot(c-2)} + O(\tfrac{1}{c^2}) < \frac{1}{\ln(4)\cdot(c-2)} \cdot (1 + \tfrac{2}{7k+5}) + O(\tfrac{1}{c^2})$$

(as $1/\ln(4) > 5/7$). For which we obtain:

$$\mathrm{E}_{\mathcal{R}}\left[\left(\frac{|S \cap \mathcal{R}(q)|}{|S|}\right)^{-\rho} \middle| \mathcal{R}(q) = \mathcal{R}(p^*)\right] \geq e^{r/|D|}.$$

The probability that p^* and q collide is:

$$\Pr_{\mathcal{R}}[\mathcal{R}(q) = \mathcal{R}(p^*)] \geq \frac{|D|}{|D| + r} \geq e^{-r/|D|}.$$

We have thus established that (3.1).

We are now ready to complete the proof to our main theorem:

Proof. [Proof of Theorem 4.1] The proof follows immediately from Lemma 4.3. In particular, we prepare $O(n^\rho)$ trees using the generic LSH Forest algorithm, Algorithm 1 with the pivot set, of size k, chosen as in Algorithm 2.

Acknowledgments

We thank Sariel Har-Peled for asking whether the LSH Forest can reuse randomness in different parts of the tree, which ultimately led to the algorithm based on a trie with a random permutation of the coordinates. We also thank Erik Waingarten for useful discussions.

Work is supported in part by NSF CCF-1617955 and Google Research Award.

References

[AAKK14] Amirali Abdullah, Alexandr Andoni, Ravindran Kannan, and Robert Krauthgamer. Spectral approaches to nearest neighbor search. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2014)*, pages 581–590, 2014.

[ACW16] Josh Alman, Timothy M. Chan, and Ryan Williams. Polynomial representations of threshold functions with applications. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2016)*, 2016. To appear.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '2006)*, pages 459–468, 2006.

[AI08] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[AIL+15] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In *Proceedings of Advances in Neural Information Processing Systems 28 (NIPS '2015)*, pages 1225–1233, 2015. Available as arXiv:1509.02897.

[AINR14] Alexandr Andoni, Piotr Indyk, Huy L. Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the 25th ACM-SIAM Symposium on Discrete Algorithms (SODA '2014)*, pages 1018–1028, 2014. Available as arXiv:1306.1547.

[ALRW16] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Lower bounds on time–space trade-offs for approximate near neighbors. Available as arXiv:1605.02701, 2016.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the 47th ACM Symposium on the Theory of Computing (STOC '2015)*, pages 793–801, 2015. Available as arXiv:1501.01062.

[AR16] Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG '2016)*, pages 9:1–9:11, 2016. Available as arXiv:1507.04299.

[BCG05] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web (WWW '2005)*, pages 651–660, 2005.

[Ber13] Erik Bernhardsson. ANNOY: Approximate nearest neighbors in C++/Python optimized for memory usage and loading/saving to disk, 2013. Available as `https://github.com/spotify/annoy`.

[BGMZ97] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8–13):1157–1166, 1997.

[BKL06] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning (ICML '2006)*, pages 97–104, 2006.

[Bro97] Andrei Z. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences (SEQUENCES '1997)*, pages 21–29, 1997.

[Cla88] Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

[Cla06] Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006.

[CNBM01] Edgar Chávez, Gonzalo Navarro, Ricardo A. Baeza-Yates, and José L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[Cou13] National Research Council. *Frontiers in Massive Data Analysis*. The National Academies Press, Washington, DC, 2013.

[DF08] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the 40th ACM Symposium on the Theory of Computing (STOC '2008)*, pages 537–546, 2008.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry (SoCG '2004)*, pages 253–262, 2004.

[DS13] Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for exact nearest neighbor search. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT '2013)*, pages 317–337, 2013.

[HIM12] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8(1):321–350, 2012.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th ACM Symposium on the Theory of Computing (STOC '1998)*, pages 604–613, 1998.

[IN07] Piotr Indyk and Assaf Naor. Nearest-neighbor-preserving embeddings. *ACM Transactions on Algorithms*, 3(3), 2007.

[KKK16] Matti Karppa, Petteri Kaski, and Jukka Kohonen. A faster subquadratic algorithm for finding outlier correlations. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms (SODA '2016)*, pages 1288–1305, 2016.

[KKKÓ16] Matti Karppa, Petteri Kaski, Jukka Kohonen, and Padraig Ó Catháin. Explicit correlation amplifiers for finding outlier correlations in deterministic subquadratic time. In *Proceedings of the 24th European Symposium Of Algorithms (ESA '2016)*, 2016. To appear.

[KL04] Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA '2004)*, pages 798–807, 2004.

[KR02] David R. Karger and Matthias Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the 34th ACM Symposium on the Theory of Computing (STOC '2002)*, pages 741–750, 2002.

[McN01] James McNames. A fast nearest-neighbor algorithm based on a principal axis search tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):964–976, 2001.

[Mei93] Stefan Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

[ML09] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of the 4th International Conference on Computer Vision Theory and Applications (VISAPP '2009)*, pages 331–340, 2009.

[MNP07] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *SIAM Journal on Discrete Mathematics*, 21(4):930–935, 2007.

[OWZ14] Ryan O'Donnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory*, 6(1):5, 2014.

[SH08] Chanop Silpa-Anan and Richard Hartley. Optimised KD-trees for fast image descriptor matching. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '2008)*, pages 1–8, 2008.

[SH09] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[Spr91] Robert F. Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6(1–6):579–589, 1991.

[Val15] Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *Journal of the ACM*, 62(2):13, 2015.

[VKS09] Nakul Verma, Samory Kpotufe, and Sanjoy. Which spatial partition trees are adaptive to intrinsic dimension? In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI '2009)*, 2009.

[WLKC15] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. Available as arXiv:1509.05472, 2015.

[WSSJ14] Jingdong Wang, Heng Tao Shen, Kingkuan Song, and Jianqiu Ji. Hashing for similarity search: a survey. Available as arXiv:1408.2927, 2014.

[WTF08] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *Proceedings of Advances in Neural Information Processing Systems 21 (NIPS '2008)*, pages 1753–1760, 2008.

[YSRL11] Jay Yagnik, Dennis Strelow, David Ross, and Ruei-Sung Lin. The power of comparative reasoning. In *Proceedings of the 13th International Conference on Computer Vision (ICCV '2011)*, 2011.

A Proof of Lemma 4.1

We prove the following lemma.

LEMMA A.1. *Fix $s, n \geq 1$, $k \geq 2$ and small $\epsilon > 0$. For $i \in [n]$, let $a_i \in [1, k]$ and $x_i \in [0, 1]$. Suppose we have that:*

- $\sum_i a_i x_i \geq \frac{1}{2} \sum_i a_i (1 - \epsilon)$, *and*

- $\sum a_i(k-a_i) \geq \frac{k(k-1)\cdot s}{2}$.

Then we have that:

$$\prod_i (1-x_i) \leq \exp\left(s(1-\tfrac{1}{k})(1-\ln 4) - \sum_i x_i + O(\epsilon s)\right).$$

Proof. First we make a change of variables: $y_i = 1 - x_i \in [0,1], t = \frac{1+\epsilon}{2}$ and $f_i = a_i/k \in [1/k, 1]$.

Now our problem can be reformulated as maximizing $F(y) = \prod_{i\in[n]} (y_i \cdot \exp(1-y_i))$, subject to the following constraints:

- $\sum_i f_i(y_i - t) \leq 0$;
- $\sum f_i(1-f_i) \geq \frac{(1-1/k)\cdot s}{2}$.

Note that, there is an optimum solution where both are in fact equalities. In particular, to see this for the second part, note that for i such that $y_i < t$, we can assume $f_i > 1/2$ (otherwise take $f_i := 1 - f_i$) and increase f_i. Similarly, for $y_i > t$, we can assume $f_i < 1/2$ and we decrease it further.

We now analyze the Lagrangian of the logarithm of F. In particular for Lagrangian multipliers $\lambda, \eta \in \Re$, taking the derivatives with respect to y_i gives us that for all i:

$$1/y_i - 1 = \eta f_i, \tag{A.1}$$

and with respect f_i, for all i:

$$0 = \lambda(1-2f_i) + \eta(y_i - t). \tag{A.2}$$

Plugging-in one equation into the other, we get an equation which is polynomial of degree 2 in y_i (or, alternatively in f_i), and coefficients that depend on the Lagrange multipliers λ, η. Hence, the optimal solution can have only at most 2 distinct values for (y_i, f_i). Let these values be (y_1, f_1) and (y_2, f_2).

Assume w.l.o.g $y_1 \leq y_2$, and suppose there are $\alpha_1 s(1-1/k)$ pairs (y_1, f_1) and $\alpha_2 s(1-1/k)$ pairs (y_2, f_2), for $\alpha_1, \alpha_2 \geq 0$. Then, the first equation becomes:

$$\alpha_1 f_1(t - y_1) = \alpha_2 f_2(y_2 - t) \tag{A.3}$$

The second equation becomes $\alpha_1 s(1-1/k)\cdot f_1(1-f_1) + \alpha_2 s(1-1/k)\cdot f_2(1-f_2) = \frac{(1-1/k)s}{2}$, which simplifies to:

$$\alpha_1 f_1(1-f_1) + \alpha_2 \cdot f_2(1-f_2) = \tfrac{1}{2}, \tag{A.4}$$

The objective $\log F$ can be rewritten as maximizing:

$$[\alpha_1(\ln(y_1) + 1 - y_1) + \alpha_2(\ln(y_2) + 1 - y_2)]\,(1-1/k)s$$

It is enough to maximize $G = \alpha_1(\ln(y_1) + 1 - y_1) + \alpha_2(\ln(y_2) + 1 - y_2)$, subject to the above constraints.

Consider the Langrangian L_G of G, its derivative with respect to each α_i gives:

$$\frac{\partial L_G}{\partial \alpha_i} = \ln(y_i) + 1 - y_i - \eta f_i(y_i - t) - \lambda f_i(1-f_i)$$

Note that from Eqn. (A.1) and (A.2) (which hold true when considering L_G as well):

$$\begin{aligned} \eta f_i &= 1/y_i - 1 \\ \lambda f_i &= \frac{\eta f_i(y_i - t)}{(2f_i - 1)} \\ &= \frac{(1/y_i - 1)(y_i - t)}{(2f_i - 1)} \end{aligned}$$

Therefore, we have:

$$\begin{aligned} \frac{\partial L_G}{\partial \alpha_i} &= \ln(y_i) + 1 - y_i - (\tfrac{1}{y_i} - 1)(y_i - t) - \frac{(\frac{1}{y_i} - 1)(y_i - t)(1-f_i)}{(2f_i - 1)} \\ &= \ln(y_i) + 1 - y_i + \left(\frac{f_i - 1}{2f_i - 1} - 1\right)(1/y_i - 1)(y_i - t) \\ &= \ln(y_i) + 1 - y_i - \frac{f_i(1/y_i - 1)(y_i - t)}{2f_i - 1} \\ &= \ln(y_i) + 1 - y_i + \frac{\lambda}{\eta^2}(y_i^{-1} - 1)^2 \end{aligned}$$

The last step follows from exchanging variables according to Eqn. (A.1) and (A.2).
At the optimum of G, for each α_i, either $\frac{\partial L_G}{\partial \alpha_i} = 0$ or α_i cannot be decreased or increased which can only happen at the boundaries, i.e., where either $y_1 = 0$ or $t, y_2 = 1$ or t (otherwise we can always perturb y_i and increase/decrease α_i).
One can observe that $\frac{\partial L_G}{\partial \alpha_i} = 0$ where $y_i = 1$. We can further analyze $\frac{\partial L_G}{\partial \alpha_i}$ by taking a second derivative with respect to y_i:

$$\begin{aligned} \frac{\partial^2 L_G}{\partial \alpha_i \partial y_i} &= y_i^{-1} - 1 - 2\lambda\eta^{-2}y^{-2}(y^{-1} - 1) \\ &= (1 - 2\lambda\eta^{-2}y^{-2})(y^{-1} - 1) \end{aligned}$$

Since there can be at most 1 assignment of $0 < y_i < 1$ for which $\frac{\partial^2 L_G}{\partial \alpha_i \partial y_i} = 0$, and since $\frac{\partial L_G}{\partial \alpha_i}$ is monotonic elsewhere in the range and equals 0 where $y_i = 1$, then there can be at most 1 assignment of $0 < y_i < 1$ for which $\frac{\partial L_G}{\partial \alpha_i} = 0$, which means that any optimal solution we cannot have distinct y_1, y_2 which both lie in the interior, and therefore we must have at least of the following: $y_1 = 0, y_1 = t, y_2 = 1$ or $y_2 = t$.

Now, let us analyze the above cases separately:

- If $y_1 = 0$, the function is at minimum.
- If either $y_i = t$, then according to Eqn. (A.3) the other y_j must also be t (or otherwise $f_j = 0$ for which in optimum $\alpha_j = 0$).
- If $y_2 = 1$, then according to Eqn. (A.1): $1 = \eta f_2 + 1$ and therefore $f_2 = 0$, and according to Eqn. (A.3): $y_1 = t$ (and since $f_2 = 0$, α_2 is irrelevant).

In either of the last 2 scenarios, it is enough to optimize:

$$F(\alpha) = (t^{\alpha} e^{(1-t)\alpha})^{(1-1/k)s} = \exp\left(\alpha \cdot s \cdot (1 - 1/k) \cdot (1 - t - \ln(1/t))\right)$$

subject to:

- $\alpha f(1-f) = \frac{1}{2}$.

The above is optimized at $f = 1/2; \alpha = 2$, and hence the optimal value of F is:

$$F^* = \exp\left(s \cdot (1 - 1/k) \cdot 2(1 - t - \ln(1/t))\right)$$

Therefore, the required inequality holds:

$$\prod_i (1-x_i) \leq \exp\left(s \cdot (1 - 1/k) \cdot (1 - \ln(4)) - \sum_i x_i + O(\epsilon s)\right).$$