58th Annual IEEE Symposium on Foundations of Computer Science

Optimal Las Vegas Locality Sensitive Data Structures

IT University of Copenhagen

Thomas D. Ahle

***Abstract*—We show that approximate similarity (near neighbour) search can be solved in high dimensions with performance matching state of the art (data independent) Locality Sensitive Hashing, but with a guarantee of no false negatives. Specifically we give two data structures for common problems. For c-approximate near neighbour in Hamming space we get query time $dn^{1/c+o(1)}$ and space $dn^{1+1/c+o(1)}$ matching that of [Indyk and Motwani, 1998] and answering a long standing open question from [Indyk, 2000a] and [Pagh, 2016] in the affirmative. For (s_1, s_2)-approximate Jaccard similarity we get query time $d^2 n^{\rho+o(1)}$ and space $d^2 n^{1+\rho+o(1)}$, $\rho = \log \frac{1+s_1}{2s_1} \big/ \log \frac{1+s_2}{2s_2}$, when sets have equal size, matching the performance of [Pagh and Christiani, 2017].**

The algorithms are based on space partitions, as with classic LSH, but we construct these using a combination of brute force, tensoring and splitter functions à la [Naor et al., 1995]. We also show two dimensionality reduction lemmas with 1-sided error.

I. Introduction

Locality Sensitive Hashing has been a leading approach to high dimensional similarity search (nearest neighbour search) data structures for the last twenty years. Intense research [Indyk and Motwani, 1998, Gionis et al., 1999, Kushilevitz et al., 2000, Indyk, 2000b, Indyk, 2001, Charikar, 2002, Datar et al., 2004, Lv et al., 2007, Panigrahy, 2006, Andoni and Indyk, 2006, Andoni et al., 2014, Andoni et al., 2017a, Becker et al., 2016, Ahle et al., 2017, Aumüller et al., 2017] has applied the concept of space partitioning to many different problems and similarity spaces. These data structures are popular in particular because of their ability to overcome the 'curse of dimensionality' and conditional lower bounds by [Williams, 2005], and give sub-linear query time on worst case instances. They achieve this by being approximate and Monte Carlo, meaning they may return a point that is slightly further away than the nearest, and with a small probability they may completely fail to return any nearby point.

Definition 1 ((c, r)-Approximate Near Neighbour)**.** *Given a set P of n data points in a metric space (X, dist), build a data structure, such that given any $q \in X$, for which there is an $x \in P$ with $\mathrm{dist}(q, x) \le r$, we return a $x' \in P$ with $\mathrm{dist}(q, x') \le cr$.*

A classic problem in high dimensional geometry has been whether data structures existed for (c, r)-Approximate Near Neighbour with Las Vegas guarantees, and performance matching that of Locality Sensitive Hashing. That is, whether we could guarantee that a query will always return an approximate near neighbour, if a near neighbour exists; or simply, if we could rule out false negatives? The problem has seen practical importance as well as theoretical. There is in general no way of verifying that an LSH algorithm is correct when it says 'no near neighbours' - other than iterating over every point in the set, in which case the data structure is entirely pointless. This means LSH algorithms can't be used for many critical applications, such as finger print data bases. Even more applied, it has been observed that tuning the error probability parameter is hard to do well, when implementing LSH [Gionis et al., 1999, Arya et al., 1998]. A Las Vegas data structure entirely removes this problem. Different authors have described the problem with different names, such as 'Las Vegas' [Indyk, 2000a], 'Have no false negatives' [Goswami et al., 2017, Pagh, 2016], 'Have total recall' [Pham and Pagh, 2016], 'Are exact' [Arasu et al., 2006] and 'Are explicit' [Karppa et al., 2016].

Recent years have shown serious progress towards finally solving the problem. In particular [Pagh, 2016] showed that the problem in Hamming space admits a Las Vegas algorithm with query time $dn^{1.38/c+o(1)}$, matching the $dn^{1/c}$ data structure of [Indyk and Motwani, 1998] up to a constant factor in the exponent. In this paper we give an algorithm in the Locality Sensitive Filter framework [Becker et al., 2016, Christiani, 2017], which not only removes the factor 1.38, but improves to $dn^{1/(2c-1)+o(1)}$ in the case $cr \approx d/2$, matching the algorithms of [Andoni et al., 2015] for Hamming space.

We would like to find an approach to Las Vegas LSH that generalizes to the many different situations where LSH is useful. Towards that goal, we present as second algorithm for the approximate similarity search problem under Braun-Blanquet similarity, which is defined for sets $x, y \subseteq [d]$ as $\mathrm{sim}(x, y) = |x \cap y| / \max(|x|, |y|)$. We refer to the following problem definition:

Definition 2 (Approximate similarity search). *Let $P \subseteq \mathcal{P}([d])$ be a set of $|P| = n$ subsets of $[d]$; (here $\mathcal{P}(X)$ denotes the powerset of X.) let $\mathrm{sim} : \mathcal{P}([d]) \times \mathcal{P}([d]) \to [0,1]$ be a similarity measure. For given $s_1, s_2 \in [0,1]$, $s_1 > s_2$, a solution to the "(s_1, s_2)-similarity search problem under sim" is a data structure that supports the following query operation: on input $q \subseteq [d]$, for which there exists a set $x \in P$ with $\mathrm{sim}(x,q) \geq s_1$, return $x' \in P$ with $\mathrm{sim}(x',q) > s_2$.*

The problem has traditionally been solved using the Min-Hash LSH [Broder et al., 1997, Broder, 1997], which combined with the results of Indyk and Motwani [Indyk and Motwani, 1998] gives a data structure with query time dn^ρ and space $dn^{1+\rho}$ for $\rho = \log s_1 / \log s_2$. Recently it was shown by [Pagh and Christiani, 2017] that this could be improved for vectors of equal weight to $\rho = \log \frac{2s_1}{1+s_1} / \log \frac{2s_2}{1+s_2}$. We show that it is possible to achieve this recent result with a data structure that has no false negatives.

A. Summary of Contributions

We present the first Las Vegas algorithm for approximate near neighbour search, which gives sub-linear query time for any approximation factor $c > 1$. This solves a long standing open question from [Indyk, 2000a] and [Pagh, 2016]. In particular we get the following two theorems:

Theorem 1. *Let $X = \{0,1\}^d$ be the Hamming space with metric $\mathrm{dist}(x,y) = \|x \oplus y\| \in [0,d]$ where $\oplus$ is "xor" or addition in $\mathbb{Z}_2$. For every choice of $0 < r$, $1 < c$ and $cr \leq d/2$, we can solve the (c,r)-approximate near neighbour problem in Hamming space with query time dn^ρ and space usage $dn^{1+\rho}$ where $\rho = 1/c + \hat{O}((\log n)^{-1/4})$.*

Note: $\hat{O}$ hides $\log\log n$ factors.

Corollary 1. *When $r/d = \Omega((\log n)^{-1/6})$, we get the improved exponent $\rho = \frac{1-cr/d}{c(1-r/d)} + \hat{O}((\log n)^{-1/3} d/r)$.*

This improves upon theorem 1 when r/d is constant (or slightly sub-constant), including in the important "random case", when $r/d = 1/(2c)$ where we get $\rho = 1/(2c-1) + o(1)$.

Theorem 2. *Let sim be the Braun-Blanquet similarity $\mathrm{sim}(x,y) = |x \cap y| / \max(|x|,|y|)$. For every choice of constants $0 < s_2 < s_1 < 1$, we can solve the (s_1,s_2)-similarity problem over sim with query time $d^2 n^\rho$ and space usage $d^2 n^{1+\rho}$ where $\rho = \log s_1 / \log s_2 + \hat{O}((\log n)^{-1/3}(s_1/s_2)^{1/3})$.*

The first result matches the lower bounds by [O'Donnell et al., 2014] for "data independent" LSH data structures for Hamming distance and improves upon [Pagh, 2016] by a factor of $\log 4 > 1.38$ in the exponent. By deterministic reductions from ℓ_2 to ℓ_1 [Indyk, 2007] and ℓ_1 to hamming, this also gives the best currently known Las Vegas data structures for ℓ_1 and ℓ_2 in $\mathbb{R}^d$. The second result matches the corresponding lower bounds by [Pagh and Christiani, 2017] for Braun-Blanquet similarity and, by reduction, Jaccard similarity. See table I for more comparisons.

Detaching the data structures from our constructions, we get the first explicit constructions of large Turán Systems [Sidorenko, 1995]. Our systems can even be efficiently decoded, which is likely to have other algorithmic applications.

B. Background and Related Work

The arguably most successful technique for similarity search in high dimensions is Locality-Sensitive Hashing (LSH), introduced in 1998 by [Indyk and Motwani, 1998, Har-Peled et al., 2012]. The idea is to make a random space partition in which similar points are likely to be stored in the same region, thus allowing the search space to be pruned substantially. The granularity of the space partition (the size/number of regions) is chosen to balance the expected number of points searched against keeping a (reasonably) small probability of pruning away the actual nearest point. To ensure a high probability of success (good recall) one repeats the above construction, independently at random, a small polynomial (in n) number of times.

In [Pagh, 2016, Arasu et al., 2006] it was shown that one could change the above algorithm to not do the repetitions independently. (Eliminating the error probability of an algorithm by independent repetitions, of course, takes an infinite number of repetitions.) By making correlated repetitions, it was shown possible to reach zero false negatives much faster, after only polynomially many repetitions. This means, for example, that they needed more repetitions than LSH does to get 0.99 success rate, but fewer than LSH needs for success rate $1 - 2^{-n}$.

An alternative to LSH was introduced by [Becker et al., 2016, Dubiner, 2010]. It is referred to as Locality Sensitive Filters, or LSF. While it achieves the same bounds as LSH, LSF has the advantage of giving more control to the algorithm designer for balancing different performance metrics. For example, it typically allows better results for low dimensional data, $d = O(\log n)$, and space/time trade-offs [Andoni et al., 2017a]. The idea is to sample a large number of random sections of the space. In contrast to LSH these sections are not necessarily partitions and may overlap heavily. For example, for points on the sphere S^{d-1} the sections may be defined by balls around the points of a spherical code. One issue compared to LSH is that the number of sections in LSF is very large. This means we need to impose some structure so we can efficiently find all sections containing a particular point. With LSH the space partitioning automatically provided such an algorithm, but for LSF it is common to use a kind of random product code. (An interesting alternative is [Pagh and Christiani, 2017], which uses a random branching processes.) LSF is similar to LSH in that it only approaches 100% success rate as the number of sections goes to infinity.

The work in this paper can be viewed as way of constructing correlated, efficiently decodable filters for Hamming space and Braun-Blanquet similarity. That is, our filters guarantee that any two close points are contained in a shared section, without having an infinite number of sections. Indeed the number of sections needed is equal to that needed by random constructions for achieving constant success probability, up to $n^{o(1)}$ factors. It is not crucial that our algorithms are in the LSF framework rather than LSH. Our techniques can make correlated LSH space partitions of optimal size as well as filters. However the more general LSF framework allows for us to better show of the strength of the techniques.

One very important line of LSH/LSF research, that we don't touch upon in this paper, is that of data dependency. In the seminal papers [Andoni et al., 2014, Andoni and Razenshteyn, 2015, Andoni et al., 2017a] it was shown that the performance of space partition based data structures can be improved, even in the worst case, by considering the layout of the points in the data base. Using clustering, certain bad cases for LSH/LSF can be removed, leaving only the case of "near random" points to be considered, on which LSH works very well. It seems possible to make Las Vegas versions of these algorithms as well, since our approach gives the optimal performance in these near random cases. However one would need to find a way to derandomize the randomized clustering step used in their approach.

There is of course also a literature of deterministic and Las Vegas data structures not using LSH. As a baseline, we note that the "brute force" algorithm that stores every data point in a hash table, and given a query, $q \in \{0,1\}^d$, looks up every $\sum_{k=1}^{r} \binom{d}{k}$ point of Hamming distance most r. This of course requires $r \log(d/r) < \log n$ to be sub-linear, and for a typical example of $d = (\log n)^2$ and $r = d/10$ it won't be practical. In [Cole et al., 2004] this was somewhat improved to yield $n(\log n)^r$ time, but it still requires $r = o(\frac{\log n}{\log\log n})$ for queries to be sub-linear. We can also imagine storing the nearest neighbour for every point in $\{0,1\}^d$. Such an approach would give fast (constant time) queries, but the space required would be exponential in r.

In Euclidean space (ℓ_2 metric) the classical K-d tree algorithm [Bentley, 1975] is of course deterministic, but it has query time $n^{1-1/d}$, so we need $d = O(1)$ for it to be strongly sub-linear. Allowing approximation, but still deterministically, [Arya et al., 1998] found a $(d/(c-1))^d$ algorithm for a $c > 1$ approximation. This allows us to set $d = o(\frac{\log n}{\log\log n})$.

For large approximation factors [Har-Peled et al., 2012] gave a deterministic data structure with query time $O(d \log n)$, but space and preprocessing more than $n \cdot O(1/(c-1))^d$. In a different line of work, [Indyk, 2000a] gave a deterministic $(d\epsilon^{-1} \log n)^{O(1)}$ query time, fully deterministic algorithm with space usage $n^{O(1/\epsilon^6)}$ for a $3 + \epsilon$ approximation.

See Table I for an easier comparison of the different results and spaces.

C. Techniques

Our main new technique is a combination of 'splitters' as defined by [Naor et al., 1995, Alon et al., 2006], and 'tensoring' which is a common technique in the LSH literature.

Tensoring means constructing a large space partition $P \subseteq \mathcal{P}(X)$ by taking multiple smaller random partitions $P_1, P_2, \dots$ and taking all the intersections $P = \{p_1 \cap p_2, \dots \mid p_1 \in P_1, p_2 \in P_2, \dots\}$. Often the implicit partition P is nearly as good as a fully random partition of equal size, while it is cheaper to store in memory and allows much faster lookups of which section covers a given point. In this paper we are particularly interested in P_i's that partition different small sub-spaces, such that P is used to increase the dimension of a small, explicit, good partition.

Unfortunately tensoring doesn't seem to be directly applicable for deterministic constructions, since deterministic space partitions tend to have some overhead that gets amplified by the product construction. This is the reason why [Pagh, 2016] constructs hash functions directly using algebraic methods, rather than starting with a small hash function and 'amplifying' as is common for LSH. Algebraic methods are great when they exist, but they tend to be hard to find, and it would be a tough order to find them for every similarity measure we would like to make a data structure for.

It turns out we can use splitters to help make tensoring work deterministically. Roughly, these are generalizations of perfect hash functions. However, where a (d, m, k)-perfect hash family guarantees that for any set $S \subseteq [d]$ of size k, there is a function $\pi : [d] \to [m]$ such that $|\pi(S)| = k$, a (d, m)-splitter instead guarantees that the is some π such that $|S \cap \pi^{-1}(i)| = d/m$ for each $i = 1, \dots, m$; or as close as possible if m does not divide d. That is, for any S there is some π that 'splits' S evenly between m buckets.

Using splitters with tensoring, we greatly limit the number of combinations of smaller space partitions that are needed to guarantee covering. We use this to amplify partitions found probabilistically and verified

Reference	Space	Exponent, search time	Comments
[Bentley, 1975]	ℓ_2	$1 - 1/d$	Exact algorithm, Fully deterministic.
[Cole et al., 2004]	Hamming	$r\frac{\log\log n}{\log n}$	Sub-linear for $r < \frac{\log n}{\log\log n}$. Exact.
[Arya et al., 1998]	ℓ_2	$d^{\frac{\log(d/(c-1))}{\log n}}$	Sub-linear for $d < \frac{\log n}{\log\log n}$.
[Har-Peled et al., 2012]	Hamming	$o(1)$	c-approximation, Fully deterministic, $(1/(c-1))^d$ space.
[Indyk, 2000a]	Hamming	$o(1)$	$(3+\epsilon)$-approximation, Fully deterministic, $n^{\Omega(1/\epsilon^6)}$ space.
[Arasu et al., 2006]	Hamming	$\approx 3/c$	The paper makes no theoretical claims on the exponent.
[Pagh, 2016]	Hamming	$1.38/c$	Exponent $1/c$ when $r = o(\log n)$ or $(\log n)/(cr) \in \mathbb{N}$.
[Pacuk et al., 2016]	ℓ_p	$O(d^{1-1/p}/c)$	Sub-linear for ℓ_2 when $c = \omega(\sqrt{d})$.
This paper	Hamming	$1/c$	Actual exponent is $\frac{1-cr/d}{c(1-r/d)}$ which improves to $1/(2c-1)$ for $cr \approx d/2$.
[Pagh, 2016]	Braun-Blanquet	$1.38\,\frac{1-b_1}{1-b_2}$	Via reduction to Hamming. Requires sets of equal weight.
This paper	Braun-Blanquet	$\frac{\log 1/b_1}{\log 1/b_2}$	See [Pagh and Christiani, 2017] figure 2 for a comparison with [Pagh, 2016].

TABLE I
COMPARISON OF LAS VEGAS ALGORITHMS FOR HIGH DIMENSIONAL NEAR NEIGHBOUR PROBLEMS. THE EXPONENT IS THE VALUE ρ, SUCH THAT THE DATA STRUCTURE HAS QUERY TIME $n^{\rho+o(1)}$. ALL LISTED ALGORITHMS, EXCEPT FOR [INDYK, 2000A] USE LESS THAN n^2 SPACE. ALL ALGORITHMS GIVE c-APPROXIMATIONS, EXCEPT FOR THE FIRST TWO, AND FOR [INDYK, 2000A], WHICH IS A $(3+\epsilon)$-APPROXIMATION.

deterministically. The random aspect is however only for convenience, since the greedy set cover algorithm would suffice as well, as is done in [Alon et al., 2006]. We don't quite get a general reduction from Monte Carlo to Las Vegas LSH data structures, but we show how two state of the art algorithms may be converted at a negligible overhead.

A final technique to make everything come together is the use of dimensionality reductions. We can't quite use the standard bit-sampling and Johnson–Lindenstrauss lemmas, since those may (though unlikely) increase the distance between originally near points. Instead we use two dimensionality reduction lemmas based on partitioning. Similarly to [Pagh, 2016] and others, we fix a random permutation. Then given a vector $x \in \{0,1\}^d$ we permute the coordinates and partition into blocks $x_1, \dots, x_{d/B}$ of size B. For some linear distance function, $\mathrm{dist}(x,y) = \mathrm{dist}(x_1,y_1) + \dots + \mathrm{dist}(x_{d/B}, y_{d/B})$, which implies that for some i we must have $\mathrm{dist}(x_i,y_i) \le \mathrm{dist}(x,y)B/d$. Running the algorithm separately for each set of blocks guarantee that we no pair gets mapped too far away from each other, while the randomness of the permutation lets us apply standard Chernoff bounds on how close the remaining points get.

Partitioning, however, doesn't work well if distances are very small, $cr << d$. This is because we need $B \approx d/(cr)\epsilon^{-2}\log n$ to get the said Chernoff bounds on distances for points at distance cr. We solve this problem by hashing coordinates into buckets of $\approx cr/\epsilon$ and taking the xor of each bucket. This has the effect of increasing distances and thereby allowing us to partition into blocks of size $\approx \epsilon^{-3}\log n$. A similar technique was used for dimensionality reduction in [Kushilevitz et al., 2000], but without deterministic guarantees. The problem is tackled fully deterministically in [Indyk, 2000a] using codes, but with the slightly worse bound of $\epsilon^{-4}\log n$.

In the process of showing our results, we show a useful bound on the ratio between two binomial coefficients, which may be of separate interest.

D. Notation

We use $[d] = \{1,\dots,d\}$ as convenient notation sets of a given size. Somewhat overloading notation, for a predicate P, we also use the Iversonian notation $[P]$ for a value that is 1 if P is true and 0 otherwise.

For a set $x \subseteq [d]$, we will sometimes think of it as a subset of the universe $[d]$, and at other times as a vector $x \in \{0,1\}^d$, where $x_i = 1$ indicates that $i \in x$. This correspondence goes further, and we may refer to the set size $|x|$ or the vector norm $\|x\|$, which is always the Hamming norm, $\|x\| = \sum_{i=1}^d x_i$. Similarly for two sets or points $x, y \in \{0,1\}^d$, we may refer to the inner product $\langle x,y\rangle = \sum_{i=1}^d x_i y_i$ or to the size of their intersection $|x \cap y|$.

We use $S \times T = \{(s,t) : s \in S, t \in T\}$ for the cross product, and $x \oplus y$ for symmetric difference (or 'xor'). $\mathcal{P}(X)$ is the power set of X, such that $x \subseteq X \equiv x \in \mathcal{P}(X)$.

For at set $S \subseteq [d]$ and a vector $x \in \{0,1\}^d$, we let x_S be the projection of x onto S. This is an $|S|$-dimensional vector, consisting of the coordinates $x_S = \langle x_i : i \in S\rangle$ in the natural order of i. For a function $f : [a] \to [b]$ we let $f^{-1} : \mathcal{P}([b]) \to \mathcal{P}([a])$ be the 'pullback' of f, such that $f^{-1}(S) = \{i \in [a] \mid f(i) \in S\}$. For example, for $x \in \{0,1\}^a$, we may write $x_{f^{-1}(1)}$ to be the vector x projected onto the coordinates of $f^{-1}(\{1\})$.

Sometimes when a variable is $\omega(1)$ we may assume it is integral, when this is achievable easily by rounding that only perturbs the result by an insignificant $o(1)$ amount.

The functional $\operatorname{poly}(a,b,\dots)$ means any polynomial combination of the arguments, essentially the same set as $(a \cdot b \dots)^{\pm O(1)}$.

E. Organization

We start by laying out the general framework shared between our algorithms. We use a relatively common approach to modern near neighbour data structures, but the overview also helps establish some notation used in the later sections.

The second part of section II describes the main ideas and intuition on how we achieve our results. In particular it defines the concept of 'splitters' and how they may be used to create list-decodable codes for various measures. The section finally touches upon the issues we encounter on dimensionality reduction, which we can use to an extent, but which is restricted by our requirement of '1-sided' errors.

In sections III and IV we prove the main theorems from the introduction. The sections follow a similar pattern: First we introduce a filter family and prove its existence, then we show a dimensionality reduction lemma and analyze the resulting algorithm.

II. Overview

Both algorithms in this paper follow the structure of the Locality Sensitive Filter framework, which is as follows: For a given universe U, we define a family $\mathcal{F}$ of 'filters' equipped with a function $F : U \to \mathcal{P}(\mathcal{F})$, which assigns every point a set of filters.

Typically, $\mathcal{F}$ will be a covering of U, and $F(x)$ will be the sets that cover the point x. Intuitively we want points that share a filter to be close/similar, and we want points that are sufficiently close/similar to always share a filter. There will usually also be some non-similar points in a particular filter. To get good expected performance, some randomness, such as a rotation that does not change distances, is performed to make this only happen with low probability.

To construct the data structure, we are given a set of data points $P \subseteq U$. We compute $F(x)$ for every $x \in P$ and store the points in a (hash) map $T : \mathcal{F} \to \mathcal{P}(P)$. For any point $x \in P$ and filter $f \in F(x)$, we store $x \in T[f]$. Note that the same x may be stored in multiple different buckets.

To query the data structure with a point $x \in U$, we compute the distance/similarity between x and every point $y \in \bigcup_{f \in F(x)} T[f]$, returning the first suitable candidate, if any.

There are many possible variations of the scheme, such as sampling $\mathcal{F}$ from a distribution of filter families. In case we want a data structure with space/time trade-offs, we can use different $\mathcal{F}$ functions for data points and query points. However in this article we will not include these extensions.

We note that while it is easy to delete and insert new points in the data structure after creation, we are going to choose $\mathcal{F}$ parametrized on the total number of points, $|P|$. This makes our data structure essentially static, but luckily [Overmars and van Leeuwen, 1981] have found general, deterministic reductions from dynamic to static data structures.

A. Intuition

The main challenge in this paper will be the construction of filter families $\mathcal{F}$ which are: (i) not too large; (ii) have a $F(\cdot)$ function that is efficient to evaluate; and most importantly, (iii) guarantee that all sufficiently close/similar points always share a filter. The last requirement is what makes our algorithm different from previous results, which only had this guarantee probabilistically.

For concreteness, let us consider the Hamming space problem. Observe that for very low dimensional spaces, $d = (1+o(1))\log n$, we can afford to spend exponential time designing a filter family. In particular we can formulate a set cover problem, in which we wish to cover each pair of points at distance $\le r$ with Hamming balls of radius s. This gives a family that is not much larger than what can be achieved probabilistically, and which is guaranteed to work. Furthermore, this family has sublinear size ($n^{o(1)}$), making $F(x)$ efficient to evaluate, since we can simply enumerate all of the Hamming balls and check if x is contained.

The challenge is to scale this approach up to general d.

Using a standard approach of randomly partitioning the coordinates, we can reduce the dimension to $(\log n)^{1+\epsilon}$. This is basically dimensionality reduction by bit sampling, but it produces $d/\log n$ different subspaces, such that for any pair x, y there is at least one subspace in which their distance is not increased. We are left with a gap from $(\log n)^{1+\epsilon}$ down to $\log n$. Bridging this gap turns out to require a lot more work. Intuitively we cannot hope to simply use a stronger dimensionality reduction, since $\log n$ dimensions only just fit n points in Hamming space and would surely make too many non-similar points collide to be effective.

A natural idea is to construct higher-dimensional filter families by combining multiple smaller families. This is a common technique from the first list decodable error correcting codes, for example [Elias, 1957]: Given a code $\mathcal{C} \subseteq \{0,1\}^d$ with covering radius r, we can create a new code $\mathcal{C}_2 \subseteq \{0,1\}^{2d}$ of size $|\mathcal{C}|^2$ with covering radius $2r$ by taking every pair of code words and concatenating them. Then for a given point $x \in \{0,1\}^{2d}$ we can decode the first and last d coordinates of $x = x_1 x_2$ separately in $\mathcal{C}$. This returns two code words c_1, c_2 such that $\mathrm{dist}(x_1, c_1) \le r$ and $\mathrm{dist}(x_2, c_2) \le r$. By construction $c_1 c_2$ is in $\mathcal{C}_2$ and $\mathrm{dist}(x_1 x_2, c_1 c_2) \le 2r$.

This combination idea gives is nice when it applies. When used with high quality inner codes, the combined code is close to optimal as well. In most cases the properties of $\mathcal{C}$ that we are interested in won't decompose as nicely. With the example of our Hamming ball filter family, consider $x, y \in \{0,1\}^{2d}$ with distance $\mathrm{dist}(x,y) = r$. If we split $x = x_1 x_2$ an $y = y_1 y_2$ we could decode the smaller vectors individually in a smaller family, however we don't have any guarantee on $\mathrm{dist}(x_1, y_1)$ and $\mathrm{dist}(x_2, y_2)$ individually, so the inner code might fail to return anything at all.

To solve this problem, we use a classic tool for creating combinatorial objects, such as our filter families, called 'splitters'. Originally introduced by [Mairson, 1983, Naor et al., 1995] they are defined as follows:

Definition 3 (Splitter). *A (B,l)-splitter H is a family of functions from $\{1,\dots,B\}$ to $\{1,\dots,l\}$ such that for all $S \subseteq \{1,\dots,B\}$, there is a $h \in H$ that splits S perfectly, i.e., into equal-sized parts $h^{-1}(j) \cap S$, $j = 1,2,\dots,l$. (or as equal as possible, if l does not divide $|S|$).*

The size of H is at most B^l, and using either a generalization by [Alon et al., 2006] or a simple combinatorial argument, it is possible to ensure that the size of each part $|h^{-1}(j)|$ equals B/l (or as close as possible).

We now explain how splitters help us combine filter families. Let H be a splitter from $\{1,\dots,2d\}$ to $\{1,2\}$. For any $x, y \in \{0,1\}^{2d}$ we can let S be the set of coordinates on which x and y differ. Then there is a function $h \in H$ such that $|h^{-1}(1) \cap S| = |h^{-1}(2) \cap S| = |S|/2$. (Or as close as possible if $|S|$ is odd.) If we repeat the failed product combination from above for every $h \in H$ we get a way to scale our family from d to $2d$ dimensions, taking the size from $|\mathcal{F}|$ to $(2d)^2 |\mathcal{F}|^2$. That is, we only suffer a small polynomial loss. In the end it turns out that the loss suffered from creating filter families using this divide and conquer approach can be contained, thus solving our problem.

An issue that comes up, is that the 'property' we are splitting (such as distance) can often be a lot smaller than the dimensionality d of the points. In particular this original dimensionality reduction may suffer an overhead factor $d/|S|$, which could make it nearly useless if $|S|$ is close to 1. To solve this problem, both of our algorithms employ special half-deterministic dimensionality reductions, which ensures that the interesting properties get 'boosted' and end up taking a reasonable amount of 'space'. These reductions are themselves not complicated, but they are somewhat non-standard, since they can only have a one sided error. For example for Hamming distance we need that the mapped distance is never larger than its expected value, since otherwise we could get false negatives.

For Hamming distance our dimension reduction works by hashing the coordinates randomly from [d] to [m] taking the xor of the coordinates in each bucket. This is related to the β-test in [Kushilevitz et al., 2000]. The idea is that if x and y are different in only a few coordinates, then taking a small random group of coordinates, it is likely to contain at most one where they differ. If no coordinates differ, then after taking the xor the result will still be the same, but if exactly one (or an odd number) of coordinates differ, the resulting coordinate will be different.

For set similarity it turns out that we need less dimensionality reduction and simple partitioning suffices. This is due to the verification algorithm being efficient when sets have small similarity, while partitioning works well when the similarity is large.

III. Hamming Space Data Structure

We will give an efficient filter family for LSF in Hamming space. Afterwards we will analyze it and choose the most optimal parameters, including dimensionality reduction.

Lemma 1. *For every choice of parameters $B, b \in \mathbb{N}$, b B, $0 < r < B/2$ and $s^2 = O(B/\sqrt{b})$, there exists a code $\mathcal{C} \subseteq \{0,1\}^B$ of size $|\mathcal{C}| = \mathrm{poly}(B^{B/b}) \exp(\frac{s^2}{2(1-r/d)})$ with the following properties:*

1) *Given $x \in \{0,1\}^B$ we can find a subset $C(x) \subseteq \{c \in \mathcal{C} : \mathrm{dist}(x,c) \le B/2 - s\sqrt{B}/2\}$ in time $|C(x)| + \mathrm{poly}(B^{B/b}, e^{s^2 b/B})$.*
2) *For all pairs $x, y \in \{0,1\}^B$ with $\mathrm{dist}(x,y) \le r$ there is some common nearby code word $c \in C(x) \cap C(y)$.*
3) *The code requires $4^b \mathrm{poly}(B^{B/b}, e^{s^2 b/B})$ time for preprocessing and $\mathrm{poly}(B^{B/b}, e^{s^2 b/B})$ space.*

Note that we don't actually guarantee that our 'list-decoding' function $C(x)$ returns *all* nearby code words, just that it returns enough for property (2) to hold. (This is just to make the proof a bit shorter though; using a decoding algorithm similar to [Becker et al., 2016] would change this.)

Proof: We first show how to construct a family for $\{0,1\}^b$, then how to enlarge it for $\{0,1\}^B$. We then show that it has the covering property and finally the decoding properties. In order for our probabilistic arguments to go through, we need the following lemma, which follows from Stirling's Approximation:

Lemma 2. *For $t = \frac{d}{2} - \frac{s\sqrt{d}}{2}$, $1 \le s = O(d^{1/4})$ and $r < d/2$, Let $x, y \in \{0,1\}^d$ be two points at distance $\operatorname{dist}(x,y) = r$, and let $I = |\{z \in \{0,1\}^d : \operatorname{dist}(z,x) \le t, \operatorname{dist}(z,y) \le t\}|$ be the size of the intersection of two hamming balls around x and y of radius t, then*

$$I\,2^{-d} = \operatorname{poly}(d)\exp\left(\frac{-s^2}{2(1-r/d)}\right). \tag{1}$$

Let $s' = s\sqrt{b/B}$. Consider any two points $x, y \in \{0,1\}^b$ with distance $\le (r/d)b$. If we choose $a \in \{0,1\}^b$ uniformly at random, by lemma 2 we have probability $p = \operatorname{poly}(b)\exp(\frac{-s'^2}{2(1-r/d)})$ that both x and y have distance at most $t = b/2 - s'\sqrt{b/4}$ with c. By the union bound over pairs in $\{0,1\}^b$, if we sample $p^{-1}b\log 2$ independent as, we get constant probability that some a works for every pair. We can verify that a set A of such filters indeed works for every pair in time $4^b|A|$. By repeatedly sampling sets A and verifying them, we get a working A in expected $O(1)$ tries.[1]

Next we define $\mathcal{C}$. We build a splitter, that is a set Π of functions $\pi : [B] \to [B/b]$, such that for every set $I \subseteq [B]$ there is a $\pi \in \Pi$ such that $\lfloor |I|b/B \rfloor \le |\pi^{-1}(j) \cap I| \le \lceil |I|b/B \rceil$ for $j = 1, \dots, B/b$. By the discussion after definition 3, such a set of size $\operatorname{poly}(B^{B/b})$ exists and can be constructed in time and space proportional to its size. Implicitly we can then define $\mathcal{C}$ by making one code word $c \in \{0,1\}^B$ for every $\pi \in \Pi$ and $1 \le j_1, \dots, j_{B/b} \le |A|$, satisfying the requirement that $c_{\pi^{-1}(j_k)} = A_{j_k}$ for $k = 1 \dots B/b$. That is, for a given set of rows of A and a split of $[B]$, we combine the rows into one row c such that each row occupies a separate part of the split. Note that this is possible, since splitter has all partitions of equal size, b. The created family then has size $|\mathcal{C}| = |\Pi||A|^{B/b} = \operatorname{poly}(B^{B/b})\exp(\frac{-s^2}{2(1-r/d)})$ as promised. Since the only explicit work we had to do was finding A, we have property (3) of the lemma.

We define the decoding function $C(x) \in \mathcal{C}$ for $x \in \{0,1\}^B$ with the following algorithm: For each $\pi \in \Pi$ compute the inner decodings $A_j = \{a \in A : \operatorname{dist}(x_{\pi^{-1}(j)}, a) \le b/2 - s\sqrt{b}/2\}$ for $j = 1, \dots, B/b$. Return the set of all concatenations in the product of the A_j's: $C(x) = \{a_1 \| a_2 \| \dots \| a_{B/b} : a_1 \in A_1, \dots\}$. Computing the A_j's take time $(B/b)|A|$, while computing and concatenating the product takes linear time in the size of the output. This shows property (1).

Finally for property (2), consider a pair $x, y \in \{0,1\}^B$ of distance $\le r$. Let I be the set of coordinates on which x and y differ. Then there is a function $\pi \in \Pi$ such that x and y differ in at most $|I|b/B = rb/B$ coordinates in each subset $\pi^{-1}(1), \dots, \pi^{-1}(B/b) \subseteq [B]$. Now for each pair of projected vectors $x_{\pi^{-1}(1)}, y_{\pi^{-1}(1)}, \dots$ (let's call them $x_1, y_1, \dots$) there is an $a_j \in A$ such that $\operatorname{dist}(a_j, x_j) \le b/2 - s'\sqrt{b}/2$ and $\operatorname{dist}(a_j, y_j) \le b/2 - s'\sqrt{b}/2$. This means that x and y must both have distance at most $(b/2 - s'/2)B/b = B/2 - s\sqrt{B}/2$ to that $c \in \mathcal{C}$ which has $c_{\pi^{-1}(j)} = a_j$ for $j = 1 \dots B/b$. By the same reasoning, this c will be present in both $C(x)$ and $C(y)$, which proves the lemma. ∎

[1]The randomness is not essential, and we could as well formulate a set cover instance and solve it using the greedy algorithm, which matches the probabilistic construction up to a log factor in size and time.

Returning to the problem of near neighbour search in $\{0,1\}^d$, it is clear from the $4^b\operatorname{poly}(B^{B/b})$ construction time of the above family, that it will not be efficient for dimension $B = (\log n)^{\omega(1)}$. For this reason we will apply the following dimensionality reduction lemma:

Lemma 3. *For $n > d$, $\epsilon = \epsilon(n) > 0$, $cr > r \ge 1$ and some $B = 27\epsilon^{-3}\log n$, assume $n \ge m = 3cr/\epsilon$, then there is a random set F of at most $S = m/B$ functions $f : \{0,1\}^d \to \{0,1\}^B$ with the following properties for every $x, y \in \{0,1\}^d$:*

1) *With probability 1, there is at least one $f \in F$ st.:*

$$\operatorname{dist}(f(x), f(y)) \le \operatorname{dist}(x,y)/S.$$

2) *If $\operatorname{dist}(x,y) \ge cr$ then for every $f \in F$ with probability at least $1 - 1/n$:*

$$\operatorname{dist}(f(x), f(y)) \ge (1-\epsilon)cr/S.$$

Proof: To prove lemma 3 first notice that if $B \ge d$ we can use the identity function and we are done. If $B \ge m$, then we can duplicate the vector $\lceil m/B \rceil = O(\epsilon^{-2}\log n)$ times. Also, by adjusting ϵ by a constant factor, we can assume that B divides m.

For the construction, pick a random function $h : [d] \to [m]$. Define $g : \{0,1\}^d \to \{0,1\}^m$ by 'xor'ing the contents of each bucket, $g(x)_i = \bigoplus_{j \in h^{-1}(i)} x_j$, and let $f_i(x) = g(x)_{(iB,(i+1)B]}$ for $i = 0 \dots m/B$ be the set of functions in the lemma. We first show that this set has property (1) and then property (2).

Observe that g never increases distances, since for any $x, y \in \{0,1\}^d$ the distance

$$\operatorname{dist}(g(x), g(y)) = \sum_{i=1}^{m}\left[\bigoplus_{j \in h^{-1}(i)} x_j \ne \bigoplus_{j \in h^{-1}(i)} y_j\right]$$

is just $\sum_{i=1}^m \left(\sum_{j \in h^{-1}(i)}[x_j \ne y_j] \bmod 2\right)$ which is upper bounded by the number of coordinates at which x and y differ. By averaging, there must be one f_i such that $\operatorname{dist}(f_i(x), f_i(x)) \le \operatorname{dist}(g(x), g(y))B/m \le \operatorname{dist}(x,y)/S$.

For the second property, let $R = \operatorname{dist}(x,y) \ge cr$ and let $X_1, \dots, X_m$ be the random number of differences between x and y in each bucket under h. Let $Y_1, \dots, Y_m$ be iid. Poisson distributed variables with mean $\lambda = \operatorname{E} X_1 = R/m \ge \epsilon/3$. We use the the Poisson trick from [Mitzenmacher and Upfal, 2005] theorem 5.7:

$$\Pr[\sum_{i=1}^{B}(X_i \bmod 2) < x] \le e\sqrt{m}\Pr[\sum_{i=1}^{B}(Y_i \bmod 2) < x]$$

and notice that $(Y_i \bmod 2) \ge [Y_i = 1]$, such that the rhs. sum is bounded by a binomial $\sim \mathrm{Bin}(B,p)$ with $p = \Pr[Y_1 = 1] = \lambda e^{-\lambda} \ge \lambda(1-\lambda)$. We can then bound the probability of an f_i decreasing distances too much, using a Chernoff bound:

$$\begin{aligned}&\Pr[\mathrm{dist}(f_i(x), f_i(y)) \le (1-\epsilon)cr/S]\\ &\quad\le e\sqrt{m}\exp(-D[(1-\epsilon)\epsilon/3 \mid (1-\epsilon/3)\epsilon/3]B)\\ &\quad\le e\sqrt{m}\exp(-2\epsilon^3 B/27).\end{aligned}$$

Since $cr/S = crB/m = B\epsilon/3$. Here $D[\alpha \mid \beta] = \alpha\log\frac{\alpha}{\beta} + (1-\alpha)\log\frac{1-\alpha}{1-\beta}$ is the Kullback–Leibler divergence. For our choice of B the error probability is then $e\sqrt{m}/n^2$ which is less than $1/n$ by our assumptions. This proves the lemma. ■

Using lemma 3 we can make at most $3cr/\epsilon = O(d/\epsilon)$ data structures, as described below, and be guaranteed that in one of them, we will find a near neighbour at distance $r' = r/S = \epsilon/(3c)B$. In each data structure we will have to reduce the distance cr', at which we expect far points to appear, to $cr'(1-\epsilon)$. This ensures we see at most a constant number of false positives in each data structure, which we can easily filter out. For $\epsilon = o(1)$ this change be swallowed by the approximation factor c, and won't significantly impair our performance.

When using the filter family of lemma 1 for LSF, the time usage for queries and inserting points is dominated by two parts: 1) The complexity of evaluating $C(x)$, and 2) The expected number of points at distance larger than $cr'(1-\epsilon)$ that falls in the same filter as x.

Since lemma 3 randomly shuffled the coordinates, we can assume that x is a random point in $\{0,1\}^d$. The expected time to decode $C(x)$ is then

$$\begin{aligned}&\mathrm{E}\,|C(x)| + \mathrm{poly}(B^{B/b}, e^{s^2 b/B})\\ &= |\mathcal{C}|\Pr_x[0 \in C(x)] + \mathrm{poly}(B^{B/b}, e^{s^2 b/B})\\ &\le \mathrm{poly}(B^{B/b}, e^{s^2 b/B})\exp\left(\tfrac{s^2}{2(1-r'/B)} - \tfrac{s^2}{2}\right).\end{aligned}$$

For estimating collisions with far points, we can similarly assume that x and y are random points in $\{0,1\}^d$ with fixed distance $cr'(1-\epsilon)$:

$$\begin{aligned}&\mathrm{E}\,|\{y \in P : C(x) \cap C(y) \ne \varnothing\}|\\ &\le n\,|\mathcal{C}|\Pr_{x,y}[0 \in C(x), 0 \in C(y)]\\ &\le B^{O(B/b)}\exp\left(\tfrac{s^2}{2(1-r'/B)} - \tfrac{s^2}{2(1-c(1-\epsilon)r'/B)} + \log n\right)\\ &= B^{O(B/b)}\exp\left(\tfrac{s^2}{2}\left(\tfrac{1}{1-r'/B} - \tfrac{1}{1-cr'/B} + O(\epsilon)\right) + \log n\right).\end{aligned}$$

Finally we should recall that constructing the data structures takes time $4^b\,\mathrm{poly}(e^{s^2 b/B})$ plus n inserts.

We now choose the parameters:

$$s^2/2 = \tfrac{1-cr'/B}{cr'/B}\log n, \qquad B = 27\epsilon^{-3}\log n,$$
$$b = \log_4 n, \qquad \epsilon = (\log n)^{-1/4}.$$

This makes the code construction time $n^{1+o(1)}$ while evaluating $C(x)$ and looking at far points takes expected time at most $n^{1/c+\tilde{O}(\log n)^{-1/4}}$. To use lemma 1 we have to check that $s^2 = O(B/\sqrt{b}) = O((\log n)^{5/4})$, but $s^2/2 = \frac{1-\epsilon/3}{\epsilon/3}\log n = (\log n)^{5/4}(1-o(1))$ so everything works out. This shows theorem 1.

To get the result of corollary 1, we just need to substitute the dimensionality reduction lemma 3 for a simple partitioning approach:

Lemma 4. *For any $d \ge r \ge 1$ and $\epsilon > 0$ there is a set F of d/B functions, $f : \{0,1\}^d \to \{0,1\}^B$, where $B = 2\epsilon^{-2}d/(cr)\log n$, such that:*

1) *With probability 1, there is at least one $f \in F$ st.:*

$$\mathrm{dist}(f(x), f(y)) \le \mathrm{dist}(x,y)\,B/d.$$

2) *If $\mathrm{dist}(x,y) \ge cr$ then for every $f \in F$ with probability at least $1-1/n$:*

$$\mathrm{dist}(f(x), f(y)) \ge (1-\epsilon)cr\,B/d.$$

Proof: Fix a random permutation. Given $x \in \{0,1\}^d$, shuffle the coordinates using the permutation. Let $f_1(x)$ be the first B coordinates of x, $f_2(x)$ the next B coordinates and so forth. For any $y \in \{0,1\}^d$, after shuffling, the expected number of differences in some block of B coordinates is $\mathrm{dist}(x,y)B/d$. Then the first property follows because $\sum_i \mathrm{dist}(f_i(x), f_i(y)) = \mathrm{dist}(x,y)$ so not all distances can be below the expectation. The second property follows from the Chernoff/Hoeffding bound [Hoeffding, 1963]. ■

That is, we randomly partition the d coordinates in B parts and run the algorithm on those. The important point is that in this case $r'/B = r/d$, so the relative distance is not decreased. We choose parameters

$$s^2/2 = \tfrac{1-cr/d}{cr/d}\log n \qquad B = O(\epsilon^{-2}(cr/d)^{-1}\log n),$$
$$b = \log_4 n, \qquad \epsilon = (\log n)^{-1/3}.$$

This again makes this makes the code construction time $n^{1+o(1)}$ while evaluating $C(x)$ and looking at far points takes time $n^{\frac{1-c\delta}{c(1-\delta)}+\tilde{O}(\log n)^{-1/3}d/r}$ as in the corollary. Again we need need to check that $s^2 = O(B/\sqrt{b}) = O((\log n)^{7/6})$. This works as long as $r/d = \Omega((\log n)^{-1/6})$, which is the assumption of the corollary.

IV. Set Similarity Data Structure

The structure of the algorithm follows the LSF framework as in the previous section. We will try to follow the same structure of presenting a filter family, then analyze it with a simple dimensionality reduction, and then iron out remaining kinks.

One technicality is that we will assume the weight of the query points $\|q\|$ and any data point $\|x\|$ are known in advance. This can be accomplished by making $(d+1)^2$ data structures, saving each point in $d+1$ of them and querying $d+1$ of them for each query. For each such data structure we define $t = \max(t_q, t_x)$.

If we are trying to solve the (s_1, s_2)-approximate similarity search problem for our particular weight pair t_q, t_x, then the inner product between the query and the 'close' data points we are interested in is at least $s_1 t$, while the inner product between the query and the 'far' points we are not interested in is at most $s_2 t$.

A good filter family for set similarity turns out to be the r-element blocks of a Turán system:

Definition 4 ([Turán, 1961, Colbourn and Dinitz, 2006])**.** *A Turán (n,k,r)-system is a collection of r-element subsets, called 'blocks', of an n element set X such that every k element subset of X contains at least one of the blocks.*

This choice is inspired by [Pagh and Christiani, 2017], and is generally well studied. However none of the known constructions are explicit, and they don't seem to allow us to avoid false negatives. The lemma below provides one such construction.

Lemma 5. *For every $B \ge b$, k, r, there exists a Turán (B,k,r)-system, $\mathcal{T}$, of size $|\mathcal{T}| \le \text{poly}(B^{B/b}, e^{r^2/k})\,(B/k)^r$ with the following properties:*

1) *Efficient decoding: Given a set $S \subseteq [B]$, we can find all the blocks it contains $\mathcal{T}(S) = \{R \in \mathcal{T} : R \subseteq S\}$ in time $|\mathcal{T}(S)| + \text{poly}(B^{B/b}, e^{r^2/k\,b/B}, (B/k)^{rb/B})$.*
2) *Correctness: For every set $K \subseteq [B]$ of size at least k, there is at least one block $R \in \mathcal{T}$ such that $R \subseteq K$.*
3) *Efficient construction: The code is constructible, implicitly, in $\text{poly}(B^{B/b}, e^{r^2/k})(eB/k)^{kb/B}$ time and space.*

A simple volume lower bound shows that an (n,k,r)-system must have at least $\binom{n}{r}/\binom{n}{k} \ge (n/k)^r$ blocks, making our construction optimal up to polynomial factors.

Proof: As in the previous section, we first show how to construct a family for $[b]$, then how to extend it to $[B]$. Finally we show correctness of the fast decoding algorithm.

To bound some of the probabilities, we will need the following lemma:

Lemma 6. *For all $n \ge m \ge k \ge 0$*

$$\left(\frac{n}{m}\right)^k \le \binom{n}{k}\Big/\binom{m}{k} \le \left(\frac{n}{m}\right)^k e^{k^2/m}.$$

Proof: For the lower bound, $\binom{n}{k}/\binom{m}{k} = \prod_{i=0}^{k-1} \frac{n-i}{m-i} \ge \prod_{i=0}^{k-1} \frac{n}{m} = \left(\frac{n}{m}\right)^k$, since $\frac{n-i}{m-i}$ is increasing in i. For the upper bound we use the sum/integral inequality to show

$$\begin{aligned}\tbinom{n}{k}/\tbinom{m}{k} &= \exp(\Sigma_{i=0}^{k-1} \log \tfrac{n-i}{m-i}) \\ &\le \exp(\int_0^k \log \tfrac{n-x}{m-x} dx) \\ &= \exp(n\,\mathrm{H}(k/n) - m\,\mathrm{H}(k/m)) \\ &\le (\tfrac{n}{m})^k \exp(\tfrac{k^2}{2}(\tfrac{1}{m-k/2} - \tfrac{1}{n})) \\ &\le (\tfrac{n}{m})^k \exp(\tfrac{k^2}{m}).\end{aligned}$$

Here $\mathrm{H}(x) = x \log 1/x + (1-x)\log 1/(1-x)$ and we used the log-inequalities $2x/(2+x) \le \log(1+x) \le x(2+x)/(2+2x)$ from [Topsøe, 2006]. ■

We now continue the proof of lemma 5. Let $r' = rb/B$ and $k' = kb/B$. Probabilistically we can build a Turán (b, k', r')-system, $\mathcal{T}'$, by sampling

$$m = \tbinom{b}{r'}/\tbinom{k'}{r'}(1 + \log \tbinom{b}{k'})$$

independent size r'-sets. For any size k' subset, K, the probability that it contains none of the r'-sets is $\left(1 - \binom{k'}{r'}/\binom{b}{r'}\right)^m \le e^{-1}/\binom{b}{k'}$. Hence by the union bound there is constant probability that all k'-sets contain an r'-set, making our $\mathcal{T}'$ a valid system.

We can test the correctness of the system in

$$\begin{aligned} m\tbinom{b}{k'} &\le \text{poly}(e^{r'^2/k'})(b/k')^{r'}(eb/k')^{k'} \\ &= \text{poly}(e^{r^2/k\,b/B}, (B/k)^{rb/B})(eB/k)^{kb/B}\end{aligned}$$

time, where we used the bound on m from lemma 6.

To scale up the system, we again take a set Π of functions $\pi_i : [B] \to [B/b]$ such that for any set $I \subseteq [B]$ there is a $\pi_i \in \Pi$ such that

$$|\pi_i^{-1}(j) \cap I| = |I| b/B \text{ for } j = 1, \dots, B/b.$$

When using combining subsets of $[b]$ or $[B]$ it is useful to think of them as binary indicator vectors in $\{0,1\}^b$ or $\{0,1\}^B$. For each i and distinct $k_1, \dots, k_{B/b} \in [m]$ we then make an r-set $R \in \{0,1\}^B$ such that $R_{\pi_i^{-1}(j)} = \mathcal{T}'_{k_j}$ for $j = 1 \dots B/b$. The created family has size $|\mathcal{T}| = |\Pi| m^{B/b} = \text{poly}(B^{B/b}, e^{r^2/k})(B/k)^r$ as we wanted.

We now show correctness. For this, consider any k-set $K \subseteq [B]$. We need to show that there is some r-set $R \in \mathcal{T}$ such that $R \subseteq K$. By construction of Π, there is some $\pi_i \in \Pi$ such that $|\pi_i^{-1}(j) \cap K| = k'$ for all $j = 1, \dots, B/b$. Considering K as an indicator vector, we look up $K_{\pi_i^{-1}(1)}, \dots, K_{\pi_i^{-1}(B/b)}$ in $\mathcal{T}'$, which gives us B/b disjoint r'-sets, $R'_1, \dots, R'_{B/b}$. By construction of $\mathcal{T}$

there is an $R \in \mathcal{T}$ such that $R_{\pi_i^{-1}(j)} = R'_j$. Since $R \subseteq K$ we have proven $\mathcal{T}$ to be a correct (B, k, r)-system.

Decoding $\mathcal{T}$ is fast, since we only have to do $|\Pi|$ times B/b enumerations of $\mathcal{T}'$. When evaluating $\mathcal{T}(K)$ we make sure we return every r-set in K. Hence we return the entire "cross product" of unions:

$$\mathcal{T}(K) = \bigcup_i \{R_1 \cup \dots \cup R_{B/b} : \\ R_1 \in \mathcal{T}'(K_{\pi_i^{-1}(1)}), R_2 \in \dots\}.$$

In total this takes time $|\mathcal{T}(K)| + |\Pi|(B/b)m = |\mathcal{T}(K)| + \operatorname{poly}(B^{B/b}, e^{r^2/k\, b/B}, (B/k)^{rb/B})$, which was the last thing we needed to show lemma 5. ■

Before we apply the filter family to the problem, we will do dimensionality reduction by partitioning, as in the proof of corollary 1. In particular we use 4 with $B = 2\epsilon^{-2} d/(b_2 t) \log n$, so that we may assume far points have inner product at most $(1+\epsilon) b_2 t B/d$, all sets have size at most $(1+\epsilon)tB/d$ and for any close pair of points there is one partition in which they have inner product at least $b_1 t B/d$.

As in the analysis in the previous section, the time usage for queries and inserting points is dominated by two parts: 1) The complexity of evaluating $\mathcal{T}(x)$, and 2) The expected number of sets with inner product less than $b_2 t(1+\epsilon)$ that fall in the same filter as x.

Since lemma 4 randomly shuffled the coordinates, we can assume that x is a random weight $(1+\epsilon)tB/d$ set. The expected time to decode $C(x)$ is then

$$\begin{aligned} \operatorname{E}|T(x)| &= \sum_{R \in \mathcal{T}} \Pr_x[R \subseteq x] \\ &= |\mathcal{T}| \binom{(1+\epsilon)tB/d}{r} / \binom{B}{r} \\ &\le \operatorname{poly}(B^{B/b}, e^{r^2/k}, e^{\epsilon r})(tB/(dk))^r \end{aligned}$$

For estimating collisions with far points, we can similarly assume that x and y are random sets with inner product at most $(1+\epsilon) b_2 t B/d$.

$$\begin{aligned} &\sum_{Y \in P} \operatorname*{E}_{x,y} |\{R \in \mathcal{T} : R \in x \cap y\}| \\ &\quad = n|\mathcal{T}| \binom{(1+\epsilon)b_2 tB/d}{r} / \binom{B}{r} \\ &\quad \le \operatorname{poly}(B^{B/b}, e^{r^2/k}, e^{r\epsilon}) n (b_2 tB/(dk))^r \end{aligned}$$

And finally we have from the lemma, that the system needs $\operatorname{poly}(B^{B/b}, e^{r^2/k})(eB/k)^{kb/B}$ preprocessing.

We choose the parameters:

$$r = \frac{\log n}{\log(1/b_2)}, \quad B = 2\epsilon^{-2} d/(b_2 t) \log n, \quad k = b_1 t B/d,$$
$$b = d \log n/(b_1 t \log(eB/k)), \quad \epsilon = (b_1/b_2)^{1/3} (\log n)^{-1/3}$$

such that $(tB/(dk))^r = n(b_2 tB/(dk)) = n^{\log b_1/\log b_2}$, and the Turán system preprocessing is $(eB/k)^{kb/B} = n$.

Furthermore, the overhead terms are small:

$$\begin{aligned} B^{B/b} &= e^{(b_1/b_2)^{1/3} (\log n)^{2/3} \log(B) \log(B/k)} = n^{\hat{O}((\log n)^{-1/3})}, \\ e^{r\epsilon} &= e^{(b_1/b_2)^{1/3} (\log n)^{2/3} / \log(1/b_2)} = n^{\hat{O}((\log n)^{-1/3})}, \\ e^{r^2/k} &= n^{\hat{O}(\log n)^{-2/3}}. \end{aligned}$$

By the reduction mentioned in the beginning of the section, we make a data structure like the above for each of $(d+1)^2$ possible sizes for query and data sets. Hence the final construction takes space $O(d^2\, n^{\rho+o(1)})$ and has query time $O(d^2\, n^{\rho+o(1)})$, which is theorem 2.

V. Conclusion and Open Problems

We have seen that, perhaps surprisingly, there exists a relatively general way of creating efficient Las Vegas versions of state-of-the art high-dimensional search data structures.

As tools we found efficient, explicit constructions of large Turán systems and covering codes for pairs. We also showed an efficient way to do dimensionality reduction in hamming space without false negatives.

The work leaves a number of open questions for further research:

1) Can we make a completely deterministic high-dimensional data structure for the proposed problems? Cutting the number of random bits used for Las Vegas guarantees would likewise be interesting. The presented algorithms both use $O(d \log d)$ bits, but perhaps limited independence could be used to show that $O(\log d)$ suffice?
2) Does there exist Las Vegas data structures with performance matching that of data-dependent LSH data structures? This might connect to the previous question, since a completely deterministic data structure would likely have to be data-dependent. However the most direct approach would be to repeat [Andoni et al., 2017b], but find Las Vegas constructions for the remaining uses of Monte Carlo randomness, such as clustering.
3) By reductions, our results extend to ℓ_2 and ℓ_1 with exponent $n^{1/c}$. This is optimal for ℓ_1, but for ℓ_2 we would hope to get n^{1/c^2}. Can our techniques be applied to yield such a data structure? Are there other interesting LSH algorithms that can be made Las Vegas using our techniques? The author conjectures that a space/time trade-off version of the presented algorithm should follow easily following the approach of [Andoni et al., 2017b, Laarhoven, 2015, Christiani, 2017].
4) In the most general version, we we get the overhead term $(\log n)^{-1/4}$ in our ρ value. Some previous known LSH data structures also had large terms, such as [Andoni and Indyk, 2006], which had a $(\log n)^{-1/3}$ term and [Andoni et al., 2017b], which

has $(\log \log n)^{-\Theta(1)}$, but in general most algorithms have at most a $(\log n)^{-1/2}$ term.

Can we improve the overhead of the approach given in this paper? Alternatively, is there a completely different approach, that has a smaller overhead?

5) Finally, our data structure for the set-similarity search problem has a factor $(b_1/b_2)^{1/3}$ in its overhead term. This could potentially be a problem in cases where $b_1 >> b_2$, so it would be highly interesting to get rid of it. In the approach of this framework, that could happen either by finding a faster way to verify a Turán system, or by finding a more clever dimensionality reduction to apply as we did for Hamming distance.

A. Acknowledgements

The author would like to thank Rasmus Pagh, Tobias Christiani and the members of the Scalable Similarity Search project for many rewarding discussions on derandomization and set similarity data structures. Further thanks to Omri Weinstein, Rasmus Pagh, Martin Aumüller, Johan Sivertsen and the anonymous reviewers for useful comments on the manuscript; and to the people at University of Texas, Austin, for hosting me while doing this work.

The research leading to these results has received funding from the European Research Council under the European Union's 7th Framework Programme (FP7/2007-2013) / ERC grant agreement no. 614331.

References

[Ahle et al., 2017] Ahle, T. D., Aumüller, M., and Pagh, R. (2017). Parameter-free locality sensitive hashing for spherical range reporting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 239–256. SIAM.

[Alon et al., 2006] Alon, N., Moshkovitz, D., and Safra, S. (2006). Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms (TALG)*, 2(2):153–177.

[Andoni and Indyk, 2006] Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE.

[Andoni et al., 2015] Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., and Schmidt, L. (2015). Practical and optimal lsh for angular distance. In *Advances in Neural Information Processing Systems*, pages 1225–1233.

[Andoni et al., 2014] Andoni, A., Indyk, P., Nguyen, H. L., and Razenshteyn, I. (2014). Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1018–1028. Society for Industrial and Applied Mathematics.

[Andoni et al., 2017a] Andoni, A., Laarhoven, T., Razenshteyn, I., and Waingarten, E. (2017a). Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 47–66. SIAM.

[Andoni et al., 2017b] Andoni, A., Laarhoven, T., Razenshteyn, I., and Waingarten, E. (2017b). Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 47–66. Society for Industrial and Applied Mathematics.

[Andoni and Razenshteyn, 2015] Andoni, A. and Razenshteyn, I. (2015). Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, pages 793–801. ACM.

[Arasu et al., 2006] Arasu, A., Ganti, V., and Kaushik, R. (2006). Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, pages 918–929. VLDB Endowment.

[Arya et al., 1998] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923.

[Aumüller et al., 2017] Aumüller, M., Christiani, T., Pagh, R., and Silvestri, F. (2017). Distance-sensitive hashing. *arXiv preprint arXiv:1703.07867*.

[Becker et al., 2016] Becker, A., Ducas, L., Gama, N., and Laarhoven, T. (2016). New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24. SIAM.

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

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

[Broder et al., 1997] Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166.

[Charikar, 2002] Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing*, pages 380–388. ACM.

[Christiani, 2017] Christiani, T. (2017). A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 31–46. SIAM.

[Colbourn and Dinitz, 2006] Colbourn, C. J. and Dinitz, J. H. (2006). *Handbook of combinatorial designs*. CRC press.

[Cole et al., 2004] Cole, R., Gottlieb, L.-A., and Lewenstein, M. (2004). Dictionary matching and indexing with errors and don't cares. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 91–100. ACM.

[Datar et al., 2004] Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM.

[Dubiner, 2010] Dubiner, M. (2010). Bucketing coding and information theory for the statistical high-dimensional nearest-neighbor problem. *IEEE Transactions on Information Theory*, 56(8):4166–4179.

[Elias, 1957] Elias, P. (1957). List decoding for noisy channels. In *1957-IRE WESCON Convention Record, Pt.* Citeseer.

[Gionis et al., 1999] Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529. Morgan Kaufmann Publishers Inc.

[Goswami et al., 2017] Goswami, M., Pagh, R., Silvestri, F., and Sivertsen, J. (2017). Distance sensitive bloom filters without false negatives. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 257–269. SIAM.

[Har-Peled et al., 2012] Har-Peled, S., Indyk, P., and Motwani, R. (2012). Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350.

[Hoeffding, 1963] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30.

[Indyk, 2000a] Indyk, P. (2000a). Dimensionality reduction techniques for proximity problems. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 371–378. Society for Industrial and Applied Mathematics.

[Indyk, 2000b] Indyk, P. (2000b). *High-dimensional computational geometry*. PhD thesis, Stanford University.

[Indyk, 2001] Indyk, P. (2001). On approximate nearest neighbors under l_∞ norm. *Journal of Computer and System Sciences*, 63(4):627–638.

[Indyk, 2007] Indyk, P. (2007). Uncertainty principles, extractors, and explicit embeddings of l2 into l1. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 615–620. ACM.

[Indyk and Motwani, 1998] Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM.

[Karppa et al., 2016] Karppa, M., Kaski, P., Kohonen, J., and Catháin, P. Ó. (2016). Explicit correlation amplifiers for finding outlier correlations in deterministic subquadratic time. *Proceedings of the 24th European Symposium Of Algorithms*.

[Kushilevitz et al., 2000] Kushilevitz, E., Ostrovsky, R., and Rabani, Y. (2000). Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474.

[Laarhoven, 2015] Laarhoven, T. (2015). Tradeoffs for nearest neighbors on the sphere. *arXiv preprint arXiv:1511.07527*.

[Lv et al., 2007] Lv, Q., Josephson, W., Wang, Z., Charikar, M., and Li, K. (2007). Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 950–961. VLDB Endowment.

[Mairson, 1983] Mairson, H. G. (1983). The program complexity of searching a table. In *Foundations of Computer Science, 1983., 24th Annual Symposium on*, pages 40–47. IEEE.

[Mitzenmacher and Upfal, 2005] Mitzenmacher, M. and Upfal, E. (2005). *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press.

[Naor et al., 1995] Naor, M., Schulman, L. J., and Srinivasan, A. (1995). Splitters and near-optimal derandomization. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 182–191. IEEE.

[Overmars and van Leeuwen, 1981] Overmars, M. H. and van Leeuwen, J. (1981). Worst-case optimal insertion and deletion methods for decomposable searching problems. *Information Processing Letters*, 12(4):168–173.

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

[Pacuk et al., 2016] Pacuk, A., Sankowski, P., Wegrzycki, K., and Wygocki, P. (2016). Locality-sensitive hashing without false negatives for lp. In *International Computing and Combinatorics Conference*, pages 105–118. Springer.

[Pagh, 2016] Pagh, R. (2016). Locality-sensitive hashing without false negatives. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1–9. SIAM.

[Pagh and Christiani, 2017] Pagh, R. and Christiani, T. (2017). Beyond minhash for similarity search. Proceedings of the forty-ninth annual ACM symposium on Theory of computing.

[Panigrahy, 2006] Panigrahy, R. (2006). Entropy based nearest neighbor search in high dimensions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1186–1195. Society for Industrial and Applied Mathematics.

[Pham and Pagh, 2016] Pham, N. and Pagh, R. (2016). Scalability and total recall with fast coveringlsh. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1109–1118. ACM.

[Sidorenko, 1995] Sidorenko, A. (1995). What we know and what we do not know about turán numbers. *Graphs and Combinatorics*, 11(2):179–199.

[Topsøe, 2006] Topsøe, F. (2006). Some bounds for the logarithmic function. *Inequality theory and applications*, 4:137.

[Turán, 1961] Turán, P. (1961). Research problems. *Közl MTA Mat. Kutató Int*, 6:417–423.

[Williams, 2005] Williams, R. (2005). A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365.