Algorithms for Similarity Search and Pseudorandomness

Tobias Christiani

IT UNIVERSITY OF COPENHAGEN

Abstract

We study the problem of approximate near neighbor (ANN) search and show the following results:

- An improved framework for solving the ANN problem using locality-sensitive hashing, reducing the number of evaluations of locality-sensitive hash functions and the word-RAM complexity compared to the standard framework.

- A framework for solving the ANN problem with space-time tradeoffs as well as tight upper and lower bounds for the space-time tradeoff of framework solutions to the ANN problem under cosine similarity.

- A novel approach to solving the ANN problem on sets along with a matching lower bound, improving the state of the art. A self-tuning version of the algorithm is shown through experiments to outperform existing similarity join algorithms.

- Tight lower bounds for asymmetric locality-sensitive hashing which has applications to the approximate furthest neighbor problem, orthogonal vector search, and annulus queries.

- A proof of the optimality of a well-known Boolean locality-sensitive hashing scheme.

We study the problem of efficient algorithms for producing high-quality pseudorandom numbers and obtain the following results:

- A deterministic algorithm for generating pseudorandom numbers of arbitrarily high quality in constant time using near-optimal space.

- An randomized construction of a family of hash functions that outputs pseudorandom numbers of arbitarily high quality with space usage and running time nearly matching known cell-probe lower bounds.

Resumé

Vi undersøger et grundlæggende problem indenfor approksimativ søgning: tilnærmelsesvis nær nabo (TNN) problemet, og viser følgende resultater:

- En forbedret generel løsning af TNN problemet som reducerer antal evalueringer af afstandsfølsomme spredefunktioner.

- En generel løsning af TNN problemet som giver mulighed for tid-plads afvejning samt tætte øvre og nedre grænser for TNN problemet med tid-plads afvejning under kosinuslighed.

- En ny tilgang til løsning af TNN problemet på mængder samt en matchende nedre grænse. En adaptiv version af algoritmen til approksimativ sammenføjning vises ved eksperimenter at være konkurrencedygtig.

- Tætte nedre grænser for asymmetrisk afstandsfølsom spredning som har anvendelser til approksimativ søgning efter fjerne naboer, ortogonale vektorer, og annulus forespørgsler.

- Et optimalitetsbevis for en velkendt familie af Boolske afstandsfølsomme spredefunktioner.

Vi undersøger problemet at finde effektive algoritmer til produktion af pseudotilfældighed af høj kvalitet og opnår følgende resultater:

- En deterministisk algoritme til generation af pseudotilfældige tal af vilkårlig høj kvalitet i konstant tid og med tæt på optimalt pladsforbrug.

- En randomiseret konstruktion af en familie af spredefunktioner som afbilder til pseudotilfældige tal af vilkårlig høj kvalitet, med evalueringstid og pladsforbrug tæt på den nedre grænse.

Acknowledgements

All that is gold does not glitter,
Not all those who wander are lost;
The old that is strong does not wither,
Deep roots are not reached by the frost.

From the ashes, a fire shall be woken,
A light from the shadows shall spring;
Renewed shall be blade that was broken,
The crownless again shall be king.

J. R. R. Tolkien (1892–1973)

Contents

Chapter 1

Introduction

1.1 Part I: Similarity search

Similarity search in large collections of high-dimensional objects is a problem that is well-motivated by a numerous applications. Consider for example the representation of an image by a d-dimensional feature vector x, where each entry x_i denotes the fraction of pixels of color i in the image. Given a collection of images P and a query image q, we could for example be interested in finding the nearest neighbor of q: the image $x \in P$ such that the distance $\text{dist}(q, x)$ is minimized, for some appropriate choice of distance function. Applications of near neighbor search include:

- Classification: Given a collection P of labelled objects and an unlabelled object q, classify q according to the label of its nearest neighbor in P.

- Recommender systems: Find similar users, movies, songs, books etc. to be used for recommendation.

- Duplicate detection: Remove near-identical objects from a collection, for example duplicate web pages from the index of a search engine.

The trivial solution to the near neighbor problem would be to iterate through every $x \in P$ and compute $\text{dist}(q, x)$ while keeping track of the nearest neighbor found so far. If we let $n = |P|$ denote the size of our collection and assume that it takes time $O(d)$ to compute the distance between a pair of objects, then the trivial solution uses time $O(dn)$.

Suppose we are interested in preprocessing the collection P into a data structure that supports answering queries faster than the trivial solution. In two-dimensional Euclidean space there exists a solution based on the Voronoi diagram of P with space usage $O(n)$ and query time $O(\log n)$ [68]. In higher dimensions, the best known solutions to the nearest neighbor problem either suffer from space usage or query time that is exponential in d [86]. This phenomenon is known as the "curse of dimensionality" and has recently been substantiated by conditional hardness results [6, 67, 171, 140], showing for example that the problem of finding all nearest neighbors in a collection of n points in d-dimensional Euclidean space cannot be solved in time subquadratic in n when $d = \omega(\log n)$ unless the Strong Exponential Time Hypothesis (SETH) is false [170].

In order to efficiently solve similarity search problems in high dimensional spaces, researchers and practitioners have turned to approximate solutions. Instead of finding the exact nearest neighbor of a query point, we settle for finding a point that is some approximation factor $c > 1$ times further away than the nearest neighbor. The algorithms and data structures for similarity search in this thesis are primarily aimed at providing efficient solutions to the *approximate near neighbor* problem defined as follows:

Definition 1.1. Let $P \subseteq X$ be a collection of $|P| = n$ points in a distance space (X, dist). A solution to the (r, cr)-*near neighbor* problem is a data structure that supports the following query operation: Given a query $q \in X$ if there exists $x \in P$ with $\text{dist}(q, x) \leq r$ return $x' \in P$ with $\text{dist}(q, x') < cr$.

The (r, cr)-near neighbor problem differs from the nearest neighbor problem by searching for any point within a fixed radius r of the query point and allowing us to return points at distance up to cr even though better candidates exist. It will be convenient to also define the (s_1, s_2)-*similarity problem* as the natural equivalent of the (r, cr)-near neighbor problem where we measure similarities rather than distances, i.e., we wish to report find a point with similarity $\text{sim}(q, x) \geq s_1$ and we are willing to accept points with similarity $s_2 < s_1$.

By allowing an approximation factor $c > 1$ it is possible to solve the (r, cr)-near neighbor problem in Euclidean space (and many other spaces) with query time that is sublinear in n and polynomial in d using space polynomial in d and n [91, 66]. However, even approximation

has its limits when it comes to alleviating the curse of dimensionality. Rubinstein [140] has recently shown that unless SETH is false, for every choice of constants $\gamma, \delta > 0$ there exists $\varepsilon > 0$ such that a solution to the $(1 + \varepsilon)$-approximate near neighbor problem with $O(n^\gamma)$ preprocessing time must use query time $\Omega(n^\delta)$.

1.1.1 Locality-sensitive hashing

One of the most successful approaches for finding solutions to the approximate near neighbor problem in various spaces is known as *locality-sensitive hashing*, commonly abbreviated as LSH (see [12, 166] for more information). The idea behind locality-sensitive hashing is to construct a distribution $\mathcal{H}$ over functions $h \colon X \to R$ that are used to partition the space X. This randomized partitioning scheme is locality-sensitive in the sense that close points $x, y \in X$ are more likely to hash to the same part of a randomly sampled partition.

Definition 1.2 (Locality-sensitive hashing [91]). Let (X, dist) be a distance space and let $\mathcal{H}$ be a distribution over functions $h \colon X \to R$. We say that $\mathcal{H}$ is (r, cr, p_1, p_2)-*sensitive* if for $x, y \in X$ and $h \sim \mathcal{H}$ we have that:

- If $\text{dist}(x, y) \leq r$ then $\Pr[h(x) = h(y)] \geq p_1$.

- If $\text{dist}(x, y) \geq cr$ then $\Pr[h(x) = h(y)] \leq p_2$.

We can speed up approximate near neighbor searches at the cost of some additional preprocessing by partitioning the set of points P according to L randomly sampled locality-sensitive hash functions $h_1, \ldots, h_L$. A query for a point q proceeds by considering the points of P that collide with q under $h_1, \ldots, h_L$. Intuitively we want to sample enough hash functions such that the ball of radius r around every potential query point $q \in X$ is covered by the union of the parts $h_1^{-1}(q), \ldots, h_L^{-1}(q)$. This approach yields the following general LSH framework for solving the approximate near neighbor problem (for more details see Chapter 2).

Theorem 1.1 (Indyk-Motwani [91, 86], simplified). *Let $\mathcal{H}$ be (r, cr, p_1, p_2)-sensitive and let $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$, then there exists a solution to the (r, cr)-near neighbor problem using $O(n^{1+\rho})$ words of space and with query time dominated by $O(n^\rho \log n)$ evaluations of functions from $\mathcal{H}$.*

1.1.2   Examples

To further introduce locality-sensitive hashing and the approach to solving the approximate near neighbor problem used in this thesis, we will present three simple and powerful families of locality-sensitive hash functions: Bit-sampling by Indyk and Motwani [91], MinHash by Broder [35], and SimHash by Charikar [47]. Indyk, Broder, and Charikar received the 2012 ACM Paris Kanellakis Theory and Practice Award *"for their groundbreaking work on Locality-Sensitive Hashing that has had great impact in many fields of computer science including computer vision, databases, information retrieval, machine learning, and signal processing"* [1]. We proceed by describing each of these families in turn, introducing relevant notation as we go along.

Bit-sampling.   Indyk and Motwani introduced a simple family of locality-sensitive hash functions $\mathcal{H}_H$ for the d-dimensional Boolean hypercube $\{0,1\}^d$ under Hamming distance $\text{dist}_H(x,y) = |\{i \in [d] \mid x_i \neq y_i\}|$ where $[d]$ denotes the set $\{1, 2, \ldots, d\}$. We sample a function $h \sim \mathcal{H}$ by sampling i uniformly at random $[d]$ and setting $h(x) = x_i$. It is easy to see that a pair of points fail to collide under a random hash function $h(x) = x_i$ if and only if i is sampled from the set of coordinates where x and y differ.

$$\Pr_{h \sim \mathcal{H}_H}[h(x) = h(y)] = 1 - \Pr_{h \sim \mathcal{H}_H}[h(x) \neq h(y)] = 1 - \text{dist}_H(x,y)/d.$$

Suppose we want to use this function to solve the (r, cr)-near neighbor problem in Hamming space $(\{0,1\}^d, \text{dist}_H)$. Then, from Theorem 1.1 we optain a query exponent of

$$\rho = \frac{\log(1/(1 - r/d))}{\log(1/(1 - cr/d))} \leq 1/c$$

where the details behind the last inequality can be found in [86]. In conclusion, bit-sampling gives a solution to the (r, cr)-near neighbor problem in Hamming space with query time roughly $n^{1/c}$ and space usage and preprocessing time roughly $n^{1+1/c}$.

MinHash.   MinHash is a family of locality-sensitive hashing with applications to similarity search and similarity estimation on sets under Jaccard similarity. Given sets $x, y \subseteq [d]$ their Jaccard similarity is defined by $\text{sim}_J(x,y) = |x \cap y|/|x \cup y|$.

A random hash function h from the MinHash family $\mathcal{H}_J$ is specified by a random permutation of $[d]$ and hashes a set x to the first element of x in this permutation. The permutation can be specified by a uniformly random hash function $f\colon [d] \to [0,1]$ where $[0,1]$ denotes the closed interval from 0 to 1. Specifically, we sample a random from $h \sim \mathcal{H}_J$ from the MinHash family by sampling a uniformly random hash function $f\colon [d] \to [0,1]$ and setting

$$h(x) = \arg\min_{i \in x} f(i).$$

Two sets x and y collide under a random hash function $h \sim \mathcal{H}_J$ if and only if the smallest element of $x \cup y$ is contained in $x \cap y$. Otherwise, the smallest element of x is in $x \backslash y$ or the smallest element of y is in $y \backslash x$ and there is no way the sets hash to the same element. Since the smallest element of $x \cup y$ is uniformly distributed we get that

$$\Pr_{h \sim \mathcal{H}_J}[h(x) = h(y)] = \frac{|x \cap y|}{|x \cup y|} = \mathrm{sim}_J(x,y).$$

MinHash gives a solution to the (s_1, s_2)-similarity problem with exponent $\rho = \log(1/s_1)/\log(1/s_2)$.

SimHash. SimHash is a family of Boolean-valued locality-sensitive hash functions for $\mathbb{R}^d$ under cosine similarity $\mathrm{sim}_C(x,y) = \cos(\theta(x,y))$ where $\theta(x,y)$ denotes the angle between x and y. We sample a function $h \sim \mathcal{H}_C$ by sampling a d-dimensional standard normal random variable $z \sim \mathcal{N}^d(0,1)$ and setting

$$h(x) = \mathrm{sign}(\langle x,z \rangle).$$

Intuitively, we sample a random hyperplane that goes through the origin and hash points depending on which side of the hyperplane they are on (the sign of the inner product $\langle x,y \rangle = \sum_i x_i z_u$). Due to the rotational invariance of the standard normal distribution the properties of this scheme can be analyzed in two dimensions. The probability that two points on the unit circle are separated by a random line through the origin is exactly

$$\Pr_{h \sim \mathcal{H}_C}[h(x) = h(y)] = 1 - \theta(x,y)/\pi.$$

This scheme yields a solution to the (s_1, s_2)-similarity problem under cosine similarity with $\rho = \log(1 - \arccos(s_1)/\pi)/\log(1 - \arccos(s_2)/\pi)$.

1.1.3   Lower bounds

Given a space (X, dist) and distance thresholds r, cr we are interested in finding a (r, cr, p_1, p_2)-sensitive family with a value of $\rho = \log(1/p_1) / \log(1/p_2)$ that is as small as possible. The primary technique for deriving locality-sensitive hashing lower bounds has been Fourier analysis of Boolean functions under noisy inputs (see the excellent book by O'Donnell for a comprehensive introduction [120]). Lower bounds for locality-sensitive hashing schemes (distributions over functions) often follow from lower bounds on the behaviour of a single function $f \colon \{-1, 1\}^d \to R$ under randomly α-correlated inputs, defined as follows:

Definition 1.3. For $-1 \leq \alpha \leq 1$ and $x, y \in \{-1, 1\}^d$ we say that (x, y) is randomly α-correlated if x is uniformly distributed over $\{-1, 1\}^d$ and each component of y is i.i.d. according to

$$y_i = \begin{cases} x_i & \text{with probability } \frac{1+\alpha}{2}, \\ -x_i & \text{with probability } \frac{1-\alpha}{2}. \end{cases}$$

If two vectors (x, y) are randomly α-correlated their expected cosine similarity is α, and their expected Hamming distance is given by $(1 - \alpha)d/2$. As the dimensionality increases, the empirical correlation between x and y will be tightly concentrated around α.

Let $0 \leq \beta < \alpha < 1$ and consider a $((1 - \alpha)d/2, (1 - \beta)d/2, p_1, p_2)$-sensitive family $\mathcal{H}$ for Hamming space $(\{-1, 1\}^d, \mathrm{dist}_H)$. Combining lower bounds by O'Donnell et al. [122] and Andoni and Razenshteyn [19] (building on work by Motwani et al. [114]), we have that

$$\rho = \frac{\log(1/p_1)}{\log(1/p_2)} \geq \max\left(\frac{\log(1/\alpha)}{\log(1/\beta)}, \frac{1 - \alpha}{1 + \alpha - 2\beta} \right) - o_d(1). \tag{1.1}$$

The lower bounds require that p_2 is not too small as a function of d. In particular, they start breaking down if p_2 is exponentially small in d, but such families are typically not of interest for high-dimensional similarity search where we want $p_2 \approx 1/n$. For a more comprehensive discussion of this issue see [122].

Compared against different constructions of locality-sensitive hash families, the two lower bounds comprising equation (1.1) reveal interesting properties of the Boolean hypercube. As α, β approach 1 the lower bound of $\log(1/\alpha) / \log(1/\beta)$ is the larger of the two bounds.

If we convert the lower bound to Hamming distance we get that $\rho \geq \log(1/(1-2r/d))/\log(1/(1-2cr/d)) \approx 1/c$ for an (r, cr, p_1, p_2)-sensitive family when $r, cr \ll d$. This lower bound is tight against the bit-sampling LSH of Indyk and Motwani. The bit-sampling family can be described as randomly partitioning the Boolean hypercube into subcubes, so in a sense subcubes are an optimal "shape" for distinguishing between very short random walks and slightly longer random walks in the Boolean hypercube. The lower bound of $1/c$ in Hamming space gives a lower bound of $1/c^p$ for ℓ_p^d-spaces (vectors in $\mathbb{R}^d$ under the ℓ_p-norm $\|x-y\|_p = (\sum_i |x_i - y_i|^p)^{1/p}$). This follows from a direct embedding of the Boolean hypercube in ℓ_p-space.

As β approaches 0 the lower bound of $(1-\alpha)/(1+\alpha-2\beta)$ dominates. Converted to Hamming distance this bound becomes $\rho \geq 1/(2c-1)$. For $\beta = 0$ this is tight against existing constructions that use balls to partition the hypercube [73, 14, 13]. Loosely speaking, in this regime we see that balls in Hamming space are optimal for simultaneously minimizing volume (capturing 0-correlated points) while maximizing the probability of capturing positively correlated points.

In Hamming space, the family of locality-sensitive hash functions that give the best known upper bound on the ρ-value can essentially be described as follows: We sample a function $h \sim \mathcal{H}$ by sampling a sequence of d balls of radius slightly below $d/2$ with the center of each ball being sampled uniformly at random from $\{-1, 1\}^d$. A point $x \in \{-1, 1\}^d$ is then hashed to the index of the first ball in the sequence that contains x. As we increase d and decrease the radius of the balls, this scheme has a ρ-value for the $((1-\alpha)d/2, (1-\beta)d/2)$-near neighbor problem of

$$\rho = \frac{1-\alpha}{1+\alpha} \bigg/ \frac{1-\beta}{1+\beta} + o_d(1). \tag{1.2}$$

This scheme also works on the unit sphere if we replace the balls by spherical caps [157, 13]. The size of the gap between the lower bound in equation (1.1) and the upper bound (1.2) is shown in Figure 1.1. Since the gap is less than 0.06 it is difficult to argue that closing the gap would have huge practical implications, especially since the lower order terms in existing constructions exceed this for most realistic applications [13]. Nevertheless, considering the tools that have gone into proving the existing lower bounds, we believe that it is of fundamental mathematical interest to understand how to best separate β-correlated points from α-correlated points.
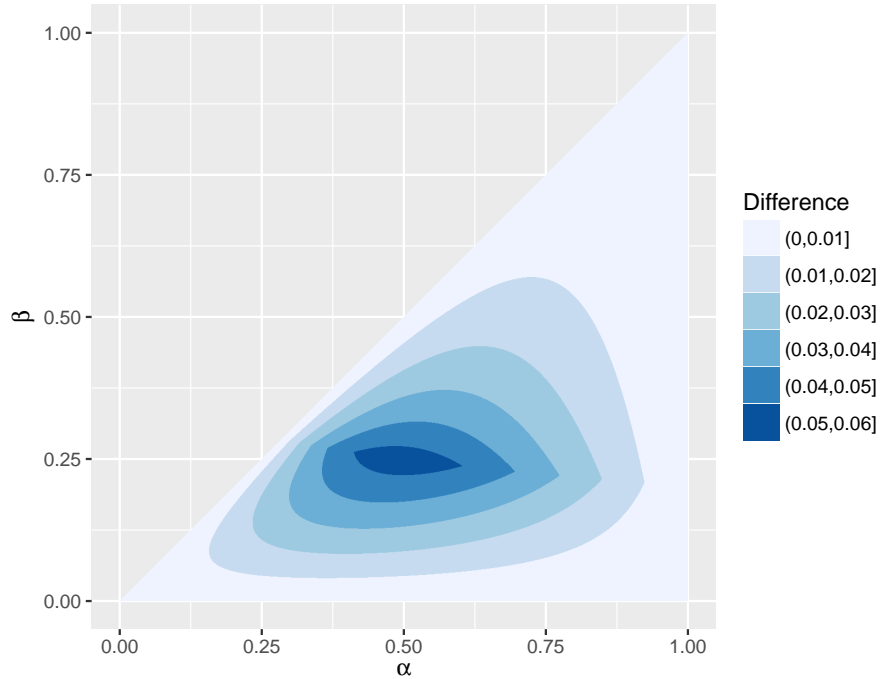
Figure 1.1: The gap in the ρ-value between the best known upper and lower bounds for families of $((1-\alpha)d/2, (1-\beta)d/2, p_1, p_2)$-sensitive hash functions in d-dimensional Hamming space.

1.1.4 Beyond locality-sensitive hashing

A common theme among recent advances in the area of theoretical approximate similarity search has been to move beyond standard locality-sensitive hashing [14, 100, 150, 24, 16, 54, 56, 22]. The results in this direction usually modify part of the framework, for example by constructing the locality-sensitive family by looking at the data, but the underlying approach of using locality-sensitive mappings from points to buckets remains the same. This thesis explores several variations of standard locality-sensitive hashing and we therefore briefly introduce some of this work here.

Data-dependent locality-sensitive hashing. A sequence of papers [14, 17, 19, 18, 16] has explored the idea of data-dependent locality-sensitive hashing: If we allow the construction of $\mathcal{H}$ to depend on the set of data points P, how fast can we then solve the approximate near neighbor problem? Andoni and Razenshteyn was able to show matching upper and lower bounds of $\rho = 1/(2c^2 - 1) + o_d(1)$ in Euclidean space [17, 19]. This matches standard LSH upper and lower bounds in the case of random

instances on the unit sphere, and indeed the construction by Andoni and Razenshteyn is based on a reduction to this case. Unfortunately the construction and its analysis is complicated and suffer from large lower order terms [16], although recent work has found some success in striking a balance between algorithmic simplicity and theoretical optimality using data-dependence in Hamming space [18].

Asymmetric locality-sensitive hashing.　Asymmetric locality-sensitive hashing extends the concept of standard locality-sensitive hashing to cover distributions over pairs of functions $(h, g) \sim \mathcal{A}$ and studies how the probability of collision between pairs of points can be made to depend on the distance/similarity between the points [150, 22]. This modification to standard locality-sensitive hashing opens up new applications such as approximate search for furthest neighbors, orthogonal vectors [164], and annulus queries (see [22] for an overview). In Chapter 6 we show lower bounds for asymmetric locality-sensitive hashing.

Space-time tradeoffs.　The standard locality-sensitive hashing framework offers a balanced space-time tradeoff that is the result of a symmetric query and update procedure: Every data point is stored in $O(n^\rho)$ buckets and during queries we probe $O(n^\rho)$ buckets. A line of work has investigated how the query and update parts of the algorithm can be modified to yield different tradeoffs between space usage and query time [130, 106, 9, 95, 100, 54, 16]. Typically the performance of such solutions is expressed by two exponents: ρ_u and ρ_q. During updates we store points in $O(n^{\rho_u})$ buckets and during queries we probe $O(n^{\rho_q})$ buckets.

Early work in this area focused on how to modify the standard locality-sensitive hashing query and update algorithms using an idea known as multi-probing [106]. Regular locality-sensitive hashing uses $L = O(n^\rho)$ hash functions $h_1, \ldots, h_L$. Suppose $h_l(q)$ denotes the lth bucket to be probed during the standard LSH query algorithm. By inspecting buckets in the neighborhood of $h_l(q)$, for example by adding some noise z to q and probing $h(q + z)$, we can increase the probability of finding a near neighbor of q, which in turn allows us to reduce L while maintaining correctness.

Recent breakthroughs in this area have come by abandoning the locality-sensitive hashing framework in favor of a more direct approach based on locality-sensitive filtering [100, 54]. Finally, Andoni et al. [16]

have combined their data-dependent approach to locality-sensitive hashing with the best known space-time tradeoff solutions for random data to obtain optimal space-time tradeoffs, as shown by matching lower bounds. The optimal trade-off between $\rho_q, \rho_u \geq 0$ for the (r, cr)-near neighbor problem in Euclidean space can be described by the equation

$$c^2 \sqrt{\rho_q} + (c^2 - 1) \sqrt{\rho_u} = \sqrt{2c^2 - 1}.$$

For a balanced tradeoff this collapses to $1/(2c^2 - 1)$ which is tight for data-dependent locality-sensitive hashing, but the bound has been shown to be tight for every choice of ρ_q, ρ_u that satisfies the equation.

Locality-sensitive filters and maps.   Locality-sensitive filtering [24] differs from locality-sensitive hashing in that it uses locality-sensitive subsets of space (filters) rather than locality-sensitive partitions (hash functions) to solve the approximate near neighbor problem. An example of a locality-sensitive filter family is the distribution over balls of some fixed radius in Hamming space. This idea is further extended to allow asymmetry by using different filters for queries and updates [100, 54]. It turns out that the filter family of consisting of pairs of concentric balls in Hamming space can be used to solve the approximate near-neighbor problem with optimal space-time tradeoffs, matching the lower bound of Andoni et al. [16] for random data. Chapter 3 further introduces locality-sensitive filtering and space-time tradeoffs.

   In even greater generality we can think of locality-sensitive hashing and filtering as being approaches to constructing randomized mappings $M: X \to 2^R$ (where 2^R denotes the power set of R) from a space (X, dist) to a collection of $|R|$ buckets that satisfy certain properties. Recent work on set similarity search (Chapter 4) and improvements to the standard locality-sensitive hashing framework (Chapter 2) explores these ideas and obtains efficient search algorithms by deviating from the standard approach.

1.2   Part II: Pseudorandom hashing and generators

The second part of this thesis contains results on efficient pseudorandom hash functions and pseudorandom number generators. We are interested in replacing the use of true randomness in randomized algorithms and data structures with the output of a pseudorandom hash function or

generator, stretching a small random seed into a much larger output of pseudorandom values, while retaining guarantees on the performance of these algorithms. For a primer on the general study of pseudorandom generators see [79].

Universal hashing.   The pseudorandomness part of this thesis focuses on one specific type of pseudorandomness known as k-wise independence or k-independence, first introduced to the field of computer science through the concept of universal hashing by Carter and Wegman [40].

Definition 1.4. Let k be a positive integer and let $\mathcal{F}$ be a family of functions from U to R. We say that $\mathcal{F}$ is a k-*independent* family of functions if for every choice of k distinct keys $x_1, \ldots, x_k$ and arbitrary values $y_1, \ldots, y_k$ we have that

$$\Pr_{f \sim \mathcal{F}}[f(x_1) = y_1 \wedge f(x_2) = y_2 \wedge \cdots \wedge f(x_k) = y_k] = |R|^{-k}.$$

Furthermore, we say that f is k-independent when it is selected uniformly at random from a family of k-independent functions.

We can sample a k-independent hash function $f(x) = \sum_{i=0}^{k-1} a_i x^i \mod p$ by sampling each a_i uniformly at random from the set $\{0, 1, \ldots, p-1\}$ where p is prime. In fact, the family of polynomials of degree at most $k-1$ over a finite field is k-independent [92]. We are typically interested in applications where the size of the universe $u = |U|$ is much larger than the degree of independence k.

Different types of hashing-based dictionaries work for k-independent hash functions with k much smaller than the number of elements in the dictionary which we denote by n. For example, it was shown that 5-independence suffices for linear probing to ensure expected constant time per operation [125]. It is known that $\Theta(\log n)$-independence suffices for Cuckoo hashing [128], but 5-independence is not enough to ensure constant amortized cost per operation [63]. For a brief introduction to the use of random hashing in algorithms and data structures see [69].

Fast hashing and lower bound.   For applications that require super-constant independence, the time to evelute the hash function can be a performance bottleneck. A k-independent polynomial hash function can be stored using $O(k)$ words and evaluated using time $O(k)$ on a word-RAM, assuming constant time arithmetic over the finite field. What

if we are willing to use more space to represent a k-independent hash function $f \sim \mathcal{F}$ in order to reduce the evaluation time? Siegel [152] gave a powerful cell-probe lower bound for this problem, showing that for a k-independent hash function with domain size u, even if we use space roughly $O(ku^{1/t})$ for some $t \geq 1$ the evaluation time has to be $\Omega(t)$.

Siegel also showed the existence of a matching upper bound based on highly unbalanced bipartite expander graphs $G = (U \cup V, E)$ with left vertex set U corresponding to the domain of the hash function, right vertex set V of size $|V| = O(ku^{1/t})$, and left outdegree $d = O(t)$. Given an appropriate expander graph G we can sample a k-independent hash function $f \colon U \to R$ by associating each vertex $v \in V$ with a random element from R where we assume that $(R, +)$ is an abelian group, such as the the integers under modular arithmetic. To compute $f(x)$ we take the sum of the random elements associated with the neighbors of the vertex $x \in U$ and return the result.

Unfortunately we only know of the existence of such optimal expander graphs by the probablistic method: a random bipartite graph has the right properties for optimal k-independent hashing with overwhelming probability if we parameterize the graph generation process correctly. Several works, Siegel's original paper included, attempt to approach the performance of such optimal bipartite expander graph by the use of probabilistic constructions [72, 124, 159, 58]. In Chapter 9 we show a probablistic construction with space usage and evaluation time that almost matches the lower bound. Finding optimal explicit constructions remains a major open problem.

Other approaches to the problem of finding fast hash functions with theoretical guarantees include the study of tabulation hashing and its variations which has guarantees beyond what can be derived from the degree of k-independence [160], to simulate uniformly random hashing in constant time on a subset of the universe [124], reusing randomness by splitting the problem into sub-problems that share a single highly random hash function [71], or extracting additional randomness from the input to the hash function [59].

Generating k-independent random variables. The generation of k-independent random variables differs from random hashing by allowing the algorithm designer to specify where to evaluate a k-independent function f in order to generate a sequence of variables $f(x_1), f(x_2), \ldots$ that is k-independent. The problem of generating a sequence of k-independent

random variables is therefore easier than the problem of constructing a data structure to represent a random k-independent hash function that an adversary can choose to evaluate in an arbitrary point.

We can take a standard k-independent polynomial hash function and evaluate it in k points in time $k \operatorname{poly} \log k$ using fast multipoint evaluation algorithms [27, 165], giving us a generator of k-independent random variables with generation time $\operatorname{poly} \log k$ per variable that uses space $O(k)$. This in itself shows that the task of generation is easier than hashing, as it would be impossible to evaluate a k-independent hash function in time $\operatorname{poly} \log k$ using space $O(k)$, if for example $k = \Theta(\log u)$. In Chapter 8 we show how to generate k-independent variables in constant time, independent of k, using space $k \operatorname{poly} \log k$.

1.3 Overview and contributions

This thesis is divided into two parts. The first part presents algorithms and lower bounds for various problems related to similarity search. The second part presents algorithms for the efficient generation of high-quality pseudorandom numbers, as well as efficient hash functions. The chapters are based on the following papers:

  I. Similarity search.

  2. Tobias Christiani: Fast locality-sensitive hashing frameworks for approximate near neighbor search [53]. 2017. Unpublished.

  3. Tobias Christiani: A Framework for Similarity Search with Space-Time Tradeoffs using Locality-Sensitive Filtering [54]. SODA 2017.

  4. Tobias Christiani and Rasmus Pagh: Set similarity search beyond MinHash [56]. STOC 2017.

  5. Tobias Christiani, Rasmus Pagh and Johan Sivertsen: Scalable and robust set similarity join [57]. ICDE 2018.

  6. Martin Aumüller, Tobias Christiani, Rasmus Pagh and Francesco Silvestri: Distance-sensitive hashing [22]. PODS 2018.

  7. Tobias Christiani: Optimal Boolean locality-sensitive hashing. 2018. Unpublished.

  II. Pseudorandomness.

8. Tobias Christiani and Rasmus Pagh: Generating k-independent variables in constant time [55]. FOCS 2014.

9. Tobias Christiani, Rasmus Pagh and Mikkel Thorup: From Independence to Expansion and Back Again [58]. STOC 2015.

We proceed by giving a brief description of the contribution of each chapter.

1.3.1   Part I: Similarity search

Chapter 2: Fast locality-sensitive hashing frameworks.   This chapter begins by surveying different techniques for constructing a solution to the approximate near neighbor problem from a family of locality-sensitive hash functions. Given a family $\mathcal{H}$ of locality-sensitive hash functions, the standard Indyk-Motwani framework (Theorem 1.1) uses $O(n^\rho \log n)$ functions from $\mathcal{H}$ to solve the approximate near neighbor problem. During a query all of these hash functions are evaluated, dominating the query time. For many LSH schemes the time to evaluate a single function is $O(d)$ or greater, as witnessed for example by SimHash or MinHash, further exacerbating the problem. Building on recent work by Dahlgaard et al. [64] we show that the number of locality-sensitive hash functions can be reduced to $O(\log^2 n)$ in general, yielding an improved LSH framework. We combine this result with a technique from another LSH framework by Andoni and Indyk [10] to reduce the word-RAM complexity of this improved framework by a logarithmic factor to $O(n^\rho)$.

Chapter 3: Space-time tradeoffs for similarity search.   This chapter introduces a framework for solving the approximate near neighbor problem with space-time tradeoffs using locality-sensitive filtering. We show concrete solutions on the unit sphere under cosine similarity with extensions to ℓ_p-space for every $0 < p \leq 2$. These results improve and generalize prior work [100, 95]. We also include a lower bound on space-time tradeoff that is tight, but suffers from some important restrictions. A paper by Andoni et al. [16] has since shown a strengthened lower bound and an improved upper bound through the use of data-dependent techniques. An early version of the paper behind this chapter formed part of my master's thesis. At the end of the chapter we have added an improved locality-sensitive filtering framework compared to the one in the main text, building on ideas introduced in Chapter 2 and 4.

Chapter 4: Set similarity search beyond MinHash. In this chapter we consider the problem of set similarity search under Braun-Blanquet similarity $\text{sim}_B(x, y) = |x \cap y| / \max(|x|, |y|)$. We show that the (s_1, s_2)-similarity problem in this setting can be solved with an exponent of $\rho = \log(1/s_1) / \log(1/s_2)$ and that this is tight among solutions based on data-independent locality-sensitive maps. The upper bound is based on a novel construction inspired by branching processes and interestingly, although it is data-independent, it outperforms the best known data-dependent techinques for a large portion of the parameter space $0 \leq s_2 < s_1 < 1$. The lower bound follows from a reduction to the standard (r, cr)-near neighbor problem in Hamming space for $r, cr \ll d/2$. In this setting the lower bound by O'Donnell et al. [122] is tight and we are able to show that it extends to Braun-Blanquet similarity for *every* choice of $0 \leq s_2 < s_1 < 1$. This is interesting in the light of the gap in our knowledge when it comes to the usual (α, β)-similarity problem for cosine similarity, as explained in the introduction.

Chapter 6: Lower bounds for asymmetric locality-sensitive hashing. In this chapter we derive lower bounds (on the ρ-value) for asymmetric locality-sensitive hashing. Our lower bound covers the case of asymmetric families for approximate near neighbor search, as well as the case of approximate furthest neighbor search where we are interested in having the collision probability of $(h, g) \sim \mathcal{A}$ increase in the distance between points. We show that our lower bounds are tight against existing symmetric constructions in the case of the application to near neighbor search, and that this construction can easily be modified to yield an optimal asymmetric construction for furthest neighbor search.

Chapter 7: Optimal Boolean locality-sensitive hashing. In this chapter we show that, among the class of Boolean locality-sensitive hash functions $h \colon \{-1, 1\}^d \to \{-1, 1\}$, bit-sampling is an optimal LSH (minimizes the ρ-value) for the $((1 - \alpha)d/2, (1 - \beta)d/2)$-near neighbor problem in Hamming space for every choice of $0 \leq \beta < \alpha < 1$. This stands in contrast to the lower bound by O'Donnell et al. [122] which is unrestricted with respect to the range of the locality-sensitive hash functions. Bit-sampling only matches this unrestricted lower bound in the case where α, β approach 1. Our result settles the question of optimal Boolean locality-sensitive hashing for Hamming space and shows that we have to look towards families of hash functions with a larger range in order to

further improve the ρ-value compared to bit-sampling. Andoni et al. [13] have shown lower bounds on the ρ-value on the unit sphere as a function of the size of the range of the hash function.

1.3.2   Part II: Pseudorandom hashing and number generation

Chapter 8:   Generating k-independent random variables in constant time.   We investigate the problem of efficiently generating k-independent random variables and give an explicit generator of k-independent random variables with constant generation time, independent of k. The explicit construction combines multipoint evaluation of polynomials over finite fields with a cascading construction of explicit bipartite expander graphs by Capalbo et al. [39]. The space usage of this construction is $k \operatorname{poly} \log k$ with a very large exponent in the polynomial. We also show a randomized version of the same construction that uses a randomly generated bipartite graph. This reduces the space overhead to $O(\log^3 k)$ at the cost of introducing an error probability (the generated sequence may fail to be k-independent) that is polynomially small in k. We implement a version of the generator that combines a random bipartite expander with fast multipoint evaluation of polynomials over $\mathbb{F}_{2^{64}}$ and show that it scales well, even for generating $k = 2^{20}$-independent variables.

Chapter 9: Near-optimal k-independent hashing.   In this chapter we attack the problem of constructing fast k-independent random hash functions. We use the fact that there is a sort of duality between randomized bipartite expander graphs and k-independent random hash functions. A bipartite expander graph that expands on subsets of size k can be used to construct a k-independent family of functions, and a k-independent function is likely to represent a bipartite expander that expands on subsets of size k. We take a small bipartite expander graph and apply an inefficient graph product that preserves its expansion properties while increasing the size of the left vertex set (the size of the domain of the resulting hash function). Then we use this resulting bipartite expander graph to construct a k-independent random hash function that now represents a new expander on a larger domain with optimal properties. By applying this strategy recursively using different graph products we are able to give randomized constructions of k-independent hash functions in the word-RAM model that almost match Siegel's cell probe lower bound [152].

1.4   Conclusion and open problems

1.4.1   Similarity search

We have shown new upper and lower bounds for problems related to approximate similarity search in high-dimensional spaces, showing improved locality-sensitive hashing frameworks, lower bounds for Boolean locality-sensitive hashing, and going beyond locality-sensitive hashing in several different directions with asymmetric locality-sensitive hashing, space-time tradeoffs through locality-sensitive filtering, and locality-sensitive maps for set similarity search.

Optimal data-independent locality-sensitive hashing.   It remains open to close the gap between the upper and lower bounds on the ρ-value of $((1-\alpha)d/2, (1-\beta)d/2, p_1, p_2)$-sensitive families in Hamming space (shown in Figure 1.1). Existing lower bounds seem to have explored the limits of what can be shown with our current understanding of hypercontractive inequalities and Fourier analysis of Boolean functions. We conjecture that the ball-based LSH construction with the ρ-value given in equation 1.2 is asymptotically optimal for every choice of $0 \le \beta < \alpha < 1$.

Orthogonal search.   Suppose we are interested in an asymmetric locality-sensitive hashing scheme for the unit sphere under cosine similarity that can be used to search for orthogonal vectors. For this purpose we want the probability of collison to be as high as possible for 0-correlated (orthogonal) vectors and have the probability of collision decrease at the correlation becomes positive or negative. Let $p(\alpha)$ denote the probability of collision of the asymmetric locality-sensitive hashing scheme for a pair of α-correlated vectors. The current best upper bound on $\rho = \log(1/p(0))/\log(1/\max(p(\alpha), p(-\alpha)))$ is given by $(1-\alpha^2)/(1+\alpha^2)$ [22]. The lower bound presented in Chapter 6 only implies $\rho \ge (1-|\alpha|)/(1+|\alpha|)$. Obtaining a "two-sided" lower bound that simultanously relates $p(0)$ to both $p(\alpha)$ and $p(-\alpha)$ has close ties to the open symmetric Gaussian problem [119]. It is conjectured that the upper bound is tight.

Simple data-dependent constructions.   It is an important open problem to find simpler data-dependent solutions to approximate near neighbor search. Despite the intuitive appeal of using the data to inform the

construction of the solution, relatively few people have succeded in making theoretical progress in this area [14, 17, 18]. Perhaps by relaxing the problem slightly, for example by only requiring that queries that follow a specific distribution succeed with constant probability, progress can be made. An example of such a query distribution could be to sample one of the n data points uniformly at random and sample the query from a ball around the data point. Attacking the problem for data structures that use near-linear space in n also seems like a promising approach.

1.4.2 k-independent hashing and generation

We have shown near-optimal results for k-independent hashing and generation.

Optimal explicit unbalanced bipartite expander graphs. The main open problem in this area is the explicit construction of highly unbalanced bipartite expander graphs with optimal properties. We would like to be able to evaluate the neighbor function $\Gamma \colon U \to V^d$ of a left d-regular bipartite expander graph with optimal parameters (matching Siegel's lower bound for k-independent hashing) using time that is at most polynomial in the bit-length of the input. For the application to random hashing we would furthermore like to be able to list the d neighbors of a vertex in time $O(d)$. The construction in Chapter 9 is essentially able to solve this task in time $O(d \log d)$, so it would require a very clean explicit construction to yield an improvement to the efficiency of random hashing in practice. Results on the construction of explicit bipartite expanders by Guruswami et al. [84] and preprocessing polynomials [96] are based directly on results such as the fundamental theorem of algebra and the Chinese remainder theorem and give hope that there exists a simple explicit construction.

Constant time generators with minimal space. The fast generators in Chapter 8 uses polynomials over finite fields and require space $k \operatorname{poly} \log k$. Through the sequential evaluation of hash functions presented in Chapter 9 we can remove the need for arithmetic over finite fields, but it seems that if we want to use minimal space the evalution time will still be $O(\log u)$ with space usage $k \operatorname{poly}(\log u, \log k)$. Is it possible to get constant-time generation in a restricted word-RAM model without multiplication using space $O(k)$?

Part I

Similarity search

Chapter 2

Fast locality-sensitive hashing frameworks

'Renewed shall be blade that was broken'

The Indyk-Motwani Locality-Sensitive Hashing (LSH) framework (STOC 1998) is a general technique for constructing a data structure to answer approximate near neighbor queries by using a distribution $\mathcal{H}$ over locality-sensitive hash functions that partition space. For a collection of n points, after preprocessing, the query time is dominated by $O(n^\rho \log n)$ evaluations of hash functions from $\mathcal{H}$ and $O(n^\rho)$ hash table lookups and distance computations where $\rho \in (0,1)$ is determined by the locality-sensitivity properties of $\mathcal{H}$. It follows from a recent result by Dahlgaard et al. (FOCS 2017) that the number of locality-sensitive hash functions can be reduced to $O(\log^2 n)$, leaving the query time to be dominated by $O(n^\rho)$ distance computations and $O(n^\rho \log n)$ additional word-RAM operations. We state this result as a general framework and provide a simpler analysis showing that the number of lookups and distance computations closely match the Indyk-Motwani framework. Using ideas from another locality-sensitive hashing framework by Andoni and Indyk (SODA 2006) we are able to reduce the number of additional word-RAM operations to $O(n^\rho)$.

2.1 Introduction

The (r_1, r_2)-approximate near neighbor problem is the problem of preprocessing a collection P of n points in a space (X, dist) into a data structure that after preprocessing supports the following query operation: Given a query point $q \in X$, if there exists a point $x \in P$ with $\text{dist}(q, x) \leq r_1$, then the data structure is guaranteed to return a point $x' \in P$ such that $\text{dist}(q, x') < r_2$.

Indyk and Motwani [91] introduced a general framework for constructing solutions to the approximate near neighbor problem using a technique known as locality-sensitive hashing (LSH). The framework takes a distribution over hash functions $\mathcal{H}$ with the property that near points are more likely to collide under a random $h \sim \mathcal{H}$. During preprocessing a number of locality-sensitive hash functions are sampled from $\mathcal{H}$ and used to hash the points of P into buckets. The query algorithm evaluates the same hash functions on the query point and looks into the associated buckets to find an approximate near neighbor.

The locality-sensitive hashing framework of Indyk and Motwani has had a large impact in both theory and practice (see surveys [12] and [166] for an introduction), and many of the best known (data-independent) solutions to the approximate near neighbor problem in high-dimensional spaces, such as Euclidean space [11], the unit sphere under inner product similarity [13], and sets under Jaccard similarity [33] come in the form of families of locality-sensitive hash functions that can be plugged into the Indyk-Motwani LSH framework. Recent work on data-dependent locality-sensitive hashing has further improved solutions for ℓ_p-spaces and cosine similarity [14, 17, 16], but these solutions typically do not come directly in the form of a distribution over locality-sensitive hash functions and as such it is unclear whether the techniques in this paper can yield further speedups to these results.

Definition 2.1 (Locality-sensitive hashing [91]). Let (X, dist) be a distance space and let $\mathcal{H}$ be a distribution over functions $h \colon X \to R$. We say that $\mathcal{H}$ is (r_1, r_2, p_1, p_2)-sensitive if for $x, y \in X$ and $h \sim \mathcal{H}$ we have that:

- If $\mathrm{dist}(x, y) \leq r_1$ then $\Pr[h(x) = h(y)] \geq p_1$.

- If $\mathrm{dist}(x, y) \geq r_2$ then $\Pr[h(x) = h(y)] \leq p_2$.

The Indyk-Motwani framework takes a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ and constructs a data structure that solves the approximate near neighbor problem for parameters $r_1 < r_2$ with some positive constant probability of success. We will refer to this randomized approximate version of the near neighbor problem as the (r_1, r_2)-near neighbor problem, where we require queries to succeed with probability at least $1/2$ (see Definition 2.2). To simplify the exposition we will assume throughout the introduction, unless otherwise stated, that $0 < p_1 < p_2 < 1$ are constant, that a hash function $h \in \mathcal{H}$ can be stored in $n / \log n$ words of space, and for

$\rho = \log(1/p_1)/\log(1/p_2) \in (0,1)$ that a point $x \in X$ can be stored in $O(n^\rho)$ words of space. The assumption of a constant gap between p_1 and p_2 allows us to avoid performing distance computations by instead using the 1-bit sketching scheme of Li and König [103] together with the family $\mathcal{H}$ to approximate distances (see Section 2.4.1 for details). In the remaining part of the paper we will state our results without any such assumptions to ensure, for example, that our results hold in the important case where p_1, p_2 may depend on n or the dimensionality of the space [11, 13].

Theorem 2.1 (Indyk-Motwani [91, 86], simplified)**.** *Let $\mathcal{H}$ be (r_1, r_2, p_1, p_2)-sensitive and let $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$, then there exists a solution to the (r_1, r_2)-near neighbor problem using $O(n^{1+\rho})$ words of space and with query time dominated by $O(n^\rho \log n)$ evaluations of functions from $\mathcal{H}$.*

The query time of the Indyk-Motwani framework is dominated by the number of evaluations of locality-sensitive hash functions. To make matters worse, almost all of the best known and most widely used locality-sensitive families have an evalution time that is at least linear in the dimensionality of the underlying space [33, 47, 66, 11, 13]. Significant effort has been devoted to the problem of reducing the evaluation complexity of locality-sensitive hash families [157, 75, 65, 13, 97, 148, 149, 64], while the question of how many independent locality-sensitive hash functions are actually needed to solve the (r_1, r_2)-near neighbor problem has received relatively little attention [10, 64].

This paper aims to bring attention to, strengthen, generalize, and simplify results that reduce the number of locality-sensitive hash functions used to solve the (r_1, r_2)-near neighbor problem. In particular, we will extract a general framework from a technique introduced by Dahlgaard et al. [64] in the context of set similarity search under Jaccard similarity, showing that the number of locality-sensitive hash functions can be reduced to $O(\log^2 n)$ in general. Reducing the number of locality-sensitive hash functions allows us to spend time $O(n^\rho / \log^2 n)$ per hash function evaluation without increasing the overall complexity of the query algorithm — something which is particularly useful in Euclidean space where the best known LSH upper bounds offer a tradeoff between the ρ-value that can be achieved and the evaluation complexity of the locality-sensitive hash function [11, 13, 97].

The main technical contribution of this paper is to reduce the word-RAM complexity of the general LSH framework from $O(n^\rho \log n)$ to

$O(n^\rho)$ by combining techniques from Dahlgaard et al. and Andoni and Indyk [10].

2.1.1   Related work

Indyk-Motwani.   The Indyk-Motwani framework uses $L = O(n^\rho)$ independent partitions of space, each formed by overlaying $k = O(\log n)$ random partitions induced by k random hash functions from a locality-sensitive family $\mathcal{H}$. The parameter k is chosen such that a random partition has the property that a pair of points $x, y \in X$ with $\text{dist}(x, y) \leq r_1$ has probability $n^{-\rho}$ of ending up in the same part of the partition, while a pair of points with $\text{dist}(x, y) \geq r_2$ has probability n^{-1} of colliding. By randomly sampling $L = O(n^\rho)$ such partitions we are able to guarantee that a pair of near points will collide with constant probability in at least one of them. Applying these L partitions to our collection of data points P and storing the result of each partition of P in a hash table we obtain a data structure that solves the (r_1, r_2)-near neighbor problem as outlined in Theorem 2.1 above. Section 2.3 and 2.3.1 contains a more complete description of LSH-based frameworks and the Indyk-Motwani framework.

Andoni-Indyk.   As previously mentioned, many locality-sensitive hash functions happen to have a super-constant evaluation time. This motivated Andoni and Indyk to introduce a replacement to the Indyk-Motwani framework in a paper on substring near neighbor search [10]. The key idea is to re-use hash functions from a small collection of size $m \ll L$ by forming all combinations of $\binom{m}{t}$ hash functions. This technique is also known as tensoring and has seen some use in the work on alternative solutions to the approximate near neighbor problem, in particular the work on locality-sensitive filtering [73, 24, 54]. By applying the tensoring technique the Andoni-Indyk framework reduces the number of hash functions to $O(\exp(\sqrt{\rho \log n \log \log n})) = n^{o(1)}$ as stated in Theorem 2.2.

Theorem 2.2 (Andoni-Indyk [10], simplified). *Let $\mathcal{H}$ be (r_1, r_2, p_1, p_2)-sensitive and let $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$, then there exists a solution to the (r_1, r_2)-near neighbor problem using $O(n^{1+\rho})$ words of space and with query time dominated by $O(\exp(\sqrt{\rho \log n \log \log n}))$ evaluations of functions from $\mathcal{H}$ and $O(n^\rho)$ other word-RAM operations.*

The paper by Andoni and Indyk did not state this result explicitly as a theorem in the same form as the Indyk-Motwani framework; the analysis made some implicit restrictive assumptions on p_1, p_2 and ignored integer constraints. Perhaps for these reasons the result does not appear to have received much attention, although it has seen some limited use in practice [153]. In Section 2.3.2 we present a slightly different version of the Andoni-Indyk framework together with an analysis that satisfies integer constraints, providing a more accurate assessment of the performance of the framework in the general, unrestricted case.

Dahlgaard-Knudsen-Throup.   The paper by Dahlgaard et al. [64] introduced a different technique for constructing the L hash functions/partitions from a smaller collection of m hash functions from $\mathcal{H}$. Instead of forming all combinations of subsets of size t as the Andoni-Indyk framework they instead sample k hash functions from the collection to form each of the L partitions. The paper focused on a particular application to set similarity search under Jaccard similarity, and stated the result in terms of a solution to this problem. In Section 2.3.3 we provide a simplified and tighter analysis to yield a general framework:

Theorem 2.3 (Dahlgaard-Knudsen-Thorup [64], simplified). *Let $\mathcal{H}$ be (r_1, r_2, p_1, p_2)-sensitive and let $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$, then there exists a solution to the (r_1, r_2)-near neighbor problem using $O(n^{1+\rho})$ words of space and with query time dominated by $O(\log^2 n)$ evaluations of functions from $\mathcal{H}$ and $O(n^\rho \log n)$ other word-RAM operations.*

The analysis of [64] indicates that the Dahlgaard-Knudsen-Thorup framework, when compared to the Indyk-Motwani framework, would use at least 50 times as many partitions (and a corresponding increase in the number of hash table lookups and distance computations) to solve the (r_1, r_2)-near neighbor problem with success probability at least $1/2$. Using elementary tools, the analysis in this paper shows that we only have to use twice as many partitions as the Indyk-Motwani framework to obtain the same guarantee of success.

Number of hash functions.   To provide some idea of the number of hash functions H used by the different frameworks, Figure 2.1 shows the value of $\log_2 H$ that is obtained by the Indyk-Motwani (IM), Andoni-Indyk (AI), and Dahlgaard-Knudsen-Thorup (DKT) frameworks according to the analysis in Section 2.3 for $p_1 = 1/2$ and every value of
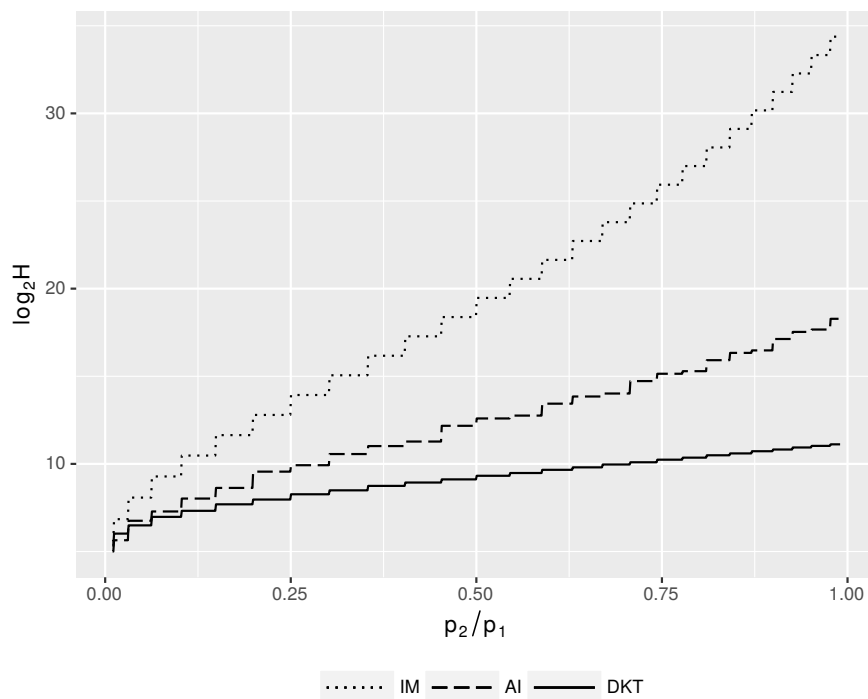
Figure 2.1: Upper bounds on the number of locality-sensitive hash functions from a $(r_1, r_2, 0.5, p_2)$-sensitive family used by different frameworks to solve the (r_1, r_2)-near neighbor problem on a collection of 2^{30} points according to the analysis in this paper.

$0 < p_2 < 1/2$ for a solution to the (r_1, r_2)-near neighbor problem on a collection of $n = 2^{30}$ points with success probability at least $1/2$. Note that Figure 2.1 shows an upper bound on the number of hash functions used by the frameworks according to the analysis in order to provide a solution with theoretical guarantees to the approximate near neighbor problem for any data set, and not the actual setting required for a particular data set (we haven't actually performed an experiment on 2^{30} points). In the analysis behind Figure 2.1 we have attempted to minimize H within each respective framework.

   Figure 2.1 reveals that the number of hash functions used by the Indyk-Motwani framework exceeds 2^{30}, the size of the collection of points P, as p_2 approaches p_1. In addition, locality-sensitive hash functions used in practice such as Charikar's SimHash [47] and p-stable LSH [66] have evaluation time $O(d)$ for points in $\mathbb{R}^d$. These two factors might help explain why a linear scan over sketches of the entire collection of points is a popular approach to solve the approximate near neighbor problem in practice [169, 80]. The Andoni-Indyk framework reduces the number

of hash functions by several orders of magnitude, and the Dahlgaard-Knudsen-Thorup framework presents another improvement of several orders of magnitude. Since the word-RAM complexity of the DKT framework matches the the number of hash functions used by the IM framework, the gap between the solid line (DKT) and the dotted line (IM) gives some indication of the time we can spend on evaluating a single hash function in the DKT framework without suffering a noticeable increase in the query time.

2.1.2 Contribution

Improved word-RAM complexity. In addition to our work on the Andoni-Indyk and Dahlgaard-Knudsen-Thorup frameworks as mentioned above, we show how the word-RAM complexity of the DKT framework can be reduced by a logarithmic factor. The solution is a simple combination of the DKT sampling technique and the AI tensoring technique: First we use the DKT sampling technique twice to construct two collections of $\sqrt{L}$ partitions. Then we use the AI tensoring technique to form $L = \sqrt{L} \times \sqrt{L}$ pairs of partitions from the two collections. Below we state our main Theorem 2.4 in its general form where we make no implicit assumptions about $\mathcal{H}$ (p_1 and p_2 are not assumed to be constant and can depend on for example n) or about the complexity of storing a point or a hash function, or computing the distance between pairs of points in the space (X, dist).

Theorem 2.4. *Let $\mathcal{H}$ be (r_1, r_2, p_1, p_2)-sensitive and let $\rho = \log(1/p_1)/\log(1/p_2)$, then there exists a solution to the (r_1, r_2)-near neighbor with the following properties:*

- *The query complexity is dominated by $O(\log^2_{1/p_2}(n)/p_1)$ evaluations of functions from $\mathcal{H}$, $O(n^\rho)$ distance computations, and $O(n^\rho/p_1)$ other word-RAM operations.*

- *The solution uses $O(n^{1+\rho}/p_1)$ words of space in addition to the space required to store the data and $O(\log^2_{1/p_2}(n)/p_1)$ functions from $\mathcal{H}$.*

Under the same simplifying assumptions used in the statements of Theorem 2.1, 2.2, and 2.3, our main Theorem 2.4 can be stated as Theorem 2.3 with the word-RAM complexity reduced by a logarithmic factor to $O(n^\rho)$. This improvement in the word-RAM complexity comes at the

cost of a (rather small) constant factor increase in the number of hash functions, lookups, and distance computations compared to the DKT framework. By varying the size m of the collection of hash functions from $\mathcal{H}$ and performing independent repetitions we can obtain a tradeoff between the number of hash functions and the number of lookups. In Section 2.5 we remark on some possible improvements in the case where p_2 is large.

Distance sketching using LSH. Finally, we combine Theorem 2.4 with the 1-bit sketching scheme of Li and König [103] where we use the locality-sensitive hash family to create sketches that allow us to leverage word-level parallelism and avoid direct distance computations. This sketching technique is well known and has been used before in combination with LSH-based approximate similarity search [57], but we believe there is some value in the simplicity of the analysis and in a clear statement of the combination of the two results as given in Theorem 2.5, for example in the important case where $0 < p_2 < p_1 < 1$ are constant.

Theorem 2.5. *Let $\mathcal{H}$ be (r_1, r_2, p_1, p_2)-sensitive and let $\rho = \log(1/p_1)/\log(1/p_2)$, then there exists a solution to the (r_1, r_2)-near neighbor with the following properties:*

- *The complexity of the query operation is dominated by $O(\log^2(n)/(p_1 - p_2)^2)$ evaluations of hash functions from $\mathcal{H}$ and $O(n^\rho/(p_1 - p_2)^2)$ other word-RAM operations.*

- *The solution uses $O(n^{1+\rho}/p_1 + n/(p_1 - p_2)^2)$ words of space in addition to the space required to store the data and $O(\log^2(n)/(p_1 - p_2)^2)$ hash functions from $\mathcal{H}$.*

2.2   Preliminaries

Problem and dynamization. We begin by defining the version of the approximate near neighbor problem that the frameworks presented in this paper will be solving:

Definition 2.2. Let $P \subseteq X$ be a collection of $|P| = n$ points in a distance space (X, dist). A solution to the (r_1, r_2)-near neighbor problem is a data structure that supports the following query operation: Given a query point $q \in X$, if there exists a point $x \in P$ with $\text{dist}(q, x) \leq r_1$, then, with probability at least $1/2$, return a point $x' \in P$ such that $\text{dist}(q, x') < r_2$.

We aim for solutions with a failure probability that is upper bounded by $1/2$. The standard trick of using η independent repetitions of the data structure allows us to reduce the probability of failure to $1/2^\eta$. For the sake of simplicity we restrict our attention to static solutions, meaning that we do not concern ourselves with the complexity of updates to the underlying set P, although it is simple to modify the static solutions presented in this paper to dynamic solutions where the update complexity essentially matches the query complexity [123, 86]

LSH powering. The Indyk-Motwani framework and the Andoni-Indyk framework will make use of the following standard powering technique described in the introduction as "overlaying partitions". Let $k \geq 1$ be an integer and let $\mathcal{H}$ denote a locality-sensitive family of hash functions as in Definition 2.1. We will use the notation $\mathcal{H}^k$ to denote the distribution over functions $h' \colon X \to R^k$ where

$$h'(x) = (h_1(x), \ldots, h_k(x))$$

and $h_1, \ldots, h_k$ are sampled independently at random from $\mathcal{H}$. It is easy to see that $\mathcal{H}^k$ is (r_1, r_2, p_1^k, p_2^k)-sensitive. To deal with some special cases we define $\mathcal{H}^0$ to be the family consisting of a single constant function.

Model of computation. We will work in the standard word-RAM model of computation [85] with a word length of $\Theta(\log n)$ bits where n denotes the size of the collection P to be searched in the (r_1, r_2)-near neighbor problem. During the preprocessing stage of our solutions we will assume access to a source of randomness that allows us to sample independently from a family $\mathcal{H}$ and to seed pairwise independent hash functions [41, 42]. The latter can easily be accomplished by augmenting the model with an instruction that generates a uniformly random word in constant time and using that to seed the tables of a Zobrist hash function [174].

2.3 Frameworks

Overview. We will describe frameworks that take as input a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ and a collection P of n points and constructs a data structure that solves the (r_1, r_2)-near neighbor problem. The frameworks described in this paper all use the same high-level technique

of constructing L hash functions $g_1, \ldots, g_L$ that are used to partition space such that a pair of points x, y with $\mathrm{dist}(x, y) \leq r_1$ will end up in the same part of one of the L partitions with probability at least $1/2$. That is, for x, y with $\mathrm{dist}(x, y) \leq r_1$ we have that $\Pr[\exists l \in [L]\colon g_l(x) = g_l(y)] \geq 1/2$ where $[L]$ is used to denote the set $\{1, 2, \ldots, L\}$. At the same time we ensure that the expected number of collisions between pairs of points x, y with $\mathrm{dist}(x, y) \geq r_2$ is at most one in each partition.

Preprocessing and queries. During the preprocessing phase, for each of the L hash functions $g_1, \ldots, g_L$ we compute the partition of the collection of points P induced by g_l and store it in a hash table in the form of key-value pairs $(z, \{x \in P \mid g_l(x) = z\})$. To reduce space usage we store only a single copy of the collection P and store references to P in our L hash tables. To guarantee lookups in constant time we can use the perfect hashing scheme by Fredman et al. [76] to construct our hash tables. We will assume that hash values $z = g_l(x)$ fit into $O(1)$ words. If this is not the case we can use universal hashing [40] to operate on fingerprints of the hash values.

We perform a query for a point q as follows: for $l = 1, \ldots, L$ we compute $g_l(q)$, retrieve the set of points $\{x \in P \mid g_l(x) = g_l(q)\}$, and compute the distance between q and each point in the set. If we encounter a point x' with $\mathrm{dist}(q, x') < r_2$ then we return x' and terminate. If after querying the L sets no such point is encountered we return a special symbol $\varnothing$ and terminate.

We will proceed by describing and analyzing the solutions to the (r_1, r_2)-near neighbor problem for different approaches to sampling, storing, and computing the L hash functions $g_1, \ldots, g_L$, resulting in the different frameworks as mentioned in the introduction.

2.3.1   Indyk-Motwani

To solve the (r_1, r_2)-near neighbor problem using the Indyk-Motwani framework we sample L hash functions $g_1, \ldots, g_L$ independently at random from the family $\mathcal{H}^k$ where we set $k = \lceil \log(n) / \log(1/p_2) \rceil$ and $L = \lceil (\ln 2) / p_1^k \rceil$. Correctness of the data structure follows from the observation that the probability that a pair of points x, y with $\mathrm{dist}(x, y) \leq r_1$ does not collide under a randomly sampled $g_l \sim \mathcal{H}^k$ is at most $1 - p_1^k$. We can therefore upper bound the probability that a near pair of points does not collide under any of the hash functions

by $(1 - p_1^k)^L \leq \exp(-p_1^k L) \leq 1/2$ using a standard bound stated as Lemma 2.3 in Appendix 2.7.

In the worst case, the query operation computes L hash functions from $\mathcal{H}^k$ corresponding to Lk hash functions from $\mathcal{H}$. For a query point q the expected number of points $x' \in P$ with $\text{dist}(q, x') \geq r_2$ that collide with q under a randomly sampled $g_l \sim \mathcal{H}^k$ is at most $np_2^k \leq np_2^{\log(n)/\log(1/p_2)} = 1$. It follows from linearity of expectation that the total expected number of distance computations during a query is at most L. The result is summarized in Theorem 2.6 from which the simplified Theorem 2.1 follows.

Theorem 2.6 (Indyk-Motwani [91, 86]). *Given a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ we can construct a data structure that solves the (r_1, r_2)-near neighbor problem such that for $k = \lceil \log(n)/\log(1/p_2) \rceil$ and $L = \lceil (\ln 2)/p_1^k \rceil$ the data structure has the following properties:*

- *The query operation uses at most Lk evaluations of hash functions from $\mathcal{H}$, expected L distance computations, and $O(Lk)$ other word-RAM operations.*

- *The data structure uses $O(nL)$ words of space in addition to the space required to store the data and Lk hash functions from $\mathcal{H}$.*

Theorem 2.6 gives a bound on the expected number of distance computations while the simplified version stated in Theorem 2.1 uses Markov's inequality and independent repetitions to remove the expectation from the bound by treating an excessive number of distance computations as a failure.

2.3.2   Andoni-Indyk

In 2006 Andoni and Indyk, as part of a paper on the substring near neighbor problem, introduced an improvement to the Indyk-Motwani framework that reduces the number of locality-sensitive hash functions [10]. Their improvement comes from the use of a technique that we will refer to as tensoring: setting the hash functions $g_1, \ldots, g_L$ to be all t-tuples from a collection of m functions sampled from $\mathcal{H}^{k/t}$ where $m \ll L$. The analysis in [10] shows that by setting $m = n^{\rho/t}$ and repeating the entire scheme $t!$ times, the total number of hash functions can be reduced to $O(\exp(\sqrt{\rho \log n \log \log n}))$ when setting $t = \sqrt{\frac{\rho \log n}{\log \log n}}$. This analysis ignores integer constraints on t, k, and m, and implicitly place restrictions

on p_1 and p_2 in relation to n (e.g. $0 < p_2 < p_1 < 1$ are constant). We will introduce a slightly different scheme that takes into account integer constraints and analyze it without restrictions on the properties of $\mathcal{H}$.

Assume that we are given a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$. Let $\eta, t, k_1, k_2, m_1, m_2$ be non-negative integer parameters. Each of the L hash functions $g_1, \ldots, g_L$ will be formed by concatenating one hash function from each of t collections of m_1 hash functions from $\mathcal{H}^{k_1}$ and concatenating a last hash function from a collection of m_2 hash functions from $\mathcal{H}^{k_2}$. We take all $m_1^t m_2$ hash functions of the above form and repeat η times for a total of $L = \eta m_1^t m_2$ hash functions constructed from a total of $H = \eta(m_1 k_1 t + m_2 k_2)$ hash functions from $\mathcal{H}$. In Appendix 2.8 we set parameters, leaving t variable, and provide an analysis of this scheme, showing that L matches the Indyk-Motwani framework bound of $O(1/p_1^k)$ up to a constant where $k = \lceil \log(n) / \log(1/p_2) \rceil$ as in Theorem 2.6.

Setting t. It remains to show how to set t to obtain a good bound on the number of hash functions H. Note that in practice we can simply set $t = \arg \min_t H$ by trying $t = 1, \ldots, k$. If we ignore integer constraints and place certain restrictions of $\mathcal{H}$ as in the original tensoring scheme by Andoni and Indyk we want to set t to minimize the expression $t^t n^{\rho/t}$. This minimum is obtained when setting t such that $t^2 \log t = \rho \log n$. We therefore cannot do much better than setting $t = \sqrt{\rho \log(n) / \log \log n}$ which gives the bound $H = O(\exp(\sqrt{\rho \log(n) \log \log n}))$ as shown in [10]. To allow for easy comparison with the Indyk-Motwani framework without placing restrictions on $\mathcal{H}$ we set $t = \lceil \sqrt{k} \rceil$, resulting in Theorem 2.7.

Theorem 2.7. *Given a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ we can construct a data structure that solves the (r_1, r_2)-near neighbor problem such that for $k = \lceil \log(n) / \log(1/p_2) \rceil$, $H = k(\sqrt{k}/p_1)^{\sqrt{k}}$, and $L = \lceil 1/p_1^k \rceil$ the data structure has the following properties:*

- *The query operation uses $O(H)$ evaluations of functions from $\mathcal{H}$, $O(L)$ distance computations, and $O(L + H)$ other word-RAM operations.*

- *The data structure uses $O(nL)$ words of space in addition to the space required to store the data and $O(H)$ hash functions from $\mathcal{H}$.*

Thus, compared to the Indyk-Motwani framework we have gone from using $O(k(1/p_1)^k)$ locality-sensitive hash functions to $O(k(\sqrt{k}/p_1)^{\sqrt{k}})$ locality-sensitive hash functions. Figure 2.1 shows the actual number of hash functions of the revised version of the Andoni-Indyk scheme as analyzed in Appendix 2.8 when t is set to minimize H.

2.3.3 Dahlgaard-Knudsen-Thorup

In a recent paper Dahlgaard et al. [64] introduce a different technique for reducing the number of locality-sensitive hash functions. The idea is to construct each hash value $g_l(x)$ by sampling and concatenating k hash values from a collection of km pre-computed hash functions from $\mathcal{H}$. Dahlgaard et al. applied this technique to provide a fast solution the approximate near neighbor problem for sets under Jaccard similarity. In this paper we use the same technique to derive a general framework solution that works with every family of locality-sensitive hash functions, reducing the number of locality-sensitive hash functions compard to the Indyk-Motwani and Andoni-Indyk frameworks.

Let $[n]$ denote the set of integers $\{1, 2, \ldots, n\}$. For $i \in [k]$ and $j \in [m]$ let $h_{i,j} \sim \mathcal{H}$ denote a hash function in our collection. To sample from the collection we use k pairwise independent hash functions [42] of the form $f_i \colon [L] \to [m]$ and set

$$g_l(x) = (h_{1,f_1(l)}(x), \ldots, h_{k,f_k(l)}(x)).$$

To show correctness of this scheme we will use make use of an elementary one-sided version of Chebyshev's inequality stating that for a random variable Z with mean $\mu > 0$ and variance $\sigma^2 < \infty$ we have that $\Pr[Z \leq 0] \leq \sigma^2/(\mu^2 + \sigma^2)$. For completeness we have included the proof of this inequality in Lemma 2.5 in Appendix 2.7. We will apply this inequality to lower bound the probability that there are no collisions between close pairs of points. For two points x and y let $Z_l = \mathbb{1}\{g_l(x) = g_l(y)\}$ so that $Z = \sum_{l=1}^{L} Z_l$ denotes the sum of collisions under the L hash functions. To apply the inequality we need to derive an expression for the expectation and the variance of the random variable Z. Let $p = \Pr_{h \sim \mathcal{H}}[h(x) = h(y)]$ then by linearity of expectation we have that $\mu = \mathbb{E}[Z] = Lp^k$. To bound $\sigma^2 = \mathbb{E}[Z^2] - \mu^2$ we proceed by bounding $\mathbb{E}[Z^2]$ where we note that $Z_l = \Pi_{i=1}^{k} Y_{l,i}$ for $Y_{l,i} = 1\{h_{i,f_i(l)}(x) = h_{i,f_i(l)}(x)\}$ and make use of the

independence between $Y_{l,i}$ and $Y_{l',i'}$ for $i \neq i'$.

$$\mathbb{E}[Z^2] = \sum_{\substack{l,l' \in [L] \\ l \neq l'}} \mathbb{E}[Z_l Z_{l'}] + \sum_{l=1}^{L} \mathbb{E}[Z_l]$$

$$= (L^2 - L) \, \mathbb{E}[Z_l Z_{l'}] + \mu$$

$$\leq L^2 \, \mathbb{E} \left[\Pi_{i=1}^{k} Y_{l,i} Y_{l',i} \right] + \mu$$

$$= L^2 \left(\mathbb{E}[Y_{l,i} Y_{l',i}] \right)^k + \mu.$$

We have that $\mathbb{E}[Y_{l,i}Y_{l',i}] = \Pr[f_i(l) = f_i(l')]p + \Pr[f_i(l) \neq f_i(l')]p^2 = (1/m)p + (1 - 1/m)p^2$ which follows from the pairwise independence of f_i. Let $\varepsilon > 0$ and set $m = \lceil \frac{1-p_1}{p_1} \frac{k}{\ln(1+\varepsilon)} \rceil$ then for $p \geq p_1$ we have that $(\mathbb{E}[Y_{l,i}Y_{l',i}])^k \leq (1 + \varepsilon)p^{2k}$. This allows us to bound the variance of Z by $\sigma^2 \leq \varepsilon\mu^2 + \mu$ resulting in the following lower bound on the probability of collision between similar points.

Lemma 2.1. *For $\varepsilon > 0$ let $m \geq \lceil \frac{1-p_1}{p_1} \frac{k}{\ln(1+\varepsilon)} \rceil$, then for every pair of points x, y with $\mathrm{dist}(x,y) \leq r_1$ we have that*

$$\Pr[\exists l \in [L] \colon g_l(x) = g_l(y)] \geq \frac{1 + \varepsilon\mu}{1 + (1 + \varepsilon)\mu}.$$

By setting $\varepsilon = 1/4$ and $L = \lceil (2\ln(2))/p_1^k \rceil$ we obtain an upper bound on the failure probability of $1/2$. Setting the size of each of the k collections of pre-computed hash values to $m = \lceil 5k/p_1 \rceil$ is sufficient to yield the following solution to the (r_1, r_2)-near neighbor problem where provide exact bounds on the number of lookups L and hash functions H:

Theorem 2.8 (Dahlgaard-Knudsen-Thorup [64])**.** *Given a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ we can construct a data structure that solves the (r_1, r_2)-near neighbor problem such that for $k = \lceil \log(n)/\log(1/p_2) \rceil$, $H = k\lceil 5k/p_1 \rceil$, and $L = \lceil (2\ln(2))/p_1^k \rceil$ the data structure has the following properties:*

- *The query operation uses at most H evaluations of hash functions from $\mathcal{H}$, expected L distance computations, and $O(Lk)$ other word-RAM operations.*

- *The data structure uses $O(nL)$ words of space in addition to the space required to store the data and H hash functions from $\mathcal{H}$.*

Compared to the Indyk-Motwani framework we have reduced the number of locality-sensitive hash functions H from $O(k(1/p_1)^k)$ to $O(k^2/p_1)$ at the cost of using twice as many lookups. To reduce the number of lookups further we can decrease ε and perform several independent repetitions. This comes at the cost of an increase in the number of hash functions H.

2.4   Reducing the word-RAM complexity

One drawback of the DKT framework is that each hash value $g_l(x)$ still takes $O(k)$ word-RAM operations to compute, even after the underlying locality-sensitive hash functions are known. This results in a bound on the total number of additional word-RAM operations of $O(Lk)$. We show how to combine the DKT universal hashing technique with the AI tensoring technique to ensure that the running time is dominated by $O(L)$ distance computations and $O(H)$ hash function evaluations. The idea is to use the DKT scheme to construct two collections of respectively L_1 and L_2 hash functions, and then to use the AI tensoring approach to form $g_1, \ldots, g_L$ as the $L = L_1 \times L_2$ combinations of functions from the two collections. The number of lookups can be reduced by applying tensoring several times in independent repetitions, but for the sake of simplicity we use a single repetition. For the usual setting of $k = \lceil \log(n)/\log(1/p_2) \rceil$ let $k_1 = \lceil k/2 \rceil$ and $k_2 = \lfloor k/2 \rfloor$. Set $L_1 = \lceil 6(1/p_1)^{k_1} \rceil$ and $L_2 = \lceil 6(1/p_1)^{k_2} \rceil$. According to Lemma 2.1 if we set $\varepsilon = 1/6$ the success probability of each collection is at least $3/4$ and by a union bound the probability that either collection fails to contain a colliding hash function is at most $1/2$. This concludes the proof of our main Theorem 2.4.

2.4.1   Sketching

The theorems of the previous section made no assumptions on the word-RAM complexity of distance computations and instead stated the number of distance computations as part of the query complexity. We can use a (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ to create sketches that allows us to efficiently approximate the distance between pairs of points, provided that the gap between p_1 and p_2 is sufficiently large. In this section we will re-state the results of Theorem 2.4 when applying the family $\mathcal{H}$ to create sketches using the 1-bit sketching scheme of Li and König [103].

Let b be a positive integer denoting the length of the sketches in bits. The advantage of this scheme is that we can use word level parallelism to evaluate a sketch of b bits in time $O(b/\log n)$ in our word-RAM model with word length $\Theta(\log n)$.

For $i = 1,\dots,b$ let $h_i\colon X \to R$ denote a randomly sampled locality-sensitive hash function from $\mathcal{H}$ and let $f_i\colon R \to \{0,1\}$ denote a randomly sampled universal hash function. We let $s(x) \in \{0,1\}^b$ denote the sketch of a point $x \in X$ where we set the ith bit of the sketch $s(x)_i = f_i(h(x))$. For two points $x,y \in X$ the probability that they agree on the ith bit is 1 if the points collide under h_i and $1/2$ otherwise.

$$\Pr[s(x)_i = s(y)_i] = \Pr[h_i(x) = h_i(y)] + (1 - \Pr[h_i(x) = h_i(y)])/2$$
$$= (1 + \Pr[h_i(x) = h_i(y)])/2.$$

We will apply these sketches during our query procedure instead of direct distance computations when searching through the points in the L buckets, comparing them to our query point q. Let $\lambda \in (0,1)$ be a parameter that will determine whether we report a point or not. For sketches of length b we will return a point x if $\|s(q) - s(x)\|_1 > \lambda b$. An application of Hoeffiding's inequality gives us the following properties of the sketch:

Lemma 2.2. *Let $\mathcal{H}$ be a (r_1, r_2, p_1, p_2)-sensitive family and let $\lambda = (1 + p_2)/2 + (p_1 - p_2)/4$, then for sketches of length $b \geq 1$ and for every pair points $x, y \in X$:*

- *If $\mathrm{dist}(x,y) \leq r_1$ then $\Pr[\|s(x) - s(y)\|_1 \leq \lambda b] \leq e^{-b(p_1-p_2)^2/8}$.*

- *If $\mathrm{dist}(x,y) \geq r_2$ then $\Pr[\|s(x) - s(y)\|_1 > \lambda b] \leq e^{-b(p_1-p_2)^2/8}$.*

If we replace the exact distance computations with sketches we want to avoid two events: Failing to report a point with $\mathrm{dist}(q,x) \leq r_1$ and reporting a point x with $\mathrm{dist}(q,x) \geq r_2$. By setting $b = O(\ln(n)/(p_1 - p_2)^2)$ and applying a union bound over the n events that the sketch fails for a point in our collection P we obtain Theorem 2.5.

2.5   The number of hash functions in corner cases

When the collision probabilities of the (r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ are close to one we get the behavior displayed in Figure 2.2 where we have
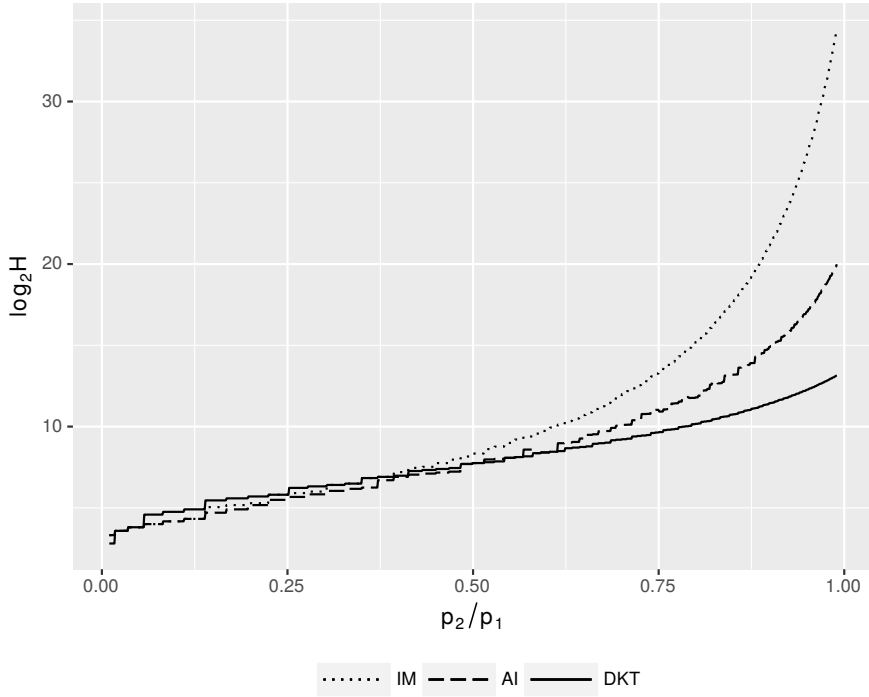
Figure 2.2: The number of locality-sensitive hash functions from a $(r_1, r_2, 0.9, p_2)$-sensitive family used by different frameworks to solve the (r_1, r_2)-near neighbor problem on a collection of 2^{30} points.

set $p_1 = 0.9$. Here it may be possible to reduce the number of hash functions by applying the DKT framework to the family $\mathcal{H}^\tau$ for some positive integer τ. That is, instead of applying the DKT technique directly to $\mathcal{H}$ we first apply the powering trick to produce the family $\mathcal{H}^\tau$. The number of locality-sensitive hash functions from $\mathcal{H}$ used by the DKT framework is given by $H = O((\log(n)/\log(1/p_2))^2/p_1)$. If we instead use the family $\mathcal{H}^\tau$ the expression becomes $H = O(\tau(\log(n)/\log(1/p_2^\tau))^2/p_1^\tau) = O((\log(n)/\log(1/p_2))^2/\tau p_1^\tau)$. Ignoring integer constraints, the value of τ that maximizes τp_1^τ, thereby minimizing H, is given by $\tau = 1/\ln(1/p_1)$. Discretizing, the resulting number of hash functions when setting $\tau = \lceil 1/\ln(1/p_1) \rceil$ is given by $H = O(\rho(\log n)^2/(p_1 \log(1/p_2)))$. For constant ρ and large p_2 this reduces the number of hash functions by a factor $1/\log(1/p_2)$. The behavior for small values of p_1 is displayed in Figure 2.3 where we have set $p_1 = 0.1$.
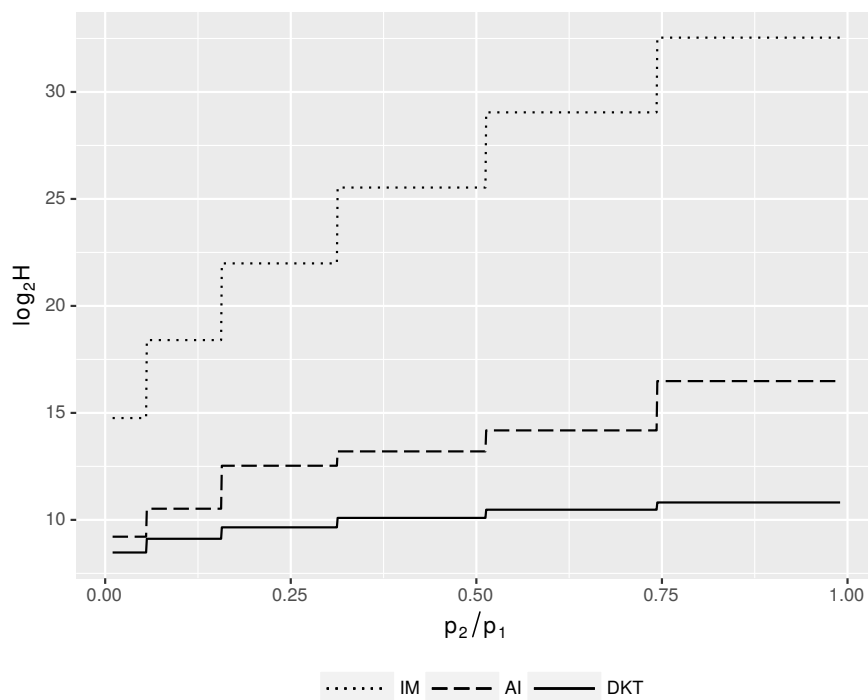
Figure 2.3: The number of locality-sensitive hash functions from a $(r_1, r_2, 0.1, p_2)$-sensitive family used by different frameworks to solve the (r_1, r_2)-near neighbor problem on a collection of 2^{30} points.

2.6   Conclusion and open problems

We have shown that there exists a simple and general framework for solving the (r_1, r_2)-near neighbor problem using only few locality-sensitive hash functions and with a reduced word-RAM complexity matching the number of lookups. The analysis in this paper indicates that the performance of the Dahlgaard-Knudsen-Thorup framework is highly competitive compared to the Indyk-Motwani framework in practice, especially when locality-sensitive hash functions are expensive to evaluate, as is often the case.

An obvious open problem is to provide a framework that uses fewer than $O(k^2/p_1)$ locality-sensitive hash function. Another direction would be to find a lower bound on the number of independent locality-sensitive hash functions required to solve the ANN problem using LSH in a suitably restricted model.

Acknowledgement

2.7 Appendix: Inequalities

We make use of the following standard inequalities for the exponential function. See [111, Chapter 3.6.2] for more details.

Lemma 2.3. *Let $n, t \in \mathbb{R}$ such that $n \geq 1$ and $|t| \leq n$ then $e^{-t}(1 - t^2/n) \leq (1 - t/n)^n \leq e^{-t}$.*

Lemma 2.4. *For $t \geq 0$ we have that $e^{-t} \leq 1 - t + t^2/2$.*

We make use of a one-sided version of Chebyshev's inequality to show correctness of the Dahlgaard-Knudsen-Thorup LSH framework.

Lemma 2.5 (Cantelli's inequality)**.** *Let Z be a random variable with $\mathbb{E}[Z] = \mu > 0$ and $\mathrm{Var}[Z] = \sigma^2 < \infty$ then $\Pr[Z \leq 0] \leq \sigma^2/(\mu^2 + \sigma^2)$.*

Proof. For every $s \in \mathbb{R}$ we have that

$$\Pr[Z \leq 0] = \Pr[-(Z - \mu) + s \geq \mu + s] \leq \Pr[(-(Z - \mu) + s)^2 \geq (\mu + s)^2].$$

Next we apply Markov's inequality

$$\Pr[(-(Z - \mu) + s)^2 \geq (\mu + s)^2] \leq \mathbb{E}[(-(Z - \mu) + s)^2]/(\mu + s)^2$$
$$= (\sigma^2 + s^2)/(\mu + s)^2$$

Set $s = \sigma^2/\mu$ and use that $\sigma^2 = s\mu$ to simplify

$$(\sigma^2 + s^2)/(\mu + s)^2 = (s\mu + s^2)/(\mu + s)^2 = \sigma^2/(\mu^2 + \sigma^2).$$

$\square$

To analyze the 1-bit sketching scheme by Li and König we make use of Hoeffding's inequality:

Lemma 2.6 (Hoeffding [88, Theorem 1])**.** *Let $X_1, X_2, \ldots, X_n$ be independent random variables satisfying $0 \leq X_i \leq 1$ for $i \in [n]$. Define $\bar{X} = (X_1 + X_2 + \cdots + X_n)/n$ and $\mu = \mathbb{E}[\bar{X}]$, then:*

- *For $0 < \varepsilon < 1 - \mu$ we have that $\Pr[\bar{X} - \mu \geq \varepsilon] \leq e^{-2n\varepsilon^2}$.*

- *For $0 < \varepsilon < \mu$ we have that $\Pr[\bar{X} - \mu \leq -\varepsilon] \leq e^{-2n\varepsilon^2}$.*

2.8   Appendix: Analysis of the Andoni-Indyk framework

Let φ denote the probability that a pair of points x, y with $\text{dist}(x,y) \leq r_1$ collide in a single repetition of the scheme. A collision occurs if and only if there there exists at least one hash function in each of the underlying $t + 1$ collections where the points collide. It follows that

$$\varphi = (1 - (1 - p_1^{k_1})^{m_1})^t (1 - (1 - p_1^{k_2})_2^m).$$

To guarantee a collision with probability at least $1/2$ it suffices to set $\eta = \lceil \ln(2)/\varphi \rceil$.

We will proceed by analyzing this scheme where we let $t \geq 1$ be variable and set parameters as followers:

$$k = \lceil \log(n) / \log(1/p_2) \rceil$$
$$k_1 = \lfloor k/t \rfloor$$
$$k_2 = k - tk_1$$
$$m_1 = \lceil 1/tp_1^{k_1} \rceil$$
$$m_2 = \lceil 1/p_1^{k_2} \rceil$$
$$\eta = \lceil \ln(2)/\varphi \rceil.$$

To upper bound L we begin by lower bounding φ. The second part of φ can be lower bounded using Lemma 2.3 to yield $(1 - (1 - p_1^{k_2})^{m_2}) \geq 1 - 1/e$. To lower bound $(1 - (1 - p_1^{k_1})^{m_1})^t$ we first note that in the case where $p_1^{k_1} > 1/t$ we have $m_1 = 1$ and the expression can be lower bounded by $p_1^{k_1 t} = (p_1^{k_1} m_1)^t \geq (p_1^{k_1} m_1)^t / 2e$. The same lower bound holds in the case there $t = 1$. In the case where $p_1^{k_1} \leq 1/t$ and $t \geq 2$ we make use of Lemma 2.3 and 2.4 to derive the lower bound.

$$1 - (1 - p_1^{k_1})^{m_1} \geq 1 - e^{-p_1^{k_1} m_1}$$
$$\geq 1 - (1 - p_1^{k_1} m_1 + (p_1^{k_1} m_1)^2 / 2)$$
$$\geq p_1^{k_1} m_1 (1 - p_1^{k_1} (1/tp_1^{k_1} + 1)/2)$$
$$\geq p_1^{k_1} m_1 (1 - 1/t).$$

Using the bound $(p_1^{k_1} m_1 (1 - 1/t))^t \geq (p_1^{k_1} m_1)^t / 2e$ we have that

$$\varphi \geq (p_1^{k_1} m_1)^t / 4e \geq (1/t)^t / 4e.$$

We can then bound the number of lookups and the expected number of distance computations

$$L = \eta m_1^t m_2 \leq (4e/(p_1^{k_1} m_1)^t + 1) m_1^t (1/p_1^{k_2} + 1) \leq 16e(1/p_1^k).$$

Note that this matches the upper bound of the Indyk-Motwani LSH framework up to a constant factor.

To bound the number of hash functions from $\mathcal{H}$ we use that $k_1 \leq k/t \leq k$ and $k_2 < t$.

$$H = \eta(m_1 k_1 t + m_2 k_2) \leq 8et^t \left(\frac{k}{t p_1^{k/t}} + \frac{t-1}{p_1^{t-1}} \right).$$

Chapter 3

Space-time tradeoffs for similarity search

'All that is gold does not glitter'

We present a framework for similarity search based on Locality-Sensitive Filtering (LSF), generalizing the Indyk-Motwani (STOC 1998) Locality-Sensitive Hashing (LSH) framework to support space-time tradeoffs. Given a family of filters, defined as a distribution over pairs of subsets of space that satisfies certain locality-sensitivity properties, we can construct a dynamic data structure that solves the approximate near neighbor problem on a collection of n points in d-dimensional space with query time $dn^{\rho_q+o(1)}$, update time $dn^{\rho_u+o(1)}$, and space usage $dn + n^{1+\rho_u+o(1)}$. The space-time tradeoff is tied to the tradeoff between query time and update time (insertions/deletions), controlled by the exponents ρ_q, ρ_u that are determined by the filter family.

Locality-sensitive filtering was introduced by Becker et al. (SODA 2016) together with a framework yielding a single, balanced, tradeoff between query time and space, further relying on the assumption of an efficient oracle for the filter evaluation algorithm. We extend the LSF framework to support space-time tradeoffs and through a combination of existing techniques we remove the oracle assumption.

Laarhoven (arXiv 2015), building on Becker et al., introduced a family of filters with space-time tradeoffs for the high-dimensional unit sphere under inner product similarity and analyzed it for the important special case of random data. We show that a small modification to the family of filters gives a simpler analysis that we use, together with our framework, to provide guarantees for worst-case data. Through an application of Bochner's Theorem from harmonic analysis by Rahimi & Recht (NIPS 2007), we are able to extend our solution on the unit sphere to $\mathbb{R}^d$ under

the class of similarity measures corresponding to real-valued character-istic functions. For the characteristic functions of s-stable distributions we obtain a solution to the (r, cr)-near neighbor problem in ℓ_s^d-spaces with query and update exponents $\rho_q = \frac{c^s(1+\lambda)^2}{(c^s+\lambda)^2}$ and $\rho_u = \frac{c^s(1-\lambda)^2}{(c^s+\lambda)^2}$ where $\lambda \in [-1, 1]$ is a tradeoff parameter. This result improves upon the space-time tradeoff of Kapralov (PODS 2015) and is shown to be optimal in the case of a balanced tradeoff, matching the LSH lower bound by O'Donnell et al. (ITCS 2011) and a similar LSF lower bound proposed in this paper. Finally, we show a lower bound for the space-time tradeoff on the unit sphere that matches Laarhoven's and our own upper bound in the case of random data.

3.1 Introduction

Let (X, dist) denote a space over a set X equipped with a symmetric measure of dissimilarity dist (a distance function in the case of metric spaces). We consider the (r, cr)-*near neighbor problem* first introduced by Minsky and Papert [110, p. 222] in the 1960's. A solution to the (r, cr)-near neighbor problem for a set P of n points in (X, dist) takes the form of a data structure that supports the following operation: given a query point $x \in X$, if there exists a data point $y \in P$ such that $\text{dist}(x, y) \leq r$ then report a data point $y' \in P$ such that $\text{dist}(x, y') \leq cr$. In some spaces it turns out to be convenient to work with a measure of similarity rather than dissimilarity. We use sim to denote a symmetric measure of similarity and define the (α, β)-*similarity problem* to be the $(-\alpha, -\beta)$-near neighbor problem in $(X, -\text{sim})$.

A solution to the (r, cr)-near neighbor problem can be viewed as a fundamental building block that yields solutions to many other similarity search problems such as the c-approximate *nearest* neighbor problem [89, 86]. In particular, the (r, cr)-near neighbor problem is well-studied in ℓ_s^d-spaces where the data points lie in $\mathbb{R}^d$ and distances are measured by $\text{dist}(x, y) = \|x - y\|_s = (\sum_{i=1}^d |x_i - y_i|^s)^{1/s}$. Notable spaces include the Euclidean space $(\mathbb{R}^d, \|\cdot\|_2)$, Hamming space $(\{0, 1\}^d, \|\cdot\|_1)$, and the d-dimensional unit sphere $\mathbb{S}^d = \{x \in \mathbb{R}^d \mid \|x\|_2 = 1\}$ under inner product (cosine) similarity $\text{sim}(x, y) = \langle x, y \rangle = \sum_{i=1}^d x_i y_i$.

Curse of dimensionality.   All known solutions to the (r, cr)-near neighbor problem for $c = 1$ (the exact near neighbor problem) either suffer

from a space usage that is exponential in d or a query time that is linear in n [86]. This phenomenon is known as the "curse of dimensionality" and has been observed both in theory and practice. For example, Alman and Williams [6] recently showed that the existence of an algorithm for determining whether a set of n points in d-dimensional Hamming space contains a pair of points that are exact near neighbors with a running time strongly subquadratic in n would refute the Strong Exponential Time Hypothesis (SETH) [170]. This result holds even when d is rather small, $d = O(\log n)$. From a practical point of view, Weber et al. [168] showed that the performance of many of the tree-based approaches to similarity search from the field of computational geometry [68] degrades rapidly to a linear scan as the dimensionality increases.

Approximation to the rescue. If we allow an approximation factor of $c > 1$ then there exist solutions to the (r, cr)-near neighbor problem with query time that is strongly sublinear in n and space polynomial in n where both the space and time complexity of the solution depends only polynomially on d. Techniques for overcoming the curse of dimensionality through approximation were discovered independently by Kushilevitz et al. [99] and Indyk and Motwani [91]. The latter, classical work by Indyk and Motwani [91, 86] introduced a general framework for solving the (r, cr)-near neighbor problem known as Locality-Sensitive Hashing (LSH). The introduction of the LSH framework has inspired an extensive literature (see e.g. [12, 166] for surveys) that represents the state of the art in terms of solutions to the (r, cr)-near neighbor problem in high-dimensional spaces [91, 47, 66, 130, 11, 12, 9, 13, 17, 95, 19, 24, 100].

Hashing and filtering frameworks. The LSH framework and the more recent LSF framework introduced by Becker et al. [24] produce data structures that solve the (r, cr)-near neighbor problem with query and update time $dn^{\rho+o(1)}$ and space usage $dn + n^{1+\rho+o(1)}$. The LSH (LSF) framework takes as input a distribution over partitions (subsets) of space with the locality-sensitivity property that close points are more likely to be contained in the same part (subset) of a randomly sampled element from the distribution. The frameworks proceed by constructing a data structure that associates each point in space with a number of memory locations or "buckets" where data points are stored. During a query operation the buckets associated with the query point are searched by computing the distance to every data point in the bucket, returning the

first suitable candidate. The set of memory locations associated with a particular point is independent of whether an update operation or a query operation is being performed. This symmetry between the query and update algorithm results in solutions to the near neighbor problem with a balanced space-time tradeoff. The exponent ρ is determined by the locality-sensitivity properties of the family of partitions/hash functions (LSH) or subsets/filters (LSF) and is typically upper bounded by an expression that depends only on the aproximation factor c. For example, Indyk and Motwani [91] gave a simple locality-sensitive family of hash functions for Hamming space with an exponent of $\rho \leq 1/c$. This exponent was later shown to be optimal by O'Donnell et al. [122] who gave a lower bound of $\rho \geq 1/c - o_d(1)$ in the setting where r and cr are small compared to d. The advantage of having a general framework for similarity search lies in the reduction of the (r, cr)-near neighbor problem to the, often simpler and easier to analyze, problem of finding a locality-sensitive family of hash functions or filters for the space of interest.

Space-time tradeoffs. Space-time tradeoffs for solutions to the (r, cr)-near neighbor problem is an active line of research that can be motivated by practical applications where it is desirable to choose the tradeoff between query time and update time (space usage) that is best suited for the application and memory hierarchy at hand [130, 106, 9, 95, 100]. Existing solutions typically have query time $dn^{\rho_q + o(1)}$, update time (insertions/deletions) $dn^{\rho_u + o(1)}$, and use space $dn + n^{1 + \rho_u + o(1)}$ where the query and update exponents ρ_q, ρ_u that control the space-time tradeoff depend on the approximation factor c and on a tradeoff parameter $\lambda \in [-1, 1]$.

This paper combines a number of existing techniques [24, 100, 73] to provide a general framework for similarity search with space-time tradeoffs. The framework is used to show improved upper bounds on the space-time tradeoff in the well-studied setting of ℓ_s-spaces and the unit sphere under inner product similarity. Finally, we show a new lower bound on the space-time tradeoff for the unit sphere that matches an upper bound for random data on the unit sphere by Laarhoven [100]. We proceed by stating our contribution and briefly surveying the relevant literature in terms of frameworks, upper bounds, and lower bounds as well as some recent developments. See table Table 3.1 for an overview.

Table 3.1: Overview of data-independent locality-sensitive hashing (LSH) and filtering (LSF) results

Reference	Setting	ρ_q	ρ_u
LSH [91, 86], LSF [24]	(X, dist), (X, sim)	$\dfrac{\log(1/p)}{\log(1/q)}$	
Theorem 3.1		$\dfrac{\log(p_q/p_1)}{\log(p_q/p_2)}$	$\dfrac{\log(p_u/p_1)}{\log(p_q/p_2)}$
Cross-poly. LSH [13]	(α, β)-sim., $(\mathsf{S}^d, \langle \cdot, \cdot \rangle)$	$\dfrac{1-\alpha}{1+\alpha} \Big/ \dfrac{1-\beta}{1+\beta}$	
Spherical cap LSF [100]	$(\alpha, o_d(1))$-sim., $(\mathsf{S}^d, \langle \cdot, \cdot \rangle)$	$\dfrac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2}$	$\dfrac{(\alpha^\lambda - \alpha)^2}{1-\alpha^2}$
Theorem 3.2	(α, β)-sim., $(\mathsf{S}^d, \langle \cdot, \cdot \rangle)$	$\dfrac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2} \Big/ \dfrac{(1-\alpha^\lambda\beta)^2}{1-\beta^2}$	$\dfrac{(\alpha^\lambda - \alpha)^2}{1-\alpha^2} \Big/ \dfrac{(1-\alpha^\lambda\beta)^2}{1-\beta^2}$
Ball-carving LSH [11]		$1/c^2$	
Ball-search LSH* [95]	(r, cr)-nn. in ℓ_2^d	$\dfrac{c^2(1+\lambda)^2}{(c^2+\lambda)^2 - c^2(1+\lambda^2)/2 - \lambda^2}$	$\dfrac{c^2(1-\lambda)^2}{(c^2+\lambda)^2 - c^2(1+\lambda^2)/2 - \lambda^2}$
Theorem 3.3		$\dfrac{c^2(1+\lambda)^2}{(c^2+\lambda)^2}$	$\dfrac{c^2(1-\lambda)^2}{(c^2+\lambda)^2}$
Lower bound [122]	LSH in ℓ_2^d	$\geq 1/c^2$	
Theorem 3.4	LSF in ℓ_2^d	$\geq 1/c^2$	
Lower bound [114, 19]	LSH in $(\mathsf{S}^d, \langle \cdot, \cdot \rangle)$	$\geq \dfrac{1-\alpha}{1+\alpha}$	
Theorem 3.5, [16]	LSF in $(\mathsf{S}^d, \langle \cdot, \cdot \rangle)$	$\geq \dfrac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2}$	$\geq \dfrac{(\alpha^\lambda - \alpha)^2}{1-\alpha^2}$

TABLE NOTES: Space-time tradeoffs for dynamic randomized solutions to similarity search problems in the LSH and LSF frameworks with query time $dn^{\rho_q+o(1)}$, update time $dn^{\rho_u+o(1)} + dn^{o(1)}$ and space usage $dn + n^{1+\rho_u+o(1)}$. Lower bounds are for the exponents ρ_q, ρ_u within their respective frameworks. Here $\varepsilon > 0$ denotes an arbitrary constant and $\lambda \in [-1, 1]$ controls the space-time tradeoff. We have hidden $o_n(1)$ terms in the upper bounds and $o_d(1)$ terms in the lower bounds.
*Assumes $c^2 \geq (1+\lambda)^2/2 + \lambda + \varepsilon$.

3.1.1 Contribution

Before stating our results we give a definition of locality-sensitive filtering that supports asymmetry in the framework query and update algorithm, yielding space-time tradeoffs.

Definition 3.1. Let (X, dist) be a space and let $\mathcal{F}$ be a probability distribution over $\{(Q, U) \mid Q \subseteq X, U \subseteq X\}$. We say that $\mathcal{F}$ is $(r, cr, p_1, p_2, p_q, p_u)$-sensitive if for all points $x, y \in X$ and (Q, U) sampled randomly from $\mathcal{F}$ the following holds:

- If $\mathrm{dist}(x,y) \leq r$ then $\Pr[x \in Q, y \in U] \geq p_1$.

- If $\mathrm{dist}(x,y) > cr$ then $\Pr[x \in Q, y \in U] \leq p_2$.

- $\Pr[x \in Q] \leq p_q$ and $\Pr[x \in U] \leq p_u$.

We refer to (Q,U) as a filter and to Q as the query filter and U as the update filter.

Our main contribution is a general framework for similarity search with space-time tradeoffs that takes as input a locality-sensitive family of filters.

Theorem 3.1. *Suppose we have access to a family of filters that is* $(r, cr, p_1, p_2, p_q, p_u)$-*sensitive. Then we can construct a fully dynamic data structure that solves the* (r, cr)-*near neighbor problem with query time* $dn^{\rho_q + o(1)}$, *update time* $dn^{\rho_u + o(1)}$, *and space usage* $dn + n^{1 + \rho_u + o(1)}$ *where* $\rho_q = \frac{\log(p_q/p_1)}{\log(p_q/p_2)}$ *and* $\rho_u = \frac{\log(p_u/p_1)}{\log(p_q/p_2)}$.

We give a worst-case analysis of a slightly modified version of Laarhoven's [100] filter family for the unit sphere and plug it into our framework to obtain the following theorem.

Theorem 3.2. *For every choice of* $0 \leq \beta < \alpha < 1$ *and* $\lambda \in [-1, 1]$ *there exists a solution to the* (α, β)-*similarity problem in* $(\mathrm{S}^d, \langle \cdot, \cdot \rangle)$ *that satisfies the guarantees from Theorem 3.1 with exponents* $\rho_q = \frac{(1 - \alpha^{1 + \lambda})^2}{1 - \alpha^2} \Big/ \frac{(1 - \alpha^\lambda \beta)^2}{1 - \beta^2}$ *and* $\rho_u = \frac{(\alpha^\lambda - \alpha)^2}{1 - \alpha^2} \Big/ \frac{(1 - \alpha^\lambda \beta)^2}{1 - \beta^2}$.

We show how an elegant and powerful application of Bochner's Theorem [141] by Rahimi and Recht [139] allows us to extend the solution on the unit sphere to a large class of similarity measures, yielding as a special case solutions for ℓ_s-space.

Theorem 3.3. *For every choice of* $c \geq 1$, $s \in (0, 2]$, *and* $\lambda \in [-1, 1]$ *there exists a solution to the* (r, cr)-*near neighbor problem in* ℓ_s^d *that satisfies the guarantees from Theorem 3.1 with exponents* $\rho_q = \frac{c^s (1 + \lambda)^2}{(c^s + \lambda)^2}$ *and* $\rho_u = \frac{c^s (1 - \lambda)^2}{(c^s + \lambda)^2}$.

This result improves upon the state of the art for every choice of asymmetric query/update exponents $\rho_q \neq \rho_u$ [130, 11, 9, 95]. We conjecture that this tradeoff is optimal among the class of algorithms that

independently of the data determine which locations in memory to probe during queries and updates. In the case of a balanced space-time tradeoff where we set $\rho_q = \rho_u$ our approach matches existing, optimal [122], data-independent solutions in ℓ_s-spaces [91, 66, 11, 117].

The LSF framework is very similar to the LSH framework, especially in the case where the filter family is symmetric ($Q = U$ for every filter in $\mathcal{F}$). In this setting we show that the LSH lower bound by O'Donnell et al. applies to the LSF framework as well [122], confirming that the results of Theorem 3.3 are optimal when we set $\rho_q = \rho_u$.

Theorem 3.4 (informal)**.** *Every filter family that is symmetric and* $(r, cr, p_1, p_2, p_q, p_u)$*-sensitive in ℓ_s^d must have* $\rho = \frac{\log(p_u/p_1)}{\log(p_q/p_2)} \geq 1/c^s - o_d(1)$ *when $r = \omega_d(1)$ is chosen to be sufficiently small.*

Finally we show a lower bound on the space-time tradeoff that can be obtained in the LSF framework. Our lower bound suffers from two important restrictions. First the filter family must be regular, meaning that all query filters and all update filters are of the same size. Secondly, the size of the query and update filter cannot differ by too much.

Theorem 3.5 (informal)**.** *Every regular filter family that is* $((1 - \alpha)d/2, (1 - \beta)d/2, p_1, p_2, p_q, p_u)$*-sensitive in d-dimensional Hamming space with asymmetry controlled by $\lambda \in [-1, 1]$ cannot simultaneously have that* $\rho_q < \frac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2} - o_d(1)$ *and* $\rho_u < \frac{(\alpha^\lambda - \alpha)^2}{1-\alpha^2} - o_d(1)$.

Together our upper and lower bounds imply that the filter family of concentric balls in Hamming space is asymptotically optimal for random data.

Techniques. The LSF framework in Theorem 3.1 relies on a careful combination of "powering" and "tensoring" techniques. For positive integers m and τ with $m \gg \tau$ the tensoring technique, a variant of which was introduced by Dubiner [73], allows us to simulate a collection of $\binom{m}{\tau}$ filters from a collection of m filters by considering the intersection of all τ-subsets of filters. Furthermore, given a point $x \in X$ we can efficiently list the simulated filters that contain x. This latter property is crucial as we typically need $\text{poly}(n)$ filters to split our data into sufficiently small buckets for the search to be efficient. The powering technique lets us amplify the locality-sensitivity properties of a filter family in the same way that powering is used in the LSH framework [91, 12, 122].

To obtain results for worst-case data on the unit sphere we analyze a filter family based on standard normal projections using the same techniques as Andoni et al. [13] together with existing tail bounds on bivariate Gaussians. The approximate kernel embedding technique by Rahimi and Recht [139] is used to extend the solution on the unit sphere to a large class of similarity measures, yielding Theorem 3.3 as a special case.

The lower bound in Theorem 3.4 relies on an argument of contradiction against the LSH lower bounds by O'Donnell [122] and uses a theoretical, inefficient, construction of a locality-sensitive family of hash functions from a locality-sensitive family of filters that is similar to the spherical LSH by Andoni et al. [14].

Finally, the space-time tradeoff lower bound from Theorem 3.5 is obtained through an application of an isoperimetric inequality by O'Donnell [121, Ch. 10] and is similar in spirit to the LSH lower bound by Motwani et al. [114].

3.1.2   Related work

The LSH framework takes a distribution $\mathcal{H}$ over hash functions that partition space with the property that the probability of two points landing in the same partition is an increasing function of their similarity.

Definition 3.2. Let (X, dist) be a space and let $\mathcal{H}$ be a probability distribution over functions $h\colon X \to R$. We say that $\mathcal{H}$ is (r, cr, p, q)-sensitive if for all points $x, y \in X$ and h sampled randomly from $\mathcal{H}$ the following holds:

- If $\text{dist}(x, y) \leq r$ then $\Pr[h(x) = h(y)] \geq p$.

- If $\text{dist}(x, y) > cr$ then $\Pr[h(x) = h(y)] \leq q$.

The properties of $\mathcal{H}$ determines a parameter $\rho < 1$ that governs the space and time complexity of the solution to the (r, cr)-near neighbor problem.

Theorem 3.6 (LSH framework [91, 86]). *Suppose we have access to a (r, cr, p, q)-sensitive hash family. Then we can construct a fully dynamic data structure that solves the (r, cr)-near neighbor problem with query time $dn^{\rho+o(1)}$, update time $dn^{\rho+o(1)}$, and with a space usage of $dn + n^{1+\rho+o(1)}$ where $\rho = \frac{\log(1/p)}{\log(1/q)}$.*

The LSF framework by Becker et al. [24] takes a symmetric $(r, cr, p_1, p_2, p_q, p_u)$-sensitive filter family $\mathcal{F}$ and produces a data structure that solves the (r, cr)-near neighbor problem with the same properties as the one produced by the LSH framework where instead we have $\rho = \frac{\log(p_q/p_1)}{\log(p_q/p_2)}$. In addition, the framework assumes access to an oracle that is able to efficiently list the relevant filters containing a point $x \in X$ out of a large collection of filters. The LSF framework in this paper removes this assumption, showing how to construct an efficient oracle as part of the framework.

In terms of frameworks that support space-time tradeoffs, Panigrahy [130] developed a framework based on LSH that supports the two extremes of the space-time tradeoff. In the language of Theorem 3.1, Panigrahy's framework supports either setting $\rho_u = 0$ for a solution that uses near-linear space at the cost of a slower query time, or setting $\rho_q = 0$ for a solution with query time $n^{o(1)}$ at the cost of a higher space usage. To obtain near-linear space the framework stores every data point in $n^{o(1)}$ partitions induced by randomly sampled hash functions from a (r, cr, p, q)-sensitive LSH family $\mathcal{H}$. In comparison, the standard LSH framework from Theorem 3.6 uses n^ρ such partitions where ρ is determined by $\mathcal{H}$. For each partition induced by $h \in \mathcal{H}$ the query algorithm in Panigrahy's framework generates a number of random points z in a ball around the query point x and searches the parts of the partition $h(z)$ that they hash to. The query time is bounded by $n^{\hat{\rho}+o(1)}$ where $\hat{\rho} = \frac{I(h(z)|x,h)}{\log(1/q)}$ and $I(h(z)|x, h)$ denotes conditional entropy, i.e. the query time is determined by how hard it is to guess where z hashes to given that we know x and h. Panigrahy's technique was used in a number of follow-up works that improve on solutions for specific spaces, but to our knowledge none of them state a general framework with space-time tradeoffs [106, 9, 95].

Upper bounds. As is standard in the literature we state results in ℓ_s-spaces in terms of the properties of a solution to the (r, cr)-near neighbor problem. For results on the unit sphere under inner product similarity $(\mathbb{S}^d, \langle \cdot, \cdot \rangle)$ we instead use the (α, β)-similarity terminology, defined in the introduction, as we find it to be cleaner and more intuitive while aligning better with the analysis. The ℓ_s-spaces, particularly ℓ_1 and ℓ_2, as well as $(\mathbb{S}^d, \langle \cdot, \cdot \rangle)$ are some of most well-studied spaces for similarity search and are also widely used in practice [166]. Furthermore, fractional norms (ℓ_s

for $s \neq 1, 2$) have been shown to perform better than the standard norms in certain use cases [2] which motivates finding efficient solutions to the near neighbor problem in general ℓ_s-space.

In the case of a balanced space-time tradeoff the best data-independent upper bound for the (r, cr)-near neighbor problem in ℓ_s^d are solutions with an LSH exponent of $\rho = 1/c^s$ for $0 < s \leq 2$. This result is obtained through a combination of techniques. For $0 < s \leq 1$ the LSH based on s-stable distributions by Datar et al. [66] can be used to obtain an exponent of $(1 + \varepsilon)/c^s$ for an arbitrarily small constant $\varepsilon > 0$. For $1 < s \leq 2$ the ball-carving LSH by Andoni and Indyk [11] for Euclidean space can be extended to ℓ_s using the technique described by Nguyen [117, Section 5.5]. Theorem 3.3 matches (and potentially improves in the case of $0 < s < 1$) these results with a single unified technique and analysis that we find to be simpler.

For space-time tradeoffs in Euclidean space (again extending to ℓ_s for $1 < s < 2$) Kapralov [95], improving on Panigrahy's results [130] in Euclidean space and using similar techniques, obtains a solution with query exponent $\rho_q = \frac{c^2(1+\lambda)^2}{(c^2+\lambda)^2 - c^2(1+\lambda^2)/2 - \lambda^2}$ and update exponent $\rho_u = \frac{c^2(1-\lambda)^2}{(c^2+\lambda)^2 - c^2(1+\lambda^2)/2 - \lambda^2}$ under the condition that $c^2 \geq (1+\lambda)^2/2 + \lambda + \varepsilon$ where $\varepsilon > 0$ is an arbitrary positive constant. Comparing to our Theorem 3.3 it is easy to see that we improve upon Kapralov's space-time tradeoff for all choices of c and λ. In addition, Theorem 3.3 represents the first solution to the (r, cr)-near neighbor problem in Euclidean space that for every choice of constant $c > 1$ obtains sublinear query time ($\rho_q < 1$) using only near-linear space ($\rho_u = 0$). Due to the restrictions on Kapralov's result he is only able to obtain sublinear query time for $c > \sqrt{3}$ when the space usage is restricted to be near-linear. It appears that our improvements can primarily be attributed to our techniques allowing a more direct analysis. Kapralov uses a variation of Panigrahy's LSH-based technique of, depending on the desired space-time tradeoff, either querying or updating additional memory locations around a point $x \in X$ in the partition induced by $h \in \mathcal{H}$. For a query point x and a near neighbor y his argument for correctness is based on guaranteeing that both the query algorithm and update algorithm visit the part $h(z)$ where z is a point lying between x and y, possibly leading to a loss of efficiency in the analysis. More details on the comparison of Theorem 3.3 to Kapralov's result can be found in Appendix 3.11.

In terms of space-time tradeoffs on the unit sphere, Laarhoven [100] modifies a filter family introduced by Becker et al. [24] to support space-

time tradeoffs, obtaining a solution for random data on the unit sphere (the (α, β)-similarity problem with $\beta = o_d(1)$) with query exponent $\rho_q = \frac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2}$ and update exponent $\rho_u = \frac{(\alpha^\lambda-\alpha)^2}{1-\alpha^2}$. Theorem 3.2 extends this result to provide a solution to the (α, β)-similarity problem on the unit sphere for every choice of $0 \leq \beta < \alpha < 1$. This extension to worst case data is crucial for obtaining our results for ℓ_s-spaces in Theorem 3.3. We note that there exist other data-independent techniques (e.g. Valiant [163, Alg. 25]) for extending solutions on the unit sphere to ℓ_2, but they also require a solution for worst-case data on the unit sphere to work.

Lower bounds   The performance of an LSH-based solution to the near neighbor problem in a given space that uses a (r, cr, p, q)-sensitive family of hash functions $\mathcal{H}$ is summarized by the value of the exponent $\rho = \frac{\log(1/p)}{\log(1/q)}$. It is therefore of interest to lower bound ρ in terms of the approximation factor c. Motwani et al. [114] proved the first lower bound for LSH families in d-dimensional Hamming space. They show that for every choice of $c \geq 1$ then for some choice of r it must hold that $\rho \geq 0.462/c$ as d goes to infinity under the assumption that q is not too small ($q \geq 2^{-o(d)}$).

As part of an effort to show lower bounds for data-dependent locality-sensitive hashing, Andoni and Razenshteyn [19] strengthened the lower bound by Motwani et al. to $\rho \geq 1/(2c-1)$ in Hamming space. These lower bounds are initially shown in Hamming space and can then be extended to ℓ_s-space and the unit sphere by the fact that a solution in these spaces can be used to yield a solution in Hamming space, contradicting the lower bound if ρ is too small. Translated to (α, β)-similarity on the unit sphere, which is the primary setting for the lower bounds on LSF space-time tradeoffs in this paper, the lower bound by Andoni and Razenshteyn shows that an LSH on the unit sphere must have $\rho \geq \frac{1-\alpha}{1+\alpha}$ which is tight in the case of random data [13].

The lower bound uses properties of random walks over a partition of Hamming space: A random walk starting from a random point $x \in \{-1, 1\}^d$ is likely to "walk out" of the the part identified by $h(x)$ in the partition induced by h. The space-time tradeoff lower bound in Theorem 3.5 relies on a similar argument that lower bounds the probability that a random walk starting from a subset Q ends up in another subset U, corresponding nicely to query and update filters in the LSF framework.

Using related techniques O'Donnell [122] showed tight LSH lower bounds for ℓ_s-space of $\rho \geq 1/c^s$. The work by Andoni et al. [15] and

Panigrahy et al. [131, 132] gives cell probe lower bounds for the (r, cr)-near neighbor problem, showing that in Euclidean space a solution with a query complexity of t probes require space at least $n^{1+\Omega(1/tc^2)}$. For more details on these lower bounds and how they relate to the upper bounds on the unit sphere see [16, 100].

Data-dependent solutions The solutions to the (r, cr)-near neighbor problems considered in this paper are all *data-independent*. For the LSH and LSF frameworks this means that the choice of hash functions or filters used by the data structure, determining the mapping between points in space and the memory locations that are searched during the query and update algorithm, is made without knowledge of the data. Data-independent solutions to the (r, cr)-near neighbor problem for worst-case data have been the state of the art until recent breakthroughs by Andoni et al. [14] and Andoni and Razenshteyn [17] showing improved solutions to the (r, cr)-near neighbor problem in Euclidean space using *data-dependent* techniques. In this setting the solution obtained by Andoni and Razenshteyn has an exponent of $\rho = 1/(2c^2 - 1)$ compared to the optimal data-independent exponent of $\rho = 1/c^2$. Furthermore, they show that this exponent is optimal for data-dependent solutions in a restricted model [19].

Recent developments Recent work by Andoni et al. [16], done independently of and concurrently with this paper, shows that Laarhoven's upper bound for random data on the unit sphere can be combined with data-dependent techniques [17] to yield a space-time tradeoff in Euclidean space with ρ_u, ρ_q satisfying $c^2 \sqrt{\rho_q} + (c - 1)\sqrt{\rho_u} = \sqrt{2c^2 - 1}$. This improves the result of Theorem 3.3 and matches the lower bound in Theorem 3.5. In the same paper they also show a lower bound matching our lower bound in Theorem 3.5. Their lower bound is set in a more general model that captures both the LSH and LSF framework and they are able to remove some of the technical restrictions such as the filter family being regular that weaken the lower bound in this paper. In spite of these results we still believe that this paper presents an important contribution by providing a general and simple framework with space-time tradeoffs as well as improved data-independent solutions to nearest neighbor problems in ℓ_s-space and on the unit sphere. We would also like to point out the simplicity and power of using Rahimi and Recht's [139] result to extend solutions on the unit sphere to spaces with similarity

measures corresponding to real-valued characteristic functions, further described in Appendix 3.9.

3.2 A framework with space-time tradeoffs

We use a combination of powering and tensoring techniques to amplify the locality-sensitive properties of our initial filter family, and to simulate a large collection of filters that we can evaluate efficiently. We proceed by stating the relevant properties of these techniques which we then combine to yield our Theorem 3.1.

Lemma 3.1 (powering). *Given a $(r, cr, p_1, p_2, p_q, p_u)$-sensitive filter family $\mathcal{F}$ for (X, dist) and a positive integer κ define the family $\mathcal{F}^\kappa$ as follows: we sample a filter $F = (Q, U)$ from $\mathcal{F}^\kappa$ by sampling $(Q_1, U_1), \ldots, (Q_\kappa, U_\kappa)$ independently from $\mathcal{F}$ and setting $(Q, U) = (\bigcap_{i=1}^\kappa Q_i, \bigcap_{i=1}^\kappa U_i)$. The family $\mathcal{F}^\kappa$ is $(r, cr, p_1^\kappa, p_2^\kappa, p_q^\kappa, p_u^\kappa)$-sensitive for (X, dist).*

Let $\mathbf{F}$ denote a collection (indexed family) of m filters and let $\mathbf{Q}$ and $\mathbf{U}$ denote the corresponding collections of query and update filters, that is, for $i \in \{1, \ldots, m\}$ we have that $\mathbf{F}_i = (\mathbf{Q}_i, \mathbf{U}_i)$. Given a positive integer $\tau \leq m$ (typically $\tau \ll m$) we define $\mathbf{F}^{\otimes \tau}$ to be the collection of filters formed by taking all the intersections of τ-combinations of filters from $\mathbf{F}$, that is, for every $I \subseteq \{1, \ldots, m\}$ with $|I| = \tau$ we have that

$$\mathbf{F}_I^{\otimes \tau} = (\textstyle\bigcap_{i \in I} \mathbf{Q}_i, \bigcap_{i \in I} \mathbf{U}_i)$$

The following properties of the tensoring technique will be used to provide correctness, running time, and space usage guarantees for the LSF data structure that will be introduced in the next subsection. We refer to the evaluation time of a collection of filters $\mathbf{F}$ as the time it takes, given a point $x \in X$ to prepare a list of query filters $\mathbf{Q}(x) \subseteq \mathbf{Q}$ containing x and a list of update filters $\mathbf{U}(x) \subseteq \mathbf{U}$ containing x such that the next element of either list can be reported in constant time. We say that a pair of points (x, y) is contained in a filter (Q, U) if $x \in Q$ and $y \in U$.

Lemma 3.2 (tensoring). *Let $\mathcal{F}$ be a filter family that is $(r, cr, p_1, p_2, p_q, p_u)$-sensitive in (X, dist). Let τ be a positive integer and let $\mathbf{F}$ denote a collection of $m = \lceil \tau / p_1 \rceil$ independently sampled filters from $\mathcal{F}$. Then the collection $\mathbf{F}^{\otimes \tau}$ of $\binom{m}{\tau}$ filters has the following properties:*

- *If (x, y) have distance at most r then with probability at least $1/2$ there exists a filter in $\mathbf{F}^{\otimes \tau}$ containing (x, y).*

- *If (x, y) have distance greater than cr then the expected number of filters in $\mathbf{F}^{\otimes \tau}$ containing (x, y) is at most $p_2^{\tau} \binom{m}{\tau}$.*

- *In expectation, a point x is contained in at most $p_q^{\tau} \binom{m}{\tau}$ query filters and at most $p_u^{\tau} \binom{m}{\tau}$ update filters in $\mathbf{F}^{\otimes \tau}$.*

- *The evaluation time and space complexity of $\mathbf{F}^{\otimes \tau}$ is dominated by the time it takes to evaluate and store m filters from $\mathcal{F}$.*

Proof. To prove the first property we note that there exists a filter in $\mathbf{F}^{\otimes \tau}$ containing (x, y) if at least τ filters in $\mathbf{F}$ contain (x, y). The binomial distribution has the property that the median is at least as great as the mean rounded down [93]. By the choice of m we have that the expected number of filters in $\mathbf{F}$ containing (x, y) is at least τ and the result follows. The second and third properties follow from the linearity of expectation and the fourth is trivial. $\qquad \square$

3.2.1 The LSF data structure

We will introduce a dynamic data structure that solves the (r, cr)-near neighbor problem on a set of points $P \subseteq X$. The data structure has access to a $(r, cr, p_1, p_2, p_q, p_u)$-sensitive filter family $\mathcal{F}$ in the sense that it knows the parameters of the family and is able to sample, store, and evaluate filters from $\mathcal{F}$ in time $dn^{o(1)}$.

The data structure supports an initialization operation that initializes a collection of filters $\mathbf{F}$ where for every filter we maintain a (possibly empty) set of points from X. After initialization the data structure supports three operations: INSERT, DELETE, and QUERY. The INSERT (DELETE) operation takes as input a point $x \in X$ and adds (removes) the point from the set of points associated with each update filter in $\mathbf{F}$ that contains x. The QUERY operation takes as input a point $x \in X$. For each query filter in $\mathbf{F}$ that contains x we proceed by computing the dissimilarity $\mathrm{dist}(x, y)$ to every point y associated with the filter. If a point y satisfying $\mathrm{dist}(x, y) \leq cr$ is encountered, then y is returned and the query algorithm terminates. If no such point is found, the query algorithm returns a special symbol "$\varnothing$" and terminates.

The data structure will combine the powering and tensoring techniques in order to simulate the collection of filters $\mathbf{F}$ from two smaller

collections: $\mathbf{F}_1$ consisting of m_1 filters from $\mathcal{F}^{\kappa_1}$ and $\mathbf{F}_2$ consisting of m_2 filters from $\mathcal{F}^{\kappa_2}$. The collection of simulated filters $\mathbf{F}$ is formed by taking all filters $(Q_1 \cap Q_2, U_1 \cap U_2)$ where (Q_1, U_1) is a member of $\mathbf{F}_1^{\otimes \tau}$ and (Q_2, U_2) is a member of $\mathbf{F}_2$. It is due to the integer constraints on the parameter τ in the tensoring technique and the parameter κ in the powering technique that we simulate our filters from two underlying collections instead of just one. This gives us more freedom to hit a target level of amplification of the simulated filters which in turn makes it possible for the framework to support efficient solutions for a wider range of parameters of LSF families.

The initialization operation takes $\mathcal{F}$ and parameters $m_1, \kappa_1, \tau, m_2, \kappa_2$ and samples and stores $\mathbf{F}_1$ and $\mathbf{F}_2$. The filter evaluation algorithm used by the insert, delete, and query operation takes a point $x \in X$ and computes for $\mathbf{F}_1$ and $\mathbf{F}_2$, depending on the operation, the list of update or query filters containing x. From these lists we are able to generate the list of filters in $\mathbf{F}$ containing x.

Setting the parameters of the data structure to guarantee correctness while balancing the contribution to the query time from the filter evaluation algorithm, the number of filters containing the query point, and the number of distant points examined, we obtain a partially dynamic data structure that solves the (r, cr)-near neighbor problem with failure probability $\delta \leq 1/2 + 1/e$. Using a standard dynamization technique by Overmars and Leeuwen [123, Thm. 1] we obtain a fully dynamic data structure resulting in Theorem 3.1. The details of the proof have been deferred to Appendix 3.7.

3.3   Gaussian filters on the unit sphere

In this section we show properties of a family of filters for the unit sphere $\mathbb{S}^d$ under inner product similarity. Later we will show how to make use of this family to solve the near neighbor problem in other spaces, including ℓ_s for $0 < s \leq 2$.

Lemma 3.3. *For every choice of $0 \leq \beta < \alpha < 1$, $\lambda \in [-1, 1]$, and $t > 0$ let $\mathcal{G}$ denote the family of filters defined as follows: we sample a filter (Q, U) from $\mathcal{G}$ by sampling $z \sim \mathcal{N}^d(0, 1)$ and setting*

$$Q = \{x \in \mathbb{R}^d \mid \langle x, z \rangle > \alpha^\lambda t\},$$
$$U = \{x \in \mathbb{R}^d \mid \langle x, z \rangle > t\}.$$

Then $\mathcal{G}$ is locality-sensitive on the unit sphere under inner product similarity with exponents

$$\rho_q \leq \left(\frac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2} + \frac{\ln(2\pi(1+t/\alpha)^2)}{t^2/2} \right) \bigg/ \frac{(1-\alpha^\lambda\beta)^2}{1-\beta^2} \, ,$$

$$\rho_u \leq \left(\frac{(\alpha^\lambda-\alpha)^2}{1-\alpha^2} + \frac{\ln(2\pi(1+t/\alpha)^2)}{t^2/2} \right) \bigg/ \frac{(1-\alpha^\lambda\beta)^2}{1-\beta^2} \, .$$

Laarhoven's filter family [100] is identical to $\mathcal{G}$ except that he normalizes the projection vectors z to have unit length. The properties of $\mathcal{G}$ can easily be verified with a simple back-of-the-envelope analysis using two facts: First, for a standard normal random variable Z we have that $\Pr[Z > t] \approx e^{-t^2/2}$. Secondly, the invariance of Gaussian projections $\langle x, z \rangle$ to rotations, allowing us to analyze the projection of arbitrary points $x, y \in \mathsf{S}^d$ with inner product $\langle x, y \rangle = \alpha$ in a two-dimensional setting $x = (1, 0)$ and $y = (\alpha, \sqrt{1-\alpha^2})$ without any loss of generality. The proof of Lemma 3.3 as well as the proof of Theorem 3.2 has been deferred to Appendix 3.8.

3.4   Space-time tradeoffs under kernel similarity

In this section we will show how to combine the Gaussian filters for the unit sphere with kernel approximation techniques in order to solve the (α, β)-similarity problem over $(\mathbb{R}^d, S)$ for the class of similarity measures of the form $S(x, y) = k(x - y)$ where $k \colon \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a real-valued characteristic function [162]. For this class of functions there exists a feature map ψ into a (possibly infinite-dimensional) dot product space such that $k(x, y) = \langle \psi(x), \psi(y) \rangle$. Through an elegant combination of Bochner's Theorem and Euler's Theorem, detailed in Appendix 3.9, Rahimi and Recht [139] show how to construct approximate feature maps, i.e., for every k we can construct a function v with the property that $\langle v(x), v(y) \rangle \approx \langle \psi(x), \psi(y) \rangle = k(x - y)$. We state a variant of their result for a mapping onto the unit sphere.

Lemma 3.4. *For every real-valued characteristic function k and every positive integer l there exists a family of functions $\mathcal{V} \subseteq \{v \mid v \colon \mathbb{R}^d \to \mathsf{S}^l\}$ such that for every $x, y \in \mathbb{R}^d$ and $\varepsilon > 0$ we have that*

$$\Pr_{v \sim \mathcal{V}}[|\langle v(x), v(y) \rangle - k(x, y)| \geq \varepsilon] \leq e^{-\Omega(l\varepsilon^2)}.$$

Theorem 3.10 in Appendix 3.9 shows that Theorem 3.2 holds with the space $(\mathsf{S}^d, \langle \cdot, \cdot \rangle)$ replaced by $(\mathbb{R}^d, k)$.

3.4.1   Tradeoffs in ℓ_s^d-space

Consider the (r, cr)-near neighbor problem in ℓ_s^d for $0 < s \leq 2$. We solve this problem by first applying the approximate feature map from Lemma 3.4 for the characteristic function of a standard s-stable distribution [175], mapping the data onto the unit sphere, and then applying our solution from Theorem 3.2 to solve the appropriate (α, β)-similarity problem on the unit sphere. The characteristic functions of s-stable distributions take the following form:

Lemma 3.5 (Lévy [102]). *For every positive integer d and $0 < s \leq 2$ there exists a characteristic function $k \colon \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ of the form*

$$k(x, y) = k(x - y) = e^{-\|x - y\|_s^s}.$$

A result by Chambers et al. [46] shows how to sample efficiently from an s-stable distributions.

To sketch the proof of Theorem 3.3 we proceed by upper bounding the exponents ρ_q, ρ_u from Theorem 3.2 when applying Lemma 3.4 to get $\alpha \geq e^{-r^s} - \varepsilon$ and $\beta \leq e^{-c^s r^s} - \varepsilon$. We make use of the following standard fact (see e.g. [143]) that can be derived from the Taylor expansion of the exponential function: for $x \geq 0$ it holds that $1 - x \leq e^{-x} \leq 1 - x + x^2/2$. Scaling the data points such that $r^s = o(1)$ and inserting the above values of $\alpha \approx 1 - r^s$ and $\beta \approx 1 - c^s r^s$ into the expressions for ρ_q, ρ_u in Lemma 3.3 we can set parameters t and l such that Theorem 3.3 holds.

3.5   Lower bounds

We begin by stating the lower bound on the LSH exponent $\rho = \log(1/p)/\log(1/q)$ by O'Donnell et al. [122].

Theorem 3.7 (O'Donnell et al. [122]). *Fix $d \in \mathbb{N}$, $1 < c < \infty$, $0 < s < \infty$ and $0 < q < 1$. Then for a certain choice of $r = \omega_d(1)$ and under the assumption that $q \geq 2^{-o(d)}$ we have that every (r, cr, p, q)-sensitive family of hash functions for ℓ_s^d must satisfy*

$$\rho = \frac{\log(1/p)}{\log(1/q)} \geq \frac{1}{c^s} - o_d(1).$$

The following lemma shows how to use a filter family $\mathcal{F}$ to construct a hash family $\mathcal{H}$.

Lemma 3.6. *Given a symmetric family of filters that is $(r, cr, p_1, p_2, p_q, p_u)$-sensitive in (X, dist) we can construct a $(r, cr, p_1/(2p_q), p_2/p_q)$-sensitive family of hash functions in (X, dist).*

Proof. Given the filter family $\mathcal{F}$ we sample a random function h from the hash family $\mathcal{H}$ taking an infinite sequence of independently sampled filters $(F_i)_{i=0}^{\infty}$ from $\mathcal{F}$ and setting $h(x) = \min\{i \mid x \in F_i\}$. The probability of collision is given by

$$\Pr_{h \sim \mathcal{H}}[h(x) = h(y)] = \frac{\Pr_{F \sim \mathcal{F}}[x \in F \wedge y \in F]}{\Pr_{F \sim \mathcal{F}}[x \in F \vee y \in F]}$$

and the result follows from the properties of $\mathcal{F}$. $\square$

If the LSH family in Lemma 3.6 had $p = p_1/p_q$ and $q = p_2/p_q$ then the lower bound would follow immediately. We apply the powering technique from Lemma 3.1 to the underlying filter family in order make the factor 2 in $p_1/(2p_q)$ disappear in the statement of ρ as d tends to infinity.

Theorem 1.4. *Every symmetric $(r, cr, p_1, p_2, p_q, p_u)$-sensitive filter family $\mathcal{F}$ for ℓ_s^d must satisfy the lower bound of Theorem 3.7 with $p = p_1/p_q$ and $q = p_2/p_q$.*

Proof. Given a family $\mathcal{F}$ that satisfies the requirements from Theorem 3.7 there exists an integer $\kappa = \omega_d(1)$ such the hash family $\mathcal{H}$ that results from applying Lemma 3.6 to the powered family $\mathcal{F}^\kappa$ also satisfies the requirements from Theorem 3.7. The constructed family $\mathcal{H}$ is (r, cr, p, q)-sensitive for $p = (1/2) \cdot (p_1/p_q)^\kappa$ and $q = (p_2/p_q)^\kappa$. By our choice of κ we have that $\log(1/p)/\log(1/q) = \log(p_q/p_1)/\log(p_q/p_2) + o_d(1)$ and the lower bound on $\log(1/p)/\log(1/q)$ from Theorem 3.7 applies. $\square$

3.5.1 Asymmetric lower bound

The lower bound is based on an isoperimetric-type inequality that holds for randomly correlated points in Hamming space. We say that the pair of points (x, y) is α-correlated if x is a random point in $\{-1, 1\}^d$ and y is formed by taking x and independently flipping each bit with probability $(1 - \alpha)/2$. We are now ready to state O'Donnell's generalized small-set expansion theorem. Notice the similarity to the value of p_1 for the Gaussian filter family described in Section 3.3 and Appendix 3.8.

Lemma 3.7 ([121, p. 285]). *For every $0 \leq \alpha < 1$, $-1 \leq \lambda \leq 1$, and $Q, U \subseteq \{-1, 1\}^d$ satisfying that $|Q|/2^d = (|U|/2^d)^{\alpha^{2\lambda}}$ we have*

$$\Pr_{\substack{(x,y) \\ \alpha\text{-correlated}}} [x \in Q, y \in U] \leq (|U|/2^d)^{\frac{1+\alpha^{2\lambda}-2\alpha^{1+\lambda}}{1-\alpha^2}}.$$

The argument for the lower bound assumes a regular $(r, cr, p_1, p_2, p_q, p_u)$-sensitive filter family $\mathcal{F}$ for Hamming space where we set $r = (1 - \alpha)d/2$ and $cr = (1 - \beta)d/2$ for some choice of $0 < \beta < \alpha < 1$. We then proceed by deriving constraints on p_1, p_2, p_q, p_u, and minimize ρ_q and ρ_u subject to those constrains. The proof of Theorem 1.5 is provided in Appendix 3.10.

Theorem 1.5. *Fix $0 < \beta < \alpha < 1$. Then for every regular $((1-\alpha)d/2, (1-\beta)d/2, p_1, p_2, p_q, p_u)$-sensitive filter family in d-dimensional Hamming space with and $|Q|/2^d = (|U|/2^d)^{\alpha^{2\lambda}}$ where λ satisfies $\alpha + 2\sqrt{\ln(d)/d} \leq \alpha^\lambda \leq 1/(\alpha - 2\sqrt{\ln(d)/d})$ it must hold that*

$$\rho_q = \frac{\log(p_q/p_1)}{\log(p_q/p_2)} \geq \frac{(1 - \alpha^{1+\lambda})^2}{1 - \alpha^2} - o_d(1),$$

$$\rho_u = \frac{\log(p_u/p_1)}{\log(p_q/p_2)} \geq \frac{(\alpha^\lambda - \alpha)^2}{1 - \alpha^2} - o_d(1)$$

when p_q is set to minimize ρ_q and we assume that $|U|/2^d \geq 2^{-o_d(1)}$.

3.6 Open problems

An important open problem is to find simple and practical data-dependent solutions to the (r, cr)-near neighbor problem. Current solutions, the Gaussian filters in this paper included, suffer from $o(1)$ terms in the exponents that decrease very slowly in n. A lower bound for the unit sphere by Andoni et al. [13] indicates that this might be unavoidable.

Another interesting open problem is finding the shape of provably exactly optimal filters in different spaces. In the random data setting in Hamming space, this problem boils down to maximizing the number of pairs of points below a certain distance threshold that is contained in a subset of the space of a certain size. This is a fundamental problem in combinatorics that has been studied by among others [94], but a complete answer remains elusive. The LSH and LSF lower bounds [114, 122, 19],

along with classical isoperimetric inequalities such as Harper's Theorem and more recent work summarized in the book by O'Donnell [121] hints that the answer is somewhere between a subcube and a generalized sphere.

A recent result by Chierichetti and Kumar [49] characterizes the set of transformations of LSH-able similarity measures as the set of probability-generating functions. This seems to have deep connections to result of this paper that uses characteristic functions that allow well-known kernel transformations. It seems possible that this paper can be viewed as a semi-explicit construction of their result, or that their result can be described as an application of Bochner's Theorem.

Acknowledgment

3.7   Appendix: Framework

We state a version of Theorem 3.1 where the parameters of the filter family are allowed to depend on n.

Theorem 3.1. *Suppose we have access to a filter family that is $(r, cr, p_1, p_2, p_q, p_u)$-sensitive. Then we can construct a fully dynamic data structure that solves the (r, cr)-near neighbor problem. Assume that $1/p_1$, $1/\log(p_q/p_2)$, and $\exp(\log(1/p_1)/\log(\min(p_q, p_u)/p_1))$ are $n^{o(1)}$, then the data structure has*

- *query time $dn^{\rho_q + o(1)}$,*

- *update time $n^{\rho_u + o(1)} + dn^{o(1)}$,*

– *space usage $n^{1+\rho_u+o(1)} + dn + dn^{o(1)}$*

where

$$\rho_q = \frac{\log p_q/p_1}{\log p_q/p_2}, \quad \rho_u = \frac{\log p_u/p_1}{\log p_q/p_2}.$$

To prove Theorem 3.1, we begin by setting the parameters mentioned in the description of the LSF data structure in Section 3.2.1.

$$\kappa_1 = \left\lceil \frac{\min(\rho_q, \rho_u) \log n}{\log(1/p_1)} \right\rceil$$

$$\tau = \left\lfloor \frac{\log n}{\kappa_1 \log(p_q/p_2)} \right\rfloor \leq \frac{\log(1/p_1)}{\log(\min(p_q, p_u)/p_1)}$$

$$m_1 = \lceil \tau/p_1^{\kappa_1} \rceil$$

$$\kappa_2 = \max(0, \lceil \log(n)/\log(p_q/p_2) \rceil - \tau\kappa_1)$$

$$m_2 = \lceil 1/p_1^{\kappa_2} \rceil$$

We will now briefly explain the reasoning behind the parameter settings. Begin by observing that the powering and tensoring techniques both amplify the filters from $\mathcal{F}$. Let $m = \binom{m_1}{\tau} \cdot m_2$ denote the number of simulated filters in our collection **F** and let $a = \tau\kappa_1 + \kappa_2$ be an integer denoting the number of times each filter has been amplified. Ignoring the time it takes to evaluate the filters, the query time is determined by the sum of the number of filters that contain a query point and the number of distant points associated with those filters that the query algorithm inspects. The expected number of activated filters is given by mp_q^a while the worst case expected number of distant points to be inspected by the query algorithm is given by nmp_2^a. Balancing the contribution to the query time from these two effects (ignoring the $O(d)$ factor from distance computations) results in a target value of $a = \lceil \log(n)/\log(p_q/p_2) \rceil$. Compared to having an oracle that is able to list the filters from a collection that contains a point, there is a small loss in efficiency from using the tensoring technique due to the increase in the number of filters required to guarantee correctness. The parameters of the LSF data structure are therefore set to minimize the use of tensoring such that the time spent evaluating our collection of filters roughly matches the minimum of the query and update time.

Consider the initialization operation of the LSF data structure with the parameters setting from above. We have that $\kappa_2 \leq \kappa_1$ implying that $m_2 = O(m_1)$. The initialization time and the space usage of the data

structure prior to any insertions is dominated by the time and space used to sample and store the filters in $\mathbf{F}_1$. By the assumption that a filter from $\mathcal{F}$ can be sampled in $O(d)$ operations and stored using $O(d)$ words, we get a space and time bound on the initialization operation of

$$O(d\kappa_1 m_1) = O\left(dn^{\min(\rho_q, \rho_u)} \frac{p_1 \log(n)}{\log(p_q/p_2)}\right).$$

Importantly, this bound also holds for the running time of the filter evaluation algorithm, that is, the preprocessing time required for constant time generation of the next element in the list of filters in $\mathbf{F}$ containing a point. In the following analysis of the update and query time we will temporarily ignore the running time of the filter evaluation algorithm.

The expected time to insert or delete a point is dominated by the number of update filters in $\mathbf{F}$ that contains it. The probability that a particular update filter in $\mathbf{F}$ contains a point is given by p_u^a. Using a standard upper bound on the binomial coefficient we get that $m = O(e^\tau/p_1^a)$ resulting in an expected update time of

$$O(mp_u^a + d) = O(n^{\rho_u}(p_u/p_1)e^\tau + d).$$

In the worst case where every data point is at distance greater than cr from the query point and has collision probablity p_2 the expected query time can be upper bounded by

$$O(mp_q^a + dnmp_2^a) = O(n^{\rho_q}e^\tau(p_q/p_1 + d)).$$

With respect to the correctness of the query algorithm, if a near neighbor y to the query point x exists in P, then it is found by the query algorithm if (x, y) is contained in a filter in $\mathbf{F}_1^{\otimes \tau}$ as well as in a filter in $\mathbf{F}_2$. By Lemma 3.2 the first event happens with probability at least $1/2$ and by the choice of m_2, the second event happens with probability at least $1 - (1 - p_1^{\kappa_2})^{p_1^{\kappa_2}} \geq 1 - 1/e$. From the independence between $\mathbf{F}_1$ and $\mathbf{F}_2$ we can upper bound the failure probability $\delta \leq (1/2)(1 + 1/e)$. This completes the proof of Theorem 3.1.

3.8   Appendix: Gaussian filters

In this section we upper and lower bound the probability mass in the tail of the bivariate standard normal distribution when the correlation

between the two standard normals is at most β (upper bound) or at least α (lower bound). We make use of the following upper and lower bounds on the univariate standard normal as well as an upper bound for the multivariate case.

Lemma 3.8 (Follows Szarek & Werner [154]). *Let Z be a standard normal random variable. Then, for every $t \geq 0$ we have that*

$$\frac{1}{\sqrt{2\pi}} \frac{1}{t+1} e^{-t^2/2} \leq \Pr[Z \geq t] \leq \frac{1}{\sqrt{\pi}} \frac{1}{t+1} e^{-t^2/2}.$$

Lemma 3.9 (Lu & Li [104]). *Let z be a d-dimensional vector of i.i.d. standard normal random variables and let $D \subset \mathbb{R}^d$ be a closed convex domain that does not contain the origin. Let Δ denote the Euclidean distance to the unique closest point in D, then we have that*

$$\Pr[z \in D] \leq e^{-\Delta^2/2}.$$

Lemma 3.10 (Tail upper bound). *For $\alpha, \lambda, t, \beta$ satisfying $0 < \alpha < 1$, $-1 \leq \lambda \leq 1$, $t > 0$, and $-1 < \beta < \alpha$ every pair of standard normal random variables (X, Y) with correlation $\beta' \leq \beta$ satisfies*

$$\Pr[X \geq t \wedge Y \geq \alpha^\lambda t] \leq e^{-\Delta^2/2}$$

where $\Delta^2 = (1 + \frac{(\alpha^\lambda - \beta)^2}{1-\beta^2})t^2$.

Proof. For $\beta' = -1$ the result is trivial. For values of β' in the range $-1 < \beta' \leq \beta$ we use the 2-stability of the normal distribution to analyze a tail bound for (X, Y) in terms of a Gaussian projection vector $z = (Z_1, Z_2)$ applied to unit vectors $x, y \in \mathbb{R}^2$. That is, we can define $X = \langle z, x \rangle$ and $Y = \langle z, x \rangle$ for some appropriate choice of x and y. Without loss of generality we set $x = (1, 0)$ and note that for $\mathbb{E}[XY] = \beta'$ we must have that $y = (\beta', \sqrt{1 - \beta'^2})$. If we consider the region of $\mathbb{R}^2$ where z satisfies $X \geq t \wedge Y \geq \alpha^\lambda t$ we get a closed domain D defined by $z = (Z_1, Z_2)$ such that $Z_1 \geq t$ and $Z_2 \geq (\alpha^\lambda t - \beta' Z_1)/(\sqrt{1 - \beta'^2})$. The squared Euclidean distance from the origin to the closest point in D at least Δ^2 as can be seen by the fact that Δ^2 decreasing in β. Combining this observation with Lemma 3.9 we get the desired result. $\qquad\square$

Lemma 3.11 (Tail lower bound). *For α, λ, t satisfying $0 < \alpha < 1$, $-1 \leq \lambda \leq 1$, and $t > 0$ every pair of standard normal random variables (X, Y) with correlation $\alpha' \geq \alpha$ satisfies*

$$\Pr[X \geq t \wedge Y \geq \alpha^\lambda t] \geq \frac{e^{-\Delta^2/2}}{2\pi(1 + t/\alpha)^2}$$

where $\Delta^2 = (1 + \frac{(\alpha^\lambda - \alpha)^2}{1 - \alpha^2})t^2$.

Proof. For $\alpha' = 1$ the result follows directly from Lemma 3.8. For $\alpha' < 1$ we use the trick from the proof of Lemma 3.10 and define $X = \langle z, x \rangle$ and $Y = \langle z, x \rangle$ where $x = (1, 0)$ and $y = (\alpha, \sqrt{1 - \alpha^2})$ and $z = (Z_1, Z_2)$ is a vector of two i.i.d. standard normal random variables. This allows us to rewrite the probability as follows:

$$\Pr[Z_1 \geq t \wedge \alpha Z_1 + \sqrt{1 - \alpha^2}Z_2 \geq \alpha^\lambda t]$$
$$= \Pr[Z_1 \geq t]\Pr[\alpha Z_1 + \sqrt{1 - \alpha^2}Z_2 \geq \alpha^\lambda t \mid Z_1 \geq t]$$
$$\geq \Pr[Z_1 \geq t]\Pr[\alpha t + \sqrt{1 - \alpha^2}Z_2 \geq \alpha^\lambda t]$$

By the restrictions on α and λ we have that $(\alpha^\lambda - \alpha)t/\sqrt{1 - \alpha^2} \leq t/\alpha$. The result follows from applying the lower bound from Lemma 3.8 and noting that the bound is increasing in α. $\qquad\square$

3.8.1   Space-time tradeoffs on the unit sphere

Summarizing the bound from the previous section, the family $\mathcal{G}$ from Lemma 3.3 satisfies that

$$p_1 \geq \frac{e^{-(1 + \frac{(\alpha^\lambda - \alpha)^2}{1 - \alpha^2})t^2/2}}{2\pi(1 + t/\alpha)^2}$$
$$p_2 \leq e^{-(1 + \frac{(\alpha^\lambda - \beta)^2}{1 - \beta^2})t^2/2}$$
$$p_q \leq e^{-\alpha^{2\lambda}t^2/2}$$
$$p_u \leq e^{-t^2/2}.$$

We combine the Gaussian filters with Theorem 3.1 to show that we can solve the (α, β)-similarity problem efficiently for the full range of space/time tradeoffs, even when α, β are allowed to depend on n, as long as the gap $\alpha - \beta$ is not too small.

Theorem 3.2. *For every choice of $0 \leq \beta < \alpha < 1$ and $\lambda \in [-1, 1]$ we can construct a fully dynamic data structure that solves the (α, β)-similarity problem in $(\mathbb{S}^d, \langle \cdot, \cdot \rangle)$. Suppose that $\alpha - \beta \geq (\ln n)^{-zeta}$ for some constant $zeta < 1/2$, that satisfies the guarantees from Theorem 3.1 with exponents $\rho_q = \frac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2} \Big/ \frac{(1-\alpha^\lambda \beta)^2}{1-\beta^2}$ and $\rho_u = \frac{(\alpha^\lambda - \alpha)^2}{1-\alpha^2} \Big/ \frac{(1-\alpha^\lambda \beta)^2}{1-\beta^2}$.*

Proof. Assuming that $\alpha - \beta \geq (\ln n)^{-zeta}$ there exists a constant $\varepsilon > 0$ where by setting the parameter t of $\mathcal{G}$ such that $t^2/2 = \frac{1-\beta^2}{(1-\alpha^\lambda \beta)^2}(\ln n)^\varepsilon$ the family of filters satisfies the assumptions in Theorem 3.1 while guaranteeing that the second term in ρ_q and ρ_u from Lemma 3.3 are $o(1)$.   □

Remark 3.1. Theorem 3.2 aims for simplicity and generality while allowing α and β to depend on n. For specific values of α, β, λ it is easy to find better bounds on the probabilties (e.g. the bounds by Savage [143]) and to adjust t in Lemma 3.3 to avoid powering (setting $\kappa_1 = 1, \kappa_2 = 0$) in the LSF framework.

3.9 Appendix: Approximate feature maps, characteristic functions, and Bochner's Theorem

We begin by defining what a characteristic function is and listing some properties that are useful for our application. More information about characteristic functions can be found in the books by Lukacs [105] and Ushakov [162].

Lemma 3.12 ([105, 162]). *Let Z denote a random variable with distribution function μ. Then the characteristic function $k(\Delta)$ of Z is defined as*

$$k(\Delta) = \int_{-\infty}^{\infty} \mu(t) e^{i\Delta t} dt$$

and it has the following properties:

- *A distribution function is symmetric if and only if its characteristic function is real and even.*

- *Every characteristic function $k(\Delta)$ is uniformly continuous, has $k(0) = 1$, and $|k(\Delta)| \leq 1$ for all real Δ.*

- *Suppose that $k(\Delta)$ denotes the characteristic function of an absolutely continuous distribution then $\lim_{\Delta \to \infty} |k(\Delta)| = 0$.*

    - *Let X and Y be independent random variables with characteristic functions k_X and k_Y. Then the characteristic function of $Z = (X, Y)$ is given by $k(x, y) = k_X(x)k_Y(y)$.*

Bochner's Theorem reveals the relation between characteristic functions and the class of real-valued functions $k(x, y)$ that admit a feature space representation $k(x, y) = \langle \phi(x), \phi(y) \rangle$

Theorem 3.8 (Bochner's Theorem [141]). *A function $k : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ is positive definite if and only if it can be written on the form*

$$k(x, y) = \int_{\mathbb{R}^d} \mu(v) e^{i\langle v, x-y \rangle} dv$$

where μ is the probability density function of a symmetric distribution.

Rahimi & Recht's [139] family of approximate feature maps $\mathcal{V}$ is constructed from Bochner's Theorem by making use of Euler's Theorem as follows:

$$\begin{aligned}
k(x, y) &= \int_{\mathbb{R}^d} \mu(v) e^{i\langle v, x-y \rangle} dv \\
&= \int_{\mathbb{R}^d} \mu(v) (\cos(\langle v, x-y \rangle) + i \sin(\langle v, x-y \rangle)) dv \\
&= \mathop{\mathbb{E}}_{v} [\cos(\langle v, x-y \rangle)] \\
&= \mathop{\mathbb{E}}_{v,b} [\cos(\langle v, x-y \rangle) + \cos(\langle v, x \rangle + \langle v, y \rangle + 2b)] \\
&= 2 \mathop{\mathbb{E}}_{v,b} [\cos(\langle v, x \rangle + b) \cdot \cos(\langle v, y \rangle + b)].
\end{aligned}$$

Where the third equality makes use of the fact that $k(x, y)$ is real-valued to remove the complex part of the integral and the fifth equality uses that $2\cos(x)\cos(y) = \cos(x + y) + \cos(x - y)$.

Now that we have an approximate feature map onto the sphere for the class of shift-invariant kernels, we will take a closer look at what functions this class contains, and what their applications are for similarity search. Given an arbitrary similarity function, we would like to be able to determine whether it is indeed a characteristic function. Unfortunately, there are no known simple techniques for answering this question in general. However, the machine learning literature contains many applications of different shift-invariant kernels [145] and many common distributions have real characteristic functions (see Appendix

B in [162] for a long list of examples). Characteristic functions are also well studied from a mathematical perspective [105, 162], and a number of different necessary and sufficient conditions are known. A classical result by Pólya [136] gives simple sufficient conditions for a function to be a characteristic function. Through the vectorization property from Lemma 3.12, Pólya's conditions directly imply the existence of a large class of similarity measures on $\mathbb{R}^d$ that can fit into the above framework.

Theorem 3.9 (Pólya [136]). *Every even continuous function $k : \mathbb{R} \to \mathbb{R}$ satisfying the properties*

- $k(0) = 1$

- $\lim_{\Delta \to \infty} k(\Delta) = 0$

- $k(\Delta)$ *is convex for $\Delta > 0$*

is a characteristic function.

Based on the results of Section 3.4.1 one could hope for the existence of characteristic functions of the form $k(\Delta) = e^{-|\Delta|^s}$ for $s > 2$ but it is known that such functions cannot exist [25, Theorem D.8]. Furthermore, Marcinkiewicz [108] shows that a function of the form $k(\Delta) = \exp(-\operatorname{poly}(\Delta))$ cannot be a characteristic function if the degree of the polynomial is greater than two.

We state a more complete, constructive version of Lemma 3.4 as well as the proof here.

Lemma 3.13. *Let k be a real-valued characteristic function with associated distribution function μ and let l be a positive integer. Consider the family of functions $\mathcal{V} \subseteq \{v \mid v \colon \mathbb{R}^d \to \mathbb{S}^l\}$ where a randomly sampled function v is defined by, independently for $j = 1, \ldots, l$, sampling v from μ and b uniformly on $[0, 2\pi]$, letting $\hat{v}(x)_j = \sqrt{(2/l)} \cos(\langle v, x \rangle + b)$ and normalizing $v(x)_j = \frac{\hat{v}(x)}{\|\hat{v}(x)\|}$. The family $\mathcal{V}$ has the property that for every $x, y \in \mathbb{R}^d$ and $\varepsilon > 0$ we have that*

$$\Pr_{v \sim \mathcal{V}}[|\langle v(x), v(y) \rangle - k(x, y)| \geq \varepsilon] \leq 6e^{-l\varepsilon^2/128}.$$

Proof. Since $l \cdot \hat{v}(x)_j \hat{v}(y)_j$ is bounded between 2 and -2, and we have independence for different values of j, Hoeffding's inequality [88] can be

applied to show that for every fixed pair of points x, y and $\hat{\varepsilon} > 0$ it holds that

$$\Pr[|\langle \hat{v}(x), \hat{v}(y) \rangle - k(x, y)| \geq \hat{\varepsilon}] \leq 2e^{-l\hat{\varepsilon}^2/8}.$$

From the properties of characteristic functions we have that $k(x, x) = 1$ and $k(x, y) \leq 1$. The bound on the deviation of

$$\langle v(x), v(y) \rangle = \frac{\langle \hat{v}(x), \hat{v}(y) \rangle}{\sqrt{\langle \hat{v}(x), \hat{v}(x) \rangle \langle \hat{v}(y), \hat{v}(y) \rangle}}$$

from $k(x, y)$ follows from setting $\hat{\varepsilon} = \varepsilon/4$ and using a union bound over the probabilities that the deviation of one of the inner products is too large. $\qquad\square$

Combining the approximate feature map onto the unit sphere with Theorem 3.2 we obtain the following:

Theorem 3.10. *Let* $k \colon \mathbb{R}^d \to \mathbb{R}$ *be a characteristic function and define the similarity measure* $S(x, y) = k(x - y)$. *Assume that we have access to samples from the distribution associated with k, then Theorem 3.2 holds with* $(\mathbb{S}^d, \langle \cdot, \cdot \rangle)$ *replaced by* $(\mathbb{R}^d, S)$.

Proof. According to Lemma 3.13, we can set $l = n^{o(1)}$ to obtain a map $v \colon \mathbb{R}^d \to \mathbb{S}^l$ such that the the inner product on $\mathbb{S}^l$ preserves the pairwise similarity between $n^{O(1)}$ points with additive error $\varepsilon = o(1)$. This map has a space and time complexity of $O(dl) = dn^{o(1)}$. After applying v to the data we can solve the (α, β)-similarity problem on $(\mathbb{R}^d, k(x - y))$ by solving the $(\alpha - \varepsilon, \beta + \varepsilon)$-similarity problem on $(\mathbb{S}^d, \langle \cdot, \cdot \rangle)$. We can use Theorem 3.2 to construct a fully dynamic data structure for solving this problem, adjusting the parameter λ so that it continues to lie in the admissible range. The space and time complexities follow. $\qquad\square$

3.10   Appendix: Proof of tradeoff lower bound

Consider $\rho_q = \frac{\log(p_q/p_1)}{\log(p_q/p_2)}$. Subject to the (implicit) LSF constraint that $p_q, p_u > p_1 > p_2 > 0$ we see that ρ_q is minimized by setting p_q, p_2 as small as possible and p_1 as large as possible. We will therefore derive lower bounds on p_q, p_2 and an upper bound on p_1. For every value of p_1 and p_2 we minimize ρ_q, ρ_u by choosing p_q as small as possible.

For a random point $x \in \{-1, 1\}^d$ it must hold that $\Pr_{\mathcal{F}}[x \in Q] = |Q|/2^d$. This implies the existence of a fixed point $y \in \{-1, 1\}^d$ with

the property that $\Pr_{\mathcal{F}}[y \in Q] \geq |Q|/2^d$. A regular filter family must therefore satisfy that $p_q \geq |Q|/2^d$ and $p_u \geq |U|/2^d$. Let λ be defined as in Lemma 3.7 then by a similar argument we have that $p_2 \geq (U/2^d)^{1+\alpha^{2\lambda}}$.

In order to upper bound p_1 we make use of Lemma 3.7 together with the following lemma that follows directly from an application of Hoeffding's inequality [88].

Lemma 3.14. *For every $0 < \varepsilon < (1 - \alpha)/2$ we have that*

$$\Pr_{\substack{(x,y) \\ \alpha+\varepsilon\text{-correlated}}} \left[\frac{1}{d}\sum_{i=1}^{d} x_i y_i \leq \alpha\right] \leq e^{-\varepsilon^2 d/2}.$$

In the following derivation, assume that α, ε satisfies $0 < \varepsilon < (1 - \alpha)/2$, let x, y denote randomly $(\alpha + \varepsilon)$-correlated vectors in $\{-1, 1\}^d$, and assume that $\alpha + \varepsilon \leq \alpha^\lambda \leq 1/(\alpha + \varepsilon)$, then

$$(|U|/2^d)^{\frac{1+\alpha^{2\lambda}-2\alpha^\lambda(\alpha+\varepsilon)}{1-(\alpha+\varepsilon)^2}} \geq \Pr[x \in Q, y \in U]$$
$$\geq \Pr[x \in Q, y \in U \mid \langle x, y \rangle \geq \alpha] \Pr[\langle x, y \rangle \geq \alpha]$$
$$\geq p_1(1 - e^{-\varepsilon^2 d/2})$$

Summarizing the bounds:

$$p_1 \leq \frac{(|U|/2^d)^{\frac{1+\alpha^{2\lambda}-2\alpha^\lambda(\alpha+\varepsilon)}{1-\alpha^2}}}{1 - e^{-\varepsilon^2 d/2}}$$
$$p_2 \geq (|U|/2^d)^{1+\alpha^{2\lambda}}$$
$$p_q \geq |Q|/2^d$$
$$p_u \geq |U|/2^d.$$

When minimizing ρ_q we have that $\log(p_q/p_2) = -\log(|U|/2^d)$. Setting $\varepsilon = 2\sqrt{\ln(d)/d}$ results in $\log(1/p_1) \geq -\frac{1+\alpha^{2\lambda}-2\alpha^\lambda(\alpha+\varepsilon)}{1-\alpha^2}\log(|U|/2^d) -$

$O(1/d^2)$. Putting things together:

$$
\begin{aligned}
\frac{\log(p_q/p_1)}{\log(p_q/p_2)} &\geq -\frac{\alpha^{2\lambda}\log(|U|/2^d)}{\log(|U|/2^d)} \\
&\quad + \frac{\frac{1+\alpha^{2\lambda}-2\alpha^\lambda(\alpha+\varepsilon)}{1-\alpha^2}\log(|U|/2^d)+O(1/d^2)}{\log(|U|/2^d)} \\
&= \frac{(1-\alpha^{1+\lambda})^2-2\alpha^\lambda\varepsilon}{1-\alpha^2} + \frac{O(1/d^2)}{\log(|U|/2^d)} \\
&= \frac{(1-\alpha^{1+\lambda})^2}{1-\alpha^2} - O(\sqrt{\log(d)/d}).
\end{aligned}
$$

The derivation of the lower bound for ρ_u is almost the same and the resulting expression is

$$
\frac{\log(p_u/p_1)}{\log(p_q/p_2)} \geq \frac{(\alpha^\lambda-\alpha)^2}{1-\alpha^2} - O(\sqrt{\log(d)/d}).
$$

3.11 Appendix: Comparison to Kapralov

Kapralov uses α to denote a parameter controlling the space-time trade-off for his solution to the (r, cr)-near neighbor problem in Euclidean space. For every choice of tradeoff parameter $\alpha \in [0, 1]$, assuming that $c^2 \geq 3(1-\alpha)^2 - \alpha^2 + \varepsilon$ for arbitrarily small constant $\varepsilon > 0$, Kapralov [95] obtains query and update exponents

$$
\begin{aligned}
\rho_q &= \frac{4(1-\alpha)^2}{c^2+(1-\alpha)^2-3\alpha^2}, \\
\rho_u &= \frac{4\alpha^2}{c^2+(1-\alpha)^2-3\alpha^2}.
\end{aligned}
$$

We convert Kapralov's notation to our own by setting $\lambda = 1 - 2\alpha$. To compare, Kapralov sets $\alpha = 0$ for near-linear space and we set $\lambda = 1$. We want to write Kapralov's exponents on the form

$$
\rho_q = \frac{c^2(1+\lambda)^2}{(c^2+\lambda)^2+x}, \quad \rho_u = \frac{c^2(1-\lambda)^2}{(c^2+\lambda)^2+x}
$$

for some x that we will proceed to derive. We have that $(1-\alpha)^2 = (1+\lambda)^2/4$ and $\alpha^2 = (1-\lambda)^2/4$. Multiplying the numerator and denominator

in Kapralov's exponents by c^2 we can write Kapralov's exponents as

$$\rho_q = \frac{c^2(1+\lambda)^2}{c^4 + c^2(1+\lambda)^2/4 - 3c^2(1-\lambda)^2/4},$$

$$\rho_u = \frac{c^2(1-\lambda)^2}{c^4 + c^2(1+\lambda)^2/4 - 3c^2(1-\lambda)^2/4}.$$

We have that

$$x = c^4 + c^2(1+\lambda)^2/4 - 3c^2(1-\lambda)^2/4 - (c^2+\lambda)^2$$
$$= -c^2(1+\lambda^2)/2 - \lambda^2.$$

For every choice of $\lambda \in [-1,1]$, and under the assumption that $c^2 \geq (1+\lambda)^2/2 + \lambda + \varepsilon$ for an arbitrarily small constant $\varepsilon > 0$, this allows us to write Kapralov's exponents as

$$\rho_q = \frac{c^2(1+\lambda)^2}{(c^2+\lambda)^2 - c^2(1+\lambda^2)/2 - \lambda^2},$$

$$\rho_u = \frac{c^2(1-\lambda)^2}{(c^2+\lambda)^2 - c^2(1+\lambda^2)/2 - \lambda^2}.$$

To compare Kapralov's result against our own for search in ℓ_s-spaces we consider the exponents from Theorem 3.3, ignoring additive $o(1)$ terms:

$$\rho_q = \frac{c^s(1+\lambda)^2}{(c^s+\lambda)^2}, \quad \rho_u = \frac{c^s(1-\lambda)^2}{(c^s+\lambda)^2}.$$

Setting $\lambda = 1$ we obtain a data structure that uses near-linear space and we get a query exponent $\rho_q = 16/25$ while Kapralov obtains an exponent of $\rho_q = 16/20$, ignoring $o(1)$ terms. At the other end of the tradeoff, setting $\lambda = -1$, we get a data structure with query time $n^{o(1)}$ and update exponent $\rho_u = 16/9$ while Kapralov gets an update exponent of $\rho_u = 4$, again ignoring additive $o(1)$ terms.

The assumption made by Kapralov that $c^2 \geq (1+\lambda)^2/2 + \lambda + \varepsilon$ means that in the case of a near-linear space data structure ($\lambda = 1$) sublinear query time can only be obtained for $c > \sqrt{3}$. In contrast, Theorem 3.3 gives sublinear query time for every constant $c > 1$.

3.12 Appendix: Details about dynamization and the model of computation

In order to obtain fully dynamic data structures we apply a powerful dynamization result of Overmars and Leeuwen [123] for decomposable searching problems. Their result allows us to turn a partially dynamic data structure into a fully dynamic data structure, supporting arbitrary sequences of queries and updates, at the cost of a constant factor in the space and running time guarantees. Suppose we have a partially dynamic data structure that solves the (r, cr)-near neighbor problem on a set of n points. By partially dynamic we mean that, after initialization on a set P of n points, the data structure supports $\Theta(n)$ updates without changing the query time by more than a constant factor. Let $T_q(n)$, $T_u(n)$, and $T_c(n)$ denote the query time, update time, and construction time of such a data structure containing n points. Then, by Theorem 1 of Overmars and Leeuwen [123], there exists a fully dynamic version of the data structure with query time $O(T_q(n))$ and update time $O(T_u(n) + T_c(n)/n)$ that uses only a constant factor additional space. The data structures presented in this paper, as well as most related constructions from the literature, have the property that $T_c(n)/n = O(T_u(n))$, allowing us to go from a partially dynamic to a fully dynamic data structure "for free" in big O notation.

In terms of guaranteeing that the query operation solves the (r, cr)-near neighbor problem on the set of points P currently inserted into the data structure, we allow a constant failure probability $\delta < 1$, typically around $1/2$, and omit it from our statements. We make the standard assumption that the adversary does not have knowledge of the randomness used by the data structure. Say we have a data structure with constant failure probability and a bound on the expected space usage. Then, for every positive integer T we can create a collection of $O(\log T)$ independent repetitions of the data structure such that for every sequence of T operations it holds with high probability in T that the space usage will never exceed the expectation by more than a constant factor and no query will fail.

3.12.1 Model of computation

We use the standard word RAM model as defined by Hagerup [85] with a word size of $\Theta(\log n)$ bits. Unless otherwise stated, we make the assumption that a point in (X, D) can be stored in d words and

that the dissimilarity between two arbitrary points can be computed in d operations where d is a positive integer that corresponds to the dimension in the various well-studied settings mentioned in the main text. Furthermore, when describing framework-based solutions to the (r, cr)-near neighbor problem, we make the assumption that we can sample, evaluate, and represent elements from $\mathcal{F}$ and $\mathcal{H}$ with neglible error using space and time $dn^{o(1)}$.

Many of the LSH and LSF families rely on random samples from the standard normal distribution. We will ignore potential problems resulting from rounding due to the fact that our model only supports finite precision arithmetic. This approach is standard in the literature and can be justified by noting that the error introduced by rounding is neglible. Furthermore, there exists small pseudorandom standard normal distributions that support sampling using only few uniformly distributed bits as noted by Charikar [47]. In much of the related literature the model of computation is left unspecified and statements about the complexity of solutions to the (r, cr)-near neighbor problem are usually made with respect to particular operations such as the hash function computations, distance computations, etc., leaving out other details [91, 86].

3.13   Addendum: An improved framework

The LSF framework in Theorem 3.1 suffers from large lower-order terms that depend on the $(r, cr, p_1, p_2, p_q, p_u)$-sensitivity properties of $\mathcal{F}$. With the parameterization in Appendix 3.7 the framework uses $O(\tau n^{\min(\rho_q, \rho_u)})$ filters from $\mathcal{F}$ where $\tau \leq \log(1/p_1) / \log(\min(p_q, p_u)/p_1)$. In addition, the query and update time have a multiplicative factor e^τ which can potentially be very large and where we have to assume explicitly that $e^\tau = n^{o(1)}$. We will use a combination of techniques in recent work on set similarity search [56] and fast locality-sensitive hashing frameworks [64, 53] to give an improved LSF framework with more precise complexity bounds.

The data structure produced by the framework follows the high-level approach as outlined in Section 3.2.1: queries and updates are mapped to a collection of buckets that are searched for similar points in the case of a query, or updated to store a reference to the point in the case of an update. Let $V \colon X \to R$ denote the mapping from query points to buckets and $W \colon X \to R$ denote the corresponding map for updates. The

set of buckets $V(x)$ will be identified by the "survivors" of w branching processes through k collections of m filters, similarly to the Chosen Path algorithm [56].

The data structure is initialized by sampling k collections of m filters. We will use the notation $Q_{i,j}$ ($U_{i,j}$) to denote the jth query (update) filter in the ith collection. For $i = 1, \ldots, k$ let $h_i \colon [w] \times [m]^i \to [0,1]$ denote a pairwise independent random hash function. Let $\lambda \in [0,1]$ be a parameter to be determined later and let $\circ$ denote vector concatenation, then the locality-sensitive map V is defined recursively as follows:

$$V_i(x) = \begin{cases} \{p \circ j \mid p \in V_{i-1}(x) \wedge h_i(p \circ j) < \lambda \wedge x \in Q_{i,j}\} & \text{if } i > 0 \\ [w] & \text{if } i = 0. \end{cases}$$

The map W is defined in the same way except it uses $U_{i,j}$ instead of $Q_{i,j}$.

Properties.   To show that the maps V, W provide an efficient solution to the (r, cr)-near neighbor problem we need to show the following:

- An upper bound on the expected size of $V(x)$ and $W(x)$ to bound the expected number of buckets probed during queries/updates.

- An upper bound on the expected size of $V(x) \cap W(y)$ when $\text{dist}(x, y) > cr$ to bound the expected number of distant points that will be encountered during the linear scan part of the query algorithm.

- That $V(x) \cap W(y)$ is non-empty with constant probability when $\text{dist}(x, y) \leq r$ to guarantee that the query algorithm encounters a point at distance at most r with constant probability, provided such a point exists.

By the independence between the different levels $i = 1, \ldots, k$ in the branching process we have that

$$\mathbb{E}[|V_i(x)|] = \mathbb{E}[|V_{i-1}(x)|] m \lambda p_q = w(m \lambda p_q)^i.$$

Given $x, y \in X$ define $Z_i = V_i(x) \cap W_i(y)$. Define $p = \Pr[x \in Q, y \in U]$ where (Q, U) is sampled from $\mathcal{F}$. The expected number of collisions between x and y at level i is then given by

$$\mathbb{E}[|Z_i|] = \mathbb{E}[|Z_{i-1}|] m \lambda p = w(m \lambda p)^i.$$

To show correctness of the scheme we will use Chebyshev's inequality to show that with constant probability we have $|Z_i| > 0$ for points x, y with $\text{dist}(x, y) \leq r$. We proceed by upper bounding $\mathbb{E}[|Z_i|^2]$ in order to bound the variance $\text{Var}[|Z_i|] = \mathbb{E}[|Z_i|^2] - \mathbb{E}[|Z_i|]^2$. To ease the derivation we define $Y_{p,j} = \mathbb{1}\{h_i(p \circ j) < \lambda \wedge (x, y) \in (Q_{i,j}, U_{i,j})\}$ where we suppress the subscript i. Without loss of generality we can assume that $p = p_1$ since $\text{dist}(x, y) \leq r$.

$$
\begin{aligned}
\mathbb{E}[|Z_i|^2] &= \mathbb{E}\left[\left(\sum_{p \in Z_{i-1}} \sum_{j \in [m]} Y_{p,j}\right)^2\right] \\
&= \mathbb{E}\left[\sum_{p \neq p'}\left(\sum_{j \neq j'} Y_{p,j} Y_{p',j'} + \sum_{j=j'} Y_{p,j} Y_{p',j'}\right)\right] \\
&\quad + \mathbb{E}\left[\sum_{p=p'}\left(\sum_{j \neq j'} Y_{p,j} Y_{p',j'} + \sum_{j=j'} Y_{p,j} Y_{p',j'}\right)\right] \\
&= \mathbb{E}[|Z_{i-1}|^2 - |Z_{i-1}|]((m^2 - m)\lambda^2 p_1^2 + m\lambda^2 p_1) \\
&\quad + \mathbb{E}[|Z_{i-1}|]((m^2 - m)\lambda^2 p_1^2 + m\lambda p_1) \\
&\leq \mathbb{E}[|Z_{i-1}|^2](1 + 1/mp_1)(m\lambda p_1)^2 + \mathbb{E}[|Z_{i-1}|]m\lambda p_1.
\end{aligned}
$$

Since $i \leq k$ if we set $m \geq \ln(1 + \varepsilon)k/p_1$ we have that

$$
\mathbb{E}[|Z_i|^2] \leq (1 + \varepsilon)w^2(m\lambda p_1)^{2i} + (1 + \varepsilon)\,\mathbb{E}[|Z_i|]\sum_{s=0}^{i-1}(m\lambda p_1)^s.
$$

We will set the parameters in order to give a simple upper bound the worst-case performance of the data structure. The constants can be improved.

$$
\begin{aligned}
\varepsilon &= 1/10, \\
k &= \lceil \log(n) / \log(p_q/p_2) \rceil, \\
m &= \lceil \ln(1 + \varepsilon)k/p_1 \rceil, \\
\lambda &= 1/(m\min(p_q, p_u)), \\
w &= \lceil 10k(\min(p_q, p_u)/p_1)^k \rceil.
\end{aligned}
$$

We can now bound the variance of $|Z_k|$ as follows:

$$\text{Var}[|Z_k|] = \mathbb{E}[|Z_k|^2] - \mathbb{E}[|Z_k|]^2$$

$$\leq \mathbb{E}[|Z_k|]^2/10 + (1+1/10)\,\mathbb{E}[|Z_k|]\sum_{s=0}^{k-1}(p_1/\min(p_q,p_u))^s$$

$$\leq \mathbb{E}[|Z_k|]^2/10 + (1+1/10)k\,\mathbb{E}[|Z_k|]$$

where we use the fact that $p_1 \leq \min(p_q, p_u)$. By Chebyshev's inequality we have that

$$\Pr[|Z_k| < 0] \leq \text{Var}[|Z_k|]/\mathbb{E}[|Z_k|]^2$$

$$\leq 1/10 + (1+1/10)k/\mathbb{E}[|Z_k|]$$

By our parameter setting we have $\mathbb{E}[|Z_k|] = w(m\lambda p_1)^k \geq 10k$ so x, y collide with probability at least $7/10$ under V, W, ensuring correctness.

3.13.1   Fast evaluation

We will use a hashing trick to compute $V_k(x)$ in expected time $O(k\,\mathbb{E}[|V_k(x)|])$. This technique is only briefly mentioned in [56]. Observe that for the correctness argument to hold, it suffices that the hash functions $h_1, \ldots, h_k$ are sampled independently from a pairwise independent family [41, 42]. At the ith step in the computation of $V_k(x)$ we wish to determine, for each $p \in V_{i-1}(x)$ the set of $j \in [m]$ satisfying $h_i(p \circ j) < \lambda$ and $x \in Q_{i,j}$. In order to answer this efficiently we will make use of the property that a pairwise independent hash function can be decomposed as

$$h_i(p \circ j) = g_i(p) \oplus f_i(j)$$

where g_i, f_i are pairwise independent and $\oplus$ denotes addition in an abelian group. For concreteness assume that g_i, f_i map to b-bit strings and let $\oplus$ denote the exclusive-or operator. If we view the b-bit output of $h_i(p \circ j)$ as an integer in the set $0, 1, \ldots, 2^b - 1$ using the standard base two representation, the original condition $h_i(p \circ j) < \lambda$ can be transformed into the condition $h_i(p \circ j) < \lambda M$ where $M = 2^b - 1$. By choosing $b = \Theta(\log n)$ we can with high probability determine whether the condition is satisfied without reading more than b bits, so we can effectively treat the output of the hash function as a real number at the cost of a small increase in the failure probability of the data structure.

Continuing with the new representation, in order for $h_i(p \circ j) < \lambda M$ we must have that the leading $\kappa = b - \lceil \log_2(\lambda M) \rceil - 1$ bits of the output of $g_i(p) \oplus f_i(j)$ is all zeroes. Given the leading κ bits of $g_i(p)$ we can restrict our attention to $j \in [m]$ with the same value in the leading κ bits of $f_i(j)$. At the beginning of the query algorithm, for each $i \in k$ we determine the subset $J \subseteq [m]$ such that $x \in Q_{i,j}$ We then create a table with 2^{κ} linked lists and for each $j \in J$ we append j to the linked list at the table entry given by the leading κ bits of $f_i(j)$. The running time and space usage of preparing these additional data structures is dominated by the complexity of evaluating and storing $O(k^2/p_1)$ filters from $\mathcal{F}$.

Now, given $V_{i-1}(x)$ we can compute $V_i(x)$ in expected time $O(m\lambda p_q |V_{i-1}(x)|)$ by, for each $p \in V_i(x)$, looking up the relevant table entry (given by the leading κ bits of $g_i(p)$) and verifying whether the elements of the linked list satisfy the hashing condition. Every element of the linked list found in this way satisfies the hashing condition with constant probability by our setting of κ. To implement g_i and f_i we can use simple tabulation hashing [174].

One problem remains: long paths $p \in [w] \times [m]^i$ can take super constant time to hash. To prevent this we again use hashing to create $O(\log n)$-bit fingerprints of the paths that we work on instead. A conservative upper bound on the expected time to compute $V_k(x)$ is $O(k \, \mathbb{E}[|V_k(x)|])$ since $\mathbb{E}[|V_i(x)|]$ is non-decreasing in i and the expected time spent at level i is upper bounded by $O(\mathbb{E}[|V_i(x)|])$. We use the same approach to compute $W_k(x)$.

3.13.2 Framework

We are now ready to state the properties of the new framework.

Theorem 3.11. *Given a $(r, cr, p_q, p_u, p_1, p_2)$-sensitive family $\mathcal{F}$ we can construct a fully dynamic data structure that solves the (r, cr)-near neighbor problem. Define $k = \lceil \log(n) / \log(p_q/p_2) \rceil$, then:*

- *The data structure uses $O(kn(p_u/p_1)^k)$ words of space in addition to the space required to store n data points and $O(k^2/p_1)$ filters from $\mathcal{F}$.*

- *The query operation uses $O(k^2(p_q/p_1)^k))$ word-RAM operations, $O(k(p_q/p_1)^k)$ distance computations, and $O(k^2/p_1)$ filter evaluations.*

- *The update operation uses $O(k^2(p_u/p_1)^k))$ word-RAM operations and $O(k^2/p_1)$ filter evaluations.*

Compared to the usual formulation where the query time is stated as $n^{\rho_q + o(1)}$ Theorem 3.11 offers a more precise statement of the complexity and can be converted to the other formulation. The lower order terms are now confined to the multiplicative factor k which is a standard expression that also appears in the LSH framework as $k = \lceil \log(n) / \log(1/p_2) \rceil$ where p_2 is an upper bound on the collision probability between pairs of points x, y with $\mathrm{dist}(x, y) > cr$. The analysis can be tightened further by not using k as an upper bound for $\sum_{s=0}^{k-1} (p_1 / \min(p_q, p_u))^s$ when bounding the variance, but removing the multiplicative dependence on k entirely as in the improved LSH framework [53] is an interesting open problem.

Chapter 4

Set similarity search beyond MinHash

'From the ashes, a fire shall be woken'

We consider the problem of approximate set similarity search under Braun-Blanquet similarity $B(x,y) = |x \cap y| / \max(|x|, |y|)$. The (b_1, b_2)-approximate Braun-Blanquet similarity search problem is to preprocess a collection of sets P such that, given a query set q, if there exists $x \in P$ with $B(q, x) \geq b_1$, then we can efficiently return $x' \in P$ with $B(q, x') > b_2$.

We present a simple data structure that solves this problem with space usage $O(n^{1+\rho} \log n + \sum_{x \in P} |x|)$ and query time $O(|q| n^\rho \log n)$ where $n = |P|$ and $\rho = \log(1/b_1) / \log(1/b_2)$. Making use of existing lower bounds for locality-sensitive hashing by O'Donnell et al. [122] we show that this value of ρ is tight across the parameter space, i.e., for every choice of constants $0 < b_2 < b_1 < 1$.

In the case where all sets have the same size our solution strictly improves upon the value of ρ that can be obtained through the use of state-of-the-art data-independent techniques in the Indyk-Motwani locality-sensitive hashing framework [91] such as Broder's MinHash [35] for Jaccard similarity and Andoni et al.'s cross-polytope LSH [13] for cosine similarity. Surprisingly, even though our solution is data-independent, for a large part of the parameter space we outperform the currently best data-*dependent* method by Andoni and Razenshteyn [17].

4.1 Introduction

In this paper we consider the approximate set similarity problem or, equivalently, the problem of approximate Hamming near neighbor search in sparse vectors. Data that can be represented as sparse vectors is

ubiquitous — a typical example is the representation of text documents as *term vectors*, where non-zero vector entries correspond to occurrences of words (or shingles). In order to perform identification of near-identical text documents in web-scale collections, Broder et al. [30, 36] designed and implemented *MinHash* (a.k.a. min-wise hashing), now understood as a locality-sensitive hash function [86]. This allowed approximate answers to similarity queries to be computed much faster than by other methods, and in particular made it possible to cluster the web pages of the AltaVista search engine (for the purpose of eliminating near-duplicate search results). Almost two decades after it was first described, MinHash remains one of the most widely used locality-sensitive hashing methods as witnessed by thousands of citations of [30, 36] as well as the ACM Paris Kanellakis Theory and Practice Award that Broder shared with Indyk and Charikar in 2012.

A *similarity measure* maps a pair of vectors to a similarity score in $[0, 1]$. It will often be convenient to interpret a vector $x \in \{0, 1\}^d$ as the set $\{i \mid x_i = 1\}$. With this convention the *Jaccard similarity* of two vectors can be expressed as $J(x, y) = |x \cap y| / |x \cup y|$. In *approximate similarity search* we are interested the problem of searching a data set $P \subseteq \{0, 1\}^d$ for a vector of similarity at least j_1 with a query vector $q \in \{0, 1\}^d$, but allow the search algorithm to return a vector of similarity $j_2 < j_1$. To simplify the exposition we will assume throughout the introduction that all vectors are t-sparse, i.e., have the same Hamming weight t.

Recent theoretical advances in data structures for approximate *near neighbor* search in Hamming space [17] make it possible to beat the asymptotic performance of MinHash-based Jaccard similarity search (using the LSH framework of [86]) in cases where the similarity threshold j_2 is not too small. However, numerical computations suggest that MinHash is always better when $j_2 < 1/45$.

In this paper we address the problem: Can similarity search using MinHash be improved *in general*? We give an affirmative answer in the case where all sets have the same size t by introducing CHOSEN PATH: a simple data-independent search method that strictly improves MinHash, and is always better than the data-dependent method of [17] when $j_2 < 1/9$. Similar to data-independent locality-sensitive filtering (LSF) methods [24, 100, 54] our method works by mapping each data (or query) vector to a set of keys that must be stored (or looked up). The name CHOSEN PATH stems from the way the mapping is constructed: As paths in a layered random graph where the vertices at each layer is
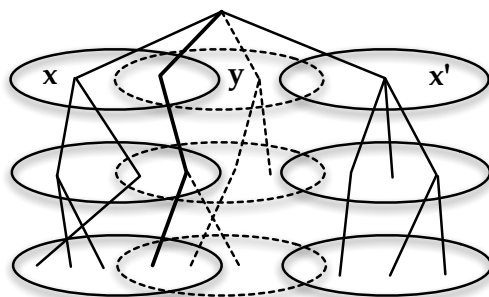
Figure 4.1: Chosen Path uses a branching process to associate each vector $x \in \{0,1\}^d$ with a set $M_k(x) \subseteq \{1,\ldots,d\}^k$ of paths of lengtk k (in the picture $k = 3$). The paths associated with x are limited to indices in the set $\{i \mid x_i = 1\}$, represented by an ellipsoid at each level in the illustration. In the example the set sizes are: $|M_3(x)| = 4$ and $|M_3(y)| = |M_3(x')| = 3$. Parameters are chosen such that a query y that is similar to $x \in P$ is likely to have a common path in $x \cap y$ (shown as a bold line), whereas it shares few paths in expectation with vectors such as x' that are not similar.

identified with the set $\{1,\ldots,d\}$ of dimensions, and where a vector x is only allowed to choose paths that stick to non-zero components x_i. This is illustrated in Figure 4.1.

4.1.1   Related Work

High-dimensional approximate similarity search methods can be characterized in terms of their *ρ-value* which is the exponent for which queries can be answered in time $O(dn^\rho)$, where n is the size of the set P and d denotes the dimensionality of the space. Here we focus on the "balanced" case where we aim for space $O(n^{1+\rho} + dn)$, but note that there now exist techniques for obtaining other trade-offs between query time and space overhead [16, 54].

Locality-sensitive hashing methods.   We begin by describing results for Hamming space, which is a special case of similarity search on the unit sphere (many of the results cited apply to the more general case). In Hamming space the focus has traditionally been on the ρ-value that can be obtained for solutions to the (r, cr)-*approximate near neighbor* problem: Preprocess a set of points $P \subseteq \{0,1\}^d$ such that, given a query point q, if there exists $x \in P$ with $\|x - q\|_1 \le r$, then return $x' \in P$ with $\|x' - q\|_1 < cr$. In the literature this problem is often presented as the c-approximate near neighbor problem where bounds for the ρ-value are

stated in terms of c and, in the case of upper bounds, hold for every choice of r, while lower bounds may only hold for specific choices of r.

O'Donnell et al. [122] have shown that the value $\rho = 1/c$ for c-approximate near neighbor search in Hamming space, obtained in the seminal work of Indyk and Motwani [91], is the best possible in terms of c for schemes based on Locality-Sensitive Hashing (LSH). However, the lower bound only applies when the distances of interest, r and cr, are relatively small compared to d, and better upper bounds are known for large distances. Notably, other LSH schemes for angular distance on the unit sphere such as cross-polytope LSH [13] give lower ρ-values for large distances. Extensions of the lower bound of [122] to cover more of the parameter space were recently given in [16, 54]. Until recently the best ρ-value known in terms of c was $1/c$, but in a sequence of papers Andoni et al. [14, 17] have shown how to use *data-dependent* LSH techniques to achieve $\rho = 1/(2c - 1) + o_n(1)$, bypassing the lower bound framework of [122] which assumes the LSH to be independent of data.

Set similarity search.   There exists a large number of different measures of set similarity with various applications for which it would be desirable to have efficient approximate similarity search algorithms [51]. Given a measure of similarity assume that we have access to a family $\mathcal{H}$ of locality-sensitive hash functions (defined in Section 4.2) such that for every pair of sets x, y it holds that

$$\Pr[h(x) = h(y)] = \text{sim}(x, y).$$

when h is sampled randomly from $\mathcal{H}$. We will refer to a family of locality-sensitive hash functions with this specific property as a *similarity-sensitive* family. Given a similarity-sensitive family we can use the LSH framework to construct a solution for the (s_1, s_2)-approximate similarity search problem with exponent $\rho = \log(1/s_1) / \log(1/s_2)$.

Regarding the existence of similarity-sensitive families it was shown by Charikar [47] that if the similarity measure $\text{sim}(x, y)$ admits a similarity-sensitive LSH, then $1 - \text{sim}(x, y)$ must be a metric. Recently, Chierichetti and Kumar [49] showed that, given a similarity $\sim$ that admits a similarity-sensitive LSH, the transformed similarity $f(\sim)$ will continue to admit an LSH if $f(\cdot)$ is a probability generating function. The existence of an LSH that admits a similarity measure sim will therefore give rise to the existence of solutions to the approximate similarity

search problem for the much larger class of similarities $f(\text{sim})$. However, this still leaves open the problem of finding efficient explicit constructions, and as it turns out, the property of similarity-sensitive families $\Pr[h(x) = h(y)] = \text{sim}(x,y)$, while intuitively appealing and useful for similarity estimation, does not necessarily imply that the LSH is optimal for solving the approximate search problem. In fact, it was recently shown [50] that for Braun-Blanquet there does not exist a LSH scheme with $\Pr[h(x) = h(y)] = B(x,y) = |x \cap y| / \max(|x|, |y|)$. Moreover, it was shown that MinHash achieves a two-approximation to Braun-Blanquet similarity and that this is optimal for LSH schemes.

The problem of finding tight upper and lower bounds on the ρ-value that can be obtained through the LSH framework for data-independent (s_1, s_2)-approximate similarity search across the entire parameter space (s_1, s_2) remains open for two of the most common measures of set similarity: Jaccard similarity $J(x,y) = |x \cap y| / |x \cup y|$ and cosine similarity $C(x,y) = |x \cap y| / \sqrt{|x||y|}$.

A random function from the MinHash family $\mathcal{H}_{\text{minhash}}$ hashes a set $x \subseteq \{1, \dots, d\}$ to the first element of x in a random permutation of the set $\{1, \dots, d\}$. For $h \sim \mathcal{H}_{\text{minhash}}$ we have that $\Pr[h(x) = h(y)] = J(x,y)$, yielding an LSH solution to the approximate Jaccard similarity search problem. For cosine similarity the SimHash family $\mathcal{H}_{\text{simhash}}$, introduced by Charikar [47], works by sampling a random hyperplane in $\mathbb{R}^d$ that passes through the origin and hashing x according to what side of the hyperplane it lies on. For $h \sim \mathcal{H}_{\text{simhash}}$ we have that $\Pr[h(x) = h(y)] = 1 - \arccos(C(x,y))/\pi$, which can be used to derive a solution for cosine similarity, although not the clean solution that we could have hoped for in the style of MinHash for Jaccard similarity. There exists a number of different data-independent LSH approaches [157, 14, 13] that improve upon the ρ-value of SimHash. Perhaps surprisingly, it turns out that these approaches yield lower ρ-values for the (j_1, j_2)-approximate Jaccard similarity search problem compared to MinHash for certain combinations of (j_1, j_2). Unfortunately, while asymptotically superior these techniques suffer from a non-trivial $o_n(1)$-term in the exponent that only decreases very slowly with n. In comparison, both MinHash and SimHash are simple to describe and have closed expressions for their ρ-values. Furthermore, MinHash and SimHash both have the advantage of being efficient in the sense that a hash function can be represented using space $O(d)$ and the time to compute $h(x)$ is $O(|x|)$. In Table 4.1 we show how the upper bounds for similarity search under different

Table 4.1: Overview of ρ-values for similarity search with Hamming vectors of equal weight t.

Ref. \ Measure	Hamming $r_1 < r_2$	Braun-Blanquet $b_1 > b_2$	Jaccard $j_1 > j_2$
Bit-sampling [91]	r_1/r_2	$\frac{1-b_1}{1-b_2}$	$\frac{1-j_1}{1+j_1} / \frac{1-j_2}{1+j_2}$
MinHash [30]	$\log \frac{1-r_1}{1+r_1} / \log \frac{1-r_2}{1+r_2}$	$\log \frac{b_1}{2-b_1} / \log \frac{b_2}{2-b_2}$	$\log(j_1) / \log(j_2)$
Cross-poly. [13]	$\frac{r_1}{r_2} \frac{1-r_2/2}{1-r_1/2}$	$\frac{1-b_1}{1+b_1} / \frac{1-b_2}{1+b_2}$	$\frac{1-j_1}{1+3j_1} / \frac{1-j_2}{1+3j_2}$
Data-dep. [17]	$\frac{r_1}{r_2} \frac{1}{2-r_1/r_2}$	$\frac{1-b_1}{1+b_1-2b_2}$	$\frac{(1-j_1)(1+j_2)}{1-j_1j_2+3(j_1-j_2)}$
Theorem 4.1	$\log(1-r_1) / \log(1-r_2)$	$\log(b_1) / \log(b_2)$	$\log \frac{2j_1}{1+j_1} / \log \frac{2j_2}{1+j_2}$

Notes: While most results in the literature are stated for a single measure, the fixed weight restriction gives a 1-1 correspondence that makes it possible to express the results in terms of other similarity measures. Hamming distances are normalized by a factor $2t$ to lie in $[0,1]$. Lower order terms of ρ-values are suppressed, and for bit-sampling LSH we assume that the Hamming distances are small relative to the dimensionality of the space, i.e., that $2r_1t/d = o(1)$.

measures of set similarity relate to each other in the case where all sets are t-sparse. In addition to Hamming distance and Jaccard similarity, we consider Braun-Blanquet similarity [28] defined as

$$B(x,y) = |x \cap y| / \max(|x|, |y|), \tag{4.1}$$

which for t-sparse vectors is identical to cosine similarity. When the query and the sets in P can have different sizes the picture becomes muddled, and the question of which of the known algorithms is best for each measure of similarity is complicated and can depend on (s_1, s_2). In Section 4.5 we treat the problem of different set sizes and provide a brief discussion for Jaccard similarity, specifically in relation to our upper bound for Braun-Blanquet similarity.

Similarity search under set similarity and the batched version often referred to as *set similarity join* [20, 23] have also been studied extensively in the information retrieval and database literature, but mostly without providing theoretical guarantees on performance. Recently the notion of containment search, where the similarity measure is the (unnormalized) intersection size, was studied in the LSH framework [151]. This is a special case of *maximum inner product* search [151, 5]. However, these techniques do not give improvements in our setting.

Similarity estimation. Finally, we mention that another application of MinHash [30, 36] is the (easier) problem of *similarity estimation*, where the task is to condense each vector x into a short signature $s(x)$ in such a way that the similarity $J(x, y)$ can be estimated from $s(x)$ and $s(y)$. A related similarity estimation technique was independently discovered by Cohen [60]. Thorup [158] has shown how to perform similarity estimation using just a small amount of randomness in the definition of the function $s(\cdot)$. In another direction, Mitzenmacher et al. [112] showed that it is possible to improve the performance of MinHash for similarity estimation when the Jaccard similarity is close to 1, but for smaller similarities it is known that succinct encodings of MinHash such as the one in [103] comes within a constant factor of the optimal space for storing $s(x)$ [129]. Curiously, our improvement to MinHash in the context of similarity *search* comes when the similarity is neither too large nor too small. Our techniques do not seem to yield any improvement for the similarity *estimation* problem.

4.1.2 Contribution

We show the following upper bound for approximate similarity search under Braun-Blanquet similarity:

Theorem 4.1. *For every choice of constants* $0 < b_2 < b_1 < 1$ *we can solve the* (b_1, b_2)*-approximate similarity search problem under Braun-Blanquet similarity with query time* $O(|q|n^\rho \log n)$ *and space usage* $O(n^{1+\rho} \log n + \sum_{x \in P} |x|)$ *where* $\rho = \log(1/b_1) / \log(1/b_2)$.

In the case where the sets are t-sparse our Theorem 4.1 gives the first strict improvement on the ρ-value for approximate Jaccard similarity search compared to the data-independent LSH approaches of MinHash and Angular LSH. Figure 4.2 shows an example of the improvement for a slice of the parameter space. The improvement is based on a new locality-sensitive mapping that considers a specific random collection of length-k paths on the vertex set $\{1, \ldots, d\}$, and associates each vector x with the paths in the collection that only visits vertices in $\{i \mid x_i = 1\}$. Our data structure exploits that similar vectors will be associated with a common path with constant probability, while vectors with low similarity have a negligible probability of sharing a path. However, the collection of paths has size superlinear in n, so an efficient method is required for locating the paths associated with a particular vector. Our choice of
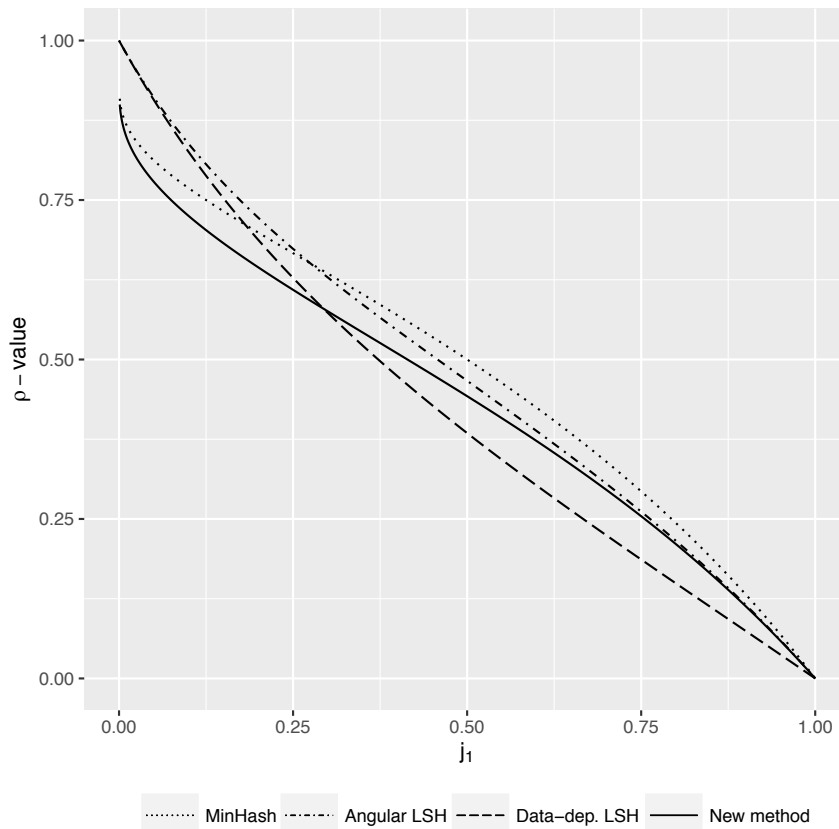
Figure 4.2: Exponent when searching for a vector with Jaccard similarity j_1 with approximation factor 2 (i.e., guaranteed to return a vector with Jaccard similarity $j_1/2$) for various methods in the setting where all sets have the same size. Our new method is the best data-independent method, and is better than data-dependent LSH up to about $j_1 \approx 0.3$.

the collection of paths balances two opposing constraints: It is random enough to match the filtering performance of a truly random collection of sets, and at the same time it is structured enough to allow efficient search for sets matching a given vector. The search procedure is comparable in simplicity to the classical techniques of bit sampling, MinHash, SimHash, and p-stable LSH, and we believe it might be practical. This is in contrast to most theoretical advances in similarity search in the past ten years that suffer from $o(1)$ terms in the exponent of complexity bounds.

Intuition. Recall that we will think of a vector $x \in \{0,1\}^d$ also as a set, $\{i \mid x_i = 1\}$. MinHash can be thought of as a way of sampling an element

i_x from x, namely, we let $i_x = \arg\min_{i \in x} h(i)$ where h is a random hash function. For sets x and y the probability that $i_x = i_y$ equals their Jaccard similarity $J(x, y)$, which is much higher than if the samples had been picked independently. Consider the case in which $|x| = |y| = t$, so $J(x, y) = \frac{|x \cap y|}{2t - |x \cap y|}$. Another way of sampling is to compute $I_x = x \cap b$, where $\Pr[i \in b] = 1/t$, independently for each $i \in [d]$. The expected size of I_x is 1, so this sample has the same expected "cost" as the MinHash-based sample. But if the Jaccard similarity is small, the latter samples are more likely to overlap:

$$\Pr[I_x \cap I_y \neq \varnothing] = 1 - (1 - 1/t)^{|x \cap y|} \approx 1 - e^{-|x \cap y|/t} \approx |x \cap y|/t,$$

nearly a factor of 2 improvement. In fact, whenever $|x \cap y| < 0.6\,t$ we have $\Pr[I_x \cap I_y \neq \varnothing] > \Pr[i_x = i_y]$. So in a certain sense, MinHash is not the best way of collecting evidence for the similarity of two sets. (This observation is not new, and has been made before e.g. in [62].)

Locality-sensitive maps. The intersection of the samples I_x does not correspond directly to hash collisions, so it is not clear how to turn this insight into an algorithm in the LSH framework. Instead, we will consider a generalization of both the locality sensitive filtering (LSF) and LSH frameworks where we define a distribution $\mathcal{M}$ over maps $M \colon \{0, 1\}^d \to 2^R$. The map M performs the same task as the LSH data structure: It takes a vector x and returns a set of memory locations $M(x) \subseteq \{1, \ldots, R\}$. A randomly sampled map $M \sim \mathcal{M}$ has the property that if a pair of points x, y are close then $M(x) \cap M(y) \neq \varnothing$ with constant probability, while if x, y are distant then the expected size $M(x) \cap M(y)$ is small (much smaller than 1). It is now straightforward to see that this distribution can be used to construct a data structure for similarity search by storing each data point $x \in P$ in the set of memory locations or buckets $M(x)$. A query for a point y is performed by computing the similarity between y and every point x contained in the set buckets $M(y)$, reporting the first sufficiently similar point found.

Chosen Path. It turns out that to most efficiently filter out vectors of low similarity in the setting where all sets have equal size, we would like to map each data point $x \in \{0, 1\}^d$ to a collection $M(x)$ of random subsets of $\{0, 1\}^d$ that are contained in x. Furthermore, to best distuinguish

similar from dissimilar vectors when solving the approximate similarity search problem, we would like the random subsets of $\{0,1\}^d$ to have size $\Theta(\log n)$. This leads to another obstacle: The collection of subsets of $\{0,1\}^d$ required to ensure that $M(x) \cap M(y) \neq \varnothing$ for similar points, i.e., that M maps to a subset contained in $x \cap y$, is very large. The space usage and evaluation time of a locality-sensitive map M to fully random subsets of $\{0,1\}^d$ would far exceed n, rendering the solution useless. To overcome this we create the samples in a gradual, correlated way using a pairwise independent branching process that turns out to yield "sufficiently random" samples for the argument to go through.

Lower bound. On the lower bound side we show that our solution for Braun-Blanquet similarity is best possible in terms of parameters b_1 and b_2 within the class of solutions that can be characterized as data-independent locality-sensitive maps. The lower bound works by showing that a family of locality-sensitive maps for Braun-Blanquet similarity with a ρ-value below $\log(1/b_1)/\log(1/b_2)$ can be used to construct a locality-sensitive hash family for the c-approximate near neighbor problem in Hamming space with a ρ-value below $1/c$, thereby contradicting the LSH lower bound by O'Donnell et al. [122]. We state the lower bound here in terms of locality-sensitive hashing, formally defined in Section 4.2.

Theorem 4.2. *For every choice of constants* $0 < b_2 < b_1 < 1$ *any* (b_1, b_2, p_1, p_2)-*sensitive hash family* $\mathcal{H}_B$ *for* $\{0,1\}^d$ *under Braun-Blanquet similarity must satisfy*

$$\rho(\mathcal{H}_B) = \frac{\log(1/p_1)}{\log(1/p_2)} \geq \frac{\log(1/b_1)}{\log(1/b_2)} - O\left(\frac{\log(d/p_2)}{d}\right)^{1/3}.$$

The details showing how this LSH lower bound implies a lower bound for locality-sensitive maps are given in Section 4.4.

4.2 Preliminaries

As stated above we will view $x \in \{0,1\}^d$ both as a vector and as a subset of $[d] = \{1, \ldots, d\}$. Define x to be *t-sparse* if $|x| = t$; we will be interested in the setting where $t \leq d/2$, and typically the sparse setting $t \ll d$. Although many of the concepts we use hold for general spaces, for simplicity we state definitions in the same setting as our

results: the boolean hypercube $\{0,1\}^d$ under some measure of similarity $\text{sim}\colon \{0,1\}^d \times \{0,1\}^d \to [0,1]$.

Definition 4.1. (Approximate similarity search) Let $P \subset \{0,1\}^d$ be a set of $|P| = n$ data vectors, let $\text{sim}\colon \{0,1\}^d \times \{0,1\}^d \to [0,1]$ be a similarity measure, and let $s_1, s_2 \in [0,1]$ such that $s_1 > s_2$. A solution to the (s_1, s_2)-*similarity search problem* is a data structure that supports the following query operation: on input $q \in \{0,1\}^d$ for which there exists a vector $x \in P$ with $\text{sim}(x, q) \geq s_1$, return $x' \in P$ with $\text{sim}(x', q) > s_2$.

Our data structures are randomized, and queries succeed with probability at least $1/2$ (the probability can be made arbitrarily close to 1 by independent repetition). Sometimes similarity search is formulated as searching for vectors that are near q according to the distance measure $\text{dist}(x, y) = 1 - \text{sim}(x, y)$. For our purposes it is natural to phrase conditions in terms of similarity, but we compare to solutions originally described as "near neighbor" methods.

Many of the best known solutions to approximate similarity search problems are based on a technique of randomized space partitioning. This technique has been formalized in the locality-sensitive hashing framework [91] and the closely related locality-sensitive filtering framework [24, 54].

Definition 4.2. (Locality-sensitive hashing [91]) A (s_1, s_2, p_1, p_2)-sensitive family of hash functions for a similarity measure $\text{sim}\colon \{0,1\}^d \times \{0,1\}^d \to [0,1]$ is a distribution $\mathcal{H}_{\text{sim}}$ over functions $h\colon \{0,1\}^d \to R$ such that for all $x, y \in \{0,1\}^d$ and random h sampled according to $\mathcal{H}_{\text{sim}}$:

- If $\text{sim}(x, y) \geq s_1$ then $\Pr[h(x) = h(y)] \geq p_1$.

- If $\text{sim}(x, y) \leq s_2$ then $\Pr[h(x) = h(y)] \leq p_2$.

The range R of the family will typically be fairly small such that an element of R can be represented in a constant number of machine words. In the following we assume for simplicity that the family of hash functions is *efficient* such that $h(x)$ can be computed in time $O(|x|)$. Furthermore, we will assume that the time to compute the similarity $\sim (x, y)$ can be upper bounded by the time it takes to compute the size of the intersection of preprocessed sets, i.e., $O(\min(|x|, |y|))$.

Given a locality-sensitive family it is quite simple to obtain a solution to the approximate similarity search problem, essentially by hashing

points to buckets such that close points end up in the same bucket while distant points are kept apart.

Lemma 4.1 (LSH framework [91, 86]). *Given a (s_1, s_2, p_1, p_2)-sensitive family of hash functions it is possible to solve the (s_1, s_2)-similarity search problem with query time $O(|q|n^\rho \log n)$ and space usage $O(n^{1+\rho} + \sum_{x \in P} |x|)$ where $\rho = \log(1/p_1) / \log(1/p_2)$.*

The upper bound presented in this paper does not quite fit into the existing frameworks. However, we would like to apply existing LSH lower bound techniques to our algorithm. Therefore we define a more general framework that captures solutions constructed using the LSH and LSF framework, as well as the upper bound presented in this paper.

Definition 4.3 (Locality-sensitive map). A (s_1, s_2, m_1, m_2)-sensitive family of maps for a similarity measure sim: $\{0, 1\}^d \times \{0, 1\}^d \to [0, 1]$ is a distribution $\mathcal{M}_{\text{sim}}$ over mappings $M: \{0, 1\}^d \to 2^R$ (where 2^R denotes the power set of R) such that for all $x, y \in \{0, 1\}^d$ and random $M \in \mathcal{M}_{\text{sim}}$:

1. $\mathbb{E}[|M(x)|] \le m_1$.

2. If $\text{sim}(x, y) \le s_2$ then $\mathbb{E}[|M(x) \cap M(y)|] \le m_2$.

3. If $\text{sim}(x, y) \ge s_1$ then $\Pr[M(x) \cap M(y) \ne \varnothing] \ge 1/2$.

Once we have a family of locality-sensitive maps $\mathcal{M}$ we can use it to obtain a solution to the (s_1, s_2)-similarity search problem.

Lemma 4.2. *Given a (s_1, s_2, m_1, m_2)-sensitive family of maps $\mathcal{M}$ we can solve the (s_1, s_2)-similarity search problem with query time $O(m_1 + nm_2|q| + T_M)$ and space usage $O(nm_1 + \sum_{x \in P} |x|)$ where T_M is the time to evaluate a map $M \in \mathcal{M}$.*

Proof. We construct the data structure by sampling a map M from $\mathcal{M}$ and use it to place points in P into buckets. To run a query for a point q we proceed by evaluating $M(q)$ and computing the similarity between q and the points in the buckets associated with $M(q)$. If a sufficiently similar point is found we return it. We get rid of the expectation in the guarantees by independent repetitions and applying Markov's inequality. □

Model of computation. We assume the standard word RAM model [85] with word size $\Theta(\log n)$, where $n = |P|$. In order to be able to draw random functions from a family of functions we augment the model with an instruction that generates a machine word uniformly at random in constant time.

4.3 Upper Bound

We will describe a family of locality-sensitive maps $\mathcal{M}_B$ for solving the (b_1, b_2)-similarity search problem under Braun-Blanquet similarity (4.1). After describing $\mathcal{M}_B$ we will give an efficient implementation of $M \in \mathcal{M}_B$ and show how to set parameters to obtain our Theorem 4.1.

4.3.1 Chosen Path

The CHOSEN PATH family $\mathcal{M}_B$ is defined by k random hash functions $h_1, \ldots, h_k$ where $h_i \colon [w] \times [d]^i \to [0,1]$ and w is a positive integer. The evaluation of a map $M_k \in \mathcal{M}_B$ proceeds in a sequence of $k + 1$ steps that can be analyzed as a Galton-Watson branching process, originally devised to investigate population growth under the assumption of identical and independent offspring distributions. In the first step $i = 0$ we create a population of w starting points

$$M_0(x) = [w]. \tag{4.2}$$

In subsequent steps, every path that has survived so far produces offspring according to a random process that depends on h_i and the element $x \in \{0,1\}^d$ being evaluated. We use $p \circ j$ to denote concatenation of a path p with a vertex j.

$$M_i(x) = \left\{ p \circ j \mid p \in M_{i-1}(x) \wedge h_i(p \circ j) < \frac{x_j}{b_1|x|} \right\}. \tag{4.3}$$

Observe that $h_i(p \circ j) < \frac{x_j}{b_1|x|}$ can only hold when $x_j = 1$, so the paths in $M_i(x)$ are constrained to $j \in x$. The set $M(x) = M_k(x)$ is given by the paths that survive to the kth step. We will proceed by bounding the evaluation time of $M \in \mathcal{M}_B$ as well as showing the locality-sensitive properties of $\mathcal{M}_B$. In particular, for similar points $x, y \in \{0,1\}^d$ with $B(x,y) \geq b_1$ we will show that with probability at least $1/2$ there will be a path that is chosen by both x and y.

Lemma 4.3 (Properties of CHOSEN PATH). *For all $x, y \in \{0,1\}^d$, integer $i \geq 0$, and random $M \in \mathcal{M}_B$:*

1. $\mathbb{E}[|M_i(x)|] \leq (1/b_1)^i w$.

2. *If $B(x,y) < b_2$ then $\mathbb{E}[|M_i(x) \cap M_i(y)|] \leq (b_2/b_1)^i w$.*

3. *If $B(x,y) \geq b_1$ then $\Pr[M_i(x) \cap M_i(y) \neq \varnothing] \geq w/(i+w)$.*

Proof. We prove each property by induction on i. The base cases $i = 0$ follow from (4.2). Now consider the inductive step for property 1. Let $\mathbb{1}\{\mathcal{P}\}$ denote the indicator function for predicate $\mathcal{P}$. Using independence of the hash functions h_i we get:

$$\mathbb{E}[|M_i(x)|] = \mathbb{E}\left[\sum_{p \in M_{i-1}(x)} \sum_{j \in [d]} \mathbb{1}\left\{h_i(p \circ j) < \frac{x_j}{b_1|x|}\right\}\right]$$

$$= \mathbb{E}\left[\sum_{p \in M_{i-1}(x)} 1\right] \mathbb{E}\left[\sum_{j \in [d]} \mathbb{1}\left\{h_i(p \circ j) < \frac{x_j}{b_1|x|}\right\}\right]$$

$$\leq \mathbb{E}[|M_{i-1}(x)|]/b_1$$

$$\leq (1/b_1)^i w \ .$$

The last inequality uses the induction hypothesis. We use the same approach for the second property where we let $X_i = M_i(x) \cap M_i(y)$.

$$\mathbb{E}[|X_i|] = \mathbb{E}\left[\sum_{p \in X_{i-1}} \sum_{j \in [d]} \mathbb{1}\left\{h_i(p \circ j) < \frac{x_j}{b_1|x|} \wedge h_i(p \circ j) < \frac{y_j}{b_1|y|}\right\}\right]$$

$$= \mathbb{E}\left[\sum_{p \in X_{i-1}} 1\right] \sum_{j \in [d]} \Pr\left[h_i(p \circ j) < \frac{\min(x_j, y_j)}{b_1 \max(|x|, |y|)}\right]$$

$$\leq \mathbb{E}[|X_{i-1}|](B(x,y)/b_1)$$

$$\leq (B(x,y)/b_1)^i w \ .$$

To prove the third property we bound the variance of $|X_i|$ and apply Chebyshev's inequality to bound the probability of $X_i = \varnothing$. First consider the case where $|x| \leq 1/b_1$ and $|y| \leq 1/b_1$. Here it must hold that $X_i > 0$ as intersecting paths exist ($b_1 > 0$) and always activate. In all other cases we have that

$$\mathbb{E}[|X_i|] = (B(x,y)/b_1)^i w \ .$$

Knowing the expected value we can apply Chebyshev's inequality once we have an upper bound for $\mathrm{Var}[|X_i|] = \mathbb{E}[|X_i|^2] - \mathbb{E}[|X_i|]^2$. Specifically we show that $\mathbb{E}[|X_i|^2] \leq wi(B(x,y)/b_1)^{2i}$, by induction on i. To simplify notation we define the indicator variable

$$Y_{p,j} = \mathbb{1}\left\{ h_i(p \circ j) < \frac{x_j}{b_1|x|} \wedge h_i(p \circ j) < \frac{y_j}{b_1|y|} \right\}$$

where we suppress the subscript i. First observe that

$$\mathbb{E}[Y_{p,j}] = 1/(b_1 \max(|x|,|y|)) \ .$$

By (4.3) we see that $|X_i| = \sum_{p \in X_{i-1}} \sum_{j \in [d]} Y_{p,j}$, which means:

$$\mathbb{E}[|X_i|^2] = \mathbb{E}\left[\left(\sum_{p \in X_{i-1}} \sum_{j \in [d]} Y_{p,j} \right)^2 \right]$$

$$= \mathbb{E}\left[\sum_{p \in X_{i-1}} \sum_{j \in [d]} Y_{p,j}^2 \right] + \mathbb{E}\left[\sum_{p,p' \in X_{i-1}} \sum_{j,j' \in [d]} Y_{p,j} Y_{p',j'} \mathbb{1}\{(p,j) \neq (p',j')\} \right]$$

$$< \mathbb{E}[|X_{i-1}|](B(x,y)/b_1) + \mathbb{E}[|X_{i-1}|^2](B(x,y)/b_1)^2$$

$$\leq \sum_{s=1}^{i} \mathbb{E}[|X_{i-s}|](B(x,y)/b_1)^{2s-1} + \mathbb{E}[|X_0|]^2 (B(x,y)/b_1)^{2i}$$

$$= \mathbb{E}[|X_i|] \sum_{s=0}^{i-1} (B(x,y)/b_1)^s + \mathbb{E}[|X_i|]^2$$

$$\leq wi(B(x,y)/b_1)^{2i} + \mathbb{E}[|X_i|]^2 \ .$$

The third property now follows from a one-sided version of Chebychev's inequality applied to $|X_i|$. $\qquad \square$

4.3.2 Implementation details

Lemma 4.3 continues to hold when the hash functions $h_1, \dots, h_k$ are individually *2-independent* (and mutually independent) since we only use bounds on the first and second moment of the hash values. We can therefore use a simple and practical scheme such as Zobrist hashing [174] that hashes strings of $\Theta(\log n)$ bits to strings of $\Theta(\log n)$ bits in $O(1)$ time using space, say, $O(n^{1/2})$. It is not hard to see that the domain and range of $h_1, \dots, h_k$ can be compressed to $O(\log n)$ bits (causing a

neglible increase in the failure probability of the data structure). We simply hash the paths $p \in M_i(x)$ to intermediate values of $O(\log n)$ bits, avoiding collisions with high probability, and in a similar vein, with high probability $O(\log n)$ bits of precision suffice to determine whether $h_i(p \circ j) < \frac{x_j}{b_1 |x|}$.

We now consider how to parameterize $\mathcal{M}_B$ to solve the (b_1, b_2)-similarity problem for Braun-Blanquet similarity on a set P of $|P| = n$ points for every choice of constant parameters $0 < b_2 < b_1 < 1$, independent of n. Note that we exclude $b_1 = 1$ (which would correspond to identical vectors that can be found in time $O(1)$ by resorting to standard hashing) and $b_2 = 0$ (for which every data point would be a valid answer to a query). We set parameters

$$
\begin{aligned}
k &= \lceil \log(n) / \log(1/b_2) \rceil, \\
w &= 2k
\end{aligned}
$$

from which it follows that $\mathcal{M}_B$ is (b_1, b_2, m_1, m_2)-sensitive with $m_1 = n^\rho w / b_1$ and $m_2 = n^{\rho-1} w$ where $\rho = \log(1/b_1) / \log(1/b_2)$. To bound the expected evaluation time of M_k we use Zobrist hashing as well as intermediate hashes for the paths as described above. In the ith step in the branching process the expected number of hash function evaluations is bounded by $|q|$ times the number of paths alive at step $i - 1$. We can therefore bound the expected time to compute $M_k(q)$ by

$$
\sum_{i=0}^{k-1} \mathbb{E}[|q||M_i(q)|] \leq \frac{b_1^{-k} - 1}{b_1^{-1} - 1} |q| w = O(|q| n^\rho w). \tag{4.4}
$$

This completes the proof of Theorem 4.1.[1]

4.3.3 Comparison

We will proceed by comparing our Theorem 4.1 to results that can be achieved using existing techniques. Again we focus on the setting where data points and query points are exactly t-sparse. An overview of different techniques for three measures of similarity is shown in Table 4.1.

---

[1] We know of a way of replacing the multiplicative factor $|q|$ in equation (4.4) by an additive term of $O(|q|k)$ by choosing the hash functions h_i carefully, but do not discuss this improvement here since $|q|$ can be assumed to be polylogarithmic and our focus is on the exponent of n.

To summarize: The CHOSEN PATH algorithm of Theorem 4.1 improves upon all existing data-independent results over the entire $0 < b_2 < b_1 < 1$ parameter space. Furthermore, we improve upon the best known *data-dependent* techniques [17] for a large part of the parameter space (see Figure 4.5). The details of the comparisons are given in Appendix 4.8.

MinHash.   For t-sparse vectors there is a 1-1 mapping between Braun-Blanquet and Jaccard similarity. In this setting $J(x,y) = B(x,y)/(2 - B(x,y))$.   Let $b_1 = 2j_1/(j_1 + 1)$ and $b_2 = 2j_2/(j_2 + 1)$ be the Braun-Blanquet similarities corresponding to Jaccard similarities j_1 and j_2.   The LSH framework using MinHash achieves $\rho_{\text{minhash}} = \log\left(\frac{b_1}{2-b_1}\right) / \log\left(\frac{b_2}{2-b_2}\right)$; this should be compared to $\rho = \log(b_1)/\log(b_2)$ achieved in Theorem 4.1.   Since the function $f(z) = \log(\frac{z}{2-z})/\log z$ is monotonically increasing in $[0,1]$ we have that $\rho/\rho_{\text{minhash}} = f(b_2)/f(b_1) < 1$, i.e., ρ is always smaller than ρ_{minhash}.   As an example, for $j_1 = 0.2$ and $j_2 = 0.1$ we get $\rho = 0.644...$ while $\rho_{\text{minhash}} = 0.698....$ Figure 4.3 shows the difference for the whole parameter space.

Angular LSH.   Since our vectors are exactly t-sparse Braun-Blanquet similarities correspond directly to dot products (which in turn correspond to angles).   Thus we can apply angular LSH such as SimHash [47] or cross-polytope LSH [13]. As observed in [54] one can express the ρ-value of cross-polytope LSH in terms of dot products as $\rho_{\text{angular}} = \frac{1-b_1}{1+b_1} / \frac{1-b_2}{1+b_2}$. Since the function $f'(z) = (1+z)\log(z)/(1-z)$ is negative and monotonically increasing in $[0,1]$ we have that $\rho/\rho_{\text{angular}} = f'(b_1)/f'(b_2) < 1$, i.e., ρ is always smaller than ρ_{angular}. In the above example, for $j_1 = 0.2$ and $j_2 = 0.1$ we have $\rho_{\text{angular}} = 0.722...$ which is about 0.078 more than CHOSEN PATH. See Figure 4.4 for a visualization of the difference for the whole parameter space.

Data-dependent Hamming LSH.   The Hamming distance between two t-sparse vectors with Braun-Blanquet similarity b is $2t(1-b)$, since the intersection of the vectors has size tb. This means that (b_1, b_2)-similarity search under Braun-Blanquet similarity can be reduced to Hamming similarity search with approximation factor $c = (2t(1-b_1))/(2t(1-b_2)) = (1-b_1)/(1-b_2)$. As mentioned above, the *data dependent* LSH technique of [17] achieves $\rho = 1/(2c-1)$ ignoring $o_n(1)$ terms. In terms of b_1 and b_2 this is $\rho_{\text{datadep}} = \frac{1-b_1}{1+b_1-2b_2}$, which in incomparable to the ρ
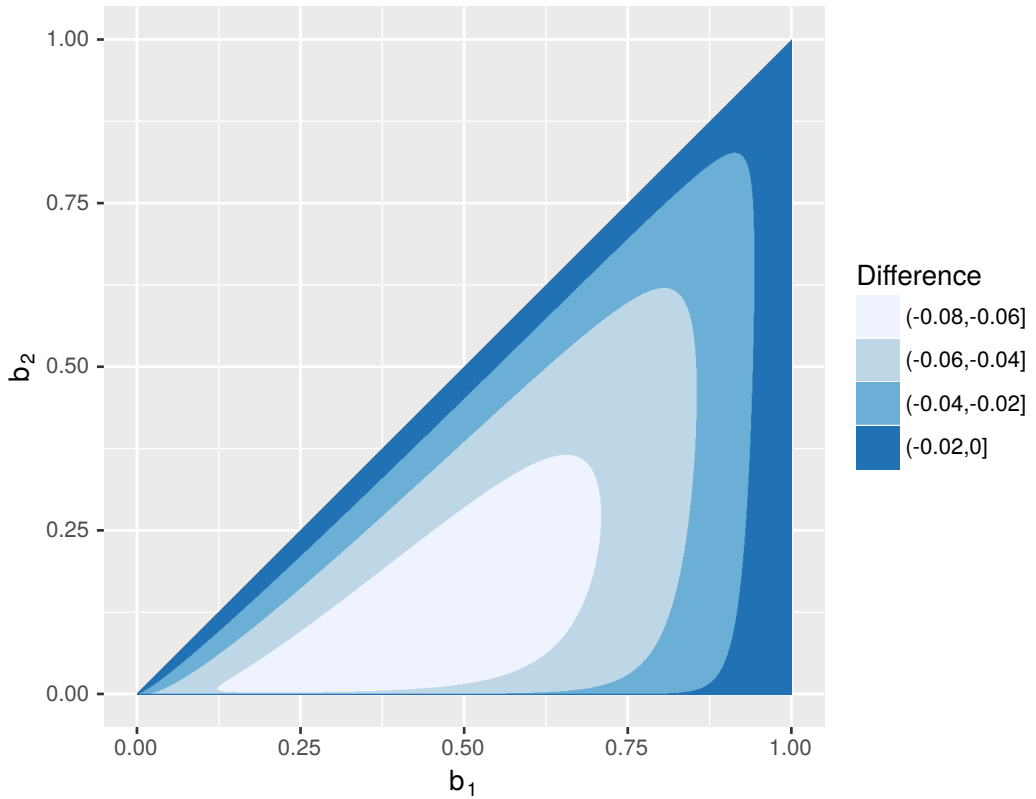
Figure 4.3: The difference $\rho - \rho_{\text{minhash}}$ comparing Chosen Path and MinHash in terms of Braun-Blanquet similarities $0 < b_2 < b_1 < 1$.

of Theorem 4.1. In Appendix 4.8 we show that $\rho < \rho_{\text{datadep}}$ whenever $b_2 \leq 1/5$, or equivalently, whenever $j_2 \leq 1/9$. Revisiting the above example, for $j_1 = 0.2$ and $j_2 = 0.1$ we have $\rho_{\text{datadep}} = 0.6875$ which is about 0.043 more than CHOSEN PATH. Figure 4.5 gives a comparison covering the whole parameter space.

4.4   Lower bound

In this section we will show a locality-sensitive hashing lower bound for $\{0, 1\}^d$ under Braun-Blanquet similarity. We will first show that LSH lower bounds apply to the class of solutions to the approximate similarity search problem that are based on locality-sensitive maps, thereby including our own upper bound. Next we will introduce some relevant tools from the literature, in particular the LSH lower bounds for Hamming
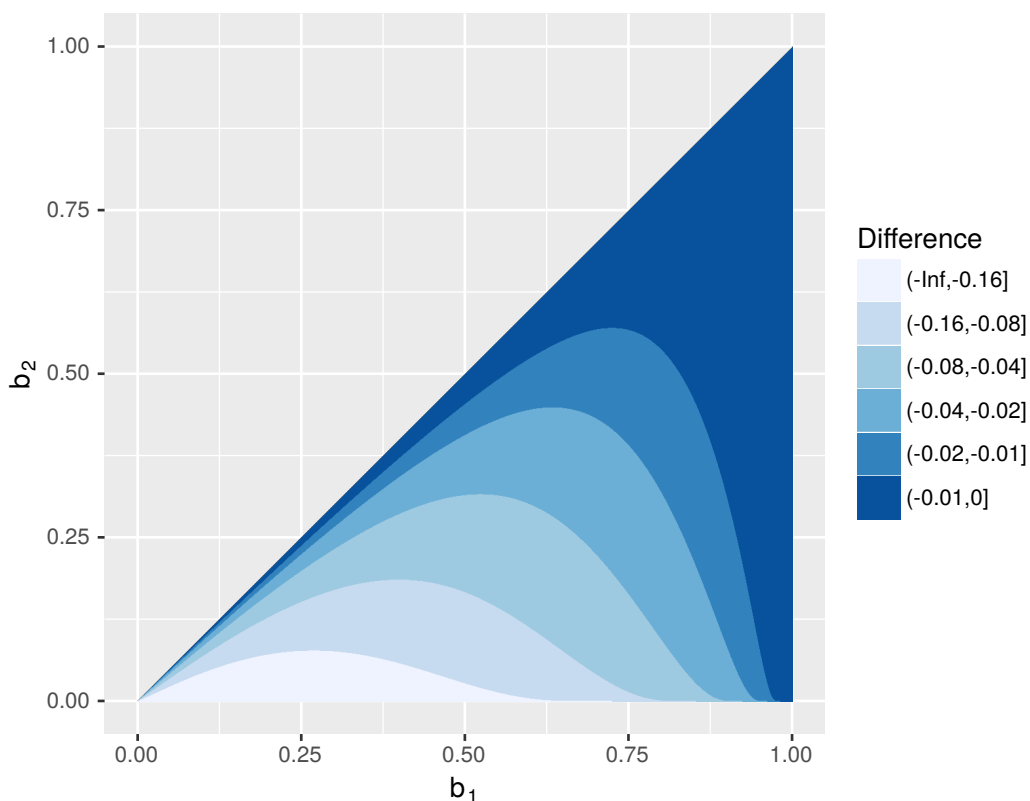
Figure 4.4: The difference $\rho - \rho_{\text{angular}}$ comparing Chosen Path and angular LSH in terms of Braun-Blanquet similarities $0 < b_2 < b_1 < 1$.

space by O'Donnell et al. [122] which we use, through a reduction, to show LSH lower bounds under Braun-Blanquet similarity.

Lower bounds for locality-sensitive maps.   Because our upper bound is based on a locality-sensitive map $\mathcal{M}_B$ and not LSH-based we first show that LSH lower bounds apply to LSM-based solutions. This is not too surprising as both the LSH and LSF frameworks produce LSM-based solutions. We note that the idea of showing lower bounds for a more general class of algorithms that encompasses both LSH and LSF was used by Andoni et al. [16] in their list-of-points data structure lower bound for the space-time tradeoff of solutions to the approximate near neighbor problem in the random data regime. We use the approach of Christiani [54] to convert an LSM family into an LSH family using MinHash.
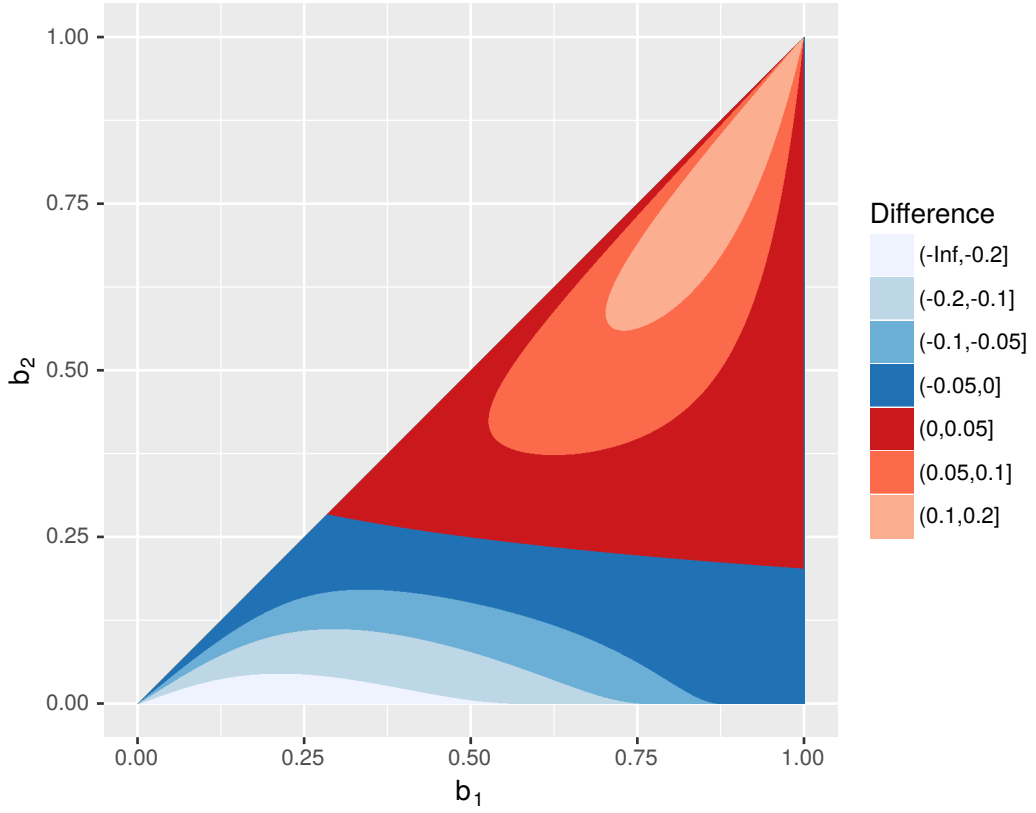
Figure 4.5: The difference $\rho - \rho_{\text{datadep}}$ comparing Chosen Path and data-dependent LSH in terms of Braun-Blanquet similarities $0 < b_2 < b_1 < 1$. In the area of the parameter space that is colored blue we have that $\rho \leq \rho_{\text{datadep}}$ while for the red area it holds that $\rho > \rho_{\text{datadep}}$.

Lemma 4.4. *Suppose we have a* (s_1, s_2, m_1, m_2)*-sensitive family of maps* $\mathcal{M}$. *Then we can construct a* (s_1, s_2, p_1, p_2)*-sensitive family of hash functions* $\mathcal{H}$ *with* $p_1 = 1/8m$ *and* $p_2 = m_2/m$ *where* $m = \lceil 8m_1 \rceil$.

Proof. We sample a function h from $\mathcal{H}$ by sampling a function M from $\mathcal{M}$, modify M to output a set of fixed size, and apply MinHash to the resulting set. For $M \in \mathcal{M}$ we define the function $\tilde{M}$ where we ensure that the size of the output set is m. We note that the purpose of this step is to be able to simultaneously lower bound p_1 and upper bound p_2 for $\mathcal{H}$ when we apply MinHash to the resulting sets.

$$\tilde{M}(x) = \begin{cases} \{(x, 1), \ldots, (x, m)\} & \text{if } |M(x)| \geq m, \\ \{(x, 1), \ldots, (x, m - |M(x)|)\} \cup M(x) & \text{otherwise.} \end{cases}$$

We proceed by applying MinHash to the set $\tilde{M}(x)$. Let π denote a random permutation of the range of $\tilde{M}$ and define

$$h(x) = \arg\min_{z \in \tilde{M}(x)} \pi(z).$$

We then have

$$\Pr[h(x) = h(y)] = \sum_{\xi} \Pr[J(\tilde{M}(x), \tilde{M}(y)) = \xi] \cdot \xi$$

summing over the finite set of all possible Jaccard similarities $\xi = a/b$ with $a, b \in \{0, 1, \ldots, 2m\}$. It is now fairly simple to lower bound p_1 and upper bound p_2. Assume that x, y satisfy that $\mathrm{sim}(x, y) \geq s_1$. To lower bound p_1 we use a union bound together with Markov's inequality to bound the following probability:

$$\begin{aligned}
\Pr[\tilde{M}(x) &\cap \tilde{M}(y) = \varnothing] \\
&\leq \Pr[M(x) \cap M(y) = \varnothing \vee |M(x)| \geq m \vee |M(y)| \geq m] \\
&\leq \Pr[M(x) \cap M(y) = \varnothing] + \Pr[|M(x)| \geq m] + \Pr[|M(y)| \geq m] \\
&\leq 1/2 + 1/8 + 1/8
\end{aligned}$$

We therefore have that $\Pr[\tilde{M}(x) \cap \tilde{M}(y) \neq \varnothing] \geq 1/4$. In the event of a nonempty intersection the probability of collision is given by $J(\tilde{M}(x) \cap \tilde{M}(y)) \geq 1/2m$ allowing us to conclude that $p_1 \geq 1/8m$.

Bounding the collision probability for distant pairs of points x, y with $\sim (x, y) \leq s_2$ we get

$$\sum_{\xi} \Pr[J(\tilde{M}(x), \tilde{M}(y)) = \xi] \cdot \xi \leq (1/m) \sum_{i=1}^{\infty} \Pr[|\tilde{M}(x) \cap \tilde{M}(y)|] \cdot i = \frac{m_2}{m}.$$

$\square$

We are now ready to justify the statement that LSH lower bounds apply to LSM, allowing us to restrict our attention to proving LSH lower bounds for Braun-Blanquet similarity.

Corollary 4.1. *Suppose that we have an LSM-based solution to the (s_1, s_2)-similarity search problem with query time $O(n^\rho)$. Then there exists a family $\mathcal{H}$ of locality-sensitive hash functions with $\rho(\mathcal{H}) = \rho + O(1/\log n)$.*

Proof. The existence of the LSM-based solution implies that for every n there exists a (s_1, s_2, m_1, m_2)-sensitive family of maps $\mathcal{M}$ with $m_1 = O(n^\rho)$ and $nm_2 = O(n^\rho)$. The upper bound on ρ follows from applying Lemma 4.4. $\square$

LSH lower bounds for Hamming space. There exist a number of powerful results that lower bound the ρ-value that is attainable by locality-sensitive hashing and related approaches in various settings [114, 133, 122, 19, 54, 16]. O'Donnell et al. [122] showed an LSH lower bound of $\rho = \log(1/p_1)/\log(1/p_2) \geq 1/c - o_d(1)$ for d-dimensional Hamming space under the assumption that p_2 is not too small compared to d, i.e., $\log(1/p_2) = o(d)$. The lower bound by O'Donnell et al. holds for (r, cr, p_1, p_2)-sensitive families for a particular choice of r that depends on d, p_2, and c, and where r is small compared to d (for instance, we have that $r = \tilde{\Theta}(d^{2/3})$ when c and p_2 are constant).

We state a simplified version of the lower bound due to O'Donnell et al. where $r = \sqrt{d}$ that we will use as a tool to prove our lower bound for Braun-Blanquet similarity. The full proof of Lemma 4.5 is given in Appendix 4.7.

Lemma 4.5. *For every $d \in \mathbb{N}$, $1/d \leq p_2 \leq 1 - 1/d$, and $1 \leq c \leq d^{1/8}$ every $(\sqrt{d}, c\sqrt{d}, p_1, p_2)$-sensitive hash family $\mathcal{H}$ for $\{0,1\}^d$ under Hamming distance must have*

$$\rho(\mathcal{H}) = \frac{\log(1/p_1)}{\log(1/p_2)} \geq \frac{1}{c} - O(d^{-1/4}). \tag{4.5}$$

In general, good lower bounds for the entire parameter space (r, cr) are not known, although the techniques by O'Donnell et al. appear to yield a bound of $\rho \gtrsim \log(1 - 2r/d)/\log(1 - 2cr/d)$. This is far from tight as can be seen by comparing it to the bit-sampling [91] upper bound of $\rho = \log(1 - r/d)/\log(1 - cr/d)$. Existing lower bounds are tight in two different settings. First, in the setting where $cr \approx d/2$ (random data), lower bounds [114, 73, 19] match various instantiations of angular LSH [157, 14, 13]. Second, in the setting where $r \ll d$, the lower bound by O'Donnell et al. [122] becomes $\rho \gtrsim \log(1 - 2r/d)/\log(1 - 2cr/d) \approx 1/c$, matching bit-sampling LSH [91] as well as Angular LSH.

4.4.1 Braun-Blanquet LSH lower bound

We are now ready to prove the LSH lower bound from Theorem 4.2. The lower bound together with Corollary 4.1 shows that the ρ-value of Theorem 4.1 is best possible up to $o_d(1)$ terms within the class of data-independent locality-sensitive maps for Braun-Blanquet similarity. Furthermore, the lower bound also applies to angular distance on the unit sphere where it comes close to matching the best known upper bounds for much of the parameter space as can be seen from Figure 4.4.

Proof sketch. The proof works by assuming the existence of a (b_1, b_2, p_1, p_2)-sensitive family $\mathcal{H}_B$ for $\{0,1\}^d$ under Braun-Blanquet similarity with $\rho = \log(1/b_1)/\log(1/b_2) - \gamma$ for some $\gamma > 0$. We use a transformation T from Hamming space to Braun-Blanquet similarity to show that the existence of $\mathcal{H}_B$ implies the existence of a (r, cr, p'_1, p'_2)-sensitive family $\mathcal{H}_H$ for D-dimensional Hamming space that will contradict the lower bound of O'Donnell et al. [122] as stated in Lemma 4.5 for some appropriate choice of $\gamma = \gamma(d, p_2)$.

We proceed by giving an informal description of a simple "tensoring" technique for converting a similarity search problem in Hamming space into a Braun-Blanquet set similarity problem for target similarity thresholds b_1, b_2. For $x \in \{0,1\}^d$ define

$$\tilde{x} = \{(i, x_i) \mid i \in [d]\}$$

and for a positive integer τ define $x^{\otimes \tau} = \{(v_1, \ldots, v_\tau) \mid v_i \in \tilde{x}\}$. We have that $|x^{\otimes \tau}| = |\tilde{x}|^\tau = d^\tau$ and

$$B(x^{\otimes \tau}, y^{\otimes \tau}) = |\tilde{x} \cap \tilde{y}|^\tau / |\tilde{x}|^\tau = (1 - r/d)^\tau$$

where $r = \|x - y\|_1$. For every choice of constants $0 < b_2 < b_1 < 1$ we can choose d, τ, r, and $c \geq 1$ such that $(1 - r/d)^\tau \approx b_1$ and $(1 - cr/d)^\tau \approx b_2$. Now, given an LSH family for Braun-Blanquet with $\rho < \log(1/b_1)/\log(1/b_2)$ we would be able to obtain an LSH family for Hamming space with

$$\rho < \frac{\log(1/b_1)}{\log(1/b_2)} = \frac{\log(1/(1 - r/d))}{\log(1/(1 - cr/d))} \leq 1/c.$$

For appropriate choices of parameters this would contradict the O'Donnell et al. LSH lower bound of $\rho \gtrsim 1/c$ for Hamming space. The proof itself is mostly an exercise in setting parameters and applying the right bounds and approximations to make everything fit together with the intuition above. Importantly, we use sampling in order to map to a dimension that is much lower than the d^τ from the proof sketch in order to make the proof hold for small values of p_2 in relation to d.

Hamming distance to Braun-Blanquet similarity. Let $d \in \mathbb{N}$ and let $0 < b_2 < b_1 < 1$ be constant as in Theorem 4.2. Let $\varepsilon \geq 1/d$ be a parameter to be determined. We want to show how to use a transformation

$T\colon \{0,1\}^D \to \{0,1\}^d$ from Hamming distance to Braun-Blanquet similarity together with our family $\mathcal{H}_B$ to construct a (r, cr, p_1', p_2')-sensitive family $\mathcal{H}_H$ for D-dimensional Hamming space with parameters

$$D = 2^d$$
$$r = \sqrt{D}$$
$$c = \frac{\ln(1/(b_2 - \varepsilon))}{\ln(1/(b_1 + \varepsilon))}$$

where p_1' and p_2' remain to be determined.

The function T takes as parameters positive integers t, l, and τ. The output of T consists of t concatenated l-bit strings, each of of Hamming weight one. Each of the t strings is constructed independently at random according to the following process: Sample a vector of indices $i = (i_1, i_2, \ldots, i_\tau)$ uniformly at random from $[D]^\tau$ and define $x_i \in \{0,1\}^\tau$ as $x_i = x_{i_1} \circ x_{i_2} \circ \cdots \circ x_{i_\tau}$. Let $z(x) \in \{0,1\}^{2^\tau}$ be indexed by $j \in \{0,1\}^\tau$ and set the bits of $z(x)$ as follows:

$$z(x)_j = \begin{cases} 1 & \text{if } x_i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Next we apply a random function $g\colon \{0,1\}^\tau \to [l]$ in order to map $z(x)$ down to an l-bit string $r(z(x))$ of Hamming weight one while approximately preserving Braun-Blanquet similarity. For $i \in [l]$ we set

$$r(z(x))_i = \bigvee_{j:g(j)=i} z(x)_j.$$

Finally we set

$$T(x) = r_1(z_1(x)) \circ r_2(z_2(x)) \circ \cdots \circ r_t(z_t(x))$$

where each $r_i(z_i(x))$ is constructed independently at random.

We state the properties of T for the following parameter setting:

$$\tau = \lfloor \sqrt{D} \ln(1/(b_1 + \varepsilon)) \rfloor$$
$$l = \lceil 8/\varepsilon \rceil$$
$$t = \lfloor d/l \rfloor.$$

Lemma 4.6. *For every $d \in \mathbb{N}$ and $D = 2^d$ there exists a distribution over functions of the form $T\colon \{0,1\}^D \to \{0,1\}^d$ such that for all $x, y \in \{0,1\}^D$ and random T:*

1. $|T(x)| = t$.

2. If $\|x - y\|_1 \leq r$ then $B(T(x), T(y)) \geq b_1$ with probability at least $1 - e^{-t\varepsilon^2/2}$.

3. If $\|x - y\|_1 > cr$ then $B(T(x), T(y)) < b_2$ with probability at least $1 - 2e^{-t\varepsilon^2/32}$.

Proof. The first property is trivial. For the second property we consider x, y with $\|x - y\|_1 \leq r$ where we would like to lower bound

$$B(T(x), T(y)) = \frac{|T(x) \cap T(y)|}{\max(|T(x)|, |T(y)|)}.$$

We know that $|T(x)| = |T(y)| = t$ so it remains to lower bound the size of the intersection $|T(x) \cap T(y)|$. Consider the expectation

$$\mathbb{E}[|T(x) \cap T(y)|] = t \Pr[z(x) = z(y)].$$

We have that $z(x) = z(y)$ if x and y take on the same value in the τ underlying bit-positions that are sampled to construct z. Under the assumption that $\varepsilon \geq 1/d$, then for d greater than some sufficiently large constant we can use a standard approximation to the exponential function (detailed in Lemma 4.10 in Appendix 4.7) to show that

$$\begin{aligned}
\Pr[z(x) = z(y)] &\geq (1 - r/D)^\tau \\
&\geq (1 - 1/\sqrt{D})^{\sqrt{D}\ln(1/(b_1+\varepsilon))} \\
&\geq e^{\ln(b_1+\varepsilon)}(1 - (\ln(b_1+\varepsilon))^2/\sqrt{D}) \\
&\geq b_1 + \varepsilon/2.
\end{aligned}$$

Seeing as $|T(x) \cap T(y)|$ is the sum of t independent Bernoulli trials we can apply Hoeffding's inequality to yield the following bound:

$$\Pr[|T(x) \cap T(y)| \leq b_1 t] \leq e^{-t\varepsilon^2/2}.$$

This proves the second property of T.

For the third property we consider the Braun-Blanquet similarity of distant pairs of points x, y with $\|x - y\|_1 > cr$. Again, under our

assumption that $\varepsilon \geq 1/d$ and for d greater than some constant we have

$$\Pr[z(x) = z(y)] \leq (1 - cr/D)^\tau$$

$$\leq \frac{\left(1 - \frac{\ln(1/(b_2 - \varepsilon))}{\sqrt{D}\ln(1/(b_1 + \varepsilon))}\right)^{\sqrt{D}\ln(1/(b_1 + \varepsilon))}}{1 - c/\sqrt{D}}$$

$$\leq (1 + 2c/\sqrt{D})(b_2 - \varepsilon)$$

$$\leq b_2 - \varepsilon/2.$$

There are two things that can cause the event $B(T(x), T(y)) < b_2$ to fail. First, the sum of the t independent Bernoulli trials for the event $z(x) = z(x')$ can deviate too much from its expected value. Second, the mapping down to l-bit strings that takes place from $z(x)$ to $r(z(x))$ can lead to an additional increase in the similarity due to collisions. Let Z denote the sum of the t Bernoulli trials for the events $z(x) = z(x')$ associated with T. We again apply a standard Hoeffding bound to show that

$$\Pr[Z \geq (b_2 - \varepsilon/4)t] \leq e^{-t\varepsilon^2/8}.$$

Let X denote the number of collisions when performing the universe reduction to l-bit strings. By our choice of l we have that $E[X] \leq (\varepsilon/8)t$. Another application of Hoeffding's inequality shows that

$$\Pr[X \geq (\varepsilon/4)t] \leq e^{-t\varepsilon^2/32}.$$

We therefore get that

$$\Pr[|T(x) \cap T(x')| \geq b_2 t] \leq 2e^{-t\varepsilon^2/32}.$$

This proves the third property of T. $\qquad\qquad\qquad\qquad\qquad\qquad\square$

Contradiction. To summarize, using the random map T together with the LSH family $\mathcal{H}_B$ we can obtain an (r, cr, p_1', p_2')-sensitive family $\mathcal{H}_H$ for D-dimensional Hamming space with $p_1' = p_1 - \delta$ and $p_2' = p_2 + \delta$ for $\delta = 2e^{-t\varepsilon^2/32}$. For our choice of $c = \frac{\ln(1/(b_2 - \varepsilon))}{\ln(1/(b_1 + \varepsilon))}$ we plug the family $\mathcal{H}_H$ into the lower bound of Lemma 4.5 and use that $O(D^{-1/4}) = O(\varepsilon)$

which follows from our constraint that $\varepsilon \geq 1/d$.

$$
\begin{aligned}
\rho(\mathcal{H}_H) &\geq 1/c - O(D^{-1/4}) \\
&= \frac{\ln(1/(1+\varepsilon/b_1)) + \ln(1/b_1)}{\ln(1/(1-\varepsilon/b_2)) + \ln(1/b_2)} - O(\varepsilon) \\
&\geq \frac{\ln(1/b_1) - \varepsilon/b_1}{\ln(1/b_2) + 2\varepsilon/b_2} - O(\varepsilon) \\
&= \frac{\ln(1/b_1)}{\ln(1/b_2)} - O(\varepsilon)
\end{aligned}
$$

Under our assumed properties of $\mathcal{H}_B$, we can upper bound the value of ρ for $\mathcal{H}_H$. For simplicity we temporarily define $\lambda = 2\delta/p_2$ and assume that $\lambda/\ln(1/p_2) \leq 1/2$ and $\ln(1/p_2) \geq 1$. The latter property holds without loss of generality through use of the standard LSH powering technique [91, 86, 122] that allows us to transform an LSH family with $p_2 < 1$ to a family that has $p_2 \leq 1/e$ without changing its associated ρ-value.

$$
\begin{aligned}
\rho(\mathcal{H}_H) &= \frac{\ln(1/p_1')}{\ln(1/p_2')} = \frac{\ln(1/p_1) + \ln(1/(1-\delta/p_1))}{\ln(1/p_2) + \ln(1/(1+\delta/p_2))} \\
&\leq \frac{\ln(1/p_1) + \lambda}{\ln(1/p_2) - \lambda} = \frac{\ln(1/p_1) + \lambda}{(\ln 1/p_2)(1 - \lambda/(\ln 1/p_2))} \\
&\leq \frac{\ln(1/p_1) + \lambda}{\ln(1/p_2)}(1 + 2\lambda/(\ln 1/p_2)) = \frac{\ln(1/p_1)}{\ln(1/p_2)} + O(\delta/p_2) \\
&\leq \frac{\ln(1/b_1)}{\ln(1/b_2)} - \gamma + O(\delta/p_2).
\end{aligned}
$$

We get a contradiction between our upper bound and lower bound for $\rho(\mathcal{H}_H)$ whenever γ violates the following relation that summarizes the bounds:

$$
\frac{\ln(1/b_1)}{\ln(1/b_2)} - O(\varepsilon) \leq \rho(\mathcal{H}_H) \leq \frac{\ln(1/b_1)}{\ln(1/b_2)} - \gamma + O(\delta/p_2).
$$

In order for a contradiction to occur, the value of γ has to satisfy

$$
\gamma > O(\varepsilon) + O(\delta/p_2).
$$

By our setting of $t = \lfloor d/l \rfloor$ and $l = \lceil 8/\varepsilon \rceil$ we have that $\delta = e^{-\Omega(d\varepsilon^3)}$. We can cause a contradiction for a setting of $\varepsilon^3 = K\frac{\ln(d/p_2)}{d}$ where K is some

constant and where we assume that d is greater than some constant. The value of γ for which the lower bound holds can be upper bounded by

$$\gamma = O\left(\frac{\ln(d/p_2)}{d}\right)^{1/3}.$$

This completes the proof of Theorem 4.2.

4.5   Equivalent set similarity problems

In this section we consider how to use our data structure for Braun-Blanquet similarity search to support other similarity measures such as Jaccard similarity. We already observed in the introduction that a direct translation exists between several similarity measures whenever the size of every sets is fixed to t. Call an (s_1, s_2)-similarity search problem *(t,t')-regular* if P is restricted to vectors of weight t and queries are restricted to vectors of weight t'. Obviously, a (t, t')-regular similarity search problem is no harder than the general similarity search problem, but it also cannot be too much easier when expressed as a function of the thresholds (s_1, s_2): For every pair $(t, t') \in \{0, \dots, d\}^2$ we can construct a (t,t')-regular data structure (such that each point $x \in P$ is represented in the $d+1$ data structures with $t = |x|$), and answer a query for $q \in \{0, 1\}^d$ by querying all data structures with $t' = |q|$. Thus, the time and space for the general (s_1, s_2)-similarity search problem is at most $d+1$ times larger than the time and space of the most expensive (t,t')-regular data structure. This does *not* mean that we cannot get better bounds in terms of other parameters, and in particular we expect that the difficulty of (t, t')-regular similarity search problems depends on parameters t and t'.

Dimension reduction.   If the dimension is large a factor of d may be significant. However, for most natural similarity measures a (s_1, s_2)-similarity problem in $d \gg (\log n)^3$ dimensions can be reduced to a logarithmic number of (s_1', s_2')-similarity problems on $P' \subseteq \{0, 1\}^{d'}$ in $d' = (\log n)^3$ dimensions with $s_1' = s_1 - O(1/\log n)$ and $s_2' = s_2 + O(1/\log n)$. Since the similarity gap is close to the one in the original problem, $s_1' - s_2' = s_1 - s_2 - O(1/\log n)$, where s_1 and s_2 are assumed to be independent of n, the difficulty (ρ-value) remains essentially the same. First, split P into $\log d$ size classes P_i such that vectors in class i have size in $[2^i, 2^{i+1})$. For each size class the reduction is done independently

and works by a standard technique: sample a sequence of random sets $I_j \subseteq \{1, \ldots, d\}$, $i = 1, \ldots, d'$, and set $x'_j = \vee_{\ell \in I_j} x_\ell$. The size of each set I_j is chosen such that $Pr[x'_j = 1] \approx 1/\log(n)$ when $|x| = 2^{i+1}$. By Chernoff bounds this mapping preserves the relative weight of vectors up to size $2^i \log n$ up to an additive $O(1/\log n)$ term with high probability. Assume now that the similarity measure is such that for vectors in P_i we only need to consider $|q|$ in the range from $2^i/\log n$ to $2^i \log n$ (since if the size difference is larger, the similarity is negligible). The we can apply Chernoff bounds to the relative weights of the dimension-reduced vectors x', q' and the intersection $x' \cap q'$. In particular, we get that the Jaccard similarity of a pair of vectors is preserved up to an additive error of $O(1/\log n)$ with high probability. The class of similarity measures for which dimension reduction to $(\log n)^{O(1)}$ dimensions is possible is large, and we do not attempt to characterize it here. Instead, we just note that for such similarity measures we can determine the complexity of similarity search up to a factor $(\log n)^{O(1)}$ by only considering regular search problems.

Equivalence of regular similarity search problems. We call a set similarity measure on $\{0,1\}^d$ *symmetric* if it can be written in the form $S(q,x) = f_{d,|q|,|x|}(|q \cap x|)$, where each function $f_{d,|q|,|x|} : \mathbb{N} \to [0,1]$ is non-decreasing. All 59 set similarity measures listed in the survey [51], normalized to yield similarities in $[0,1]$, are symmetric. In particular this is the case for Jaccard similarity (where $J(q,x) = |q \cap x|/(|q| + |x| - |q \cap x|)$) and for Braun-Blanquet similarity. For a symmetric similarity measure, the predicate $\text{sim}(q,x) \geq s_1$ is equivalent to the predicate $|q \cap x| \geq i_1$, where $i_1 = \min\{i \mid f_{d,t',t}(i) \geq s_1\}$, and $\text{sim}(q,x) > s_2$ is equivalent to the predicate $|q \cap x| \geq i_2$, where $i_2 = \min\{i \mid f_{d,t',t}(i) > s_2\}$. This means that every (t,t')-regular (s_1,s_2)-similarity search problem on $P \subseteq \{0,1\}^d$ is equivalent to an $(i_1/d, i_2/d)$-similarity search problem on P, where $\text{sim}(q,x) = |x \cap q|/d$. In other words, all symmetric similarity search problems can be translated to each other, and it suffices to study a single one, such as Braun-Blanquet similarity.

Jaccard similarity. We briefly discuss Jaccard similarity since it is the most widely used measure of set similarity. If we consider the problem of (j_1, j_2)-approximate Jaccard similarity search in the (t,t')-regular case with $t \neq t'$ then our Theorem 4.1 is no longer guaranteed to yield the lowest value of ρ among competing data-independent approaches such

as MinHash and Angular LSH. To simplify the comparision between different measures we introduce parameters β and b defined by $|y| = \beta|x|$ and $b = |x \cap y|/|x|$ (note that $0 \leq b \leq \beta \leq 1$). The three primary measures of set similarity considered in this paper can then be written as follows:

$$B(x,y) = b$$
$$J(x,y) = \frac{b}{1 + \beta - b}$$
$$C(x,y) = \frac{b}{\sqrt{\beta}}$$

As shown in Figure 4.6 among angular LSH, MinHash, and CHOSEN PATH, the technique with the lowest ρ-value is different depending on the parameters (j_1, j_2) and asymmetry β. We know that CHOSEN PATH is optimal and strictly better than the competing data-independent techniques across the entire parameter space (j_1, j_2) when $\beta = 1$, but it remains open to find tight upper and lower bounds in the case where $\beta \neq 1$.

4.6   Conclusion and open problems

We have seen that, perhaps surprisingly, there exists a relatively simple way of strictly improving the ρ-value for data-independent set similarity search in the case where all sets have the same size. To implement the required locality-sensitive map efficiently we introduce a new technique based on branching processes that could possibly lead to more efficient solutions in other settings.

It remains an open problem to find tight upper and lower bounds on the ρ-value for Jaccard and cosine similarity search that hold for the entire parameter space in the general setting with arbitrary set sizes. Perhaps a modified version of the CHOSEN PATH algorithm can yield an improved solution to Jaccard similarity search in general. One approach is to generalize the condition $h_i(p \circ j) < x_j/b_1|x|$ to use different thresholds for queries and updates. This yields different space-time tradeoffs when applying the CHOSEN PATH algorithm to Jaccard similarity search.

Another interesting question is if the improvement shown for sparse vectors can be achieved in general for inner product similarity. A similar, but possibly easier, direction would be to consider *weighted* Jaccard similarity.

(a) $\beta = 0.25$

(b) $\beta = 0.5$

(c) $\beta = 0.75$
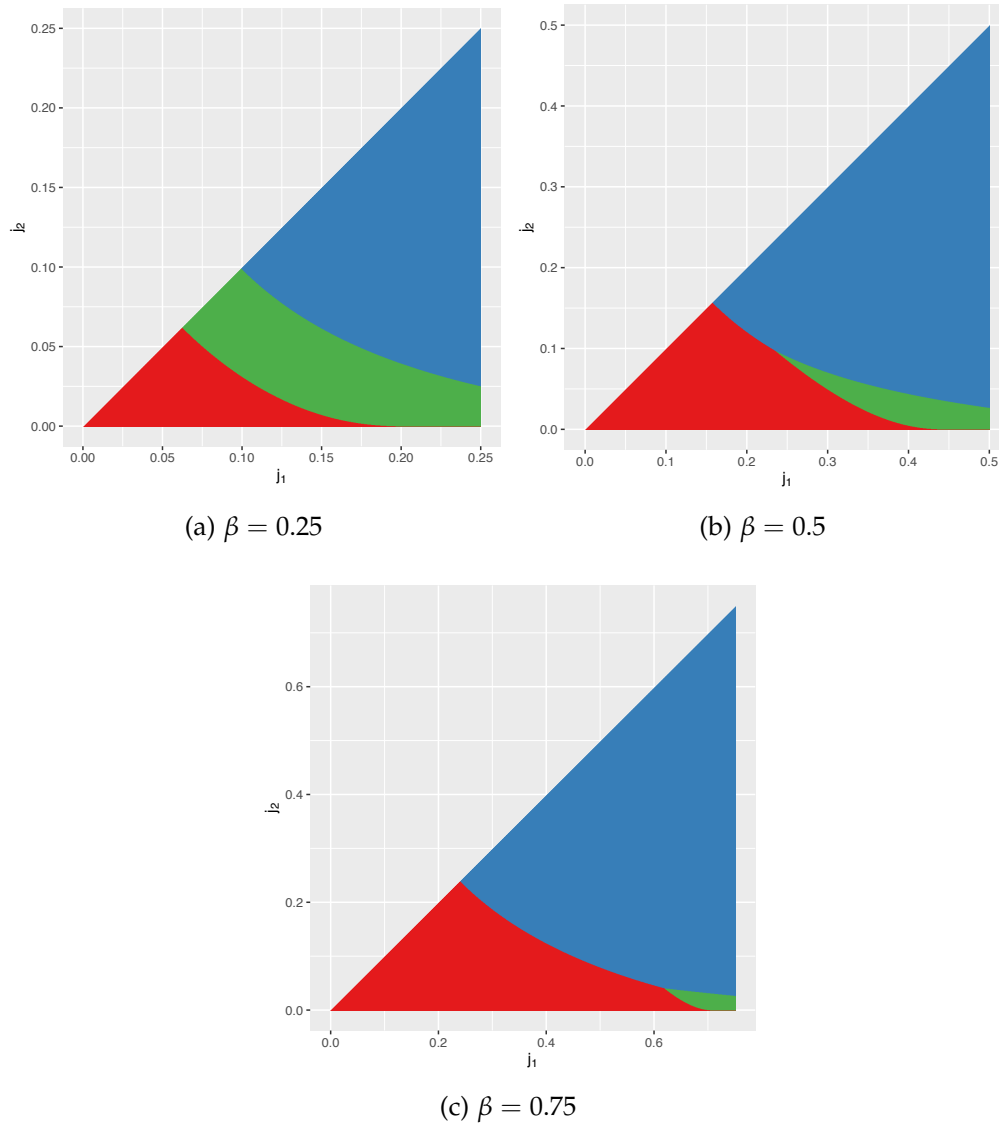
Figure 4.6: Solution with lowest ρ-value for the (j_1, j_2)-approximate Jaccard similarity search problem for different values of β. Blue is angular LSH. Green is MinHash. Red is Chosen Path. Note the difference in the axes for different values of β as it must hold that $0 \leq j_2 \leq j_1 \leq \beta$.

Acknowledgment

We thank Thomas Dybdahl Ahle for comments on a previous version of this manuscript.

4.7   Appendix: Details behind the lower bound

4.7.1   Tools

For clarity we state some standard technical lemmas that we use to derive LSH lower bounds.

Lemma 4.7 (Hoeffding [88, Theorem 1]). *Let $X_1, X_2, \ldots, X_n$ be independent random variables satisfying $0 \leq X_i \leq 1$ for $i \in [n]$. Define $X = X_1 + X_2 + \cdots + X_n$, $Z = X/n$, and $\mu = \mathbb{E}[Z]$, then:*

- *For $\hat{\mu} \geq \mu$ and $0 < \varepsilon < 1 - \hat{\mu}$ we have that $\Pr[Z - \hat{\mu} \geq \varepsilon] \leq e^{-2n\varepsilon^2}$.*

- *For $\hat{\mu} \leq \mu$ and $0 < \varepsilon < \hat{\mu}$ we have that $\Pr[Z - \hat{\mu} \leq -\varepsilon] \leq e^{-2n\varepsilon^2}$.*

Lemma 4.8 (Chernoff [113, Thm. 4.4 and 4.5]). *Let $X_1, \ldots, X_n$ be independent Poisson trials and define $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$. Then, for $0 < \varepsilon < 1$ we have*

- $\Pr[X \geq (1 + \varepsilon)\mu] \leq e^{-\varepsilon^2 \mu/3}$.

- $\Pr[X \leq (1 - \varepsilon)\mu] \leq e^{-\varepsilon^2 \mu/2}$.

Lemma 4.9 (Bounding the logarithm [161]). *For $x > -1$ we have that $\frac{x}{1+x} \leq \ln(1 + x) \leq x$.*

Lemma 4.10 (Approximating the exponential function [115, Prop. B.3]). *For all $t, n \in \mathbb{R}$ with $|t| \leq n$ we have that $e^t(1 - \frac{t^2}{n}) \leq (1 + \frac{t}{n})^n \leq e^t$.*

4.7.2   Proof of Lemma 4.5

Preliminaries.   We will reuse the notation of Section 3. from O'Donnell et al. [122].

Definition 4.4. For $0 \leq \lambda < 1$ we say that (x, y) are $(1 - \lambda)$-correlated if x is chosen uniformly at random from $\{0, 1\}^d$ and y is constructed by rerandomizing each bit from x independently at random with probability λ.

Let (x, y) be e^{-t}-correlated and let $\mathcal{H}$ be a family of hash functions on $\{0, 1\}^d$, then we define

$$\mathbb{K}_{\mathcal{H}}(t) = \Pr_{\substack{h \sim \mathcal{H} \\ (x,y)\, e^{-t}\text{- corr'd}}} [h(x) = h(y)].$$

We have that $\mathbb{K}_{\mathcal{H}}(t)$ is a log-convex function which implies the following property that underlies the lower bound:

Lemma 4.11. *For every family of hash functions $\mathcal{H}$ on $\{0, 1\}^d$, every $t \geq 0$, and $c \geq 1$ we have*

$$\frac{\ln(1/\mathbb{K}_{\mathcal{H}}(t))}{\ln(1/\mathbb{K}_{\mathcal{H}}(ct))} \geq \frac{1}{c}. \tag{4.6}$$

The idea behind the proof is to tie p_1 to $\mathbb{K}_{\mathcal{H}}(t)$ and p_2 to $\mathbb{K}_{\mathcal{H}}(ct)$ through Chernoff bounds and then apply Lemma 4.11 to show that $\rho \gtrsim 1/c$.

Proof. Begin by assuming that we have a family $\mathcal{H}$ that satisfies the conditions of Lemma 4.5. Note that the expected Hamming distance betwee $(1 - \lambda)$-correlated points x and y is given by $(\lambda/2)d$. We set $\lambda_{p_1}/2 = d^{-1/2} - d^{-5/8}$ and $\lambda_{p_2}/2 = cd^{-1/2} + 2cd^{-5/8}$ and let (x, y) denote $(1 - \lambda_{p_1})$-correlated random strings and (x, x') denote $(1 - \lambda_{p_2}q)$-correlated random strings. By standard Chernoff bounds we get the following guarantees:

$$\Pr[\|x - y\|_1 \geq r] \leq e^{-\Omega(d^{1/4})},$$
$$\Pr[\|x - x'\|_1 \leq cr] \leq e^{-\Omega(d^{1/4})}.$$

We will establish a relationship between $\mathbb{K}_{\mathcal{H}}(t_{p_1})$ and p_1 on the one hand, and $\mathbb{K}_{\mathcal{H}}(t_{p_2})$ and p_2 on the other hand, for the following choice of parameters t_{p_1} and t_{p_2}:

$$t_{p_1} = -\ln(1 - 2(d^{-1/2} - d^{-5/8}))$$
$$t_{p_2} = -\ln(1 - 2c(d^{-1/2} + 2d^{-5/8})).$$

By the properties of $\mathcal{H}$ and from the definition of $\mathbb{K}_{\mathcal{H}}$ we have that

$$\mathbb{K}_{\mathcal{H}}(t_{p_1}) \geq p_1(1 - \Pr[\|x - y\|_1 > r]) \geq p_1 - \Pr[\|x - y\|_1 \geq r]$$
$$\mathbb{K}_{\mathcal{H}}(t_{p_2}) \leq p_2(1 - \Pr[\|x - x'\|_1 \leq cr]) + \Pr[\|x - x'\|_1 \leq cr]$$
$$\leq p_2 + \Pr[\|x - x'\|_1 \leq cr].$$

Let $\delta = \max\{\Pr[\|x - y\|_1 \geq r], \Pr[\|x - x'\|_1 \leq cr]\} = e^{-\Omega(d^{1/4})}$. By Lemma 4.11 and our setting of t_{p_1} and t_{p_2} we can use the bounds on the natural logarithm from Lemma 4.9 to show the following:

$$\frac{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_1}))}{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2}))} \geq \frac{t_{p_1}}{t_{p_2}} = \frac{\ln(1 - 2(d^{-1/2} - d^{-5/8}))}{\ln(1 - 2c(d^{-1/2} + 2d^{-5/8}))}$$

$$\geq \frac{2(d^{-1/2} - d^{-5/8})}{2c(d^{-1/2} + 2d^{-5/8})} - 2(d^{-1/2} - d^{-5/8})$$

$$\geq \frac{1 - d^{-1/4}}{c + 2d^{-1/4}} - 2(d^{-1/2} - d^{-5/8})$$

$$= \frac{1}{c} - O(d^{-1/4}).$$

We proceed by lower bounding ρ where we make use of the inequalities derived above.

$$\mathbb{K}_{\mathcal{H}}(t_{p_2}) - \delta \leq p_2 < p_1 \leq \mathbb{K}_{\mathcal{H}}(t_{p_1}) + \delta.$$

By Lemma 4.11 combined with the restrictions on our parameters, for d greater than some constant we have that $\mathbb{K}_{\mathcal{H}}(t_{p_2}) \geq \mathbb{K}_{\mathcal{H}}(t_{p_1})^{2c} \geq (p_1/2)^{2c} \geq (2d)^{-2c} \geq (2d)^{-2d^{1/8}}$. Furthermore, we lower bound $\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2}))$ by using that $\mathbb{K}_{\mathcal{H}}(t_{p_2}) \leq p_2 + \delta$ together with the restriction that $p_2 \geq 1 - 1/d$ and the properties of δ. For d greater than some constant it therefore holds that $\mathbb{K}_{\mathcal{H}}(t_{p_2}) \leq 1 - 1/2d$ from which it follows that $\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2})) \geq 1/2d$.

$$\frac{\ln(1/p_1)}{\ln(1/p_2)} \geq \frac{\ln(1/(\mathbb{K}_{\mathcal{H}}(t_{p_1}) + \delta))}{\ln(1/(\mathbb{K}_{\mathcal{H}}(t_{p_2}) - \delta))}$$

$$= \frac{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_1})) - \ln(1 + \delta/\mathbb{K}_{\mathcal{H}}(t_{p_1}))}{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2})) + \ln(1/(1 - \delta/\mathbb{K}_{\mathcal{H}}(t_{p_2})))}$$

$$\geq \frac{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_1})) - \delta/\mathbb{K}_{\mathcal{H}}(t_{p_1})}{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2})) + 2\delta/\mathbb{K}_{\mathcal{H}}(t_{p_2})}$$

$$\geq \frac{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_1}))}{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2}))} - \frac{3\delta}{\mathbb{K}_{\mathcal{H}}(t_{p_2}) \ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2}))}.$$

By the arguments above we have that

$$\frac{3\delta}{\mathbb{K}_{\mathcal{H}}(t_{p_2}) \ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2}))} = e^{-\Omega(d^{1/4})} = O(d^{-1/4}).$$

Inserting the lower bound for $\frac{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_1}))}{\ln(1/\mathbb{K}_{\mathcal{H}}(t_{p_2}))}$ results in the lemma.

4.8 Appendix: Comparisons

For completeness we state the proofs behind the comparisons between the ρ-values obtained by the CHOSEN PATH algorithm and other LSH techniques.

4.8.1 MinHash

For data sets with fixed sparsity and Braun-Blanquet similarities $0 < b_2 < b_1 < 1$ we have that $\rho / \rho_{\text{minhash}} = f(b_2)/f(b_1)$ where $f(x) = \log(x/(2-x))/\log(x)$. If $f(x)$ is monotone increasing in $(0,1)$ then $\rho / \rho_{\text{minhash}} < 1$. For $x \in (0,1)$ we have that $\text{sign}(f'(x)) = \text{sign}(g(x))$ where $g(x) = \ln(x) + (2-x)\ln(2-x)$. The function $g(x)$ equals zero at $x = 1$ and has the derivative $g'(x) = \ln(x) - \ln(2-x)$ which is negative for values of $x \in (0,1)$. We can thefore see that $f'(x)$ is positive in the interval and it follows that $\rho < \rho_{\text{minhash}}$ for every choice of $0 < b_2 < b_1 < 1$.

4.8.2 Angular LSH

We have that $\rho / \rho_{\text{angular}} < 1$ if $f(x) = \ln(x)\frac{1+x}{1-x}$ is a monotone increasing function for $x \in (0,1)$. For $x \in (0,1)$ we have that $\text{sign}(f'(x)) = \text{sign}(g(x))$ where $g(x) = (1-x^2)/2 + x\ln x$. We note that $g(1) = 0$ and $g'(x) = 1 - x + \ln x$. Therefore, if $g'(x) < 0$ for $x \in (0,1)$ it holds that $g(x) > 0$ and $f(x)$ is monotone increasing in the same interval. We have that $g'(1) = 0$ and $g''(x) = -1 + 1/x > 0$ implying that $g'(x) < 0$ in the interval.

4.8.3 Data-dependent LSH

Lemma 4.12. *Let $0 < b_2 < b_1 < 1$ and fix $\rho = 1/2$ such that $b_1 = \sqrt{b_2}$. Then we have that $\rho < \rho_{datadep}$ for every value of $b_2 < 1/4$.*

Proof. We will compare $\rho = \log(b_1)/\log(b_2)$ and $\rho_{\text{datadep}} = \frac{1-b_1}{1+b_1-2b_2}$ when ρ is fixed at $\rho = 1/2$, or equivalently, $b_1 = \sqrt{b_2}$. We can solve the quadratic equation $1/2 = \frac{1-\sqrt{b_2}}{1+\sqrt{b_2}-2b_2}$ to see that for $0 < b_2 < 1$ we have that $\rho = \rho_{\text{datadep}}$ only when $b_2 = 1/4$. The derivative of ρ_{datadep} with respect to b_2 is negative when $b_1 = \sqrt{b_2}$. Under this restriction we therefore have that $\rho < \rho_{\text{datadep}}$ for $b_2 < 1/4$ which is equivalent to $j_2 < 1/7$ in the fixed-weight setting. $\qquad\square$

To compare ρ-values over the full parameter space we use the following two lemmas.

Lemma 4.13. *For every choice of fixed* $0 < \rho < 1$ *let* $b_2 = b_1^{1/\rho}$. *Then* $\rho_{datadep} = \frac{1-b_1}{1+b_1-2b_2}$ *is decreasing in* b_1 *for* $b_1 \in (0,1)$.

Proof. The sign of the derivative of ρ_{datadep} with respect to b_1 is equal to the sign of the function $g(x) = -\rho x^{-1/\rho} + \rho - 1 + x^{-1}$ for $x \in (0,1)$. We have that $g(1) = 0$ and $g'(x) = x - 1/p - 1 - x^{-2} > 0$ for $x \in (0,1)$ which shows that $g(x) < 0$ in the interval. $\qquad\square$

Lemma 4.14. *For* $1/5 = b_2 < b_1 < 1$ *we have that* $\rho < \rho_{datadep}$.

Proof. For fixed $b_2 = 1/5$ consider $f(b_1) = \rho - \rho_{\text{datadep}}$ as a function of b_1 in the interval $[1/5, 1]$. We want to show that $f(b_1) < 0$ for $b_1 \in (1/5, 1)$. In the endpoints the function takes the value 0. Between the endpoints we find that $f'(b_1) = \frac{1}{\ln(5)b_1} + \frac{8/5}{(3/5+b_1)^2}$ and that $f'(b_1) = 0$ is a quadratic form with only one solution b_1^* in $[1/5, 1]$. By Lemma 4.12 we know that that for $b_2 = 1/5$ and $b_1 = 1/\sqrt{5}$ it holds that $f(b_1) < 0$. Since $f(1/5) = f(1) = 0$, $f'(b_1) = 0$ only in a single point in $[1/5, 1]$, and $f(1/\sqrt{5}) < 0$ we can conclude that the lemma holds. $\qquad\square$

Corollary 4.2. *For every choice of* b_1, b_2 *satisfying* $0 < b_2 \leq 1/5$ *and* $b_2 < b_1 < 1$ *we have that* $\rho < \rho_{datadep}$.

Proof. If $b_2 = 1/5$ the property holds by Lemma 4.14. If $b_2 < 1/5$ we define new variables $\hat{b}_2, \hat{b}_2$, setting $\hat{b}_1 = \hat{b}_1^{\rho(b_1,b_2)}$ and initially consider $\hat{b}_2 = 1/5$. In this setting we again have that $\rho(\hat{b}_1, \hat{b}_2) < \rho_{\text{datadep}}(\hat{b}_1, \hat{b}_2)$. According to Lemma 4.13 it holds that ρ_{datadep} is decreasing in b_2 for fixed ρ. Therefore, as $\hat{b}_2$ decreases to $\hat{b}_2 = b_2$ where $\hat{b}_1 = b_1$ we have that $\rho(\hat{b}_1, \hat{b}_2) = \rho$ remains constant while ρ_{datadep} increases. Since it held that $\rho < \rho_{\text{datadep}}$ at the initial values of $\hat{b}_1, \hat{b}_2$ it must also hold for b_1, b_2. $\qquad\square$

Numerical comparison of MinHash and Data-dep. LSH. Comparing ρ_{minhash} to ρ_{datadep} we can verify numerically that even for b_2 fixed as low as $b_2 = 1/23$ we can find values of b_1 (for example $b_1 = 0.995$ such that $\rho_{\text{minhash}} > \rho_{\text{datadep}}$.

Chapter 5

Adaptive similarity join

'A light from the shadows shall spring'

Set similarity join is a fundamental and well-studied database operator. It is usually studied in the *exact* setting where the goal is to compute all pairs of sets that exceed a given similarity threshold (measured e.g. as Jaccard similarity). But set similarity join is often used in settings where 100% recall may not be important — indeed, where the exact set similarity join is itself only an approximation of the desired result set.

We present a new randomized algorithm for set similarity join that can achieve any desired recall up to 100%, and show theoretically and empirically that it significantly improves on existing methods. The present state-of-the-art exact methods are based on prefix-filtering, the performance of which depends on the prevalence of rare elements in the sets. Our method is robust against the absence of such structure in the data. At 90% recall our algorithm is often more than an order of magnitude faster than state-of-the-art exact methods, depending on how well a data set lends itself to prefix filtering. Our experiments on benchmark data sets also show that the method is several times faster than comparable approximate methods. Our algorithm makes use of recent theoretical advances in high-dimensional sketching and indexing.

5.1 Introduction

It is increasingly important for data processing and analysis systems to be able to work with data that is imprecise, incomplete, or noisy. *Similarity join* has emerged as a fundamental primitive in data cleaning and entity resolution over the last decade [21, 48, 142]. In this paper we focus on *set*

similarity join: Given collections R and S of sets the task is to compute

$$R \bowtie_\lambda S = \{(x,y) \in R \times S \mid \text{sim}(x,y) \geq \lambda\}$$

where $\text{sim}(\cdot,\cdot)$ is a similarity measure and λ is a threshold parameter. We deal with sets $x, y \subseteq \{1, \ldots, d\}$, where the number d of distinct tokens can be naturally thought of as the dimensionality of the data.

Many measures of set similarity exist [51], but perhaps the most well-known such measure is the *Jaccard similarity*,

$$J(x,y) = |x \cap y| / |x \cup y| \ .$$

For example, the sets $x = \{\texttt{IT, University, Copenhagen}\}$ and $y = \{\texttt{University, Copenhagen, Denmark}\}$ have Jaccard similarity $J(x,y) = 1/2$ which could suggest that they both correspond to the same entity. In the context of entity resolution we want to find a set T that contains $(x,y) \in R \times S$ if and only if x and y correspond to the same entity. The quality of the result can be measured in terms of *precision* $|(R \bowtie_\lambda S) \cap T|/|T|$ and *recall* $|(R \bowtie_\lambda S) \cap T|/|R \bowtie_\lambda S|$, both of which should be as high as possible. We will be interested in methods that achieve 100% precision, but that might not have 100% recall. We refer to methods with 100% recall as exact, and others as approximate.

5.1.1   Our Contributions

We present a new approximate set similarity join algorithm: Chosen Path Similarity Join (CPSJoin). We cover its theoretical underpinnings, and show experimentally that it achieves high recall with a substantial speedup compared to state-of-the-art exact techniques. The key ideas behind CPSJoin are:

- A new recursive filtering technique inspired by the recently proposed ChosenPath index for set similarity search [56], adding new ideas to make the method parameter-free, near-linear space, and adaptive to a given data set.

- Apply efficient sketches for estimating set similarity [103] that take advantage of modern hardware.

We compare CPSJoin to the exact set similarity join algorithms in the comprehensive empirical evaluation of Mann et al. [107], using the

same data sets, and to other approximate set similarity join methods suggested in the literature. We find that CPSJoin outperforms other approximate methods and scales better than exact methods when the sets are relatively large (100 tokens or more) and the similarity threshold is low (e.g. Jaccard similarity 0.5) where we see speedups of more than an order of magnitude at 90% recall. Our experiments on benchmark datasets show that exact methods are faster in the case of high similarity thresholds, when the average set size is small, and when sets have many rare elements, whereas approximate methods are faster in the case of low similarity thresholds and when sets are large. This finding is consistent with theory and is further corroborated by experiments on synthetic datasets.

5.1.2   Related Work

For space reasons we present just a sample of the most related previous work, and refer to the book of Augsten and Böhlen [21] for a survey of algorithms for exact similarity join in relational databases, covering set similarity joins as well as joins based on string similarity.

Exact Similarity Join.   Early work on similarity join focused on the important special case of detecting *near-duplicates* with similarity close to 1, see e.g. [31, 142]. A sequence of results starting with the seminal paper of Bayardo et al. [23] studied the range of thresholds that could be handled. Recently, Mann et al. [107] conducted a comprehensive study of 7 state-of-the-art algorithms for exact set similarity join for Jaccard similarity threshold $\lambda \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. These algorithms all use the idea of *prefix filtering* [48], which generates a sequence of candidate pairs of sets that includes all pairs with similarity above the threshold. The methods differ in how much additional filtering is carried out. For example, [172] applies additional *length* and *suffix* filters to prune the candidate pairs.

Prefix filtering uses an inverted index that for each element stores a list of the sets in the collection containing that element. Given a set x, assume that we wish to find all sets y such that $|x \cup y| > t$. A valid result set y must be contained in at least one of the inverted lists associated with any subset of $|x| - t$ elements of x, or we would have $|x \cup y| \leq t$. In particular, to speed up the search, prefix filtering looks at the elements of x that have the shortest inverted lists.

The main finding by Mann et al. is that while more advanced filtering techniques do yield speedups on some data sets, an optimized version of the basic prefix filtering method (referred to as "ALL") is always competitive within a factor 2.16, and most often the fastest of the algorithms. For this reason we will be comparing our results against ALL.

Locality-sensitive hashing. Locality-sensitive hashing (LSH) is a theoretically well-founded randomized method for generating candidate pairs [78]. A family of locality-sensitive hash functions is a distribution over functions with the property that the probability that similar points (or sets in our case) are more likely to have the same function value. We know only of a few papers using LSH techniques to solve similarity join. Cohen et al. [61] used LSH techniques for set similarity join in a knowledge discovery context before the advent of prefix filtering. They sketch a way of choosing parameters suitable for a given data set, but we are not aware of existing implementations of this approach. Chakrabarti et al. [44] improved plain LSH with an adaptive similarity estimation technique, *BayesLSH*, that reduces the cost of checking candidate pairs and typically improves upon an implementation of the basic prefix filtering method by 2–20×. Our experiments include a comparison against both methods [61, 44]. We refer to the survey paper [126] for an overview of newer theoretical developments on LSH-based similarity joins, but point out that these developments have not matured sufficiently to yield practical improvements.

Distance estimation. Similar to BayesLSH [44] we make use of algorithms for similarity *estimation*, but in contrast to BayesLSH we use algorithms that make use of bit-level parallelism. This approach works when there exists a way of picking a random hash function h such that

$$\Pr[h(x) = h(y)] = \text{sim}(x, y) \tag{5.1}$$

for every choice of sets x and y. Broder et al. [35] presented such a hash function for Jaccard similarity, now known as "minhash" or "minwise hashing". In the context of distance estimation, 1-bit minwise hashing of Li and König [103] maps minhash values to a compact sketch, often using just 1 or 2 machine words. Still, this is sufficient information to be able to estimate the Jaccard similarity of two sets x and y just based on the Hamming distance of their sketches.

Locality-sensitive mappings. Several recent theoretical advances in high-dimensional indexing [16, 54, 56] have used an approach that can be seen as a generalization of LSH. We refer to this approach as locality-sensitive *mappings* (also known as locality-sensitive *filters* in certain settings). The idea is to construct a function F, mapping a set x into a set of machine words, such that:

- If $\text{sim}(x, y) \geq \lambda$ then $F(x) \cap F(y)$ is nonempty with some fixed probability $\varphi > 0$.

- If $\text{sim}(x, y) < \lambda$, then the expected intersection size $\mathbb{E}[|F(x) \cap F(y)|]$ is "small".

Here the exact meaning of "small" depends on the difference $\lambda - \text{sim}(x, y)$, but in a nutshell, if it is the case that almost all pairs have similarity significantly below λ then we can expect $|F(x) \cap F(y)| = 0$ for almost all pairs. Performing the similarity join amounts to identifying all candidate pairs x, y for which $F(x) \cap F(y) \neq \varnothing$ (for example by creating an inverted index), and computing the similarity of each candidate pair. To our knowledge these indexing methods have not been tried out in practice, probably because they are rather complicated. An exception is the recent paper [56], which is relatively simple, and indeed our join algorithm is inspired by the index described in that paper.

5.2 Preliminaries

The CPSJoin algorithm solves the (λ, φ)-similarity join problem with a probabilistic guarantee on recall, formalized as follows:

Definition 5.1. An algorithm solves the (λ, φ)-similarity join problem with threshold $\lambda \in (0, 1)$ and recall probability $\varphi \in (0, 1)$ if for every $(x, y) \in S \bowtie_\lambda R$ the output $L \subseteq S \bowtie_\lambda R$ of the algorithm satisfies $\Pr[(x, y) \in L] \geq \varphi$.

It is important to note that the probability is over the random choices made by the algorithm, and *not* over a random choice of (x, y). This means that for any $(x, y) \in S \bowtie_\lambda R$ the probability that the pair is *not* reported in r independent repetitions of the algorithm is bounded by $(1 - \varphi)^r$. For example if $\varphi = 0.9$ it takes just $r = 3$ repetitions to bound the recall to at least 99.9%.

5.2.1   Similarity Measures

Our algorithm can be used with a broad range of similarity measures through randomized *embeddings*. This allows it to be used with, for example, Jaccard and cosine similarity thresholds.

Embeddings map data from one space to another while approximately preserving distances, with accuracy that can be tuned. In our case we are interested in embeddings that map data to sets of tokens. We can transform any so-called *LSHable* similarity measure sim, where we can choose h to make (5.1) hold, into a set similarity measure by the following randomized embedding: For a parameter t pick hash functions $h_1, \ldots, h_t$ independently from a family satisfying (5.1). The embedding of x is the following set of size t:

$$f(x) = \{(i, h_i(x)) \mid i = 1, \ldots, t\} \ .$$

It follows from (5.1) that the expected size of the intersection $f(x) \cap f(y)$ is $t \cdot \text{sim}(x, y)$. Furthermore, it follows from standard concentration inequalities that the size of the intersection will be close to the expectation with high probability. For our experiments with Jaccard similarity thresholds ≥ 0.5, we found that $t = 64$ gave sufficient precision for $> 90\%$ recall.

In summary we can perform the similarity join $R \bowtie_\lambda S$ for any LSHable similarity measure by creating two corresponding relations $R' = \{f(x) \mid x \in R\}$ and $S' = \{f(y) \mid y \in S\}$, and computing $R' \bowtie_\lambda S'$ with respect to the similarity measure

$$B(f(x), f(y)) = |f(x) \cap f(y)|/t \ . \tag{5.2}$$

This measure is the special case of *Braun-Blanquet* similarity where the sets are known to have size t [51]. Our implementation will take advantage of the set size t being fixed, though it is easy to extend to general Braun-Blanquet similarity.

The class of LSHable similarity measures is large, as discussed in [49]. If approximation errors are tolerable, even *edit distance* can be handled by our algorithm [45, 173].

5.2.2   Notation

We are interested in sets S where an element, $x \in S$ is a set with elements from some universe $[d] = \{1, 2, 3, \cdots, d\}$. To avoid confusion we

sometimes use "record" for $x \in S$ and "token" for the elements of x. Throughout this paper we will think of a record x both as a set of tokens from $[d]$, as well as a vector from $\{0, 1\}^d$, where:

$$x_i = \begin{cases} 1 \text{ if } i \in x \\ 0 \text{ if } i \notin x \end{cases}$$

It is clear that these representations are equivalent. The set $\{1, 4, 5\}$ is equivalent to $(1, 0, 0, 1, 1, 0, \cdots, 0)$, $\{1, d\}$ is equivalent to $(1, 0, \cdots, 0, 1)$, etc.

5.3   Overview of approach

Our high-level approach is recursive and works as follows. To compute $R \bowtie_\lambda S$ we consider each $x \in R$ and either:

1. Compare x to each record in S (referred to as "brute forcing" x), or

2. create several subproblems $S_i \bowtie_\lambda R_i$ with $x \in R_i \subseteq R$, $S_i \subseteq S$, and solve them recursively.

The approach of [56] corresponds to choosing option 2 until reaching a certain level k of the recursion, where we finish the recursion by choosing option 1. This makes sense for certain worst-case data sets, but we propose an improved parameter-free method that is better at adapting to the given data distribution. In our method the decision on which option to choose depends on the size of S and the average similarity of x to the records of S. We choose option 1 if S has size below some (constant) threshold, or if the average Braun-Blanquet similarity of x and S, $\frac{1}{|S|} \sum_{y \in S} B(x, y)$, is close to the threshold λ. In the former case it is cheap to finish the recursion. In the latter case many records $y \in S$ will have $B(x, y)$ larger than or close to λ, so we do not expect to be able to produce output pairs with x in sublinear time in $|S|$.

   If neither of these pruning conditions apply we choose option 2 and include x in recursive subproblems as described below. But first we note that the decision of which option to use can be made efficiently for each x, since the average similarity of pairs from $R \times S$ can be computed from token frequencies in time $O(t|R| + t|S|)$. Pseudocode for a self-join version of CPSJoin is provided in Algorithm 1 and 2.

5.3.1 Recursion

We would like to ensure that for each pair $(x, y) \in R \bowtie_\lambda S$ the pair is computed in one of the recursive subproblems, i.e., that $(x, y) \in R_i \bowtie_\lambda S_i$ for some i. In particular, we want the expected number of subproblems containing (x, y) to be at least 1, i.e.,

$$\mathbb{E}\left[|\{i \mid (x, y) \in R_i \bowtie_\lambda S_i\}|\right] \geq 1. \tag{5.3}$$

To achieve (5.3) for every pair $(x, y) \in R \bowtie_\lambda S$ we proceed as follows: for each $i \in \{1, \ldots, d\}$ we recurse with probability $1/(\lambda t)$ on the subproblem $R_i \bowtie_\lambda S_i$ with sets

$$R_i = \{x \in R \mid i \in x\}$$
$$S_i = \{y \in S \mid i \in y\}$$

where t denotes the size of records in R and S. It is not hard to check that (5.3) is satisfied for every pair (x, y) with $B(x, y) \geq \lambda$. Of course, expecting one subproblem to contain (x, y) does not *directly* imply a good probability that (x, y) is contained in at least one subproblem. But it turns out that we can use results from the theory of branching processes to show such a bound; details are provided in section 5.4.

5.4 Chosen Path Similarity Join

The CPSJOIN algorithm solves the (λ, φ)-set similarity join (Definition 5.1) for every choice of $\lambda \in (0, 1)$ and with a guarantee on φ that we will lower bound in the analysis.

   To simplify the exposition we focus on a self-join version where we are given a set S of n subsets of $[d]$ and we wish to report $L \subseteq S \bowtie_\lambda S$. Handling a general join $S \bowtie_\lambda R$ follows the overview in section 5.3 and requires no new ideas: Essentially consider a self-join on $S \cup R$ but make sure to consider only pairs in $S \times R$ for output. We also make the simplifying assumption that all sets in S have a fixed size t. As argued in section 5.2.1 the general case can be reduced to this one by embedding.

5.4.1 Description

The CPSJoin algorithm (see Algorithm 1 for pseudocode) works by recursively splitting the data set on elements of $[d]$ that are selected according

to a random process, forming a recursion tree with S at the root and subsets of S that are non-increasing in size as we get further down the tree. The randomized splitting has the property that the probability of a pair of sets (x, y) being in a random subproblem is increasing as a function of $|x \cap y|$.

Before each recursive splitting step we run the BRUTEFORCE subprocedure (see Algorithm 2 for pseudocode) that identifies subproblems that are best solved by brute force. It has two parts:

1. If S is below some constant size, controlled by the parameter `limit`, we report $S \bowtie_\lambda S$ exactly using a simple loop with $O(|S|^2)$ distance computations (BRUTEFORCEPAIRS) and exit the recursion. In our experiments we have set `limit` to 250, with the precise choice seemingly not having a large effect as shown experimentally in Section 5.6.2.

2. If S is larger than `limit` the second part activates: for every $x \in S$ we check whether the expected number of distance computations involving x is going to decrease by continuing the recursion. If this is not the case, we immediately compare x against every point in S (BRUTEFORCEPOINT), reporting close pairs, and proceed by removing x from S. The BRUTEFORCE procedure is then run again on the reduced set.

This procedure where we choose to handle some points by brute force crucially separates our algorithm from many other approximate similarity join methods in the literature that typically are LSH-based [127, 61]. By efficiently being able to remove points at the "right" time, before they generate too many expensive comparisons further down the tree, we are able to beat the performance of other approximate similarity join techniques in both theory and practice. Another benefit of this approach is that it reduces the number of parameters compared to the usual LSH setting where the depth of the tree has to be selected by the user.

Algorithm 1: CPSJOIN(S, λ)

---

1   *For $j \in [d]$ initialize $S_j \leftarrow \varnothing$.*
2   $S \leftarrow$ BRUTEFORCE(S, λ)
3   $r \leftarrow$ SEEDHASHFUNCTION$()$
4   **for** $x \in S$ **do**
5   **for** $j \in x$ **do**
6   **if** $r(j) < \frac{1}{\lambda |x|}$ **then** $S_j \leftarrow S_j \cup \{x\}$

7   **for** $S_j \neq \varnothing$ **do** CPSJOIN(S_j, λ)

Algorithm 2: BRUTEFORCE(S, λ)

    Global parameters: limit ≥ 1, $\varepsilon \geq 0$.

1 *Initialize empty map* count[ ] *with default value* 0.

2 **if** $|S| \leq$ limit **then**

3 | BRUTEFORCEPAIRS(S, λ)

4 | **return** $\varnothing$

5 **for** $x \in S$ **do**

6 | **for** $j \in x$ **do**

7 | | count[j] $\leftarrow$ count[j] + 1

8 **for** $x \in S$ **do**

9 | **if** $\frac{1}{|S|-1} \sum_{j \in x} (\text{count}[j]-1)/t > (1-\varepsilon)\lambda$ **then**

10 | | BRUTEFORCEPOINT(S, x, λ)

11 | | **return** BRUTEFORCE($S \setminus \{x\}, \lambda$)

12 **return** S

5.4.2 Comparison to Chosen Path

The CPSJOIN algorithm is inspired by the CHOSEN PATH algorithm [56] for the approximate near neighbor problem and uses the same underlying random splitting tree that we will refer to as the Chosen Path Tree. In the approximate near neighbor problem, the task is to construct a data structure that takes a query point and correctly reports an approximate near neighbor, if such a point exists in the data set. Using the CHOSEN PATH data structure directly to solve the (λ, φ)-set similarity join problem has several drawbacks that we avoid in the CPSJOIN algorithm. First, the CHOSEN PATH data structure is parameterized in a non-adaptive way to provide guarantees for worst-case data, vastly increasing the amount of work done compared to the optimal parameterization when data is not worst-case. Our recursion rule avoids this and instead continuously adapts to the distribution of distances as we traverse down the tree. Secondly, the data structure uses space $O(n^{1+\rho})$ where $\rho > 0$, storing the Chosen Path Tree of size $O(n^\rho)$ for every data point. The CPSJOIN algorithm, instead of storing the whole tree, essentially performs a depth-first traversal, using only near-linear space in n in addition to the space required to store the output. Finally, the CHOSEN PATH data structure only has to report a single point that is approximately similar to a query point, and can report points with similarity $< \lambda$. To solve the approximate

similarity join problem the CPSJoin algorithm has to satisfy reporting guarantees for *every* pair of points (x, y) in the exact join.

5.4.3 Analysis

The Chosen Path Tree for a set $x \subseteq [d]$ is defined by a random process: at each node, starting from the root, we sample a random hash function $r\colon [d] \to [0, 1]$ and construct children for every element $j \in x$ such that $r(j) < \frac{1}{\lambda|x|}$. Nodes at depth k in the tree are identified by their path $p = (j_1, \ldots, j_k)$. Formally, the set of nodes at depth $k > 0$ in the Chosen Path Tree for x is given by

$$F_k(x) = \left\{ p \circ j \mid p \in F_{k-1}(x) \wedge r_p(j) < \frac{x_j}{\lambda|x|} \right\}$$

where $p \circ j$ denotes vector concatenation and $F_0(x) = \varnothing$. The subset of the data set S that survives to a node with path $p = (j_1, \ldots, j_k)$ is given by

$$S_p = \{ x \in S \mid x_{j_1} = 1 \wedge \cdots \wedge x_{j_k} = 1 \}.$$

The random process underlying the Chosen Path Tree belongs to the well studied class of Galton-Watson branching processes [87]. Originally these where devised to answer questions about the growth and decline of family names in a model of population growth assuming i.i.d. offspring for every member of the population across generations [167]. In order to make statements about the properties of the CPSJoin algorithm we study in turn the branching processes of the Chosen Path Tree associated with a point x, a pair of points (x, y), and a set of points S. Note that we use the same random hash functions for different points in S.

Brute forcing.   The BruteForce subprocedure described by Algorithm 2 takes two global parameters: limit ≥ 1 and $\varepsilon \geq 0$. The parameter limit controls the minimum size of S before we discard the CPSJoin algorithm for a simple exact similarity join by brute force pairwise distance computations. The second parameter, $\varepsilon > 0$, controls the sensitivity of the BruteForce step to the expected number of comparisons that a point $x \in S$ will generate if allowed to continue in the branching process. The larger ε the more aggressively we will resort to the brute force procedure. In practice we typically think of ε as a small constant, say $\varepsilon = 0.05$, but for some of our theoretical results we will need a sub-constant setting of

$\varepsilon \approx 1/\log(n)$ to show certain running time guarantees. The BRUTEFORCE step removes a point x from the Chosen Path branching process, instead opting to compare it against every other point $y \in S$, if it satisfies the condition

$$\frac{1}{|S| - 1} \sum_{y \in S \setminus \{x\}} |x \cap y| / t > (1 - \varepsilon)\lambda.$$

In the pseudocode of Algorithm 2 we let count denote a hash table that keeps track of the number of times each element $j \in [d]$ appears in S. This allows us to evaluate the condition in equation (5.4.3) for an element $x \in S$ in time $O(|x|)$ by rewriting it as

$$\frac{1}{|S| - 1} \sum_{j \in x} (\text{count}[j] - 1) / t > (1 - \varepsilon)\lambda.$$

We claim that this condition minimizes the expected number of comparisons performed by the algorithm: Consider a node in the Chosen Path Tree associated with a set of points S while running the CPSJOIN algorithm. For a point $x \in S$, we can either remove it from S immediately at a cost of $|S| - 1$ comparisons, or we can choose to let continue in the branching process (possibly into several nodes) and remove it later. The expected number of comparisons if we let it continue k levels before removing it from every node that it is contained in, is given by

$$\sum_{y \in S \setminus \{x\}} \left(\frac{1}{\lambda} \frac{|x \cap y|}{t} \right)^k.$$

This expression is convex and increasing in the similarity $|x \cap y|/t$ between x and other points $y \in S$, allowing us to state the following remark:

Remark 5.1 (Recursion)*.* Let $\varepsilon = 0$ and consider a set S containing a point $x \in S$ such that x satisfies the recursion condition in equation (5.4.3). Then the expected number of comparisons involving x if we continue branching exceeds $|S| - 1$ at every depth $k \geq 1$. If x does not satisfy the condition, the opposite is observed.

Tree depth.   We proceed by bounding the maximal depth of the set of paths in the Chosen Path Tree that are explored by the CPSJOIN algorithm. Having this information will allow us to bound the space usage of the algorithm and will also form part of the argument for the correctness

guarantee. Assume that the parameter `limit` in the BRUTEFORCE step is set to some constant value, say `limit` $= 100$. Consider a point $x \in S$ and let $S' = \{y \in S \mid |x \cap y|/t \leq (1 - \varepsilon)\lambda\}$ be the subset of points in S that are not too similar to x. For every $y \in S'$ the expected number of vertices in the Chosen Path Tree at depth k that contain both x and y is upper bounded by

$$\mathbb{E}[|F_k(x \cap y)|] = \left(\frac{1}{\lambda} \frac{|x \cap y|}{t} \right)^k \leq (1 - \varepsilon)^k \leq e^{-\varepsilon k}.$$

Since $|S'| \leq n$ we use Markov's inequality to show the following bound:

Lemma 5.1. *Let $x, y \in S$ satisfy that $|x \cap y|/t \leq (1 - \varepsilon)\lambda$ then the probability that there exists a vertex at depth k in the Chosen Path Tree that contains x and y is at most $e^{-\varepsilon k}$.*

If x does not share any paths with points that have similarity that falls below the threshold for brute forcing, then the only points that remain are ones that will cause x to be brute forced. This observation leads to the following probabilistic bound on the tree depth:

Lemma 5.2. *With high probability the maximal depth of paths explored by the CPSJOIN algorithm is $O(\log(n)/\varepsilon)$.*

Correctness. Let x and y be two sets of equal size t such that $B(x,y) = |x \cap y|/t \geq \lambda$. We are interested in lower bounding the probability that there exists a path of length k in the Chosen Path Tree that has been chosen by both x and y, i.e. $\Pr[F_k(x \cap y) \neq \varnothing]$. Agresti [3] showed an upper bound on the probability that a branching process becomes extinct after at most k steps. We use it to show the following lower bound on the probability of a close pair of points colliding at depth k in the Chosen Path Tree.

Lemma 5.3 (Agresti [3]). *If $\text{sim}(x,y) \geq \lambda$ then for every $k > 0$ we have that $\Pr[F_k(x \cap y) \neq \varnothing] \geq \frac{1}{k+1}$.*

The bound on the depth of the Chosen Path Tree for x explored by the CPSJOIN algorithm in Lemma 5.2 then implies a lower bound on φ.

Lemma 5.4. *Let $0 < \lambda < 1$ be constant. Then for every set S of $|S| = n$ points the CPSJOIN algorithm solves the set similarity join problem with $\varphi = \Omega(\varepsilon/\log(n))$.*

Remark 5.2. This analysis is very conservative: if either x or y is removed by the BRUTEFORCE step prior to reaching the maximum depth then it only increases the probability of collision. We note that similar guarantees can be obtained when using fast pseudorandom hash functions as shown in the paper introducing the CHOSEN PATH algorithm [56].

Space usage.   We can obtain a trivial bound on the space usage of the CPSJOIN algorithm by combining Lemma 5.2 with the observation that every call to CPSJOIN on the stack uses additional space at most $O(n)$. The result is stated in terms of working space: the total space usage when not accounting for the space required to store the data set itself (our algorithms use references to data points and only reads the data when performing comparisons) as well as disregarding the space used to write down the list of results.

Lemma 5.5. *With high probability the working space of the* CPSJOIN *algorithm is at most* $O(n \log(n)/\varepsilon)$.

Remark 5.3. We conjecture that the expected working space is $O(n)$ due to the size of S being geometrically decreasing in expectation as we proceed down the Chosen Path Tree.

Running time.   We will bound the running time of a solution to the general set similarity self-join problem that uses several calls to the CPSJOIN algorithm in order to piece together a list of results $L \subseteq S \bowtie_\lambda S$. In most of the previous related work, inspired by Locality-Sensitive Hashing, the fine-grainedness of the randomized partition of space, here represented by the Chosen Path Tree in the CPSJOIN algorithm, has been controlled by a single global parameter k [78, 127]. In the Chosen Path setting this rule would imply that we run the splitting step without performing any brute force comparison until reaching depth k where we proceed by comparing x against every other point in nodes containing x, reporting close pairs. In recent work by Ahle et al. [4] it was shown how to obtain additional performance improvements by setting an individual depth k_x for every $x \in S$. We refer to these stopping strategies as global and individual, respectively. Together with our recursion strategy, this gives rise to the following stopping criteria for when to compare a point x against everything else contained in a node:

- Global: Fix a single depth k for every $x \in S$.

- Individual: For every $x \in S$ fix a depth k_x.

- Adaptive: Remove x when the expected number of comparisons is non-decreasing in the tree-depth.

Let T denote the running time of our similarity join algorithm. We aim to show the following relation between the running time between the different stopping criteria when applied to the Chosen Path Tree:

$$\mathbb{E}[T_{\text{Adaptive}}] \leq \mathbb{E}[T_{\text{Individual}}] \leq \mathbb{E}[T_{\text{Global}}].$$

First consider the global strategy. We set k to balance the contribution to the running time from the expected number of vertices containing a point, given by $(1/\lambda)^k$, and the expected number of comparisons between pairs of points at depth k, resulting in the following expected running time for the global strategy:

$$O\left(\min_k n(1/\lambda)^k + \sum_{\substack{x,y \in S \\ x \neq y}} (\text{sim}(x,y)/\lambda)^k \right).$$

The global strategy is a special case of the individual case, and it must therefore hold that $\mathbb{E}[T_{\text{Individual}}] \leq \mathbb{E}[T_{\text{Global}}]$. The expected running time for the individual strategy is upper bounded by:

$$O\left(\sum_{x \in S} \min_{k_x} \left((1/\lambda)^{k_x} + \sum_{y \in S \setminus \{x\}} (\text{sim}(x,y)/\lambda)^{k_x} \right) \right).$$

We will now argue that the expected running time of the CPSJOIN algorithm under the adaptive stopping criteria is no more than a constant factor greater than $\mathbb{E}[T_{\text{Individual}}]$ when we set the global parameters of the BRUTEFORCE subroutine as follows:

$$\texttt{limit} = \Theta(1),$$
$$\varepsilon = \frac{\log(1/\lambda)}{\log n}.$$

Let $x \in S$ and consider a path p where x is removed in from S_p by the BRUTEFORCE step. Let k'_x denote the depth of the node (length of p) at which x is removed. Compared to the individual strategy that removes x at depth k_x we are in one of three cases, also displayed in Figure 5.1.

1. The point x is removed from p at depth $k'_x = k_x$.

2. The point x is removed from p at depth $k'_x < k_x$.

3. The point x is removed from p at depth $k'_x > k_x$.
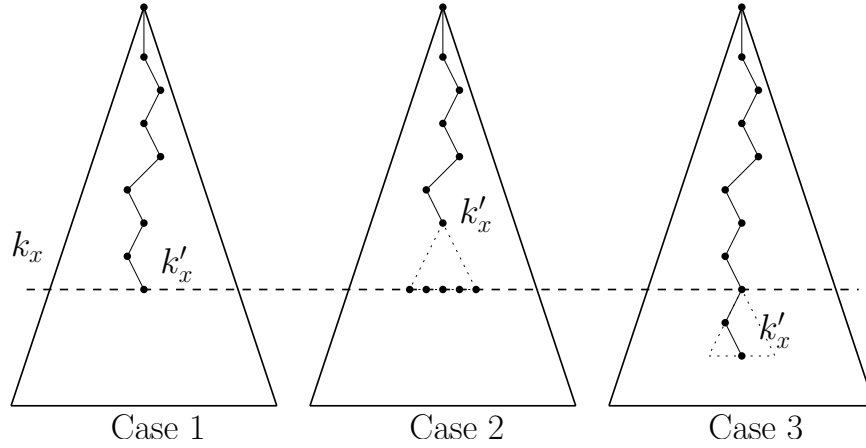
Figure 5.1: Path termination depth in the Chosen Path Tree

The underlying random process behind the Chosen Path Tree is not affected by our choice of termination strategy. In the first case we therefore have that the expected running time is upper bounded by the same (conservative) expression as the one used by the individual strategy. In the second case we remove x earlier than we would have under the individual strategy. For every $x \in S$ we have that $k_x \leq 1/\varepsilon$ since for larger values of k_x the expected number of nodes containing x exceeds n. We therefore have that $k_x - k'_x \leq 1/\varepsilon$. Let S' denote the set of points in the node where x was removed by the BRUTEFORCE subprocedure. There are two rules that could have triggered the removal of x: Either $|S'| = O(1)$ or the condition in equation (5.4.3) was satisfied. In the first case, the expected cost of following the individual strategy would have been $\Omega(1)$ simply from the $1/\lambda$ children containing x in the next step. This is no more than a constant factor smaller than the adaptive strategy. In the second case, when the condition in equation (5.4.3) is activated we have that the expected number of comparisons involving x resulting from S' if we had continued under the individual strategy is at least

$$(1 - \varepsilon)^{1/\varepsilon} |S'| = \Omega(|S'|)$$

which is no better than what we get with the adaptive strategy. In the third case where we terminate at depth $k_x' > k_x$, if we retrace the path to depth k_x we know that x was not removed in this node, implying that the expected number of comparisons when continuing the branching process on x is decreasing compared to removing x at depth k_x. We have shown that the expected running time of the adaptive strategy is no greater than a constant times the expected running time of the individual strategy.

We are now ready to state our main theoretical contribution, stated below as Theorem 5.1. The theorem combines the above argument that compares the adaptive strategy against the individual strategy together with Lemma 5.2 and Lemma 5.4, and uses $O(\log^2 n)$ runs of the CPSJOIN algorithm to solve the set similarity join problem for every choice of constant parameters λ, φ.

Theorem 5.1. *For every LSHable similarity measure and every choice of constant threshold $\lambda \in (0,1)$ and probability of recall $\varphi \in (0,1)$ we can solve the (λ, φ)-set similarity join problem on every set S of n points using working space $\tilde{O}(n)$ and with expected running time*

$$
\tilde{O}\left(\sum_{x \in S} \min_{k_x}\left(\sum_{y \in S \setminus \{x\}} (\text{sim}(x,y)/\lambda)^{k_x} + (1/\lambda)^{k_x}\right)\right).
$$

5.5 Implementation

We implement an optimized version of the CPSJOIN algorithm for solving the Jaccard similarity self-join problem. In our experiments (described in Section 5.6) we compare the CPSJOIN algorithm against the approximate methods of MinHash LSH [78, 35] and BayesLSH [44], as well as the AllPairs [23] exact similarity join algorithm. The code for our experiments is written in C++ and uses the benchmarking framework and data sets of the recent experimental survey on exact similarity join algorithms by Mann et al. [107]. For our implementation we assume that each set x is represented as a list of 32-bit unsigned integers. We proceed by describing the details of each implementation in turn.

5.5.1 Chosen Path Similarity Join

The implementation of the CPSJOIN algorithm follows the structure of the pseudocode in Algorithm 1 and Algorithm 2, but makes use of a few

heuristics, primarily sampling and sketching, in order to speed things up. The parameter setting is discussed and investigated experimentally in section 5.6.2.

Preprocessing. Before running the algorithm we use the embedding described in section 5.2.1. Specifically t independent MinHash functions $h_1, \ldots, h_t$ are used to map each set $x \in S$ to a list of t hash values $(h_1(x), \ldots, h_t(x))$. The MinHash function is implemented using Zobrist hashing [174] from 32 bits to 64 bits with 8-bit characters. We sample a MinHash function h by sampling a random Zobrist hash function g and let $h(x) = \mathrm{argmin}_{j \in x} g(j)$. Zobrist hashing (also known as simple tabulation hashing) has been shown theoretically to have strong MinHash properties and is very fast in practice [135, 160]. We set $t = 128$ in our experiments, see discussion later.

During preprocessing we also prepare sketches using the 1-bit min-wise hashing scheme of Li and König [103]. Let ℓ denote the length in 64-bit words of a sketch $\hat{x}$ of a set $x \in S$. We construct sketches for a data set S by independently sampling $64 \times \ell$ MinHash functions h_i and Zobrist hash functions g_i that map from 32 bits to 1 bit. The ith bit of the sketch $\hat{x}$ is then given by $g_i(h_i(x))$. In the experiments we set $\ell = 8$.

Similarity estimation using sketches. We use 1-bit minwise hashing sketches for fast similarity estimation in the BRUTEFORCEPAIRS and BRUTE-FORCEPOINT subroutines of the BRUTEFORCE step of the CPSJOIN algorithm. Given two sketches, $\hat{x}$ and $\hat{y}$, we compute the number of bits in which they differ by going through the sketches word for word, computing the popcount of their XOR using the gcc builtin `_mm_popcnt_u64` that translates into a single instruction on modern hardware. Let $\hat{J}(x, y)$ denote the estimated similarity of a pair of sets (x, y). If $\hat{J}(x, y)$ is below a threshold $\hat{\lambda} \approx \lambda$, we exclude the pair from further consideration. If the estimated similarity is greater than $\hat{\lambda}$ we compute the exact similarity and report the pair if $J(x, y) \geq \lambda$.

The speedup from using sketches comes at the cost of introducing false negatives: A pair of sets (x, y) with $J(x, y) \geq \lambda$ may have an estimated similarity less than $\hat{\lambda}$, causing us to miss it. We let δ denote a parameter for controlling the false negative probability of our sketches and set $\hat{\lambda}$ such that for sets (x, y) with $J(x, y) \geq \lambda$ we have that $\Pr[\hat{J}(x, y) < \hat{\lambda}] < \delta$. In our experiments we set the sketch false negative probability to be $\delta = 0.05$.

Splitting step. In the recursive step of the CPSJOIN algorithm (Algorithm 1) the set S is split into buckets S_j using the following heuristic: Instead of sampling a random hash function and evaluating it on each element $j \in x$, we sample an expected $1/\lambda$ elements from $[t]$ and split S according to the corresponding minhash values from the preprocessing step. This saves the linear overhead in the size of our sets t, reducing the time spent placing each set into buckets to $O(1)$. Internally, a collection of sets S is represented as a C++ `std::vector<uint32_t>` of set ids. The collection of buckets S_j is implemented using Google's `dense_hash` hash map implementation from the `sparse_hash` package [81].

BruteForce step. Having reduced the overhead for each set $x \in S$ to $O(1)$ in the splitting step, we wish to do the same for the BRUTEFORCE step (described in Algorithm 2), at least in the case where we do not call the BRUTEFORCEPAIRS or BRUTEFORCEPOINT subroutines. The main problem is that we spend time $O(t)$ for each set when constructing the `count` hash map and estimating the average similarity of x to sets in $S \setminus \{x\}$. To get around this we construct a 1-bit minwise hashing sketch $\hat{s}$ of length $64 \times \ell$ for the set S using sampling and our precomputed 1-bit minwise hashing sketches. The sketch $\hat{s}$ is constructed as follows: Randomly sample $64 \times \ell$ elements of S and set the ith bit of $\hat{s}$ to be the ith bit of the ith sample from S. This allows us to estimate the average similarity of a set x to sets in S in time $O(\ell)$ using word-level parallelism. A set x is removed from S if its estimated average similarity is greater than $(1 - \varepsilon)\lambda$. To further speed up the running time we only call the BRUTEFORCE subroutine once for each call to CPSJOIN, calling BRUTEFORCEPOINT on all points that pass the check rather than recomputing $\hat{s}$ each time a point is removed. Pairs of sets that pass the sketching check are verified using the same verification procedure as the ALLPAIRS implementation by Mann et al. [107]. In our experiments we set the parameter $\varepsilon = 0.1$. Duplicates are removed by sorting and performing a single linear scan.

Repetitions. In theory, for any constant desired recall $\varphi \in (0, 1)$ it suffices with $O(\log^2 n)$ independent repetitions of the CPSJOIN algorithm. In practice this number of repetitions is prohibitively large and we therefore set the number of independent repetitions used in our experiments to be fixed at ten. With this setting we were able to achieve more than 90% recall across all datasets and similarity thresholds.

5.5.2   MinHash LSH

We implement a locality-sensitive hashing similarity join using MinHash according to the pseudocode in Algorithm 3. A single run of the MIN-HASH algorithm can be divided into two steps: First we split the sets into buckets according to the hash values of k concatenated MinHash functions $h(x) = (h_1(x), \ldots, h_k(x))$. Next we iterate over all non-empty buckets and run BRUTEFORCEPAIRS to report all pairs of points with similarity above the threshold λ. The BRUTEFORCEPAIRS subroutine is shared between the MINHASH and CPSJOIN implementation. MINHASH therefore uses 1-bit minwise sketches for similarity estimation in the same way as in the implementation of the CPSJOIN algorithm described above.

The parameter k can be set for each dataset and similarity threshold λ to minimize the combined cost of lookups and similarity estimations performed by algorithm. This approach was mentioned by Cohen et al. [61] but we were unable to find an existing implementation. In practice we set k to the value that results in the minimum estimated running time when running the first part (splitting step) of the algorithm for values of k in the range $\{2, 3, \ldots, 10\}$ and estimating the running time by looking at the number of buckets and their sizes. Once k is fixed we know that each repetition of the algorithm has probability at least λ^k of reporting a pair (x, y) with $J(x, y) \geq \lambda$. For a desired recall φ we can therefore set $L = \lceil \ln(1/(1 - \varphi))/\lambda^k \rceil$. In our experiments we report the actual number of repetitions required to obtain a desired recall rather than using the setting of L required for worst-case guarantees.

Algorithm 3: MINHASH(S, λ)

    Parameters: $k \geq 1, L \geq 1$.

1 **for** $i \leftarrow 1$ **to** L **do**
2   *Initialize hash map* buckets[ ].
3   *Sample k MinHash fcts.* $h \leftarrow (h_1, \ldots, h_k)$
4   **for** $x \in S$ **do**
5   buckets$[h(x)] \leftarrow$ buckets$[h(x)] \cup \{x\}$
6   **for** $S' \in$ buckets **do**
7   BRUTEFORCEPAIRS(S', λ)

5.5.3 AllPairs

To compare our approximate methods against a state-of-the-art exact similarity join we use Bayardo et al.'s AllPairs algorithm [23] as recently implemented in the set similarity join study by Mann et al. [107]. The study by Mann et al. compares implementations of several different exact similarity join methods and finds that the simple AllPairs algorithm is most often the fastest choice. Furthermore, for Jaccard similarity, the AllPairs algorithm was at most 2.16 times slower than the best out of six different competing algorithm across all the data sets and similarity thresholds used, and for most runs AllPairs is at most 11% slower than the best exact algorithm (see Table 7 in Mann et al. [107]). Since our experiments run in the same framework and using the same datasets and with the same thresholds as Mann et al.'s study, we consider their AllPairs implementation to be a good representative of exact similarity join methods for Jaccard similarity.

5.5.4 BayesLSH

For a comparison against previous experimental work on approximate similarity joins we use an implementation of BayesLSH in C as provided by the BayesLSH authors [44, 43]. The BayesLSH package features a choice between AllPairs and LSH as candidate generation method. For the verification step there is a choice between BayesLSH and BayesLSH-lite. Both verification methods use sketching to estimate similarities between candidate pairs. The difference between BayesLSH and BayesLSH-lite is that the former uses sketching to estimate the similarity of pairs that pass the sketching check, whereas the latter uses an exact similarity computation if a pair passes the sketching check. Since the approximate methods in our CPSJoin and MinHash implementations correspond to the approach of BayesLSH-lite we restrict our experiments to this choice of verification algorithm. In our experiments we will use BayesLSH to represent the fastest of the two candidate generation methods, combined with BayesLSH-lite for the verification step.

5.6 Experiments

We run experiments using the implementations of CPSJoin, MinHash, BayesLSH, and AllPairs described in the previous section. In the

Table 5.1: Dataset size, average set size, and average number of sets that a token is contained in.

Dataset	# sets / 10^6	avg. set size	sets / tokens
AOL	7.35	3.8	18.9
BMS-POS	0.32	9.3	1797.9
DBLP	0.10	82.7	1204.4
ENRON	0.25	135.3	29.8
FLICKR	1.14	10.8	16.3
LIVEJ	0.30	37.5	15.0
KOSARAK	0.59	12.2	176.3
NETFLIX	0.48	209.8	5654.4
ORKUT	2.68	122.2	37.5
SPOTIFY	0.36	15.3	7.4
UNIFORM	0.10	10.0	4783.7
TOKENS10K	0.03	339.4	10000.0
TOKENS15K	0.04	337.5	15000.0
TOKENS20K	0.06	335.7	20000.0

experiments we perform self-joins under Jaccard similarity for similarity thresholds $\lambda \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. We are primarily interested in measuring the join time of the algorithms, but we also look at the number of candidate pairs (x, y) considered by the algorithms during the join as a measure of performance. Note that the preprocessing step of the approximate methods only has to be performed once for each set and similarity measure, and can be re-used for different similarity joins, we therefore do not count it towards our reported join times. In practice the preprocessing time is at most a few minutes for the largest data sets.

Data sets.   The performance is measured across 10 real world data sets along with 4 synthetic data sets described in Table 5.1. All datasets except for the TOKENS datasets were provided by the authors of [107] where descriptions and sources for each data set can also be found. Note that we have excluded a synthetic ZIPF dataset used in the study by Mann et al.[107] due to it having no results for our similarity thresholds of interest. Experiments are run on versions of the datasets where duplicate records are removed and any records containing only a single token are ignored.

   In addition to the datasets from the study of Mann et al. we add three synthetic datasets TOKENS10K, TOKENS15K, and TOKENS20K, designed to showcase the robustness of the approximate methods. These datasets have relatively few unique tokens, but each token appears in

many sets. Each of the TOKENS datasets were generated from a universe of 1000 tokens ($d = 1000$) and each token is contained in respectively, $10,000$, $15,000$, and $20,000$ different sets as denoted by the name. The sets in the TOKENS datasets were generated by sampling a random subset of the set of possible tokens, rejecting tokens that had already been used in more than the maximum number of sets ($10,000$ for TOKENS10K). To sample sets with expected Jaccard similarity λ' the size of our sampled sets should be set to $(2\lambda'/(1 + \lambda'))d$. For $\lambda' \in \{0.95, 0.85, 0.75, 0.65, 0.55\}$ the TOKENS datasets each have 100 random sets planted with expected Jaccard similarity λ'. This ensures an increasing number of results for our experiments where we use thresholds $\lambda \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. The remaining sets have expected Jaccard similarity 0.2. We believe that the TOKENS datasets give a good indication of the performance on real-world data that has the property that most tokens appear in a large number of sets.

Recall.   In our experiments we aim for a recall of at least 90% for the approximate methods. In order to achieve this for the CPSJoin and MinHash algorithms we perform a number of repetitions after the preprocessing step, stopping when the desired recall has been achieved. This is done by measuring the recall against the recall of AllPairs and stopping when reaching 90%. In practice this approach is not feasible as the size of the true result set is not known. However, it can be efficiently estimated using sampling if it is not too small. Another approach is to perform the number of repetitions required to obtain the theoretical guarantees on recall as described for CPSJoin in Section 5.4.3 and for MinHash in Section 5.5.2. Unfortunately, with our current analysis of the CPSJoin algorithm the number of repetitions required to guarantee theoretically a recall of 90% far exceeds the number required in practice as observed in our experiments where ten independent repetitions always suffice. For BayesLSH using LSH as the candidate generation method, the recall probability with the default parameter setting is 95%, although we experience a recall closer to 90% in our experiments.

Hardware.   All experiments were run on an Intel Xeon E5-2690v4 CPU at 2.60GHz with 35MB L3, 256kB L2 and 32kB L1 cache and 512GB of RAM. Since a single experiment is always confined to a single CPU core we ran several experiments in parallel [156] to better utilize our hardware.

Table 5.2: Join time in seconds for AllPairs (ALL) and CPSJoin (CP) with recall $\geq$ 90%.

Dataset	Threshold 0.5		Threshold 0.6		Threshold 0.7		Threshold 0.8		Threshold 0.9	
	CP	ALL	CP	ALL	CP	ALL	CP	ALL	CP	ALL
AOL	483.5	**362.1**	117.8	**113.4**	**13.7**	42.2	**4.2**	34.6	**1.6**	21.0
BMS-POS	62.5	**27.0**	20.9	**7.1**	5.6	**2.7**	1.3	**2.0**	**0.2**	0.9
DBLP	127.9	**9.2**	63.8	**2.5**	27.4	**1.1**	7.8	**0.6**	0.8	**0.3**
ENRON	78.0	**6.9**	23.2	**4.4**	6.0	**2.4**	1.6	1.6	0.4	0.7
FLICKR	**17.2**	48.6	**6.0**	30.9	**2.5**	13.8	**1.0**	6.3	**0.3**	3.4
KOSARAK	**73.1**	377.9	**14.4**	62.7	**1.6**	7.2	**0.5**	3.9	**0.1**	1.2
LIVEJ	571.7	**131.3**	145.3	**48.7**	30.6	**28.2**	7.1	16.2	1.5	9.2
NETFLIX	1354.7	**25.3**	520.4	**8.2**	177.3	**4.8**	46.2	**2.4**	5.4	**1.6**
ORKUT	359.7	**26.5**	106.4	**15.4**	36.3	**8.0**	12.2	**7.4**	3.7	4.8
SPOTIFY	**0.5**	2.5	**0.3**	1.5	**0.2**	1.0	**0.1**	1.0	**0.1**	0.5
TOKENS10K	312.1	**3.4**	236.8	**2.9**	164.0	**1.5**	114.9	**0.6**	63.2	**0.2**
TOKENS15K	688.4	**4.4**	535.3	**4.0**	390.4	**1.8**	258.2	**0.7**	140.0	**0.2**
TOKENS20K	1264.1	**5.7**	927.0	**4.0**	698.4	**2.1**	494.3	**0.8**	273.4	**0.3**
UNIFORM005	54.1	**3.9**	27.6	**1.6**	10.5	**0.9**	3.6	**0.5**	0.4	**0.1**

5.6.1   Results

Join time.   Table 5.2 shows the average join time in seconds over five independent runs, when approximate methods are required to have at least 90% recall. We have omitted timings for BayesLSH since it was always slower than all other methods, and in most cases it timed out after 20 minutes when using LSH as candidate generation method. The join time for MinHash is always greater than the corresponding join time for CPSJoin except in a single setting: the dataset KOSARAK with threshold $\lambda = 0.5$. Since CPSJoin is typically $2 - 4\times$ faster than MinHash we can restrict our attention to comparing AllPairs and CPSJoin where the picture becomes more interesting.

Figure 5.2 shows the join time speedup that CPSJoin achieves over AllPairs. We achieve speedups of between $2 - 50\times$ for most of the datasets, with greater speedups at low similarity thresholds. For a number of the datasets the CPSJoin algorithm is slower than AllPairs for the thresholds considered here. Comparing with Table 5.1 it seems that CPSJoin generally performs well on most data sets where tokens are contained in a large number of sets on average (NETFLIX, UNIFORM, DBLP), but is beaten by AllPairs on datasets that have a lot of "rare" tokens (SPOTIFY, FLICKR, AOL). This is logical because rare tokens are exploited by the sorted prefix-filtering in AllPairs. Without rare tokens AllPairs will be reading long inverted indexes. This is a common theme among all the current state-of-the-art exact methods examined in [107], including PPJoin. CPSJoin is robust in the sense that it does not depend
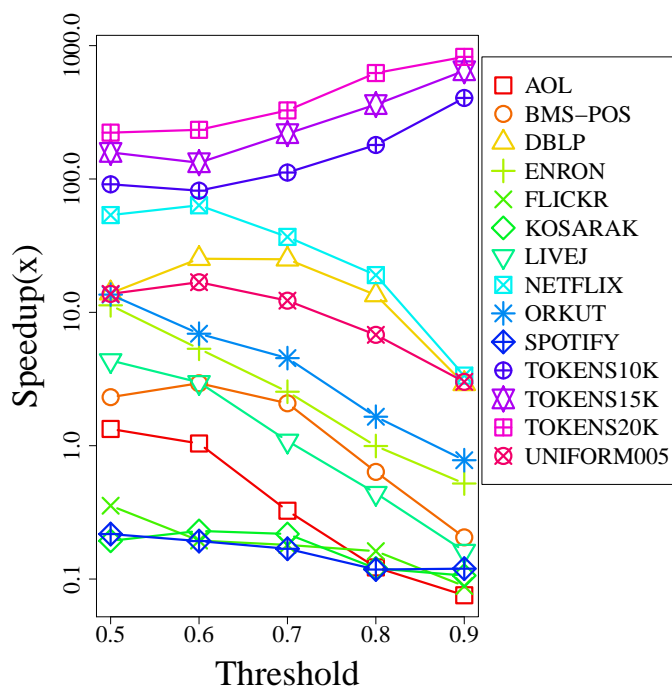
Figure 5.2: Join time of CPSJoin with at least 90% recall relative to AllPairs.

on the presence of rare tokens in the data to perform well. This difference is showcased with the synthetic TOKEN data sets.

BayesLSH. The poor performance of BᴀʏᴇsLSH compared to the other algorithms (BᴀʏᴇsLSH was always slower) can most likely be tracked down to differences in the implementation of the candidate generation methods of BᴀʏᴇsLSH. The BᴀʏᴇsLSH implementation uses an older implementation of AʟʟPᴀɪʀs compared to the implementation by Mann et al. [107] which was shown to yield performance improvements by using a more efficient verification procedure. The LSH candidate generation method used by BᴀʏᴇsLSH corresponds to the MɪɴHᴀsʜ splitting step, but with k (the number of hash functions) fixed to one. Our technique for choosing k in the MɪɴHᴀsʜ algorithm, aimed at minimizing the total join time, typically selects $k \in \{3, 4, 5, 6\}$ in the experiments. It is therefore likely that BᴀʏᴇsLSH can be competitive with the other techniques by combining it with other candidate generation procedures. Further experiments to compare the performance of BayesLSH sketching

to 1-bit minwise sketching for different parameter settings and similarity thresholds would also be instructive.

TOKEN datasets.   The TOKENS datasets clearly favor the approximate join algorithms where CPSJoin is two to three orders of magnitude faster than AllPairs. By increasing the number of times each token appears in a set we can make the speedup of CPSJoin compared to AllPairs arbitrarily large as shown by the progression from TOKENS10 to TO-KENS20. The AllPairs algorithm generates candidates by searching through the lists of sets that contain a particular token, starting with rare tokens. Since every token appears in a large number of sets every list will be long.

Interestingly, the speedup of CPSJoin is even greater for higher similarity thresholds. We believe that this is due to an increase in the gap between the similarity of sets to be reported and the remaining sets that have an average Jaccard similarity of 0.2. This is in line with our theoretical analysis of CPSJoin and most theoretical work on approximate similarity search where the running time guarantees usually depend on the approximation factor.

Candidates and verification.   Table 5.4 compares the number of pre-candidates, candidates, and results generated by the AllPairs and CP-SJoin algorithms where the desired recall for CPSJoin is set to be greater than 90%. For AllPairs the number of pre-candidates denotes all pairs (x, y) investigated by the algorithm that pass checks on their size so that it is possible that $J(x, y) \geq \lambda$. The number of candidates is simply the number of unique pre-candidates as duplicate pairs are removed explicitly by the AllPairs algorithm.

For CPSJoin we define the number of pre-candidates to be all pairs (x, y) considered by the BruteForcePairs and BruteForcePoint sub-routines of Algorithm 2. The number of candidates are pre-candidate pairs that pass size checks (similar to AllPairs) and the 1-bit minwise sketching check as described in Section 5.5.1. Note that for CPSJoin the number of candidates may still contain duplicates as this is inherent to the approximate method for candidate generation. Removing duplicates though the use of a hash table would drastically increase the space usage of CPSJoin. For both AllPairs and CPSJoin the number of candidates denotes the number of points that are passed to the exact similarity verification step of the AllPairs implementation of Mann et al. [107].

Table 5.3: Parameters of the CPSJoin algorithm, their setting during parameter experiments, and their setting for the final join time experiments

Parameter	Description	Test	Final
limit	Brute force limit	100	250
ℓ	Sketch word length	4	8
t	Size of MinHash set	128	128
ε	Brute force aggressiveness	0.0	0.1
δ	Sketch false negative prob.	0.1	0.05

Table 5.4 shows that for ALLPAIRS there is not a great difference between the number of pre-candidates and number of candidates, while for CPSJOIN the number of candidates is usually reduced by one or two orders of magnitude for datasets where CPSJOIN performs well. For datasets where CPSJOIN performs poorly such as AOL, FLICKR, and KOSARAK there is less of a decrease when going from pre-candidates to candidates. It would appear that this is due to many duplicate pairs from the candidate generation step and not a failure of the sketching technique.

5.6.2 Parameters

To investigate how parameter settings affect the performance of the CPSJOIN algorithm we run experiments where we vary the brute force parameter limit, the brute force aggressiveness parameter ε, and the sketch length in words ℓ. Table 5.3 shows the parameter settings used during theses experiments and the final setting used for our join time experiments. Figure 5.3 shows the CPSJOIN join time for different settings of the parameters. By picking one parameter at a time we are obviously ignoring possible interactions between the parameters, but the stability of the join times lead us to believe that these interactions have limited effect.

Figure 5.3 (a) shows the effect of the brute force limit on the join time. Lowering limit causes the join time to increase due to a combination of spending more time splitting sets into buckets and due to the lower probability of recall from splitting at a deeper level. The join time is relatively stable for limit $\in \{100, 250, 500\}$.

Figure 5.3 (b) shows the effect of brute force aggressiveness on the join time. As we increase ε, sets that are close to the other elements in their buckets are more likely to be removed by brute force comparing them to
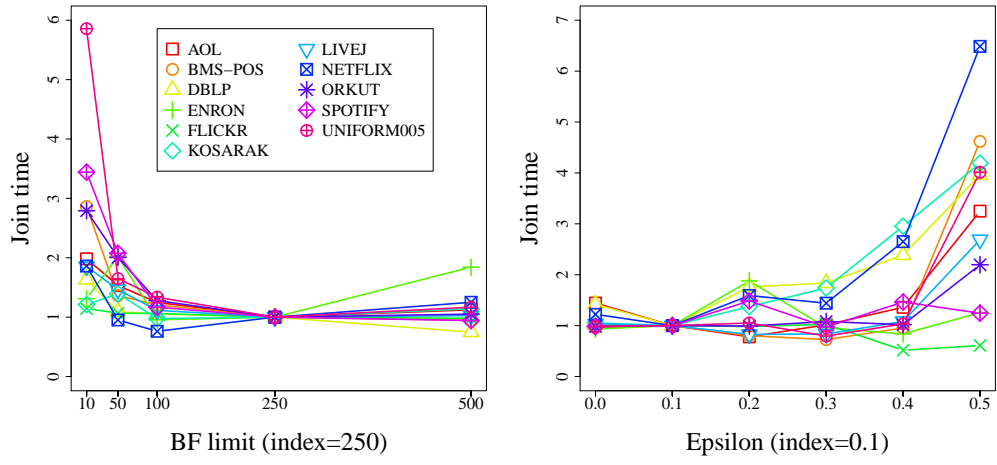
(a) `limit` $\in \{10, 50, 100, 250, 500\}$

(b) $\varepsilon \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$
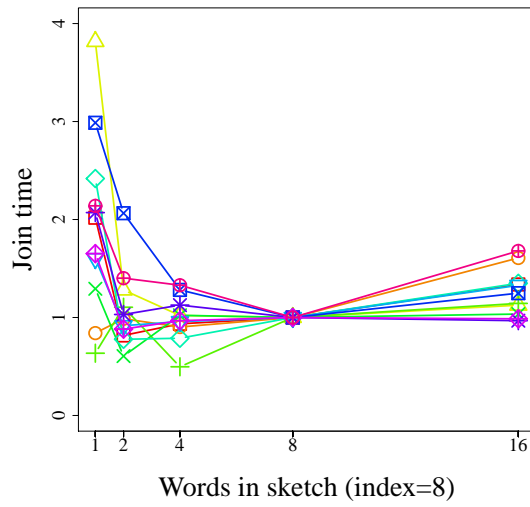
(c) $w \in \{1, 2, 4, 8, 16\}$

Figure 5.3: Relative join time for CPSJoin with at least 80% recall and similarity threshold $\lambda = 0.5$ for different parameter settings of `limit`, ε, and w.

all other points. The tradeoff here is between the loss of probability of recall by letting a point continue in the CHOSEN PATH branching process versus the cost of brute forcing the point. The join time is generally increasing with ε, but it turns out that $\varepsilon = 0.1$ is a slightly better setting than $\varepsilon = 0.0$ for almost all data sets.

Figure 5.3 (c) shows the effect of sketch length on the join time. There is a trade-off between the sketch similarity estimation time and the precision of the estimate, leading to fewer false positives. For a similarity threshold of $\lambda = 0.5$ using only a single word negatively impacts the performance on most datasets compared to using two or more words. The cost of using longer sketches seems neglible as it is only a few extra instructions per similarity estimation so we opted to use $\ell = 8$ words in our sketches.

5.7 Conclusion

We provided experimental and theoretical results on a new randomized set similarity join algorithm, CPSJOIN, and compared it experimentally to state-of-the-art exact and approximate set similarity join algorithms. CP-SJOIN is typically $2 - 4$ times faster than previous approximate methods. Compared to exact methods it obtains speedups of more than an order of magnitude on real-world datasets, while keeping the recall above 90%. Among the datasets used in these experiments we note that NETFLIX and FLICKR represents two archetypes. On average a token in the NETFLIX dataset appears in more than 5000 sets while on average a token in the FLICKR dataset appears in less than 20 sets. Our experiments indicate that CPSJOIN brings large speedups to the NETFLIX type datasets, while it is hard to improve upon the perfomance of ALLPAIRS on the FLICKR type.

A direction for future work could be to tighten and simplify the theoretical analysis. We conjecture that the running time of the algorithm can be bounded by a simpler function of the sum of similarities between pairs of points in S. We note that recursive methods such as ours lend themselves well to parallel and distributed implementations since most of the computation happens in independent, recursive calls. Further investigating this is an interesting possibility.

Table 5.4: Number of pre-candidates, candidates and results for ALL and CP with at least 90% recall.

Dataset	Threshold 0.5		Threshold 0.7	
	ALL	CP	ALL	CP
AOL	8.5E+09	7.4E+09	6.2E+08	2.9E+09
	8.5E+09	1.4E+09	6.2E+08	3.1E+07
	1.3E+08	1.2E+08	1.6E+06	1.5E+06
BMS-POS	2.0E+09	9.2E+08	2.7E+08	3.3E+08
	1.8E+09	1.7E+08	2.6E+08	4.9E+06
	1.1E+07	1.0E+07	2.0E+05	1.8E+05
DBLP	6.6E+09	4.6E+08	1.2E+09	1.3E+08
	1.9E+09	4.6E+07	7.2E+08	4.3E+05
	1.7E+06	1.6E+06	9.1E+03	8.5E+03
ENRON	2.8E+09	3.7E+08	2.0E+08	1.5E+08
	1.8E+09	6.7E+07	1.3E+08	2.1E+07
	3.1E+06	2.9E+06	1.2E+06	1.2E+06
FLICKR	5.7E+08	2.1E+09	9.3E+07	9.0E+08
	4.1E+08	1.1E+09	6.3E+07	3.8E+08
	6.6E+07	6.1E+07	2.5E+07	2.3E+07
KOSARAK	2.6E+09	4.7E+09	7.4E+07	4.2E+08
	2.5E+09	2.1E+09	6.8E+07	2.1E+07
	2.3E+08	2.1E+08	4.4E+05	4.1E+05
LIVEJ	9.0E+09	2.8E+09	5.8E+08	1.2E+09
	8.3E+09	3.6E+08	5.6E+08	1.8E+07
	2.4E+07	2.2E+07	8.1E+05	7.6E+05
NETFLIX	8.6E+10	1.3E+09	1.0E+10	4.3E+08
	1.3E+10	3.1E+07	3.4E+09	6.4E+05
	1.0E+06	9.5E+05	2.4E+04	2.2E+04
ORKUT	5.1E+09	1.1E+09	3.0E+08	7.2E+08
	3.9E+09	1.3E+06	2.6E+08	8.1E+04
	9.0E+04	8.4E+04	5.6E+03	5.3E+03
SPOTIFY	5.0E+06	1.2E+08	4.7E+05	8.5E+07
	4.8E+06	3.1E+05	4.6E+05	2.7E+03
	2.0E+04	1.8E+04	2.0E+02	1.9E+02
TOKENS10K	1.5E+10	1.7E+08	8.1E+09	4.9E+07
	4.1E+08	5.7E+06	4.1E+08	1.9E+06
	1.3E+05	1.3E+05	7.4E+04	6.9E+04
TOKENS15K	3.6E+10	3.0E+08	1.9E+10	8.1E+07
	9.6E+08	7.2E+06	9.6E+08	1.9E+06
	1.4E+05	1.3E+05	7.5E+04	6.9E+04
TOKENS20K	6.4E+10	4.4E+08	3.4E+10	1.0E+08
	1.7E+09	8.8E+06	1.7E+09	1.9E+06
	1.4E+05	1.4E+05	7.9E+04	7.4E+04
UNIFORM005	2.5E+09	3.7E+08	6.5E+08	1.3E+08
	2.0E+09	9.5E+06	6.1E+08	3.9E+04
	2.6E+05	2.4E+05	1.4E+03	1.3E+03

Acknowledgment

Chapter 6

Lower bounds for asymmetric locality-sensitive hashing

'Deep roots are not reached by the frost'

A locality-sensitive hashing (LSH) scheme [91] for a set of objects X equipped with a pairwise measure of similarity $\text{sim}\colon X \times X \to \mathbb{R}$ is a distribution $\mathcal{H}$ over functions $h\colon X \to R$ such that the probability that a pair of points in X collide under a randomly sampled $h \sim \mathcal{H}$ is a function of their similarity. That is, there exists a collision probability function (CPF) $f\colon \mathbb{R} \to [0,1]$ such that for every $x, y \in X$ we have $\Pr_{h\sim\mathcal{H}}[h(x) = h(y)] = f(\text{sim}(x,y))$.

To apply locality-sensitive hashing to solve the problem of approximate (s_1, s_2)-similarity search we are interested in finding the distribution $\mathcal{H}$ with a CPF f that magnifies the gap between the collision probability $f(s_1)$ of points with similarity at least s_1 and the collision probability $f(s_2)$ of points with similarity at most $s_2 < s_1$ where we assume that f is non-decreasing. The performance of the solution will be governed by the parameter ρ defined by $f(s_1) = f(s_2)^\rho$.

Asymmetric locality-sensitive hashing [150] extends locality-sensitive hashing by using a distribution $\mathcal{A}$ over pairs of functions $h, g\colon X \to R$ such that $\Pr_{(h,g)\sim\mathcal{A}}[h(x) = g(y)] = f(\text{sim}(x,y))$. The use of asymmetry allows us to construct schemes where the probability of collision between identical points is less than 1. Applications of asymmetric locality-sensitive hashing include maximal inner product search [150], orthogonal vector search [164], annulus queries [22], privacy-preserving similarity estimation [22], and spherical range reporting [22].

Contribution. We show lower bounds for asymmetric locality-sensitive hashing schemes for the Boolean hypercube $\{-1, 1\}^d$ under cosine similarity $\text{sim}(x, y) = \langle x, y \rangle / d$. Specifically, we will both upper and lower bound the probability of randomly α-correlated points $x, y \in \{-1, 1\}^d$ colliding under $(h, g) \sim \mathcal{A}$ as a function of α and the probability that randomly 0-correlated points collide under $(h, g) \sim \mathcal{A}$.

Theorem 6.1. *Let $\mathcal{A}$ be a distribution over function pairs $h, g \colon \{-1, 1\}^d \to R$, and define $\hat{f} \colon [-1, 1] \to [0, 1]$ as $\hat{f}(\alpha) = \Pr[h(x) = g(y)]$ where x, y are randomly α-correlated and $(h, g) \sim \mathcal{A}$. Then, for every $\alpha \in (-1, 1)$ we have*

$$\hat{f}(0)^{\frac{1+|\alpha|}{1-|\alpha|}} \leq \hat{f}(\alpha) \leq \hat{f}(0)^{\frac{1-|\alpha|}{1+|\alpha|}}.$$

As d increases the empirical correlation of a pair of randomly α-correlated points will be concentrated around α. Theorem 6.1 can therefore be used to show that for $\alpha \geq 0$, asymmetric locality-sensitive hashing for $(\alpha, 0)$-cosine similarity search must have $\rho = \log(f(\alpha)) / \log(f(0)) \geq \frac{1-\alpha}{1+\alpha}$ up to lower order terms. This lower bound on the ρ-value matches an existing lower bound for standard locality-sensitive hashing [114, 19] and matches the upper bound of cross-polytope LSH [13], showing that asymmetry cannot help provide better hashing schemes for this problem. Implicitly, this extension of the standard LSH lower bound to asymmetric LSH for $(\alpha, 0)$-cosine similarity search already follows from the space-time tradeoff lower bounds for similarity search shown independently by Andoni et al. [16] and Christiani [54].

Let $\alpha \geq 0$ and suppose we are interested in solving the $(0, -\alpha)$-similarity search problem under negative cosine similarity, i.e. we want to report points with cosine similarity at most 0 while avoiding points with cosine similarity greater than α. Theorem 6.1 lower bounds the ρ-value of asymmetric locality-sensitive hashing schemes for this problem by $\rho = \log(f(0)) / \log(f(\alpha)) \geq \frac{1-\alpha}{1+\alpha}$ up to lower order terms. This can again be matched by a simple asymmetric version of cross-polytope LSH: Let $\mathcal{H}$ denote the cross-polytope LSH, then we sample $(h, g) \sim \mathcal{A}$ by sampling $h' \sim \mathcal{H}$ and setting $h(x) = h'(x)$ and $g(x) = h'(-x)$. For more details and related work see [22].

Techniques. The proof of Theorem 6.1 combines the small-set expansion theorems by O'Donnell [120] with techniques inspired by the LSH lower bound of Motwani et al. [114]. The (reverse) small-set expansion

theorem (lower) upper bounds the probability that randomly α-correlated points (x, y) end up in a pair of subsets A, B of the Boolean hypercube, as a function of the size of these subsets. We are able to extend these bounds for pairs of subsets of the Boolean hypercube to distributions over pairs of functions through a sequence of applications of primarily Jensen's inequality.

6.1 Preliminaries

We begin by introducing the required tools from [120], starting with the definition of randomly α-correlated pairs of points.

Definition 6.1. For $-1 \leq \alpha \leq 1$ and $x, y \in \{-1, 1\}^d$ we say that (x, y) is randomly α-correlated if x is uniformly distributed over $\{-1, 1\}^d$ and each component of y is i.i.d. according to

$$
y_i = \begin{cases} x_i & \text{with probability } \frac{1+\alpha}{2}, \\ -x_i & \text{with probability } \frac{1-\alpha}{2}. \end{cases}
$$

Next we define a probabilistic version of the CPF for cosine similarity on the Boolean hypercube. Theorem 6.1 provides upper and lower bounds on the probabilistic CPF.

Definition 6.2 (Probabilistic CPF). Let $\mathcal{A}$ be a distribution over pairs $h, g \colon \{-1, 1\}^d \to R$. We define the probabilistic CPF $\hat{f} \colon [-1, 1] \to [0, 1]$ by

$$
\hat{f}(\alpha) = \Pr_{\substack{(h,g) \sim \mathcal{D} \\ (x,y) \ \alpha\text{-corr.}}} [h(x) = g(y)].
$$

The proof of the lower bound will make use of the following technical inequality that follows from two applications of Jensen's inequality.

Lemma 6.1. *Let p, q denote discrete probability distributions, then for every $c \geq 1$ we have that*

$$
\sum_i (p_i q_i)^c \geq \left(\sum_i p_i q_i \right)^{2c-1}
$$

with reverse inequality for $c \leq 1$.

Proof. Assume $c \geq 1$. By Jensen's inequality, using the fact that $x \mapsto x^c$ and $x \mapsto x^{2-1/c}$ are convex we have that

$$\sum_i (p_i q_i)^c = \sum_i p_i(p_i^{1-1/c} q_i) \geq \left(\sum_i p_i^{2-1/c} q_i\right)^c \geq \left(\sum_i p_i q_i\right)^{2c-1}.$$

For $c \leq 1$ we have that $x \mapsto x^c$ and $x \mapsto x^{2-1/c}$ are concave and the inequality is reversed.   □

6.2   Lower bounding the collision probability

The reverse small-set expansion theorem lower bounds the probability that random α-correlated points (x, y) end up in a pair of subsets A, B of the Hamming cube, as a function of the size of these subsets. In the following, for $A \subset \{-1, 1\}^d$ we refer to the quantity $|A|/2^d$ as the *volume* of A.

Theorem 6.2 (Rev. Small-Set Expansion [120]). *Let $0 \leq \alpha \leq 1$. Let $A, B \subseteq \{-1, 1\}^d$ have volumes $\exp(-a^2/2), \exp(-b^2/2)$, respectively, where $a, b \geq 0$. Then we have that*

$$\Pr_{\substack{(x,y) \\ \alpha\text{-corr.}}} [x \in A, y \in B] \geq \exp\left(-\frac{1}{2}\frac{a^2 + 2\alpha ab + b^2}{1 - \alpha^2}\right).$$

In the following lemma we convert the lower bound in Theorem 6.2 into a lower bound on the probabilistic CPF.

Lemma 6.2. *For every distribution $\mathcal{A}$ over function pairs $h, g: \{-1, 1\}^d \to R$ and $\alpha \in [0, 1)$ we have that $\hat{f}(\alpha) \geq \hat{f}(0)^{\frac{1+\alpha}{1-\alpha}}$.*

Proof. For a function $h: \{-1, 1\}^d \to R$ define its inverse image $h^{-1}: R \to 2^{\{-1,1\}^d}$ by $h^{-1}(i) = \{x \in \{-1, 1\}^d \mid h(x) = i\}$. For a pair of functions $(h, g) \in \mathcal{D}$ and $i \in R$ we define $a_{h,i}, b_{g,i} \geq 0$ such that $|h^{-1}(i)|/2^d = \exp(-a_{h,i}^2/2)$ and $|g^{-1}(i)|/2^d = \exp(-b_{g,i}^2/2)$. For fixed (h, g) define $\hat{f}_{h,g}(\alpha) = \Pr_{(x,y) \alpha\text{-corr.}}[h(x) = g(y)]$. We obtain a lower bound on $\hat{f}(\alpha)$ as

follows:

$$\hat{f}(\alpha) = \underset{(h,g)\sim\mathcal{A}}{\mathbb{E}} \left[\sum_{i\in R} \underset{(x,y)\ \alpha\text{-corr.}}{\Pr}[h(x) = g(y) = i] \right]$$

$$\overset{(1)}{\geq} \underset{(h,g)\sim\mathcal{A}}{\mathbb{E}} \left[\sum_{i\in R} \exp\left(-\frac{1}{2}\frac{a_{h,i}^2 + 2\alpha a_{h,i}b_{g,i} + b_{g,i}^2}{1-\alpha^2} \right) \right]$$

$$\overset{(2)}{\geq} \underset{(h,g)\sim\mathcal{A}}{\mathbb{E}} \left[\sum_{i\in R} \exp\left(-\frac{1}{2}\frac{a_{h,i}^2 + b_{g,i}^2}{1-\alpha} \right) \right]$$

$$\overset{(3)}{\geq} \underset{(h,g)\sim\mathcal{A}}{\mathbb{E}} \hat{f}_{h,g}(0)^{\frac{1+\alpha}{1-\alpha}} \overset{(4)}{\geq} \left(\underset{(h,g)\sim\mathcal{A}}{\mathbb{E}} \hat{f}_{h,g}(0) \right)^{\frac{1+\alpha}{1-\alpha}}$$

$$= \hat{f}(0)^{\frac{1+\alpha}{1-\alpha}}.$$

Here, (1) is due to Theorem 6.2, (2) follows from the fact that $(1+\alpha)(a^2 + b^2) \geq a^2 + 2\alpha ab + b^2$, (3) follows from Lemma 6.1 with $c = 1/(1-\alpha)$, and (4) follows from a standard application of Jensen's Inequality. $\qquad\square$

6.3   Upper bounding the collision probability

We can re-apply the techniques behind Lemma 6.2 to upper bound the probabilistic CPF. This is similar to the well-studied problem of constructing LSH lower bounds and our results match known LSH bounds [114, 19], showing that the asymmetry afforded by $\mathcal{A}$ does not help us when we wish to maximize the gap in the collision probability between random and α-correlated points as measured by the ρ-value. As with Lemma 6.2, the following theorem by O'Donnell [120] is the foundation of our upper bound on the probilistic CPF.

Theorem 6.3 (Gen. Small-Set Expansion). *Let $0 \leq \alpha \leq 1$. Let $A, B \subseteq \{-1,1\}^d$ have volumes $\exp(-a^2/2)$, $\exp(-b^2/2)$ and assume $0 \leq \alpha b \leq a \leq b$. Then,*

$$\underset{\substack{(x,y)\\ \alpha\text{-corr.}}}{\Pr}[x \in A, y \in B] \leq \exp\left(-\frac{1}{2}\frac{a^2 - 2\alpha ab + b^2}{1-\alpha^2} \right).$$

In the proof of the upper bound on the probabilistic CPF we can without loss of generality assume that the family $\mathcal{A}$ satisfies the volume restrictions that $0 \leq \alpha b \leq a \leq b$ from Theorem 6.3 on the parts of

$(h, g) \sim \mathcal{A}$. The reason is that for $a = \alpha b$ the upper bound becomes $\Pr_{x \in \{-1,1\}^d}[x \in B]$ which is an upper bound for every $0 \le a < \alpha b$ as well. For completeness we include the proof of the upper bound.

Lemma 6.3. *For every distribution $\mathcal{A}$ over function pairs $h, g \colon \{-1,1\}^d \to R$ and $\alpha \in [0,1)$ we have that $\hat{f}(\alpha) \le \hat{f}(0)^{\frac{1-\alpha}{1+\alpha}}$.*

Proof. Using the same notation as in the proof of Lemma 6.2 we derive the upper bound as follows:

$$
\begin{aligned}
\hat{f}(\alpha) &= \mathop{\mathbb{E}}_{(h,g)\sim\mathcal{A}} \left[\sum_{i \in R} \Pr_{(x,y)\ \alpha\text{-corr.}}[h(x) = g(y) = i] \right] \\
&\overset{(1)}{\le} \mathop{\mathbb{E}}_{(h,g)\sim\mathcal{A}} \left[\sum_{i \in R} \exp\left(-\frac{1}{2} \frac{a_{h,i}^2 - 2\alpha a_{h,i} b_{g,i} + b_{g,i}^2}{1 - \alpha^2} \right) \right] \\
&\overset{(2)}{\le} \mathop{\mathbb{E}}_{(h,g)\sim\mathcal{A}} \left[\sum_{i \in R} \exp\left(-\frac{1}{2} \frac{a_{h,i}^2 + b_{g,i}^2}{1 + \alpha} \right) \right] \\
&\overset{(3)}{\le} \mathop{\mathbb{E}}_{(h,g)\sim\mathcal{A}} \hat{f}_{h,g}(0)^{\frac{1+\alpha}{1-\alpha}} \overset{(4)}{\le} \left(\mathop{\mathbb{E}}_{(h,g)\sim\mathcal{A}} \hat{f}_{h,g}(0) \right)^{\frac{1-\alpha}{1+\alpha}} \\
&= \hat{f}(0)^{\frac{1-\alpha}{1+\alpha}}.
\end{aligned}
$$

Here, (1) is due to Theorem 6.3, (2) follows from the fact that $(1 - \alpha)(a^2 + b^2) \le a^2 - 2\alpha ab + b^2$, (3) follows from Lemma 6.1 with $c = 1/(1 + \alpha)$, and (4) follows from a standard application of Jensen's Inequality. $\square$

6.4　Extension to negative correlation

We can extend the bounds from Lemma 6.2 and Lemma 6.3 to negative values of α, essentially mirroring the bounds around $\alpha = 0$, by showing that an asymmetric LSH scheme $\mathcal{A}$ with a value of $\hat{f}(\alpha)$ that is too low or too high for negative values of α will contradict the bounds we have for positive values of α. The extension of the bounds to negative correlation uses the following lemma to show contradictions:

Lemma 6.4. *Suppose we have an asymmetric LSH $\mathcal{A}'$ for $\{-1,1\}^d$ with probabilistic CPF $\hat{f}'$, then the we can construct an asymmetric LSH $\mathcal{A}$ for $\{-1,1\}^d$ with probabilistic CPF $\hat{f}(\alpha) = \hat{f}'(-\alpha)$.*

Proof. We sample $(h, g) \sim \mathcal{A}$ by sampling $(h', g') \sim \mathcal{A}'$ and setting $h(x) = h'(x)$ and $g(y) = g'(-y)$. The property that $\hat{f}(\alpha) = \hat{f}'(-\alpha)$ follows from the fact that if (x, y) are randomly α-correlated then $(x, -y)$ are randomly $-\alpha$-correlated. $\qquad\square$

We can now extend the bounds on the probabilistic CPF to prove Theorem 6.1. For the upper bound, suppose for $\alpha \in [0, 1)$ that there exists a family $\mathcal{A}'$ with $\hat{f}'(-\alpha) > \hat{f}'(0)^{\frac{1-\alpha}{1+\alpha}}$. According to Lemma 6.4 this implies the existence of a family $\mathcal{A}$ with $\hat{f}(\alpha) > \hat{f}(0)^{\frac{1-\alpha}{1+\alpha}}$, contradicting Lemma 6.3. The argument for extending the lower bound is identical, completing the proof of Theorem 6.1.

6.5 Conclusion and open problems

We have shown upper and lower bounds on the collision probability of asymmetric locality-sensitive hashing schemes. The bounds are tight up to lower order terms and match an existing symmetric LSH scheme on the unit sphere and its natural asymmetric extension. An interesting consequence of the upper bound is that asymmetry cannot improve the ρ-value for the standard similarity search problem for random instances in the Boolean hypercube.

The application of asymmetric locality-sensitive hashing to orthogonal vector search seeks a family $\mathcal{A}$ with a CPF f that peaks at $f(0)$ and is decreasing in $|\alpha|$ [164]. Currently the best known upper bound on the ρ-value for orthogonal vector search is $\rho = \log(f(0))/\log(\max(f(-\alpha), f(\alpha))) = \frac{1-\alpha^2}{1+\alpha^2}$ which can be achieved by tensoring the input (squaring the cosine similarity between pairs of points) and applying the standard cross-polytope LSH family, or by combining standard cross-polytope LSH and its negated asymmetric version [22] using the same technique as in Lemma 6.4. We conjecture that the current upper bound is tight implying that the following bound must hold:

$$\max(\hat{f}(-\alpha), \hat{f}(\alpha)) \geq \hat{f}(0)^{\frac{1+\alpha^2}{1-\alpha^2}}.$$

This bound can be characterized as a two-sided bound whereas Theorem 6.1 only provides one-sided upper and lower bounds. The problem of obtaining a two-sided lower bound is related to the open symmetric Gaussian problem [119].

Chapter 7

Optimal Boolean locality-sensitive hashing

'The crownless again shall be king'

Theorem 7.1. *For $0 \leq \beta < \alpha < 1$ the distribution $\mathcal{H}$ over Boolean functions $h\colon \{-1,1\}^d \to \{-1,1\}$ that minimizes the expression*

$$\rho_{\alpha,\beta} = \frac{\log(1/\Pr_{\substack{h\sim\mathcal{H} \\ (x,y)\ \alpha\text{-corr.}}}[h(x) = h(y)])}{\log(1/\Pr_{\substack{h\sim\mathcal{H} \\ (x,y)\ \beta\text{-corr.}}}[h(x) = h(y)])}$$

assigns nonzero probability only to members of the set of dictator functions $h(x) = \pm x_i$.

7.1 Introduction

We will be studying Boolean functions, i.e., functions that for a positive integer d can be written in the form

$$h\colon \{-1,1\}^d \to \{-1,1\}.$$

We are concerned with the behavior of such Boolean functions on input pairs $x, y \in \{-1,1\}^d$ that are randomly generated.

Definition 7.1. For $-1 \leq \alpha \leq 1$ and $x \in \{-1,1\}^d$ we let $N_\alpha(x)$ denote the distribution over $\{-1,1\}^d$ where each component of $y \sim N_\alpha(x)$ is i.i.d. according to

$$y_i = \begin{cases} x_i & \text{with probability } \frac{1+\alpha}{2}, \\ -x_i & \text{with probability } \frac{1-\alpha}{2}. \end{cases}$$

We say that (x,y) is randomly α-correlated if x is uniformly distributed over $\{-1,1\}^d$ and $y \sim N_\alpha(x)$.

Let $\mathcal{H}$ denote a distribution over functions $h\colon \{-1,1\}^d \to R$ where R is a finite set and define

$$p_\alpha = \Pr_{\substack{h\sim\mathcal{H} \\ (x,y)\ \alpha\text{-corr.}}} [h(x) = h(y)].$$

For $0 \le \beta < \alpha \le 1$ we wish to characerize the distributions that minimize the expression

$$\rho_{\alpha,\beta} = \frac{\log(1/p_\alpha)}{\log(1/p_\beta)} \tag{7.1}$$

when we restrict $\mathcal{H}$ to be a distribution over Boolean functions $h\colon \{-1,1\}^d \to \{-1,1\}$. The expression for $\rho_{\alpha,\beta}$ in equation (7.1) is a well-known quantity in the study of approximate near neighbor search governing the query time and space usage of solutions based on locality-sensitive hashing [91].

7.2   Related work

Indyk and Motwani [91] introduced the uniform distribution over the set of dictator functions as a family of locality-sensitive hash functions for the Boolean hypercube. O'Donnell et al. [122] showed that for general families $\mathcal{H}$ it must hold that $\rho_{\alpha,\beta} \ge \log(1/\alpha)/\log(1/\beta)$. This matches the upper bound of Indyk and Motwani [91] when α, β approach 1. Another line of work[131, 19] using hypercontractive inequalities showed that $\rho_{\alpha,0} \ge (1-\alpha)/(1+\alpha)$, matching the upper bound of Andoni et al. [17].

The question of finding lower bounds for $\rho_{\alpha,\beta}$ for every choice of $0 \le \beta < \alpha \le 1$ is still open. In this note we answer the question for distributions over *Boolean* functions, showing that the upper bound of Indyk and Motwani is optimal. The resulting ρ-value is given by

$$\rho_{\alpha,\beta} = \frac{\log((1+\alpha)/2)}{\log((1+\beta)/2)}.$$

7.3   Preliminaries

We will be using tools from the Fourier analysis of Boolean functions to find the minimum of $\rho_{\alpha,\beta}$. For a more detailed overview we refer to the book by O'Donnell [120]. We will be using the fact that Boolean functions can be uniquely expressed as multilinear polynomials:

Theorem 7.2. *Every function $f\colon \{-1,1\}^d \to \mathbb{R}$ can be uniquely expressed as a multilinear polynomial*

$$f(x) = \sum_{S \subseteq [d]} \hat{f}(S) x^S$$

where $\hat{f}(S) \in \mathbb{R}$ and $x^S = \prod_{i \in S} x_i$.

For $S \subseteq [d]$ we refer to $\hat{f}(S)$ as the Fourier coefficient of f on S. The two following Theorems define an inner product between Boolean function and shows how it relates to their Fourier coefficents.

Theorem 7.3 (Plancherel's Theorem). *For any $f, g \colon \{-1,1\}^d \to \mathbb{R}$*

$$\langle f, g \rangle = \mathop{\mathbb{E}}_{x \sim \{-1,1\}^d} [f(x)g(x)] = \sum_{S \subseteq [d]} \hat{f}(S)\hat{g}(S).$$

The concept of Fourier weight will be useful when characterizing the how Boolean functions behave on noisy inputs:

Definition 7.2. Let $f \colon \{-1,1\}^d \to R$ then we define the Fourier weight of f at degree $k \in [d]$ by

$$W^k[f] = \sum_{\substack{S \subseteq [d] \\ |S| = k}} \hat{f}(S)^2.$$

When $f = g$ is Boolean-valued we get that the sum of the squared Fourier weights of f equals 1. This result is known as Parseval's Theorem and we will make use of it to determine where to place to Fourier weight of f in order to minimize ρ.

Theorem 7.4 (Parseval's Theorem). *For any $f \colon \{-1,1\}^d \to \{-1,1\}$*

$$\langle f, f \rangle = \mathop{\mathbb{E}}_{x \sim \{-1,1\}^d} [f(x)^2] = \sum_{S \subseteq [d]} \hat{f}(S)^2 = \sum_{i=0}^{d} W^i[f] = 1.$$

In order to study the behavior of Boolean functions under noise we introduce the noise operator T_α.

Definition 7.3. For $\alpha \in [-1,1]$ the noise operator with parameter α is the linear operator T_α on functions $f \colon \{-1,1\}^d \to \mathbb{R}$ defined by

$$T_\alpha f(x) = \mathop{\mathbb{E}}_{y \sim N_\alpha(x)} [f(y)].$$

The Fourier expansion of $T_\alpha f(x)$ is given by $\sum_{S \subseteq [d]} \alpha^{|S|} \hat{f}(S) x^S$ and it follows from Plancherel's Theorem that

$$\langle f, T_\alpha g \rangle = \mathop{\mathbb{E}}_{x \sim \{-1,1\}^d} \left[f(x) \mathop{\mathbb{E}}_{y \sim N_\alpha(x)} [g(y)] \right]$$

$$= \mathop{\mathbb{E}}_{(x,y) \; \alpha\text{-corr.}} [f(x) g(y)] = \sum_{S \subseteq [d]} \alpha^{|S|} \hat{f}(S) \hat{g}(S). \qquad (7.2)$$

In the analysis of our problem the following inequality will be used several times. For the remainder of this Chapter we will use $\log x$ to denote the natural logarithm of x.

Lemma 7.1. *For $x > 0$ we have $\log x \leq x - 1$ with equality if and only if $x = 1$.*

7.4   Bit-sampling is optimal

Our approach will be to minimize $\rho_{\alpha,\beta}$ subject to the constraint that members of $\mathcal{H}$ are Boolean functions $h: \{-1,1\}^d \to \{-1,1\}$. We begin by making some observations to simplify the problem. For $h \sim \mathcal{H}$ we can directly relate the noise-sensitivity under random α-correlated inputs to the collision probability.

$$\mathop{\mathbb{E}}_{\substack{h \sim \mathcal{H} \\ (x,y) \; \alpha\text{-corr.}}} [h(x)h(y)] = \mathop{\Pr}_{\substack{h \sim \mathcal{H} \\ (x,y) \; \alpha\text{-corr.}}} [h(x) = h(y)] - \mathop{\Pr}_{\substack{h \sim \mathcal{H} \\ (x,y) \; \alpha\text{-corr.}}} [h(x) \neq h(y)]$$

$$= p_\alpha - (1 - p_\alpha)$$

$$= 2p_\alpha - 1.$$

Using Equation (7.2) we can write p_α as follows:

$$p_\alpha = \left(1 + \mathop{\mathbb{E}}_{\substack{h \sim \mathcal{H} \\ (x,y) \; \alpha\text{-corr.}}} [h(x)h(y)]\right)/2 = \left(1 + \sum_{i=0}^{d} \alpha^i w_i\right)/2$$

where we use w_i to denote the expected Fourier weight of $h \sim \mathcal{H}$ at degree i defined by $w_i = \mathbb{E}_{h \sim \mathcal{H}} \sum_{i=0}^{d} W^i[h]$. From Plancherel's Theorem we have that $\sum_{i=0}^{d} w_i = 1$. We will now consider how to set $w_0, w_1, \ldots, w_d$ to minimize the expression

$$\rho_{\alpha,\beta} = \frac{\log((1 + \sum_{i=0}^{d} \alpha^i w_i)/2)}{\log((1 + \sum_{i=0}^{d} \beta^i w_i)/2)}.$$

An optimal solution $w_0^*, \ldots, w_d^*$ for this problem will yield an optimal solution to the original problem, provided there actually exists a Boolean-valued function satisfying the weight assignment. We will show that the assignment $w_1^* = 1$ and $w_i^* = 0$ for $i \neq 1$ minimizes $\rho_{\alpha,\beta}$. The distribution $\mathcal{H}$ therefore only assigns positive probability to functions h that have all their Fourier weight concentrated at degree 1. It turns out that a Boolean function satisfies this weight assignment if and only if it is a dictator function.

Lemma 7.2. *Let $f \colon \{-1,1\}^d \to \{-1,1\}$ and suppose that $W^1[f] = 1$, then $f(x) = \pm x_i$.*

Proof. From Parseval's Theorem we know that $\sum_i W^i[f] = 1$ and it follows that $\hat{f}(S) = 0$ for $|S| \neq 1$. The function f can therefore be written on the form $f(x) = \sum_{i=1}^d \hat{f}_i x_i$ where $\hat{f}_i = \hat{f}(S)$ for $S = \{i\}$. By the condition $W^1[f] = 1$ there exists $j \in [d]$ such that $\hat{f}_j \neq 0$. Fix the $d-1$ components $x_{i \neq j}$ of x and note that since f maps to $\{-1,1\}$ the sum $f(x) = \hat{f}_j x_j + \sum_{i \neq j} \hat{f}_i x_i$ must satisfy $f(x) = \pm 1$ when $x_j = \pm 1$. For $\hat{f}_j \neq 0$ this is only possible when $\hat{f}_j = \pm 1$ which implies that $\hat{f}_i = 0$ for $i \neq j$. It follows that f must be one of the $2d$ functions of the form $f(x) = \pm x_i$. □

7.4.1 Optimal Fourier weight at degree zero

We begin by arguing that we can restrict our attention to showing that dictator functions are optimal in the case where $0 < \beta < \alpha < 1$. If $\alpha = 1$ then for $w_1 = 1$ we have that $\rho = 0$ which is the best we can hope for (but this could also be achieved by other weight assignments, hence the statement of the main theorem is for $\alpha < 1$.). For $\beta = 0$ the following Lemma showing that $w_0^* = 0$ combined with the fact that for this setting we maximize p_α by setting $w_1 = 1$ shows that the dictator functions are optimal. We will now show that an optimal solution has no Fourier weight at degree zero.

Lemma 7.3. $w_0^* = 0$.

Proof. If $w_0 = 1$ we have $\rho = 1$ and it is clear that $\rho < 1$ if we set $w_1 = 1$. Suppose that $0 < w_0^* < 1$. We will show that in this case we can move some weight from w_0 to w_1 and decrease the value of ρ. For a given weight assignment define $s(\alpha) = \sum_i \alpha^i w_i$ and write w_1 as $w_1 =$

$1 - \sum_{j \neq i} w_j$. The partial derivative of $\rho = \log((1 + s(\alpha))/2)/\log((1 + s(\beta))/2)$ with respect to w_0 is given by

$$\frac{\partial \rho}{\partial w_0} = \frac{\frac{\partial s(\alpha)/\partial w_0}{1+s(\alpha)} \log \frac{1+s(\beta)}{2} - \frac{\partial s(\beta)/\partial w_0}{1+s(\beta)} \log \frac{1+s(\alpha)}{2}}{\log^2 \frac{1+s(\beta)}{2}}.$$

By rearranging and using that $\partial s(\alpha)/\partial w_0 = 1 - \alpha$ we find that $\frac{\partial \rho}{\partial w_0} > 0$ is equivalent to

$$\frac{1 + s(\beta)}{1 - \beta} \log \frac{1 + s(\beta)}{2} > \frac{1 + s(\alpha)}{1 - \alpha} \log \frac{1 + s(\alpha)}{2}.$$

It suffices to show that the function $g(x) = \frac{1+s(x)}{1-x} \log \frac{1+s(x)}{2}$ is decreasing for $0 < x < 1$.

$$\frac{\partial g}{\partial x} = \frac{s'(x)(1 - x) + (1 + s(x))}{(1 - x)^2} \log \frac{1 + s(x)}{2} + \frac{s'(x)}{1 - x}.$$

Rewriting, this is equivalent to showing that

$$(s'(x)(1 - x) + 1 + s(x)) \log \frac{1 + s(x)}{2} + (1 - x)s'(x) < 0.$$

By the assumption that $0 < w_0 < 1$ we have that $0 < s(x) < 1$ and using Lemma 7.1 we get that $\log((1 + s(x))/2) < (s(x) - 1)/2$. The above condition then simplifies to showing that $s'(x)(1 - x) + s(x) \leq 1$. The function $s(x) = \sum_i w_i x^i$ is a weighted sum of simple monomials where the weights sum to one. It therefore suffices to show that the inequality holds for every monomial $s_k(x) = x^k$ where $k = \{0, 1, \ldots, d\}$. For $k = 0$ and $k = 1$ we have $s'_k(x)(1 - x) + s_k(x) = 1$ satisfying the desired inequality. For $k \geq 2$ we have $s'_k(x)(1 - x) + s_k(x) = kx^{k-1} + (k - 1)x^k$. We see that $s_k(0) = 0$ and $s_k(1) = 1$ and by inspecting the derivative of $s_k(x)$ we see that it is increasing for $x \in (0, 1)$. It follows that the inequality is satisfied, completing the proof. □

7.4.2 A continuous optimization problem

In order to simplify the problem of minimizing ρ we will optimize over a larger space. In particular we will let W denote a collection of pairs (w, k) such that $\sum_{w \in W} w = 1$ where we restrict $k \in \mathbb{R}$ to satisfy $k \geq 1$.

We define $s(x) = \sum_{(w,k) \in W} w x^k$ and we will now attempt to specify the function s that minimizes

$$\rho_{\alpha,\beta} = \frac{\log \frac{1+s(\alpha)}{2}}{\log \frac{1+s(\beta)}{2}}$$

subject to the constraint that $s(\beta) = b \leq \beta$ is fixed. The constraint that $s(\beta) \leq \beta$ follows from the restrictions on s. We can therefore write $b = \beta^\gamma$ for some $\gamma \geq 1$. For fixed $s(\beta)$ it is clear that we minimize ρ by maximizing $s(\alpha)$.

Lemma 7.4. *For fixed $s(\beta) = \beta^\gamma$ we maximize $s(\alpha)$ by setting $s(x) = x^\gamma$.*

Proof. Let w denote the weight on the exponent γ in the specification W of s. We will prove that if $w < 1$ then we can increase $s(\alpha)$ by rearranging the weights of s to put more weight onto (w, γ). Note that if $w < 1$ and we have a valid configuration there must exist exponents $\gamma_0 < \gamma < \gamma_1$ such that there is positive weight on γ_0 and γ_1. If all the remaining weight was concentrated to either side of γ the condition $s(\beta) = \beta^\gamma$ would be violated. We will now move ε_0 weight from w_0 to w and ε_1 weight from w_1 to w where we set $\varepsilon_0, \varepsilon_1$ to ensure that $s(\beta) = \beta^\gamma$ after the move. It turns out that this condition is satisfied for the following ratio

$$\varphi(\beta) = \varepsilon_1/\varepsilon_0 = \frac{\beta^{\gamma_0} - \beta^\gamma}{\beta^\gamma - \beta^{\gamma_1}} > 0.$$

The change in $s(\alpha)$ due to the rearrangement of weights can be shown to be positive if $\varphi(\alpha) < \varphi(\beta)$. Therefore, it suffices to show that $\varphi(x)$ is decreasing for $0 < x < 1$ when $\gamma_0 < \gamma < \gamma_1$. To simplify further, we define $\lambda_0 = \gamma_0 - \gamma$ and $\lambda_1 = \gamma_1 - \gamma$ which satisfy $\lambda_0 < 0 < \lambda_1$. Rewriting $\varphi(x) = -(1 - x_0^\lambda)/(1 - x_1^\lambda)$ and differentiating we get

$$\frac{\partial \varphi}{\partial x} = \lambda_0 x^{\lambda_0}(1 - x^{\lambda_1} - \lambda_1 x^{\lambda_1}(1 - x^{\lambda_0}) < 0$$

$$\iff \frac{\lambda_0 x^{\lambda_0}}{1 - x^{\lambda_0}} > \frac{\lambda_1 x^{\lambda_1}}{1 - x^{\lambda_1}}.$$

It suffices to show that $\psi(x) = \frac{xa^x}{1-a^x}$ is decreasing in x for $a \in (0,1)$. We have that $\psi'(x) = a^x(1 - a^x) + a^x \log a^x$. Define $z = a^x$ and note that $z > 0$ and $z \neq 1$. We have that $z(1 - z) + z \log z < 0 \iff (1 - z) + \log z < 0$ and by Lemma 7.1 we see that $\log z < z - 1$, completing the proof. $\square$

7.4.3   Univariate analysis

According to Lemma 7.4 we can now restrict our attention to the problem of finding $\gamma \geq 1$ that minimizes the function

$$\rho(\gamma) = \frac{\log \frac{1+\alpha^\gamma}{2}}{\log \frac{1+\beta^\gamma}{2}}.$$

We will show the derivative of ρ is positive, implying that it is minimized when $\gamma = 1$.

Lemma 7.5. $\rho'(\gamma) > 0$.

Proof. From inspecting the derivative of ρ with respect to γ we see that

$$\frac{\partial \rho}{\partial \gamma} > 0$$

$$\iff \frac{\alpha^\gamma \log \alpha}{1 + \alpha^\gamma} \log \frac{1 + \beta^\gamma}{2} - \frac{\beta^\gamma \log \beta}{1 + \beta^\gamma} \log \frac{1 + \alpha^\gamma}{2} > 0$$

$$\iff \frac{1 + \beta^\gamma}{\beta^\gamma \log \beta} \log \frac{1 + \beta^\gamma}{2} > \frac{1 + \alpha^\gamma}{\alpha^\gamma \log \alpha} \log \frac{1 + \alpha^\gamma}{2}.$$

Therefore it suffices to show that the function $g(x) = \frac{1+x^\gamma}{x^\gamma \log x} \log \frac{1+x^\gamma}{2}$ is decreasing for $0 < x < 1$ and $\gamma \geq 1$. From inspecting $g'(x)$ we see that the condition that $g'(x) < 0$ is equivalent to

$$-(1 + x^\gamma + \gamma \log x) \log \frac{1 + x^\gamma}{2} + \gamma x^\gamma \log x < 0$$

If $1 + x^\gamma + \gamma \log x \geq 0$ then the condition is satisfied and we are done. Otherwise we can use the fact that $-(1 + x^\gamma + \gamma \log x) > 0$ together with Lemma 7.1 to produce following derivation:

$$- (1 + x^\gamma + \gamma \log x) \log \frac{1 + x^\gamma}{2} + \gamma x^\gamma \log x$$

$$< -(1 + x^\gamma + \log x^\gamma)\frac{x^\gamma - 1}{2} + x^\gamma \log x^\gamma$$

$$= 1 - x^{2\gamma} + (1 + x^\gamma) \log x^\gamma$$

Reapplying Lemma 7.1 we see that $(1 + x^\gamma) \log x^\gamma < (1 + x^\gamma)(x^\gamma - 1) = -(1 - x^{2\gamma})$ completing the proof. $\square$

7.4.4   Stating the result

We will now summarize how the results from the previous subsections yield Theorem 7.1. To find the the distribution over Boolean functions minimizing ρ we first considered the optimal weight assignment in the expression $s(x) = \sum_i w_i \alpha^i$ subject to the constraint that $\sum_i w_i = 1$. Finding an optimal assignment does not guarantee that we have solved the problem, because there may not exist a Boolean function with a given weight assignment, but if one or more Boolean functions that satisfy the optimal assignment exists we will have solved the problem. In Lemma 7.3 we showed that an optimal solution $w_0^*, w_1^*, \ldots w_d^*$ must have $w_0^* = 0$. Therefore the optimal solution can only have non-zero weight on exponents $k \geq 1$. Next, in Lemma 7.4, we argued that if we allow continuous exponents $k \in \mathbb{R}$ with $k \geq 1$ in $s(x)$ then the problem of minimizing ρ becomes the problem of selecting $\gamma \geq 1$ where $s(x) = x^\gamma$. Lemma 7.5 showed that $\rho(\gamma)$ is decreasing, so to minimize ρ we want to set $\gamma = 1$. The conclusion from these optimization problems is that we minimize ρ by setting $w_1^* = 1$. Finally Lemma 7.2 shows that the subset of the Boolean functions with $w_1 = 1$ is exactly the set of dictator functions $f(x) = \pm x_i$. Together with the fact that $w_1^* = 1$ is a unique minimum of ρ in the weight assignment problem we get Theorem 7.1

7.5   Open problems

Orthogonal search.   It appears that the same techniques can be used to show that pairs of functions of the form $f(x) = x_i x_j$, $g(y) = -x_i x_j$ minimize the function

$$\frac{\log(1/\min(p_\alpha, p_{-\alpha}))}{\log(1/\max(p_\beta, p_{-\beta}))}.$$

Extension to negative correlation.   It seems likely that the dictator functions or bit-sampling minimizes ρ for the entire interval $-1 \leq \beta < \alpha \leq 1$. Unfortunately the current proof breaks down in places.

General hash functions.   Showing tight bounds for hash function with an arbitrary range is an interesting open problem. For orthogonal search this is an open problem even in the case of $\rho_{\alpha,0}$. For more information see the symmetric Gaussian problem in [119].

Investigating what the implications of the results in this paper for functions with an arbitrary range through the use of 1-bit hashing is an interesting problem.

Simpler proof. The current proof seems needlessly complicated and inelegant. Perhaps some properties of the ratio of logarithms could be established and a simple proof would follow.

Exact relation to standard LSH. Find out when an LSH for Hamming space in the standard formulation violates the lower bound in this paper.

Asymmetric LSH and set similarity search. Investigate whether the same techniques can be used to show optimal 1-bit schemes for a boolean anti-LSH and for set similarity search.

Part II

Pseudorandomness

Chapter 8

Generating k-independent random variables in constant time

'The old that is strong does not wither'

The generation of pseudorandom elements over finite fields is fundamental to the time, space and randomness complexity of randomized algorithms and data structures. We consider the problem of generating k-independent random values over a finite field $\mathbb{F}$ in a word RAM model equipped with constant time addition and multiplication in $\mathbb{F}$, and present the first nontrivial construction of a generator that outputs each value in *constant time*, not dependent on k. Our generator has period length $|\mathbb{F}|\operatorname{poly}\log k$ and uses $k\operatorname{poly}(\log k)\log|\mathbb{F}|$ bits of space, which is optimal up to a $\operatorname{poly}\log k$ factor. We are able to bypass Siegel's lower bound on the time-space tradeoff for k-independent functions by a restriction to sequential evaluation.

8.1 Introduction

Pseudorandom generators transform a short random seed into a longer output sequence. The output sequence has the property that it is indistinguishable from a truly random sequence by algorithms with limited computational resources. Pseudorandom generators can be classified according to the algorithms (distinguishers) that they are able to fool. An algorithm from a class of algorithms that is fooled by a generator can have its randomness replaced by the output of the generator, while maintaining the performance guarantees from the analysis based on the assumption of full randomness. When truly random bits are costly

to generate or supplying them in advance requires too much space, a pseudorandom generator can reduce the time, space and randomness complexity of an algorithm.

This paper presents an explicit construction of a pseudorandom generator that outputs a *k*-independent sequence of values in *constant time* per value, not dependent on *k*, on a word RAM [85]. The generator works over an arbitrary finite field that allows constant time addition and multiplication over $\mathbb{F}$ on the word RAM.

Previously, the most efficient methods for generating *k*-independent sequences were either based on multipoint evaluation of degree $k - 1$ polynomials, or on direct evaluation of constant time hash functions. Multipoint evaluation has a time complexity of $O(\log^2 k \log \log k)$ field operations per value while hash functions with constant evaluation time use excessive space for non-constant *k* by Siegel's lower bound [152]. We are able to get the best of both worlds: constant time generation and near-optimal seed length and space usage.

Significance.   In the analysis of randomized algorithms and in the hashing literature in particular, *k*-independence has been the dominant framework for limited randomness. Sums of *k*-independent variables have their *j*th moment identical to fully random variables for $j \le k$ which preserves many properties of full randomness. For output length *n*, $\Theta(\log n)$-independence yields Chernoff-Hoeffding bounds [144] and random graph properties [7], while $\Theta(\text{poly} \log n)$-independence suffices to fool AC^0 circuits [29].

Our generator is particularly well suited for randomized algorithms with time complexity $O(n)$ that use a sequence of *k*-independent variables of length *n*, for non-constant *k*. For such algorithms, the generation of *k*-independent variables in constant time by evaluating a hash function over its domain requires space $O(n^\epsilon)$ for some constant $\epsilon > 0$. In contrast, our generator uses space $O(k \, \text{poly} \log k)$ to support constant time generation. Algorithms for randomized load balancing such as the simple process of randomly throwing *n* balls into *n* bins fit the above description and presents an application of our generator. Using the bounds by Schmidt et al. [144, Theorem 2] it is easy to show that $\Theta(\log n / \log \log n)$-independence suffices to obtain a maximal load of any bin of $O(\log n / \log \log n)$ with high probability. This guarantee on the maximal load is asymptotically the same as under full randomness. Using our generator, we can allocate each ball in constant time using

space $O(\log n \operatorname{poly} \log \log n)$ compared to the lower bound of $O(n^\epsilon)$ of hashing-based approaches to generating k-independence. In Section 8.7 we show how our generator improves upon existing solutions to a dynamic load balancing problem.

The generation of pseudorandomness for Monte Carlo experiments presents another application. Limited independence between Monte Carlo experiments can be shown to yield Chernoff-like bounds on the deviation of an estimator from its expected value. Consider a randomized algorithm $\mathcal{A}(Y)$ that takes m random elements from $\mathbb{F}$ encoded as a string Y and returns a value in the interval $[0,1]$. Let $\mu_\mathcal{A} > 0$ denote the expectation of the value returned by $\mathcal{A}(Y)$ under the assumption that Y encodes a truly random input. Define the estimator

$$\hat{\mu}_\mathcal{A} = \frac{1}{t} \sum_{i=1}^{t} \mathcal{A}(Y_i).$$

Due to a result by Schmidt et al. [144, Theorem 5], for every choice of constants $\epsilon, \alpha > 0$, it suffices that $Y_1, \ldots, Y_t$ encodes a sequence of $\Theta(m \log t)$-independent variables over $\mathbb{F}$ to yield the following high probability bound on the deviation of $\hat{\mu}_\mathcal{A}$ from $\mu_\mathcal{A}$.

$$Pr[|\hat{\mu}_\mathcal{A} - \mu_\mathcal{A}| \geq \epsilon \mu_\mathcal{A}] \leq O(t^{-\alpha}).$$

We hope that our generator can be a useful tool to replace heuristic methods for generating pseudorandomness in applications where theoretical guarantees are important. In order to demonstrate the practicality of our techniques, we present experimental results on a variant of our generator in Section 8.8. Our experiments show that k-independent values can be generated nearly as fast as output from heuristic pseudorandom generators, even for large k.

Methods. Our construction is a surprisingly simple combination of bipartite unique neigbor expanders with multipoint polynomial evaluation. The basic, probabilistic construction of our generator proceeds in two steps: First we use multipoint evaluation to fill a table with $\Theta(k)$-independent values from a finite field, using an average of $\operatorname{poly} \log k$ operations per table entry. Next we apply a bipartite unique neighbor expander with constant outdegree and with right side nodes corresponding to entries in the table and a left side that is $\operatorname{poly} \log k$ times larger than the right side. For each node in the left side of the expander

we generate a *k*-independent value by returning the sum of its neighboring table entries. Our main result stated in Theorem 1 uses the same idea, but instead of relying on a single randomly constructed expander graph, we employ a cascade of explicit constant degree expanders and show that this is sufficient for constant time generation.

Relation to the literature. Though the necessary ingredients have been known for around 10 years, we believe that a constant time generator has evaded discovery by residing in a blind spot between the fields of hashing and pseudorandom generators. The construction of constant time *k*-independent *hash functions* has proven to be a difficult task, and a fundamental result by Siegel [152] shows a time-space tradeoff that require hashing-based generators with sequence length n to use $O(n^\epsilon)$ space for some constant $\epsilon > 0$. On the other hand, from the point of view of pseudorandom generators, a generator of *k*-independent variables, for super-constant *k*, can not be used as an efficient method of derandomization: A lower bound by Chor et al. [52] shows that the sample space of such generators must be superpolynomial in their output length. Specifically, a *k*-independent generator that outputs n bits must have a seed length of $\Omega(k \log n)$ bits. Consequently, research shifted towards generators that produce other types of outputs such as biased sequences or almost *k*-independent variables [8, 116, 79].

It is relevant to ask whether there already exist constructions of constant time pseudorandom generators on the word RAM that can be used instead of generators that output *k*-independent variables. For example, Nisan's pseudorandom generator [118] uses constant time to generate a pseudorandom word and has remarkably strong properties: Every algorithm running in SPACE(s) that uses n random words can have its random input replaced by the output of a constant time generator with seed length $O(s \log n)$. The probability that the outcome of the algorithm differs when using pseudorandomness as opposed to statistical randomness is exponentially decreasing in the seed length.

In spite of this strong result, there are many natural applications where the restrictions on Nisan's model means that we cannot use his generator directly to replace the use of a *k*-generator. An example is the analysis that uses a union bound over all subsets of *k* words of a randomly generated structure described by n words. Algorithms shown to be derandomized by Nisan's generator are restricted to one-way access to the output of the generator. Therefore the output of Nisan's

generator can not be used to derandomize an algorithm that tests for the events of the union bound without using excessive space. In this case, *k*-independence can directly replace the use of full randomness without changing the analysis.

8.1.1 Our contribution

We present three improved constructions of *k-generators*, formally defined in Section 8.2, that are able to generate a sequence of *k*-independent values over a finite field $\mathbb{F}$. Our results are stated in a word RAM model equipped with constant time addition and multiplication in $\mathbb{F}$. Our main result is a fully explicit generator:

Theorem 8.1. *For every finite field $\mathbb{F}$ with constant time arithmetic there exists a data structure that for every choice of k is an explicit constant time k-generator with range $\mathbb{F}$, period $|\mathbb{F}|\operatorname{poly}\log k$, and seed length, space usage and initialization time $k\operatorname{poly}\log k$.*

We further investigate how the space usage and seed length may be reduced by employing a probabilistic construction that has a certain probability of error:

Theorem 8.2. *For every finite field $\mathbb{F}$ with constant time arithmetic and every choice of constants $\varepsilon, \delta > 0$ there exists a data structure that for every choice of k is a constant time k-generator with failure probability δ, range $\mathbb{F}$, period $|\mathbb{F}|$, seed length $O(k)$, space usage $O(k\log^{2+\varepsilon} k)$, and initialization time $O(k\operatorname{poly}\log k)$.*

Finally, we improve existing *k*-generators with optimal space complexity:

Theorem 8.3. *For every finite field $\mathbb{F}$ that supports computing the discrete Fourier transform of length k in $O(k\log k)$ operations, there exists a data structure that, for every choice of k and given a primitive element ω, is an explicit $O(\log k)$ time k-generator with range $\mathbb{F}$, period $|\mathbb{F}|$, seed length k, space usage $O(k)$, and initialization time $O(k\log k)$.*

Table 8.1 summarizes our results along with previous methods of generating sequences of *k*-independent values over $\mathbb{F}$. The length of the output is at least $|\mathbb{F}|$ for all the different methods.

Construction	Time	Space	Comment
Polynomials [92, 42]	$O(k)$	$O(k)$	
Multipoint [165]	$O(\log^2 k \log \log k)$	$O(k \log k)$	
Multipoint [27]	$O(\log k \log \log k)$	$O(k)$	Requires ω.
Siegel [152]	$O(1)$	$O(|\mathbb{F}|^\varepsilon)$	Probabilistic.
Theorem 8.1	$O(1)$	$k \operatorname{poly} \log k$	Explicit.
Theorem 8.2	$O(1)$	$O(k \log^{2+\varepsilon} k)$	Probabilistic.
Theorem 8.3	$O(\log k)$	$O(k)$	Requires ω_k, FFT.

Table 8.1: Overview of generators that produce a k-independent sequence over a finite field $\mathbb{F}$. We use ε to denote an arbitrary positive constant and ω and ω_k to denote, respectively, a primitive element and a k-th root of unity of $\mathbb{F}$. The unit for space is the number of elements of $\mathbb{F}$ that need to be stored, i.e., a factor $\log_2 |\mathbb{F}|$ from the number of bits. Probabilistic constructions rely on random generation of objects for which no explicit construction is known, and may fail with some probability.

Overview of paper   In Section 8.2 we define k-generators and related concepts and review results that lead up to our main results. Section 8.3 presents the details of our explicit construction of constant time generators. In Section 8.4 we apply the same techniques with a probabilistic expander construction to obtain generators with improved space and randomness complexity. Section 8.5 presents an algorithm for evaluating a polynomial over all elements of $\mathbb{F}$ that improves existing generators with optimal space. Section 8.6 shows how arithmetic over $\mathbb{F}_p$ can be implemented in constant time on a standard word RAM with integer multiplication and also reviews algorithms and the state of hardware support for $\mathbb{F}_{2^w}$. Section 8.7 applies our generator to improve the time-space tradeoff of previous solutions to a load balancing problem. Section 8.8 presents experimental results on the generation time of different k-generators for a range of values of k.

8.2   Preliminaries

We begin by defining two fundamental concepts:

Definition 8.1. A sequence $(X_1, X_2, \ldots, X_n)$ of n random variables with finite range R is an (n, k)-*sequence* if the variables at every set of k positions in the sequence are independent and uniformly distributed over R.

Definition 8.2. A family of functions $\mathcal{F} \subseteq \{f \mid f \colon U \to R\}$ is *k-independent* if for every set of k distinct inputs $x_1, x_2, \ldots, x_k$ it holds that $f(x_1), f(x_2), \ldots, f(x_k)$ are independent and uniformly distributed over R when f is selected uniformly at random from $\mathcal{F}$. We say that a function f selected uniformly at random from $\mathcal{F}$ is a *k-independent function*.

We now give a formal definition of the generator data structure.

Definition 8.3. A *k-generator* with range R, period n and failure probability δ is a data structure with the following properties:

- It supports an initialization operation that takes a random seed s as input.

- After initialization it supports an `emit()` operation that returns a value from R.

- There exists a set B such that $\Pr[s \in B] \leq \delta$ and conditioned on $s \notin B$ the sequence $(X_1, X_2, \ldots, X_n)$ of values returned by `emit()` is an (n, k)-sequence.

A *k-generator* is *explicit* if the initialization and emit operation has time complexity poly k and the probability of failure is zero. We refer to a *k-generator* as a constant time *k-generator* if the `emit()` operation has time complexity $O(1)$, not dependent on k.

A *k-generator* differs from a data structure for representing a *k-independent* hash function by only allowing sequential access to the underlying (n, k)-sequence. It is this restriction on generators that allows us to obtain a better time-space tradeoff for the problem of generating *k-independent* variables than is possible by using a *k-independent* hash function directly as a generator. We are interested in the following parameters of *k-generators*: seed length, period, probability of failure, space needed by the data structure, the time complexity of the initialization operation and the time complexity of a single `emit()` operation.

Model of computation.   Our results are stated in the word RAM model of computation with word length $w = \Theta(\log |\mathbb{F}|)$ bits. In addition to the standard bit manipulation and integer arithmetic instructions, we also assume the ability to perform arithmetic operations $(+, -, \times)$ over $\mathbb{F}$ in

constant time. In the context of our results that use abelian groups $(A, +)$ we assume that an element of A can be stored in a constant number of words and that addition can be performed in constant time.

Let $\mathbb{F}_q$ denote a field of cardinality $q = p^z$ for p prime and z a positive integer. Constant time arithmetic in $\mathbb{F}_p$ is supported on a standard word RAM with integer multiplication [82]. Section 8.6 presents additional details about the algorithms required to implement finite field arithmetic over $\mathbb{F}_p$ and $\mathbb{F}_{2^w}$ and how they relate to a standard word RAM with integer multiplication.

8.2.1   *k*-independent functions from the literature

We now review the literature on k-independent functions and how they can be used to construct k-generators. We distinguish between a k-independent function $f : U \to R$ and a k-independent hash function by letting the latter refer to a data structure that after initialization supports random access to the (n, k)-sequence defined by evaluating f over U. There exists an extensive literature that focuses on how to construct k-independent hash functions that offer a favorable tradeoff between representation space and evaluation time [69]. We note that a family of k-independent hash functions can be used to construct a k-generator by setting the seed to a random function in the family.

Constant time k-independent hash functions. A fundamental cell probe lower bound by Siegel [152] shows that a data structure to support constant time evaluation of f on every input in U cannot use less than $\Omega(|U|^\epsilon)$ space for some constant $\epsilon > 0$. This bound holds even for amortized constant evaluation time over functions in the family and elements in the domain. From Siegel's lower bound, it is clear that we cannot use k-independent hash functions directly to obtain a constant time k-generator that uses only $O(k \operatorname{poly} \log k)$ words of space.

Known constructions of k-independent hash functions with constant evaluation time are based on expander graphs. Siegel [152] gave a probabilistic construction of a family of k-independent hash functions in the word RAM model based on an iterated product of bipartite expander graphs. Thorup [159] showed that a simple tabulation hash function with high probability yields the type of expander graphs required by Siegel's construction. Unfortunately only randomized constructions of the expanders required by these hash functions is known, introducing a positive probability of error in k-generators based on them.

Polynomials. Here we briefly review the classic construction of k-independent functions based on polynomials over finite fields.

Lemma 8.1 (Joffe [92], Carter and Wegman [42]). *For every choice of finite field $\mathbb{F}$ and every $k \leq |\mathbb{F}|$, let $\mathcal{H}_k \subset \mathbb{F}[X]$ be the family of polynomials of degree at most $k - 1$ over $\mathbb{F}$. $\mathcal{H}_k \subset \{f \mid f \colon \mathbb{F} \to \mathbb{F}\}$ is a family of k-independent functions.*

An advantage of using families of polynomials as hash functions is that they use near optimal randomness, allow any choice of $k \leq |\mathbb{F}|$, and have no probability of failure. It can also be noted that in the case where $k = O(\log |\mathbb{F}|)$ and we are restricted to linear space $O(k)$, polynomial hash functions evaluated using Horner's scheme are optimal k-independent hash functions [101, 152].

Using slightly more space and for sufficiently large k, a data structure by Kedlaya and Umans [96] supports evaluation of a polynomial of degree k over $\mathbb{F}$. The space usage and preprocessing time of their data structure is $k^{1+\epsilon} \log^{1+o(1)} |\mathbb{F}|$ for constant $\epsilon > 0$. After preprocessing a polynomial f, the data structure can evaluate f in an arbitrary point of $\mathbb{F}$ using time $\operatorname{poly}(\log k) \log^{1+o(1)} |\mathbb{F}|$.

Multipoint evaluation. Using algorithms for multipoint evaluation of polynomials we are able to obtain a k-generator with $\operatorname{poly}\log k$ generation time and space usage that is linear in k. Multipoint evaluation of a polynomial $f \in \mathbb{F}[X]$ of degree at most $k - 1$ in k arbitrary points of $\mathbb{F}$ has a time complexity of $O(k \log^2 k \log \log k)$ in the word RAM model that supports field operations [165, Corollary 10.8]. Bostan and Schost [27] mention an algorithm for multipoint evaluation of f over a geometric progression of k elements with running time $O(k \log k \log \log k)$. In order to use this method to construct a k-generator with period $|\mathbb{F}|$ it is necessary to know a primitive element ω of $\mathbb{F}_q$ so we can perform multipoint evaluation over $\mathbb{F}^* = \{\omega^0, \omega^1, \ldots, \omega^{q-2}\}$. Given the prime factorization of $q - 1$ there exists a Las Vegas algorithm for finding ω with expected running time $O(\log^4 q)$ [147, Chapter 11]. In the following lemma we summarize the properties of k-generators based on multipoint evaluation of polynomials over finite fields.

Lemma 8.2 (Gathen and Gerhard [165, Corollary 10.8], Bostan and Schost [27]). *For every finite field $\mathbb{F}$ there exists for every $k \leq |\mathbb{F}|$ and bijection $\pi : [|\mathbb{F}|] \to \mathbb{F}$ an explicit k-generator with period $|\mathbb{F}|$ and seed length k.*

The space required by the generator and the initialization and generation time depends on the choice of π and multipoint evaluation algorithm.

- *For arbitrary choice of π there exists a k-generator with generation time $O(\log^2 k \log \log k)$, intialization time $O(k \log^2 k \log \log k)$ and space usage $O(k \log k)$.*

- *Given a primitive element ω of $\mathbb{F}$ and a bijection $\pi(i) = \omega^i$ there exists a generator with generation time $O(\log k \log \log k)$, initialization time $O(k \log k \log \log k)$ and space usage $O(k)$.*

Space lower bounds. Since randomness can be viewed as a resource like time and space, we are naturally interested in generators that can output long k-independent sequences using as few random bits as possible. Families of k-independent functions $f : U \to R$ with $U = R$ and $k \leq |U|$ will trivially have to use at least $k \log |U|$ random bits — a bound matched by polynomial hash functions. We are often interested in generators with $|U| \gg |R|$, for example if we wish to use a generator for randomized load balancing in the heavily loaded case. A lower bound by Chor et al. [52] shows that even in this case the minimal seed length required for k-independence is $\Omega(k \log |U|)$ for every $|R| \leq |U|$.

8.2.2 Expander graphs

All graphs in this paper are bipartite with cm vertices on the left side, m vertices on the right side and left outdegree d. Graphs are specified by their edge function $\Gamma : [cm] \times [d] \to [m]$ where the notation $[n]$ is used to denote the set $\{0, 1, \ldots, n - 1\}$. Let S be a subset of left side vertices. For convenience we use $\Gamma(S)$ to denote the neighbors of S.

Definition 8.4. The bipartite graph $\Gamma : [cm] \times [d] \to [m]$ is (c, m, d, k)-*unique* (k-*unique*) if for every $S \subseteq [cm]$ with $|S| \leq k$ there exists $y \in \Gamma(S)$ such that y has a unique neighbor in S. An expander graph is *explicit* if it has a deterministic description and Γ is computable in time polynomial in $\log cm + \log d$.

The performance of our generator constructions are directly tied to the parameters of such expanders. In particular, we would like explicit expanders that simultanously have a low outdegree d, are highly unbalanced and are k-unique for k as close to m as possible. A direct

application of a result by Capalbo et al. [39, Theorem 7.1] together with an equivalence relation between different types of expander graphs from Ta-Shma et al. [155, Theorem 8.1] yields explicit constructions of unbalanced unique neighbor expanders.[1]

Lemma 8.3 (Capalbo et al. [39, Theorem 7.1])**.** *For every choice of c and m there exists a (c, m, d, k)-unique expander with $d = \operatorname{poly} \log c$ and $k = \Omega(m/d)$. For constant c the expander is explicit.*

We note the following simple technique for constructing a larger k-unique expander from a smaller k-unique expander.

Lemma 8.4. *Let Γ be a (c, m, d, k)-unique expander with $cm \times m$ adjacency matrix M. For any positive integer b define $\Gamma^{(b)}$ as the bipartite graph with block diagonal adjacency matrix $M^{(b)} = \operatorname{diag}(M, \ldots, M)$ with b blocks in the diagonal. Then $\Gamma^{(b)}$ is a (c, bm, d, k)-unique expander.*

From expanders to independence. By associating each right vertex in a (c, m, d, k)-unique expander with a position in a (m, dk)-sequence over an abelian group $(A, +)$, we can generate a (cm, k)-sequence over A. This approach was pioneered by Siegel and has been used in different constructions of families of k-independent hash functions [152, 159].

Lemma 8.5 (Siegel [152, Lemma 2.6, Corollary 2.11])**.** *Let $\Gamma : [cm] \times [d] \to [m]$ be a k-unique expander and let $h : [m] \to A$ be a dk-independent function with range an abelian group. Let $g : [cm] \to A$ be defined as*

$$g(x) = \sum_{y \in \Gamma(\{x\})} h(y).$$

Then g is a k-independent function.

8.3 Explicit constant time generators

In this section we show how to obtain a constant time k-generator by combining an explicit $\operatorname{poly} k$-generator with a cascading composition of unbalanced unique neighbor expanders. Our technique works by generating a small number of highly independent elements in an abelian

---

[1]We state the results here without the restriction from [39] that c and m are powers of two. We do this to simplify notation and it only affects constant factors in our results.

group and then successively applying constant degree expanders to produce a greater number of less independent elements. We continue this process up until the point where the final number of elements is large enough to match the cost of generating the smaller batch of highly independent elements.

The generator has two components. The first component is an explicit m-generator $g_0 : [n] \to A$ with period n and range an abelian group A. The second component is an explicit sequence $(\Gamma_i)_{i=1}^t$ of unbalanced unique neighbor expanders. The expanders are constructed such that the left side of the ith expander matches the right side of the $(i + 1)$th expander. By Lemma 8.3, for every choice of imbalance c, target independence k and length of the expander sequence t there exists a sequence of expanders with the property that

$$\Gamma_i \text{ is } (c, c^{i-1}m, d, d^{t-i}k)\text{-unique,} \tag{8.1}$$

for $m = O(d^t k)$ and $d = \operatorname{poly} \log c$. For constant c each expander in the sequence is explicit.

We now combine the explicit m-generator g_0 and the sequence of expanders $(\Gamma_i)_{i=1}^t$ to define the k-independent function g_t. Let $b = m/n$ and assume for simplicity that m divides n. For each Γ_i we use the technique from Lemma 8.4 to construct a $(c, c^{i-1}n, d, d^{t-i}k)$-unique expander $\Gamma_i^{(b)}$. Let x_i denote a number in $[c^i n]$ corresponding to a vertex in the right side of $\Gamma_i^{(b)}$. We are now ready to give a recursive definition of $g_i : [c^i n] \to A$.

$$g_i(x_i) = \sum_{x_{i-1} \in \Gamma_i^{(b)}(\{x_i\})} g_{i-1}(x_{i-1}), \quad 1 \le i \le t. \tag{8.2}$$

Lemma 8.6. g_i is $d^{t-i}k$-independent.

Proof. We proceed by induction on i. By definition, $g_0 : [n] \to A$ is $d^t k$-independent. Assume by induction that $g_i : [c^i n] \to A$ is $d^{t-i}k$-independent. By definition $\Gamma_{i+1}^{(b)}$ is a $(c, c^i n, d, d^{t-(i+1)}k)$-unique expander. Applying Lemma 8.5 we have that $g_{i+1} : [c^{i+1}n] \to A$ is $d^{t-(i+1)}k$-independent. $\qquad \square$

We will now show that g_t supports fast sequential evaluation and prove that we can use g_t to construct an explicit constant time k-generator from any explicit m-generator, for an appropriate choice of m. Divide the

domain of each g_i evenly into $b = n/m$ batches of size $c^i m$ corresponding to each block of the adjacency matrix of Γ_i used to construct $\Gamma_i^{(b)}$ and index the batches by $j \in [b]$. In order to evaluate g_{i+1} over batch number j it suffices to know Γ_{i+1} and the values of g_i over batch number j. Fast sequential evaluation of g_t is achieved in the following steps. First we tabulate the sequence of expanders $(\Gamma_i)_{i=1}^t$ such that $\Gamma_i(\{x_i\})$ can be read in d operations. Secondly, to evaluate g_t over batch j, we begin by tabulating the output of g_0 over batch j and then successively apply our tabulated expanders to produce tables for the output of $g_1, g_2, \ldots, g_t$ over batch j.

Given tables for the sequence of expanders and assuming that the generator underlying g_0 has been initialized, we now consider the average number of operations used per output when performing batch-evaluation of g_t. The number of values output is $c^t m$. The cost of emitting m values from g_0 is by definition at most $\text{poly}(m)$. The cost of producing tables for the output of $g_1, g_2, \ldots, g_t$ for the current batch is given by $\sum_{i=1}^t dc^i m = O(dc^t m)$ for $c > 1$. The average number of operations used per output when performing batch-evaluation of g_t is therefore bounded from above by

$$\frac{O(dc^t m) + \text{poly}\, m}{c^t m} = O(d) + \frac{\text{poly}\, m}{c^t}. \tag{8.3}$$

The following lemma states that we can obtain a constant time k-generator from every explicit m-generator by setting $t = O(\log k)$ and choosing c to be an appropriately large constant.

Lemma 8.7. *Let A be an abelian group with constant time addition. Suppose there exists an explicit m-generator with range A, period n and space usage* $\text{poly}\, m$. *Then there exists a positive constant ϵ such that for every $k \le m^\epsilon$ there exists an explicit constant time k-generator with range A, period n, and seed length, space usage and initialization time* $\text{poly}\, k$.

Proof. The sequence of expanders $(\Gamma_i)_{i=1}^t$ with the properties given in (8.1) exists for $m = O(d^t k)$ and $d = \text{poly}\log c$ and is explicit for c constant. By inserting $m = O(d^t k)$ into equation (8.3) it can be seen that the average number of operations is constant for $c = O(1)$ and $t = O(\log k)$ with constants that depend on the parameters of the m-generator. The k-generator is initialized by initializing the m-generator, finding and tabulating the sequence of expanders and producing the first batch

of values, all of which can be done in poly k time and space. After initialization, each call to `emit()` will return a value from the current batch and use a constant number of operations for the task of preparing the next batch of outputs. □

We now show our main theorem about explicit constant time k-generators over finite fields. The construction uses an explicit m-generator based on multipoint evaluation. Combined with the approach of Lemma 8.7 this yields a near-optimal time-space tradeoff for k-generation.

Theorem 8.1 (Repeated). *For every finite field $\mathbb{F}$ with constant time arithmetic there exists a data structure that for every choice of k is an explicit constant time k-generator with range $\mathbb{F}$, period $|\mathbb{F}|$ poly $\log k$, and seed length, space usage and initialization time k poly $\log k$.*

Proof. Fix the choice of finite field $\mathbb{F}$. By Lemma 8.2 there exists an explicit m-generator in $\mathbb{F}$ for $m \leq |\mathbb{F}|$ with period $|\mathbb{F}|$ that uses time $O(m \log^3 m)$ to emit m values. Fix some constant $c > 1$ and let $(\Gamma_i)_{i=1}^t$ denote an explicit sequence of constant degree expanders with the properties given by (8.1). The average number of operations per k-independent value output by g_t when performing batch evaluation is given by

$$\frac{O(dc^t m) + O(m \log^3 m)}{c^t m} = O(d) + \frac{O(\log^3 d^t k)}{c^t}. \tag{8.4}$$

Setting $t = O(\log \log k)$ and following the approach of Lemma 8.7 we obtain a k-generator with the stated properties. □

Based on the discussion in a paper by Capalbo [38] that introduces unbalanced unique neighbor expanders for concrete values of c and d, it appears likely that the constants hidden in Theorem 8.1 for the current best explicit constructions make our explicit generators unsuited for practical use since c is close to 1 when d is reasonably small. The next section explores how randomly generated unique neighbor expanders can be used to show stronger existence results and yield k-generators with tractable constants.

8.4 Constant time generators with optimal seed length

Randomly constructed expanders of the type used in this paper have stronger properties than known explicit constructions, and can be generated with an overwhelming probability of success. There is no known

efficient algorithm for verifying whether a given graph is a unique neighbor expander. Therefore randomly generated expanders cannot be used to replace explicit constructions without some probability of failure.

In this section we apply the probabilistic method to show the existence of k-generators with better performance characteristics than those based on known explicit constructions of expanders. We are able to show the existence of constant time generators with optimal seed length that use $O(k \log^{2+\varepsilon} k)$ words of space for any constant $\varepsilon > 0$. Furthermore, such generators can be constructed for any choice of constant failure probability $\delta > 0$. The generators we consider in this section use only a single expander graph but are otherwise identical to the generators described in Section 8.3. Using a single expander graph suffices for constant time generation because the probabilistic constructions are powerful enough to support an imbalance of $c = \text{poly} \log k$ while maintaining constant degree. This imbalance is enough to amortize the cost of multipoint evaluation in a single expansion step as opposed to the sequence of explicit expanders employed in Theorem 8.1. Our arguments are a straightforward application of the probabilistic method, but we include them for completeness and because we are interested in somewhat nonstandard parameters.

We consider the following randomized construction of a (c, m, d, k)-unique expander Γ. For each vertex x in $[cm]$, we add an edge between x and each distinct node of d nodes selected uniformly at random from $[m]$. By a standard argument, the graph can only fail to be unique neighbor expander if there exists a subset S of left hand side vertices with $|S| \leq k$ such that $|\Gamma(S)| \leq \lfloor d|S|/2 \rfloor$ [152, Lemma 2.8]. In the following

we assume that $kd \le m$.

$$
\begin{aligned}
&\Pr[\Gamma \text{ is not a unique neighbor expander}] \\
&\le \Pr[\exists S \subseteq [cm], |S| \le k : |\Gamma(S)| \le \lfloor d|S|/2 \rfloor] \\
&\le \sum_{\substack{S \subseteq [cm] \\ |S| \le k}} \Pr[|\Gamma(S)| \le \lfloor d|S|/2 \rfloor] \\
&\le \sum_{i=1}^{k} \binom{cm}{i} \binom{m}{\lfloor id/2 \rfloor} \left(\frac{\lfloor id/2 \rfloor}{m} \right)^{id} \\
&\le \sum_{i=1}^{k} \left(\frac{cme}{i} \right)^{i} \left(\frac{me}{id/2} \right)^{id/2} \left(\frac{id/2}{m} \right)^{id} \\
&= \sum_{i=1}^{k} \left(ec(i/m)^{d/2-1}(de/2)^{d/2} \right)^{i}
\end{aligned}
\tag{8.5}
$$

If the expression in the outer parentheses in (8.5) can be bounded from above by $1/2$ for $i = 1, 2, \ldots, k$, then the expander exists. We also note that the randomized expander construction can be performed using dk-independent variables without changing the result in (8.5). Let $\gamma > 1$ be a number that may depend on k and let δ denote an upper bound on the probability that the randomized construction fails. By setting $m = O(dk\gamma)$ we are able to obtain the following expression for the relation between δ, the imbalance c and the left outdegree bound d.

$$
\delta = \frac{cd}{\gamma^{d/2-1}}
\tag{8.6}
$$

Equation (8.6) reveals tradeoffs for the parameters of the randomly constructed k-unique expander graphs. For example, increasing γ makes it possible to make the graph more unbalanced while maintaining the same upper bound on the probability of failure δ. The increased imbalance comes at the cost of an increase in m, the size of the right side of the graph. Similarly it can be seen how increasing d can be used to reduce the probability of error. Setting the parameters to minimize the space occupied by the expander while maintaining constant outdegree and by extension constant generation time, we obtain Theorem 8.2.

Theorem 8.2 (Repeated). *For every finite field $\mathbb{F}$ with constant time arithmetic and every choice of constants $\varepsilon, \delta > 0$ there exists a data structure that for every choice of k is a constant time k-generator with failure probability δ, range $\mathbb{F}$, period $|\mathbb{F}|$, seed length $O(k)$, space usage $O(k \log^{2+\varepsilon} k)$, and initialization time $O(k \operatorname{poly} \log k)$.*

Proof. Let $\tilde{\varepsilon} < \varepsilon$ be a constant and set $\gamma = \log^{\tilde{\varepsilon}} k$. Choosing d to be a sufficiently large constant (dependent on $\tilde{\varepsilon}$), equation (8.6) shows that for every $\delta > 0$ there exists a (c, m, d, k)-unique expander Γ with $c = \Omega(\log^{2+\varepsilon} k)$ and $m = O(k\gamma)$. Using multipoint evaluation, the right side vertices of Γ can be associated with $\Theta(k)$-independent variables over $\mathbb{F}$ using $O(k \log^{2+\varepsilon} k)$ operations. By the properties of Γ and applying Lemma 8.5 we are able to generate batches of k-independent variables of size $\Omega(k \log^{2+\varepsilon} k)$ using $O(k \log^{2+\varepsilon})$ operations. The seed length of $O(k)$ holds by the observation that randomized construction of the expander only requires $O(k)$-independence. The $O(k \operatorname{poly} \log k)$ initialization time is obtained by using multipoint evaluation to construct a table for Γ. $\square$

8.5   Faster multipoint evaluation for k-generators

This section presents an improved generator based directly on multipoint evaluation of a polynomial hash function $h \in \mathcal{H}_k$ over a finite field. For our purpose of generating an (n, k)-sequence from h, we are free to choose the order of elements of $\mathbb{F}$ in which to evaluate h. We present an algorithm for the systematic evaluation of h over disjoint size k subsets of $\mathbb{F}$ using Fast Fourier Transform (FFT) algorithms. Our technique yields a k-generator over $\mathbb{F}$ with generation time $O(\log k)$, and space usage and seed length that is optimal up to constant factors. The algorithm depends upon the structure of $\mathbb{F}$, similarly to other FFT algorithms over finite fields [26].

The nonzero elements of $\mathbb{F}$ form a multiplicative cyclic group $\mathbb{F}^*$ of order $q - 1$. The multiplicative group has a primitive element ω which generates $\mathbb{F}^*$.

$$\mathbb{F}^* = \{\omega^0, \omega^1, \omega^2, \ldots, \omega^{q-2}\}.$$

For k that divides $q - 1$, we can construct a multiplicative subgroup $S^*_{k,0}$ of order k with $\omega_k = \omega^{(q-1)/k}$ as the generating element. $S^*_{k,0}$ contains k distinct elements of $\mathbb{F}$. Define for $j = 0, 1, \ldots, (q-1)/k - 1$,

$$S^*_{k,j} = \omega^j S_{k,0} = \{\omega^j \omega_k^0, \omega^j \omega_k^1, \ldots, \omega^j \omega_k^{k-1}\}.$$

Viewed as subsets of $\mathbb{F}^*$ the sets $S^*_{k,j}$ form an exact cover of $\mathbb{F}^*$. We now consider how to evaluate a degree $k - 1$ polynomial $h(x) \in \mathbb{F}[X]$ in the points of $S^*_{k,j}$. The polynomial takes the form

$$h(x) = a_0 x^0 + a_1 x^1 + \cdots + a_{k-1} x^{k-1}.$$

Rewriting the polynomial evaluation over $S^*_{k,j}$ in matrix notation:

$$
\begin{bmatrix}
h(\omega^j \omega^0_k) \\
h(\omega^j \omega^1_k) \\
h(\omega^j \omega^2_k) \\
\vdots \\
h(\omega^j \omega^{k-1}_k)
\end{bmatrix}
=
\begin{bmatrix}
\omega^{0 \cdot 0}_k & \omega^{0 \cdot 1}_k & \dots & \omega^{0 \cdot (k-1)}_k \\
\omega^{1 \cdot 0}_k & \omega^{1 \cdot 1}_k & \dots & \omega^{1 \cdot (k-1)}_k \\
\omega^{2 \cdot 0}_k & \omega^{2 \cdot 1}_k & \dots & \omega^{2 \cdot (k-1)}_k \\
\vdots & \vdots & & \vdots \\
\omega^{(k-1) \cdot 0}_k & \omega^{(k-1) \cdot 1}_k & \dots & \omega^{(k-1) \cdot (k-1)}_k
\end{bmatrix}
\begin{bmatrix}
\omega^{j \cdot 0} a_0 \\
\omega^{j \cdot 1} a_1 \\
\omega^{j \cdot 2} a_2 \\
\vdots \\
\omega^{j \cdot (k-1)} a_{k-1}
\end{bmatrix}
$$

We assume that the coefficients of h and ω^j are given and consider algorithms for efficient evaluation of the matrix-vector product. The coefficients $\tilde{a}_{j,i} = \omega^{j \cdot i} a_i$ for $i = 0, 1, \dots, k-1$ can be found in $O(k)$ operations and define a polynomial $\tilde{h}_j(x) = \sum_{i=0}^{k-1} \tilde{a}_{i,j} x^i$. Evaluating $\tilde{h}_0(x)$ over $S^*_{k,0}$ corresponds to computing the Discrete Fourier Transform over a finite field.

Theorem 8.3 (Repeated). *For every finite field $\mathbb{F}$ that supports computing the discrete Fourier transform of length k in $O(k \log k)$ operations, there exists a data structure that, for every choice of k and given a primitive element ω, is an explicit $O(\log k)$ time k-generator with range $\mathbb{F}$, period $|\mathbb{F}|$, seed length k, space usage $O(k)$, and initialization time $O(k \log k)$.*

Proof. Evaluation of $\tilde{h}_j(x)$ over $S^*_{k,j}$ takes $O(k \log k)$ operations by assumption. For every batch j starting at $j = 0$, the value of ω^j is stored and used to compute the coefficients of $\tilde{h}_{j+1}(x)$ in $O(k)$ operations. □

We now discuss the validity of the assumption that we are able to compute the DFT over a finite field in $O(k \log k)$ operations. Assume that $k \mid (q-1)$ and that ω_k is known. If k is highly composite there exist Fast Fourier Transforms for computing the DFT in $O(k \log k)$ field operations [74]. If k is not highly composite there exists an algorithm for computing the DFT in $O(kz \log kz)$ operations for fields of cardinality $q = p^z$ in our model of computation [137]. For $q = p^{O(1)}$ this reduces to the desired $O(k \log k)$ operations.

8.6   Finite field arithmetic on the word RAM

Throughout the paper we have used as our model of computation a modified word RAM with constant time arithmetic $(+, -, \times)$ over a

finite field $\mathbb{F}$. In this section we show how our model relates to the more standard *multiplication model* defined as a word RAM with constant time arithmetic $(+, -, \times)$ over the integers $[2^w]$ for w-bit words [85].

Arithmetic over $\mathbb{F}_p$ for prime p is integer arithmetic modulo p. We now argue that arithmetic operations over $\mathbb{F}_p$ can be performed in $O(1)$ operations in the multiplication model. Every integer x can be written on the form $x = qp + r$ for non-negative integers q, r with $r < p$. Assume that x can be represented in a constant number of words. The problem of computing $r = x \bmod p$ can be solved by an integer division and $O(1)$ operations in the multiplication model due to the identity $r = x - \lfloor x/p \rfloor p$. An algorithm by Granlund and Montgomery [82] computes $\lfloor x/p \rfloor$ for any constant p using $O(1)$ operations in the multiplication model which gives the desired result.

Another finite field of interest is $\mathbb{F}_{2^w}$ due to the correspondence between field elements and bit vectors of length w. We will argue that a word RAM model that supports constant time multiplication over $\mathbb{F}_{2^w}$ is not unrealistic considering current hardware. Addition in $\mathbb{F}_{2^w}$ has direct support in standard CPU instruction sets through the XOR operation. A multiplication of two elements x and y in $\mathbb{F}_{2^w}$ can be viewed as a two-step process. First, we perform a carryless multiplication $z = x \cdot y$ of the representation of x and y as polynomials in $F_2[X]$. Second, we use a modular reduction to bring the product $x \cdot y$ back into $\mathbb{F}_{2^w}$, similarly to modular arithmetic over $\mathbb{F}_p$. Recently, hardware manufacturers have included partial support for multiplication in $\mathbb{F}_{2^w}$ with the CLMUL instruction for carryless multiplication [83]. The modular reduction step is performed by dividing $x \cdot y$ by an irreducible polynomial g and returning the remainder. Irreducible polynomials g that can be represented as sparse binary vectors with constant weight results in a constant time algorithm for modular reduction as presented by Gueron and Kounavis [83]. We briefly introduce the computation underlying the algorithm to show that its complexity depends on the number of 1s in the binary representation of g. Let L^w and M^w be functions that return the w least, respectively most, significant bits of their argument as represented in $\mathbb{F}_{2^{2w}}$. The complexity of Gueron and Kounavis' algorithm for modular reduction of $z = x \cdot y$ is determined by the complexity of evaluating the expression

$$L^w(L^w(g) \cdot M^w(M^w(z) \cdot g)). \tag{8.7}$$

Evaluating L^w and M^w is standard bit manipulation. For g of constant weight, the carryless multiplications denoted by $\cdot$ in equation (8.7) can

be implemented as a constant number of bit shifts and XORs. For every $w \leq 10000$ an irreducible trinomial or pentanomial (g of weight at most 5) has been found [146]. Together with the hardware support for convolutions this allows us to implement fast multiplication over fields of practical interest.

8.7   A load balancing application

We next consider how our new generator yields stronger guarantees for load balancing. Our setting is motivated by applications such as splitting a set of tasks of unknown duration among a set of m machines, in order to keep the load as balanced as possible. Once a task is assigned to a machine, it cannot be reassigned, i.e., we do not allow *migration*. For simplicity we consider the *unweighted* case where we strive to keep the *number* of tasks on each machine low, and we assume that m divides $|\mathbb{F}|$ for some field $\mathbb{F}$ with constant time operations on a word RAM. Suppose that each machine has capacity (e.g. memory enough) to handle b tasks at once, and that we are given a sequence of t tasks $T_1, \ldots, T_t$, where we identify each task with its duration (an interval in $\mathbb{R}$). Now let $k = mb$ and suppose that we use our constant time k-generator to determine for each $i = 1, \ldots, t$ which machine should handle T_i. (We emphasize that this is done without knowledge of T_i, and without coordination with the machines.) Compared to using a fully random choice this has the advantage of requiring only $k \operatorname{poly} \log k$ words of random bits, which in turn may make the algorithm faster if random number generation is a bottleneck. Yet, we are able to get essentially the same guarantee on load balancing as in the fully random case. To see this let $L(x) = \{i \mid x \in T_i\}$ be the set of tasks active at time x, and let $L_q(x)$ be the subset of $L(x)$ assigned to machine q using our generator. We have:

Lemma 8.8. *For $\varepsilon > 0$, if $|L(x)|(1 + \varepsilon) < mb$ then $\Pr[\max_q |L_q(x)| > b] < m \exp(-\varepsilon^2 b/3)$.*

Proof. Since $|L(x)| < mb = k$ we have that the assignment of tasks in $L(x)$ to machines is uniformly random and independent. This means that the number of tasks assigned to each machine follows a binomial distribution with mean $b/(1 + \varepsilon)$, and we can apply a Chernoff bound of $\exp(-\varepsilon^2 b/3)$ on the probability that more than b tasks are assigned to a particular machine. A union bound over all m machines yields the result. □

Lemma 8.8 allows us to give a strong guarantee on the probability of exceeding the capacity b of a machine at any time, assuming that the average load is bounded by $b/(1+\varepsilon)$. In particular, let $S \subseteq \mathbb{R}$ be a set of size at most $2t$ such that every workload $L(y)$ is equal to $L(x)$ for some $x \in S$. The existence of S is guaranteed since the t tasks are intervals, and they have at most $2t$ end points. This means that

$$\sup_{x \in \mathbb{R}} \max_q |L_q(x)| = \max_{x \in S} \max_q |L_q(x)|,$$

so a union bound over $x \in S$ gives

$$\Pr[\sup_{x \in \mathbb{R}} \max_q |L_q(x)| > b] < 2tm \exp(-\varepsilon^2 b/3) .$$

For constant ε and whenever $b = \omega(\log k)$ and $tm = 2^{o(b)}$ we get an error probability that is exponentially small in b. Such a strong error guarantee can not be achieved with known constant time hashing methods [152, 124, 72, 159] in reasonable space, since they all have an error probability that decreases polynomially with space usage. Even if explicit constructions for the expanders needed in Siegel's hash functions were found, the resulting space usage would be polynomially higher than with our k-generator.

8.8 Experiments

This section contains experimental results of an implementation of a k-generator over $\mathbb{F}_{2^{64}}$. There are two main components to the generator: an algorithm for filling a table of size m with dk-independent variables and a bipartite unbalanced expander graph.

For the first component, we use an implementation of Gao-Mateer's additive FFT [77, Algorithm 2.]. Utilizing the Gao-Mateer algorithm we can generate a batch of k elements of an $(|\mathbb{F}|, k)$-sequence using space $O(k)$ and $O(k \log^2 k)$ operations on a word RAM that supports arithmetic over $\mathbb{F}$. The additive complexity of the FFT algorithm is $O(k \log^2 k)$ while the multiplicative complexity is $O(k \log k)$. Addition in $\mathbb{F}_{2^{64}}$ is implemented as an XOR-operation on 64-bit words. Multiplication is implemented using the PCLMUL instruction along with the techniques for modular reduction by Gueron et al. [83] outlined in Section 8.6.

For the second component we introduce a slightly different type of expander graphs that only work in the special case of fields of characteristic two. Let $\mathbb{F}_{2^w}$ be a field of characteristic two and let M be a

$cm \times m$ adjacency matrix of a graph Γ where each entry of M is viewed as an element of $\mathbb{F}_{2^w}$. By a similar argument to the one used in Lemma 8.5 the linear system Mx defines a (cm, k)-sequence if x is a vector of dk-independent variables over $\mathbb{F}_{2^w}$ and M has row rank at least k. We consider randomized constructions of M over $\mathbb{F}_2$ with at most d 1s in each row and row rank at least k. It is easy to see that a matrix M over $\mathbb{F}_2$ with these properties also defines a matrix with the same properties over $\mathbb{F}_{2^w}$. Since k-uniqueness of Γ implies that M has row rank k, but not the other way around, we are able to obtain better performance characteristics of generators over $\mathbb{F}_{2^w}$ by focusing on randomized constructions of M.

The matrix M is constructed in the following way. Independently, for each $i \in [cm]$ sample d integers uniformly with replacement from $[m]$ and define the ith row of M as the vector constructed by taking the zero vector and adding 1s in the d positions sampled for row i. Observe that if M does not have row rank at least k then some non-empty subset of at most k rows of M sum to the zero vector. In order for a non-empty set of vectors over $\mathbb{F}_2^m$ to sum to the zero vector, the bit-parity must be even in each of the m positions of the sum. The sum of any i rows of M corresponds to a balls and bins process that distributes id balls into m bins, independently and uniformly at random. Let id be an even number. Then there are $(id - 1)!!$ ways of ordering the balls into pairs and the probability that the outcome is equal to any particular pairing is $(1/m)^{id/2}$. This yields the following upper bound on the probability that a subset of i rows sums to zero:

$$\beta_{pair}(i, d, m) = (id - 1)!! \left(\frac{1}{m} \right)^{id/2}. \tag{8.8}$$

A comparison between this bound and the bound for k-uniqueness from equation (8.5) shows that, for each term in the sum, the multiplicative factor applied to the binomial coefficient $\binom{cm}{i}$ is exponentially smaller in id for the bound in (8.8).

The pair-based approach which yields the bound β_{pair} overestimates the probability of failure on subsets of size i, increasingly as id grows large compared to m. We therefore introduce a different bound based on the Poisson approximation to the binomial distribution: the number of balls in each of the m positions can approximately be modelled as independent Poisson distributed variables [113, Ch. 5.4]. The probability

that that the parity is even in each of the m positions in a sum of i rows is bounded by

$$\beta_{poisson}(i,d,m) = e\sqrt{id}\left(\frac{1+e^{-2\frac{id}{m}}}{2}\right)^m, \tag{8.9}$$

where we use the same approach as Mitzenmacher et al. [112]. For any given subset of rows of M, we are free to choose between the two bounds. The probability that a randomly constructed matrix M fails to have rank at least k can be bounded from above using a union bound over subsets of rows of M.

$$\delta \leq \sum_{i=1}^{k}\binom{cm}{i}\min(\beta_{pair}(i,d,m),\beta_{poisson}(i,d,m)). \tag{8.10}$$

We now consider the generation time of our implementation. Let FFT_{dk} denote the time taken by the FFT algorithm to generate a dk-independent value and let $RA_{d,m}$ denote the time it takes to perform d random accesses in a table of size m. The time taken to generate a value by the implementation of our generator is then given by

$$T = \frac{FFT_{dk}}{c} + RA_{d,m}. \tag{8.11}$$

In our experiments, the choice of parameters for the expander graphs were based on a search for the fastest generation time over every combination of imbalance $c \in \{16, 32, 64\}$ and outdegree $d \in \{4, 8, 16\}$. Given choices of d, c and independence k, the size of the right side of the expander m was increased until existence could be guaranteed by the bound in (8.10). The generator in the experiments had the restriction that $m \leq 2^{26}$ and we have measured $RA_{d,m}$ assuming that the expander is read sequentially from RAM. The experiments were run on a machine with an Intel Core i5-4570 processor with 6MB cache and 8GB of RAM.

Table 8.2 shows the generation time in nanoseconds per 64-bit output using Horner's scheme, Gao-Mateer's FFT and the implementation of our generator (FFT + Expander). For the implementation of the generator, we also show the parameters of the randomly generated expander that yielded the fastest generation time among expanders in the search space.

The generation time for Horner's scheme is approximately linear in k and logarithmic in k for the FFT, as predicted by theory. The FFT is faster than using Horner's scheme already at $k = 64$ and orders of

magnitude faster for large *k*. Using our implementation of Gao-Mateer's FFT algorithm we are able to evaluate a polynomial of degree $2^{20} - 1$ in 2^{20} points in less than a second. The same task takes over an hour when using Horner's rule, even with both algorithms using the same underlying implementation of algebraic operations in the field.

For small values of *k*, our generator is an order of magnitude faster than the FFT and comes close to the performance of the 64-bit C++11 implementation of the popular Mersenne Twister. Our generator uses 25 nanoseconds to output a 1024-independent value. This is equivalent to an output of over 300MB/s. The Mersenne Twister uses around 4 nanoseconds to generate a 64-bit value.

In practice, the memory hierarchy appears to be the primary obstacle to maintaining a constant generation time as *k* increases. Our generator reads the expander graphs sequentially and performs random lookups into the table of *dk*-independent values. As *k* grows large, the table can no longer fit into cache and for large imbalance *c*, the expander can no longer be stored in main memory. Searching a wider range of expander parameters could easily yield a faster generation time, potentially at the cost of a larger imbalance *c* or higher probability of failure *δ*.

Acknowledgment

k	Horner ns	FFT ns	c	m	d	δ	ns
2^5	177	243	64	2^{13}	8	10^{-7}	15
2^6	361	294	64	2^{14}	8	10^{-8}	16
2^7	730	338	64	2^{15}	8	10^{-9}	19
2^8	1470	375	64	2^{16}	8	10^{-10}	23
2^9	2950	412	64	2^{17}	8	10^{-11}	24
2^{10}	5902	449	64	2^{18}	8	10^{-12}	25
2^{11}	11808	487	32	2^{18}	8	10^{-12}	35
2^{12}	23627	523	64	2^{18}	16	10^{-29}	43
2^{13}	47183	561	32	2^{18}	16	10^{-29}	54
2^{14}	94429	599	64	2^{22}	8	10^{-15}	68
2^{15}	188258	638	64	2^{23}	8	10^{-16}	69
2^{16}	376143	678	64	2^{24}	8	10^{-17}	77
2^{17}	751781	719	64	2^{25}	8	10^{-18}	85
2^{18}	1505016	765	64	2^{26}	8	10^{-19}	93
2^{19}	3015969	808	32	2^{26}	8	10^{-19}	110
2^{20}	6082313	864	64	2^{26}	16	10^{-46}	175

Table 8.2: Generation time in nanoseconds per 64-bit value using Horner's scheme, Gao-Mateer's FFT and an implementation of our constant-time generator

Near-optimal k-independent hashing

'Not all those who wander are lost'

We consider the following fundamental problems:

- Constructing k-independent hash functions with a space-time trade-off close to Siegel's cell-probe lower bound (SICOMP 2004).

- Constructing representations of unbalanced expander graphs having small size and allowing fast computation of the neighbor function.

It is not hard to show that these problems are intimately connected in the sense that a good solution to one of them leads to a good solution to the other one. In this paper we exploit this connection to present efficient, recursive constructions of k-independent hash functions (and hence expanders with a small representation). While the previously most efficient construction (Thorup, FOCS 2013) needed time quasipolynomial in Siegel's lower bound, our time bound is just a logarithmic factor from the lower bound.

9.1 Introduction

The problem of designing explicit unbalanced expander graphs with near-optimal parameters is of major importance in theoretical computer science. In this paper we consider bipartite graphs with edge set $E \subset U \times V$ where $|U| \gg |V|$. Vertices in U have degree d and expansion is desired for subsets $S \subset U$ with $|S| \leq k$ for some parameter k. Such expanders have numerous applications (e.g. hashing [152], routing [34],

sparse recovery [90], membership [37]), yet coming up with explicit constructions that have close to optimal parameters has proved elusive. At the same time it is easy to show that choosing E at random will give a graph with essentially optimal parameters. This means that we can efficiently and with a low probability of error produce a description of an optimal unbalanced expander that takes space proportional to $|U|$. Storing a complete description is excessive for most applications that, provided access to an explicit construction, would use space proportional to $|V|$. On the other hand, explicit constructions can be represented using constant space, but the current best explicit constructions have parameters d and $|V|$ that are polynomial in the optimal parameters of the probabilistic constructions [84]. Furthermore, existing explicit constructions have primarily aimed at optimizing the parameters of the expander, with the evaluation time of the neighbor function being of secondary interest, as long as it can be bounded by $\mathrm{poly}\log u$. This evaluation time is excessive in applications that, provided access to the neighbor function of an optimal expander, would use time proportional to d, where d is typically constant or at most logarithmic in $|U|$.

In this paper we focus on optimizing the parameters of the expander while minimizing the space usage of the representation and the evaluation time of the neighbor function. We present randomized constructions of unbalanced expanders in the standard word RAM model. Our constructions have near-optimal parameters, use space close to $|V|$, and support computing the d neighbors of a vertex in time close to d.

Hash functions and expander graphs.   There is a close connection between k-independent hash functions and expanders. A k-independent function with appropriate parameters will, with some probability of failure, represent the neighbor function of a graph that expands on subsets of size k. We refer to this as going from independence to expansion, and the fact follows from the standard union bound analysis of probabilistic constructions of expanders. Going in the other direction, from expansion to independence, was first used by Siegel [152] as a technique for showing the existence of k-independent hash functions with evaluation time that does not depend on k. We follow in Siegel's footsteps and a long line of work (see e.g. [69] for an overview) that focuses on the space-time tradeoff of k-independent hash functions over a universe of size $u = |U|$.

Ideally, we would like to construct a data structure in the word RAM model that takes as input parameters u, k, and t, and returns

a k-independent hash function over U. The hash function should use space $k(u/k)^{1/t}$ and have evaluation time $O(t)$, matching up to constant factors the space-time tradeoff of Siegel's cell probe lower bound for k-independent hashing [152]. We present the first construction that comes close to matching the space-time tradeoff of the cell probe lower bound.

Method. Our work is inspired by Siegel's graph powering approach [152] and by recent advances in tabulation hashing [159], showing that it is possible to efficiently describe expanders in space much smaller than u. Our main insight is that it is possible to make simple, recursive expander constructions by alternating between strong unbalanced expanders and highly random hash functions. Similarly to previous work, we follow the procedure of letting a k-independent function represent a bipartite graph Γ that expands on subsets of size k. We then apply a graph product to Γ in order to increase the size of the universe covered by the graph while retaining expander properties. At each step of the recursion we return to k-independence by combining the graph product with a table of random bits, leaving us with a new k-independent function that covers a larger universe. By combining the technique of alternating between expansion and independence with a new and more efficient graph product, we can improve upon existing randomized constructions of unbalanced expanders.

9.1.1 Our contribution

Table 9.1 compares previous upper and lower bounds on k-independent hashing with our results, as presented in Corollaries 1, 2, and 3. As can be seen, most results present a trade-off between time and space controlled by a parameter t. Tight lower and upper bounds have been known only in the cell probe model, but our new construction nearly matches the cell probe lower bound by Siegel [152].

The time bound for the construction using explicit expanders [84] uses the degree of the expander as a conservative lower bound, based on the possibility that the neighbor function in their construction can be evaluated in constant time in the word RAM model. The time bound that follows directly from their work is $\mathrm{poly}\log u$. While the constant factors in the exponent of the space usage of [152, 159] have likely not been optimized, their techniques do not seem to be able to yield space close to the cell probe lower bound.

Table 9.1: Space-time tradeoffs for k-independent hash functions

Reference	Space	Time
Polynomials [92, 42]	k	$O(k)$
Preprocessed polynomials [96]	$k^{1+\varepsilon}(\log u)^{1+o(1)}$	$(\text{poly} \log k)(\log u)^{1+o(1)}$
Expanders [84] + [152]	$k^{1+\varepsilon}d^2$	$d = O(\log(u)\log(k))^{1+1/\varepsilon}$
Expander powering [152]	$k^{(1-\varepsilon)t}u^\varepsilon + u^{1/t}$	$O(1/\varepsilon)^t$
Double tabulation [159]	$k^{5t} + u^{1/t}$	$O(t)$
Recursive tabulation [159]	$\text{poly}\, k + u^{1/t}$	$O(t^{\log t})$
Corollary 9.1	$ku^{1/t}t^3$	$O(t^2 + t^3\log(k)/\log(u))$
Corollary 9.2	$k^2u^{1/t}t^2$	$O(t\log t + t^2\log(k)/\log(u))$
Corollary 9.3	$ku^{1/t}t$	$O(t\log t)$
Cell probe lower bound [152]	$k(u/k)^{1/t}$	$t < k$ probes
Cell probe upper bound [152]	$k(u/k)^{1/t}t$	$O(t)$ probes

Table notes: Space-time tradeoffs for k-independent hash functions from a domain of size u, with the trade-off controlled by a parameter t. Time bounds in the last two rows are number of cell probes, and remaining rows refer to the word RAM model with word size $\Theta(\log u)$. Leading constants in the space bounds are omitted. We use t to denote an arbitrary positive integer parameter that controls the trade-off, and We use ε to denote an arbitrary positive constant. *Corollary 3 relies on the assumption $k = u^{O(1/t)}$.

As can be seen our construction polynomially improves either space or time compared to each of the previously best trade-offs. We also find our construction easier to describe and analyze than the results of [84, 96, 159], with simplicity comparable to that of Siegel's influential paper [152].

Like all other randomized constructions our data structures come with an error probability, but this error probability is *universal* in the sense that if the construction works then it provides independent hash values on *every subset* of at most k elements from U. This is in contrast to other known constructions [72, 124] that give independence with high probability on each *particular* set of at most k elements, but will fail almost surely if independence for a superpolynomial number of subsets is needed.

Applications.   Efficient constructions of highly random functions is of fundamental interest with many applications in computer science. A k-independent function can, without changing the analysis, replace a

fully random function in applications that only rely on k-subsets of inputs mapping to random values. We can therefore view k-independent functions as space and randomness efficient alternatives to fully random functions, capable of providing compact representations of complex structures such as expander graphs over very large domains. Apart from the construction of expander graphs with a small description, as an example application, k-independent functions with a universal error probability can be used to construct "real-time" dictionaries that are able to handle extremely long (in expectation) sequences of insertion and deletion operations in constant time per operation before failing.

Let $\tau > 1$ be a constant parameter. We use a k-independent hash function with $k = w^{O(\tau)}$ to split a set of n machine words of w bits into $O(n)$ subsets such that each subset has size at most k, with probability at least $1 - 2^{-w^\tau}$. Handling each subset with Thorup's recent construction of dictionaries for sets of size $w^{O(\tau)}$ using time $O(\tau)$ per operation [138] we get a dynamic dictionary in which, with high probability, every operation in a sequence of length $\ell < 2^{O(w^\tau)}$ takes constant time. In comparison the hash functions of [70, 72, 124] can only guarantee that sequences of length $\ell < \text{poly}(n)$ operations, where $n < 2^w$, succeed with high probability. The splitting hash function needs space $u^{\Omega(1)}$, which might exceed the space usage of an individual dictionary, but this can be seen as a shared resource that is used for many dictionaries (in which case we bound the total number of operations before failure).

9.2   Background and overview

In the analysis of randomized algorithms we often assume access to a fully random function of the form $f : [u] \to [r]$ where $[n]$ denotes the set $\{0, 1, \ldots, n - 1\}$. To represent such a function we need a table with u entries of $\log r$ bits. This is impractical in applications such as hashing based dictionaries where we typically have that $u \gg r$ and the goal is to use space $O(r)$ to store r elements of $[u]$. Fortunately, the analysis that establishes the performance guarantees of a randomized algorithm can often be modified to work even in the case where the function f has weaker randomness properties.

One such concept of limited randomness is k-independence, first introduced to computer science in the 1970s through the work of Carter and Wegman on universal hashing [40]. A family of functions from $[u]$ to $[r]$ is k-independent if, for every subset of $[u]$ of cardinality at most

k, the output of a random function from the family evaluated on the subset is independent and uniformly distributed in $[r]$. Trivially, the family of all functions from $[u]$ to $[r]$ is k-independent, but representing a random function from this family uses too much space. It was shown in [92] that for every finite field $\mathbb{F}$ the family of functions that consist of all polynomials over $\mathbb{F}$ of degree at most $k-1$ is k-independent. A function from this family can be represented using near-optimal space [52] by storing the k coefficients of the polynomial. The mapping defined by a function f from a k-independent polynomial family over $\mathbb{F} = \{x_1, x_2, \ldots, x_u\}$ takes the form

$$
\begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_u) \end{bmatrix} = \begin{bmatrix} x_1^0 & x_1^1 & \cdots & x_1^{k-1} \\ x_2^0 & x_2^1 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \vdots & \vdots \\ x_u^0 & x_u^1 & \cdots & x_u^{k-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{k-1} \end{bmatrix}. \tag{9.1}
$$

The k-independence of the polynomial family follows from properties of the Vandermonde matrix: every subset of k rows is linearly independent. The problem with this construction is that the Vandermonde matrix is dense, resulting in an evaluation time of $\Omega(k)$ if we simply store the coefficients of the polynomial. The lower bounds by Siegel [152], and later Larsen [101], as presented in Table 1, show that a data structure for evaluating a polynomial of degree $k-1$ using time $t < k$ must use space at least $k(u/k)^{1/t}$. The data structure of [96] presents a step in this direction, but is still far from the lower bound for k-independent functions.

The quest for k-independent families of functions with evaluation time $t < k$ can be viewed as attempts to construct compact representations of sparse matrices that fill the same role as the Vandermonde matrix. We are interested in compact representations that support fast computation of the sparse row associated with an element $x \in [u]$. An example of a sparse matrix with these properties is the adjacency matrix of a bipartite expander graph with sufficiently strong expansion properties. For the purposes of constructing k-independent hash functions we are primarily interested in expanders that are highly unbalanced.

Expander hashing.   Prior constructions of fast and highly random hash functions has followed Siegel's approach of combining expander graphs with tables of random words. If Γ is a k-unique expander graph (see Definition 9.1) then we can construct a k-independent function by composing

it with a simple tabulation function h. This approach would yield optimal k-independent hash functions if we had access to explicit expanders with optimal parameters that could be evaluated in time proportional to the left outdegree. Unfortunately, no explicit construction of a k-unique expander with optimal parameters is known.

Siegel [152] addresses this problem by storing a smaller randomly generated k-unique expander, say, one that covers a universe of size $u^{1/t}$. By the k-independent hashing lower bound, if an expander with $|U| = u^{1/t}$ has degree d, then in order for it to be k-unique it must have a right hand side of size $|V| \geq k(u^{1/t}/k)^{1/d}$. To give a space efficient construction of a k-unique expander that covers a universe of size u, Siegel repeatedly applies the Cartesian product to the graph. Applying the Cartesian product t times to a k-unique expander results in a graph that remains k-unique but with the left degree and size of the left and right vertex sets raised to the power t. Using space $u^{1/t}$ to store an expander with degree t, it follows from the lower bound that the expander resulting from repeatedly applying the Cartesian product must have

$$|V'| \geq (k(u^{1/t}/k)^{1/d})^t = k^{(1-1/d)t}u^{1/d}.$$

Setting $d = 1/\varepsilon$, the randomly generated k-unique expander that forms the basis of the construction has degree $O(1/\varepsilon)$, leading to the expression in Table 1. Since we need to store $|V'|$ random words in a table in order to create a k-independent hash function, Siegel's graph powering approach offers a space-time tradeoff that is far from the lower bound from our perspective where both u, k, and t are parameters to the hash function.

Thorup [159] shows that, for the right choice of parameters, a simple tabulation hash function is likely to form a compact representation of a k-unique expander. A simple tabulation function takes a string $x = (x_1, x_2, \ldots, x_c)$ of c characters from some input alphabet $\langle n \rangle = \{0,1\}^n$, and returns a string of d characters from some output alphabet $\langle m \rangle = \{0,1\}^m$. The simple tabulation function $h : \langle n \rangle^c \to \langle m \rangle^d$ is evaluated by taking the exclusive-or of c table-lookups

$$h(x) = h_1(x_1) \oplus h_2(x_2) \oplus \cdots \oplus h_c(x_c)$$

where $h_i : \langle n \rangle \to \langle m \rangle^d$ is a random function. The advantage of a simple tabulation function compared to a fully random function is that we only need to store the random character tables $h_1, h_2, \ldots, h_c$. Thorup is able to show that for $d \geq 6c$ a simple tabulation function is k-unique with

a low probability of failure when $k \leq (2^m)^{1/5c}$. Setting $n = m$ and composing the k-unique expander resulting from a single application of simple tabulation with another simple tabulation function, Thorup first constructs a hash function with space usage $u^{1/c}$, independence $u^{\Omega(1/c^2)}$, and evaluation time $O(c)$. He then presents a second trade-off with space $u^{1/c}$, independence $u^{\Omega(1/c)}$, and time $O(c^{\log c})$ that comes from applying simple tabulation recursively to the output of a simple tabulation function. Similar to Siegel's upper bound, the space usage of Thorup's upper bounds with respect to k is much larger than the lower bound as can be seen from Table 9.1 where the space-time tradeoff of his results have been parameterized in terms of the independence k.[1]

Explicit constructions. The literature on explicit constructions has mostly focused on optimizing the parameters of the expander, with the evaluation time of the neighbor function being of secondary interest, as long as it is bounded by poly $\log u$. As can be seen from Siegel's cell probe lower and upper bounds, optimal constructions of k-independent hash functions have evaluation time in the range $t = 1$ to $t = \log u$. Therefore, an explicit construction, even if we had one with optimal parameters, would without further guarantees on the running time not be enough to solve our problem of constructing efficient expanders. Here we briefly review the construction given by Guruswami et al. [84]. It is, to our knowledge, currently the best explicit construction of unbalanced bipartite expanders in terms of the parameters of the graph. Their construction and its analysis is, similarly to the polynomial hash function in equation (9.1), algebraic in nature and inspired by techniques from coding theory, in particular Parvaresh-Vardy codes and related list-decoding algorithms [134]. In their construction, a vertex x is identified with its Reed-Solomon message polynomial over a finite field $\mathbb{F}$. The ith neighbor of x is found by taking a sequence of powers of the message polynomial over an extension field, evaluating each of the resulting polynomials in the ith element of $\mathbb{F}$, and concatenating the output. In contrast, the constructions presented in this paper only use the subset of standard word RAM instructions that can be implemented in AC^0. In Table 1 we have assumed that we can evaluate their neighbor function in constant time as a conservative lower bound on the performance of their construction

---

[1] It should be noted that Thorup's analysis is not tuned to optimize the polynomial dependence on k, and that he gives stronger concrete bounds for some realistic parameter settings.

in the word RAM model. Other highly unbalanced explicit constructions given in [39, 155] offer a tradeoff where either one of d or $|V|$ is quasipolynomial in the lower bound. In comparison, the construction by Guruswami et al. is polynomial in both of these parameters.

9.3 Our constructions

In this section we present three randomized constructions of efficient expanders in the word RAM model. Each construction offers a different tradeoff between space, time, and the probability of failure. We present our constructions as data structures, with the randomness generated by the model during an initialization phase. The initialization time of our data structures is always bounded by their space usage, and to simplify the exposition we therefore only state the latter. Alternatively, our constructions could be viewed directly as randomized algorithms, taking as input a list of parameters, a random seed, and a vertex $x \in [u]$ and returning the list of neighbors of x. The hashing corollaries presented in Table 1 follow directly from our three main theorems using Siegel's expander hashing technique.

9.3.1 Model of computation

The algorithms presented in this paper are analyzed in the standard word RAM model with word size w as defined by Hagerup [85], modeling what can be implemented in a standard programming language like C [98]. In order to show how our algorithms benefit from word-level parallelism we use w as a parameter in the analysis. To simplify the exposition we impose the natural restriction that, for a given choice of parameters to a data structure, the word size is large enough to address the space used by the data structure. In other words, our results are stated with w as an unrestricted parameter, but are only valid when we actually have random access in constant time.

The data structures we present require access to a source of randomness in order to initialize the character tables of simple tabulation functions. To accomodate this we augment the model with an instruction that uses constant time to generate a uniformly random and independent integer in $[r]$ where $r \leq 2^w$. We note that our constructions use only the subset of arithmetic instructions required for evaluating a simple tabulation function, i.e, standard bit manipulation instructions, integer

addition, and subtraction. Our results therefore hold in a version of the word RAM model that only uses instructions that can be implemented in AC^0, known in the literature as the restricted model [85] or the Practical RAM [109].

9.3.2 Notation and definitions

Let $\langle n \rangle = \{0,1\}^n$ denote the alphabet of n-bit strings, and let $x = (x_1, x_2, \ldots, x_c) \in \langle n \rangle^c$ denote a string of n-bit characters of length c. We define a concatenation operator $\|$ that takes as input two characters $x \in \langle n \rangle$ and $y \in \langle m \rangle$, and concatenates them to form $x \| y \in \langle n + m \rangle$. The concatenation operator can also be applied to strings of equal length where it performs component-wise concatenation. Given strings $x \in \langle n \rangle^c$ and $y \in \langle m \rangle^c$ the concatenation $x \| y$ is an element of $\langle n + m \rangle^c$ with the ith component of $x \| y$ defined by $(x \| y)_i = x_i \| y_i$. We also define a prefix operator. Given $x \in \langle n \rangle$ and a positive integer m, in the case where $m \leq n$ we use $x[m] \in \langle m \rangle$ to denote the m-bit prefix of x. In the case where $m > n$ we pad the prefix such that $x[m] \in \langle m \rangle$ denotes $x[n] \| 0^{m-n}$ where 0^{m-n} is the string of $m - n$ bits all set to 0.

We will present word RAM data structures that represent functions of the form $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$. The function Γ defines a d-regular bipartite graph with input set $\langle n \rangle^c$ and output set $\{1, 2, \ldots, d\} \times \langle m \rangle$. For $S \subseteq \langle n \rangle^c$ we overload Γ and define $\Gamma(S) = \{(i, \Gamma(x)_i) \mid x \in S\}$, i.e., $\Gamma(S)$ is the set of outputs of S.

We are interested in constructing functions where every subset S of inputs of size at most k contains an input that has many unique neighbors, formally:

Definition 9.1. Let $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$ be a function satisfying the following property:

$$\forall S \subseteq \langle n \rangle^c, |S| \leq k, \exists x \in S : |\Gamma(\{x\}) \backslash \Gamma(S \backslash \{x\})| > l.$$

Then, for $l > 0$ we say that Γ is k-unique. If further $l > d/2$ we say that Γ is k-majority-unique.

For completeness we define the concept of k-independence:

Definition 9.2. Let k be a positive integer and let $\mathcal{F}$ be a family of functions from U to R. We say that $\mathcal{F}$ is a k-independent family of functions if,

for every choice of k distinct keys $x_1, \ldots, x_k$ and arbitrary values $y_1, \ldots, y_k$, then, for f selected uniformly at random from $\mathcal{F}$ we have that

$$\Pr[f(x_1) = y_1 \wedge f(x_2) = y_2 \wedge \cdots \wedge f(x_k) = y_k] = |R|^{-k}.$$

We say that f is k-independent when it is selected uniformly at random from a family of k-independent functions.

Simple tabulation functions are an important tool in our constructions. Our data structures can be made to consist entirely of simple tabulation functions and our evaluation algorithms can be viewed as a sequence of adaptive calls to this collection of simple tabulation functions.

Definition 9.3. Let $(R, \oplus)$ denote an abelian group. A *simple tabulation function* $h : \langle n \rangle^c \to R$ with k-independent character tables is defined by

$$h(x) = \bigoplus_{i=1}^{c} h_i(x_i)$$

where each *character table* $h_i : \langle n \rangle \to R$ is a k-independent function.

In this paper we consider simple tabulation functions with character tables that operate either on bit strings under the exclusive-or operation, $R = (\langle m \rangle, \oplus)$, or on sets of non-negative integers modulo some integer r, $R = ([r], +)$. For our constructions of k-independent functions we will generally assume that $r = u^{O(1)}$.

9.3.3   From k-uniqueness to k-independence

In his seminal paper Siegel [152] showed how a k-unique function can be combined with a table of random elements in order to define a k-independent family of functions. In his paper on the expansion properties of tabulation hash functions, Thorup [159, Lemma 2] used a slight variation of Siegel's technique that makes use of the position-sensitive structure of the bipartite graph defined by $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$. This is the version we state here.

Lemma 9.1 (Siegel [152], Thorup [159])**.** *Let* $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$ *be k-unique and let* $h : \langle m \rangle^d \to R$ *be a simple tabulation function with k-independent character tables. Then $h \circ \Gamma$ defines a family of k-independent functions. We sample a function from the family by sampling the character tables of h.*

9.3.4   From k-independence to k-uniqueness

A *k*-independent function has the same properties as a fully random function when considering *k*-subsets of inputs. We can therefore use the standard analysis of randomized constructions of bipartite expanders to show that, for the right choice of parameters, a *k*-independent function is likely to be *k*-unique. For completeness we provide an analysis here. In our exposition it will be convenient parameterize the *k*-uniqueness or *k*-majority-uniqueness of our constructions in terms of a positive integer κ such that $k = 2^{\kappa}$.

Lemma 9.2. *For every choice of positive integers c, n, κ let $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$ be a 2^{κ}-independent function. Then,*

- *for $m \geq n + \kappa + 1$ and $d \geq 4c$ we have that Γ is 2^{κ}-unique with probability at least $1 - 2^{-dn/2}$.*

- *for $m \geq n + \kappa + 4$ and $d \geq 8c$ we have that Γ is 2^{κ}-majority-unique with probability at least $1 - 2^{-dn/4}$.*

Proof. We will give the proof for *k*-majority-uniqueness. The proof for *k*-uniqueness uses the same technique. By a standard argument based on the pigeonhole principle, for Γ to be *k*-majority-unique it suffices that for all $S \subseteq \langle n \rangle^c$ with $|S| \leq k$ we have that $|\Gamma(S)| > (3/4)d|S|$. Given that Γ is *k*-independent, we will now bound the probability that there exists a subset S with $|S| \leq k$ such that $|\Gamma(S)| \leq (3/4)d|S|$. For every pair of sets (S, B) satisfying that $S \subseteq \langle n \rangle^c$ with $|S| \leq k$ and $B \subseteq \{1, 2, \ldots, d\} \times \langle m \rangle$ with $|B| = (3/4)d|S|$, the probability that $\Gamma(S) \subseteq B$ is given by $\prod_{i=1}^{d}(|B_i|/2^m)^{|S|}$ where $B_i = \{(i, y) \in B\}$. By the inequality of the arithmetic and geometric means we have that

$$\prod_{i=1}^{d} \left(\frac{|B_i|}{2^m} \right)^{|S|} \leq \left(\frac{|B|}{d2^m} \right)^{d|S|}.$$

This allows us to ignore the structure of B, and obtain a union bound that matches that of the standard non-compartmentalized probabilistic construction of bipartite expanders. The probability that Γ fails to be *k*-majority-unique is upper bounded by

$$\sum_{i=2}^{k} \binom{2^{cn}}{i} \binom{d2^m}{(3/4)di} \left(\frac{(3/4)di}{d2^m} \right)^{di}.$$

For every choice of positive integers c, n, κ, for $m \geq n + \kappa + 4$ and $d \geq 8c$ we get a probability of failure less than 2^{-2cn}.   □

9.3.5   A recursive construction of a k-unique function

In this section we introduce a recursive construction of a k-unique function of the form $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$. We obtain Γ as the last in a sequence $\Gamma_1, \Gamma_2, \ldots, \Gamma_c$ of k-unique functions $\Gamma_i : \langle n \rangle^i \to \langle m \rangle^d$. Each Γ_i for $i > 1$ is defined in terms of Γ_{i-1}. At the bottom of the recursion we tabulate a k-independent function $\Gamma_1 : \langle n \rangle \to \langle m \rangle^d$. In the general step we apply Γ_{i-1} to the length $i-1$ prefix of the key $(x_1, x_2, \ldots, x_{i-1})$, concatenate the result vector component-wise with the ith character x_i, and apply a simple tabulation function $h_i : \langle m + n \rangle^d \to \langle m \rangle^d$. The recursion is therefore given by

$$\Gamma_i((x_1, x_2, \ldots, x_i)) = h_i(\Gamma_{i-1}((x_1, x_2, \ldots, x_{i-1})) \parallel (x_i, x_i, \ldots, x_i)) \quad (9.2)$$

where $(x_i, x_i, \ldots, x_i)$ denotes the string of x_i repeated d times. The following theorem summarizes the properties of Γ in the word RAM model.

Theorem 9.1. *There exists a randomized data structure that takes as input positive integers c, n, κ and initializes a function $\Gamma : \langle n \rangle^c \to \langle n + \kappa + 1 \rangle^{4c}$. In the word RAM model with word size w the data structure satisfies the following:*

- *The space usage is $O(2^{2n+\kappa} c^3 (n + \kappa)/w)$ words.*

- *The evaluation time of Γ is $O(c^2 + c^3(n + \kappa)/w)$.*

- *Γ is 2^κ-unique with probability at least $1 - 2^{-cn}$.*

Proof. Set $m = n + \kappa + 1$ and $d = 4c$. We initialize Γ by tabulating a k-independent function $\Gamma_1 : \langle n \rangle \to \langle m \rangle^d$ and simple tabulation functions $h_2, h_3, \ldots, h_c : \langle m + n \rangle^d \to \langle m \rangle^c$. In total we need to store c functions that each have $O(c)$ character tables with $O(2^{2n+\kappa})$ entries of $O(c(n+\kappa))$ bits. The space usage is therefore $O(2^{2n+\kappa} c^3 (n+\kappa)/w)$ words. The same bound holds for the time to initialize the data structure.

The evaluation time of Γ can be found by considering the recursion given in equation (9.2). At each of the c steps we perform $O(c)$ lookups and take the exclusive-or of $O(c)$ bit strings of length $O(c(n+\kappa))$. The total evaluation time is therefore $O(c^2 + c^3(n+\kappa)/w)$.

Conditioned on Γ_{i-1} being k-unique, it is easy to see that Γ_i is k-unique, and by Lemma 9.1 we have that Γ_i is k-independent. For our choice of parameters, according to Lemma 9.2 the probability that Γ_i fails to be k-unique is less than 2^{-2cn}. Therefore, Γ is k-unique if $\Gamma_1, \Gamma_2, \ldots, \Gamma_c$ are k-unique. This happens with probability at least $1 - c2^{-2cn} \geq 1 - 2^{-cn}$. $\qquad\square$

Combining Theorem 9.1 and Lemma 9.1, we get k-independent hashing in the word RAM model. We state our result in terms of a data structure that represents a family of functions $\mathcal{F}$. The family is defined as in Lemma 9.1 and is represented by a particular instance of a function Γ, constructed using Theorem 9.1, together with the parameters of a family of simple tabulation functions.

Corollary 9.1. *There exists a randomized data structure that takes as input positive integers u, r, k, t and selects a family of functions $\mathcal{F}$ from $[u]$ to $[r]$. In the word RAM model with word size w the data structure satisfies the following:*

- *The space used to represent $\mathcal{F}$, as well as a function $f \in \mathcal{F}$, is $O(ku^{1/t}t^2(\log u + t \log k)/w)$ words.*

- *The evaluation time of f is $O(t^2 + t^2(\log u + t \log k)/w)$.*

- *$\mathcal{F}$ is a k-independent family with probability at least $1 - 1/u$.*

Proof. We apply Theorem 9.1, setting $c = 2t$, $n = \lceil(\log u)/2t\rceil$, $\kappa = \lceil\log k\rceil$. This gives a function $\Gamma : \langle n\rangle^c \to \langle n + \kappa + 1\rangle^{4c}$ that is k-unique over $[u]$ with probability at least $1 - 1/u$. To sample a function from the family we follow the approach of Lemma 9.1 and compose Γ with a simple tabulation function $h : \langle n + \kappa + 1\rangle^{4c} \to [r]$. The space used to store Γ follows directly from Theorem 9.1 and dominates the space used by h. Similarly, the evaluation time of $h \circ \Gamma$ is dominated by the time it takes to evaluate Γ. $\qquad\square$

Remark 9.1. For every integer $\tau \geq 1$ we can construct a family $\mathcal{F}^{(\tau)}$ that is k-independent with probability at least $1 - u^{-\tau}$ at the cost of increasing the space usage and evaluation time by a factor τ. The family is defined by

$$\mathcal{F}^{(\tau)} = \{f = \bigoplus_{i=1}^{\tau} f_i \mid f_i \in \mathcal{F}_i\}$$

where each $\mathcal{F}_i$ is constructed independently.

Remark 9.2. The recursion in equation (9.2) is well suited for sequential evaluation where the task is to evaluate Γ in an interval of $[u]$ in order to generate a sequence of k-independent random variables. To see this, note that once we have evaluated Γ on a key $x = (x_1, x_2, \ldots, x_c)$, a change in the last character only changes the last step of the recursion. It follows that we can generate k-independent variables using amortized time $O(t)$

and space close to $O(ku^{1/t})$. To our knowledge, this presents the best space-time tradeoff for the generation of k-independent variables in the case where we do not have access to multiplication over a suitable finite field as in [55].

9.3.6   A divide and conquer recursion

In this section we introduce a data structure for representing a k-majority-unique function that offers a faster evaluation time at the cost of using more space. As in the construction from Theorem 9.1 we use the technique of alternating between expansion and independence, but rather than reading a single character at the time, we view the key as composed of two characters $x = (x_1, x_2)$ and recurse on each. In the previous section we increased the size of the domain of our k-unique function by concatenating part of the key. If we use only a few large characters this approach becomes very costly in terms of the space required to store the simple tabulation function h_i in the composition $h_i(\Gamma_{i-1}((x_1, x_2, \ldots, x_{i-1})) \,\|\, (x_i, x_i, \ldots, x_i))$. To be able to efficiently recurse on large characters we show that the function $Y((x_1, x_2)) = \Gamma(x_1) \,\|\, \Gamma(x_2)$ is k-unique when Γ is k-majority-unique.

Lemma 9.3. *Let $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$ be a k-majority-unique function. Then the function $Y : \langle n \rangle^c \times \langle n \rangle^c \to \langle 2m \rangle^d$ defined by $Y((x_1, x_2)) = \Gamma(x_1) \,\|\, \Gamma(x_2)$ is k-unique.*

Proof. Let $x = (x_1, x_2)$ denote an element of $\langle n \rangle^c \times \langle n \rangle^c$. For $S \subseteq \langle n \rangle^c \times \langle n \rangle^c$ define $S_{1,a} = \{x \in S \mid x_1 = a\}$. The following holds for every $x = (x_1, x_2) \in S$.

$$
\begin{aligned}
|Y(\{x\}) \backslash Y(S \backslash \{x\})| &= |Y(\{x\}) \backslash (Y(S \backslash S_{1,x_1}) \cup Y(S_{1,x_1} \backslash \{x\}))| \\
&= |(Y(\{x\}) \backslash Y(S \backslash S_{1,x_1})) \cap (Y(\{x\}) \backslash Y(S_{1,x_1} \backslash \{x\}))| \\
&\geq |Y(\{x\}) \backslash Y(S \backslash S_{1,x_1})| + |Y(\{x\}) \backslash Y(S_{1,x_1} \backslash \{x\})| \\
&\quad - |Y(\{x\})|. \hspace{4cm} (9.3)
\end{aligned}
$$

We will show that for every $S \subseteq \langle n \rangle^c \times \langle n \rangle^c$ with $|S| \leq k$ there exists a key $(x_1, x_2) \in S$ such that $|Y(\{x\}) \backslash Y(S \backslash \{x\})| > 0$. We begin by choosing the first component of x. Let $\pi_j(S) = \{x_j \mid x \in S\}$ denote the set of jth components of elements of S. By the k-majority-uniqueness of Γ, considering the set $\pi_1(S)$, we have that

$$
\exists x_1 \in \pi_1(S) : \forall x \in S_{1,x_1} : |Y(\{x\}) \backslash Y(S \backslash S_{1,x_1})| > d/2.
$$

Fix x_1 with this property and consider the choice of x_2. By the k-majority-uniqueness of Γ, considering the set $\pi_2(S)$, we have that

$$\forall x_1 \in \pi_1(S) : \exists x_2 \in \pi_2(S_{1,x_1}) : |Y(\{x\}) \backslash Y(S_{1,x_1} \backslash \{x\})| > d/2.$$

We can therefore always find a key $(x_1, x_2) \in S$ such that both $|Y(\{x\}) \backslash Y(S \backslash S_{1,x_1})| > d/2$, $|Y(\{x\}) \backslash Y(S_{1,x_1} \backslash \{x\})| > d/2$ are satisfied. The result follows from equation (9.3) where we use the fact that $|Y(\{x\})| = d$. $\qquad\square$

We will give a recursive construction of a k-majority-unique function of the form $\Gamma_i : \langle n \rangle^{2^i} \to \langle m \rangle^{2^{i+3}}$. Let $h_i : \langle 2m \rangle^{2^{i+2}} \to \langle m \rangle^{2^{i+3}}$ be a simple tabulation function. For $i > 0$ the recursion takes the following form.

$$\Gamma_i((x_1, \ldots, x_{2^i})) = h_i(\Gamma_{i-1}((x_1, \ldots, x_{2^{i-1}})) \,\|\, \Gamma_{i-1}((x_{2^{i-1}+1}, \ldots, x_{2^i}))).$$

At the bottom of the recursion we tabulate a k-independent function Γ_0.

Theorem 9.2. *There exists a randomized data structure that takes as input positive integers λ, n, κ and initializes a function $\Gamma : \langle n \rangle^{2^\lambda} \to \langle n + \kappa + 4 \rangle^{2^{\lambda+3}}$. In the word RAM model with word size w the data structure satisfies the following:*

- *The space usage is $O(2^{2(n+\kappa+\lambda)}(n+\kappa)/w)$ words.*

- *The evaluation time of Γ is $O(2^\lambda(\lambda + 2^\lambda(n+\kappa)/w))$.*

- *Γ is 2^κ-majority-unique with probability at least $1 - 2^{-2n+1}$.*

Proof. Let $m = n + \kappa + 4$. We initialize Γ by tabulating Γ_0 and the character tables of the simple tabulation functions $h_1, h_2, \ldots, h_\lambda$ where $h_i : \langle 2m \rangle^{2^{i+2}} \to \langle m \rangle^{2^{i+3}}$. In total we have $O(2^\lambda)$ character tables with $O(2^{2(n+\kappa)})$ entries of $O(2^\lambda(n+\kappa))$ bits, resulting in the space bound.

Let $T(i)$ denote the evaluation time of Γ_i. For $i = 0$ we can evaluate Γ_0 by performing a single lookup in $O(1)$ time. For $i > 0$ evaluating $h_i \circ (\Gamma_{i-1} \,\|\, \Gamma_{i-1})$ takes two evalutions of Γ_{i-1} followed by evaluating h_i on their concatenated output using $O(2^i(1 + 2^i(n+\kappa)/w))$ operations. The recurrence takes the form

$$T(i) \leq \begin{cases} 2T(i-1) + O(2^i(1 + 2^i(n+\kappa)/w)) & \text{if } i > 0 \\ O(1) & \text{if } i = 0 \end{cases}$$

The solution to the recurrence is $O(2^i(i + 2^i(n+\kappa)/w))$.

We now turn our attention to the probability that $\Gamma_i = h_i \circ (\Gamma_{i-1} \parallel \Gamma_{i-1})$ fails to be k-majority-unique. Conditional on Γ_{i-1} being k-majority-unique, by Lemma 9.3 we have that $(\Gamma_{i-1} \parallel \Gamma_{i-1})$ is k-unique and composing it with h_i gives us a k-independent function. For our choice of parameters, according to Lemma 9.2 the probability that Γ_i fails to be k-majority-unique is less than $2^{-2^{i+1}n}$. Therefore, Γ is k-majority-unique if $\Gamma_0, \Gamma_1, \ldots, \Gamma_\lambda$ are k-majority-unique. This happens with probability at least $1 - \sum_{i=0}^{\lambda} 2^{-2^{i+1}n} \geq 1 - 2^{-2n+1}$. $\qquad\square$

Remark 9.3. The recursion behind Lemma 9.2 is well suited for parallelization. If we have c processors working in lock-step with some small shared memory we can evaluate Γ with domain $\langle n \rangle^c$ in time $O(c)$.

Corollary 9.2. *There exists a randomized data structure that takes as input positive integers u, r, k, t and selects a family of functions $\mathcal{F}$ from $[u]$ to $[r]$. In the word RAM model with word size w the data structure satisfies the following:*

- *The space used to represent $\mathcal{F}$, as well as a function $f \in \mathcal{F}$, is $O(k^2 u^{1/t} t (\log u + t \log k)/w)$ words.*

- *The evaluation time of f is $O(t \log t + t(\log u + t \log k)/w)$.*

- *$\mathcal{F}$ is k-independent with probability at least $1 - u^{-1/t}$.*

Proof. Apply Theorem 9.2 with parameters $\lambda = \lceil \log t \rceil + 1$, $n = \lceil (\log u)/2t \rceil + 1$, and $\kappa = \lceil \log k \rceil$. This gives is a function Γ that is k-unique over $[u]$ with probability at least $1 - u^{-1/t}$. The family $\mathcal{F}$ is defined by the composition of Γ with a suitable simple tabulation function following the approach of Lemma 9.1. $\qquad\square$

Remark 9.4. If we have access to a standard t-independent polynomial hash function that outputs elements of $[r]$ then the error probability can be reduced to $1/u$ at no additional cost provided that the evaluation time of such a function does not exceed the evaluation time of $f \in \mathcal{F}$. The trick is to add (modulo r) the output of such a guaranteed t-independent function to the output of our original function from $\mathcal{F}$. The reason is that most of the error probability comes from the "small sets" in the union bound in Lemma 9.2 and these will now be dealt with by our t-independent function.

9.3.7   Balancing time and space

Theorem 9.1 yielded a k-unique function over $\langle n \rangle^c$ with an evaluation time of about $O(c^2)$ while using linear space in k. Theorem 9.2 resulted in an evaluation time of about $O(c \log c)$, using quadratic space in k. Under a mild restriction on k, the two techniques can be combined to obtain an evaluation time of $O(c \log c)$ and linear space in k. We take the construction from Theorem 9.2 as our starting point, but instead of tabulating the character tables of $h_1, \ldots, h_\lambda$ we replace them with more space efficient k-independent functions that we construct using Theorem 9.1.

Theorem 9.3. *There exists a randomized data structure that takes as input positive integers λ, n, $\kappa = O(n)$ and initializes a function $\Gamma : \langle n \rangle^{2^\lambda} \to \langle n + \kappa + 4 \rangle^{2^{\lambda+3}}$. In the word RAM model with word size w the data structure satisfies the following:*

- *The space usage is $O(2^{n+\kappa+2\lambda} n / w)$ words.*

- *The evaluation time of Γ is $O(2^\lambda(\lambda + 2^\lambda n / w))$.*

- *Γ is 2^κ-majority-unique with probability at least $1 - 2^{-n+1}$.*

Proof. At the top level, the recursion underlying Γ takes the same form as in Theorem 9.2.
$$\Gamma_i = h_i \circ (\Gamma_{i-1} \,\|\, \Gamma_{i-1}).$$
The functions $h_i : \langle 2m \rangle^{2^{i+2}} \to \langle m \rangle^{2^{i+3}}$ are simple tabulation functions with $m = n + \kappa + 4$. Each h_i is constructed from 2^{i+2} character tables $h_{i,j} : \langle 2m \rangle \to \langle m \rangle^{2^{i+3}}$. Theorem 9.2 only assumes that the character tables $h_{i,j}$ are k-independent functions. We will apply Theorem 9.1 to construct a function Y that we for each character table $h_{i,j}$ compose with a simple tabulation function $g_{i,j}$ in order to construct $h_{i,j}$. By the restriction that $\kappa = O(n)$ we have that $m = O(n)$. We set the parameters of Y to $\hat{c} = O(1)$, $\hat{n} = \lceil n/2 \rceil$, $\hat{\kappa} = \kappa$ such that $\langle 2m \rangle$ can be embedded in $\langle \hat{n} \rangle^{\hat{c}}$. Furthermore, Y uses $O(2^{n+\kappa} n / w)$ words of space, can be evaluated in $O(1)$ operations, and is k-unique with probability at last $1 - 2^{-n-1}$. Because Y has $O(1)$ output characters, the time to evaluate $h_{i,j} = g_{i,j} \circ Y$ is no more than a constant times the word length of the output of $h_{i,j}$. The time to evaluate Γ therefore only increases by a constant factor compared to the evaluation time in Theorem 9.2.

The probability of failure of Γ to be k-majority-unique is the same as in Theorem 9.2, provided that Y does not fail to be k-unique. This gives a total probability of failure of less than $2^{-2n+1} + 2^{-n-1} < 2^{-n+1}$.

We only store a single Y and the character tables of $g_{i,j}$ that we use to simulate the character tables $h_{i,j}$. From the parameters of Y we have that $g_{i,j}$ uses $O(1)$ character tables with $O(2^{n+\kappa})$ entries of $O(2^i n/w)$ words. The space usage is dominated by the $O(2^\lambda)$ character tables of h_λ that use space $O(2^{n+\kappa+2\lambda} n/w)$ in total. $\square$

Corollary 9.3. *There exists a randomized data structure that takes as input positive integers $u, r, t, k = u^{O(1/t)}$ and selects a family of functions $\mathcal{F}$ from $[u]$ to $[r]$. In the word RAM model with word length w the data structure satisfies the following:*

- *The space used to represent $\mathcal{F}$, as well as a function $f \in \mathcal{F}$, is $O(k u^{1/t} t (\log u)/w)$ words.*

- *The evaluation time of f is $O(t \log t + t(\log u)/w)$.*

- *$\mathcal{F}$ is k-independent with probability at least $1 - u^{-1/t}$.*

Proof. Apply Theorem 9.3 with parameters $\lambda = \lceil \log t \rceil$, $n = \lceil (\log u)/t \rceil + 1$, and $\kappa = \lceil \log k \rceil$. This gives is a function Γ that is k-unique over $[u]$ with probability at least $1 - u^{-1/t}$. The family $\mathcal{F}$ is defined by the composition of Γ with a suitable simple tabulation function following the approach of Lemma 9.1. $\square$

9.3.8   An improvement for space close to k

In this section we present a different space efficient version of the divide-and-conquer recursion. The new recursion is based on an extension of the ideas behind the graph product from Lemma 9.3. In Lemma 9.3 we use expansion properties over subsets of size k and concatenate the output characters of Γ, resulting in an output domain of size at least k^2. By using stronger expansion properties and modifying our graph concatenation product to fit the structure of the key set, we are able to reduce the space usage at the cost of using more time. We now introduce a property of d-regular bipartite graph that we call k-super-majority-uniqueness: More than half the input vertices have more than $d/2$ unique neighbors.

Definition 9.4. Let $\Gamma : \langle n \rangle^c \to \langle m \rangle^d$ be a function satisfying the following property:

$$\forall S \subseteq \langle n \rangle^c, |S| \leq k : |\{x \in S \mid |\Gamma(\{x\}) \backslash \Gamma(S \backslash \{x\})| > d/2\}| > |S|/2.$$

Then we say that Γ is *k-super-majority-unique*.

Overview of approach. The exposition is quite technical but the underlying idea is simple: Suppose we have a set S of k keys of the form (x_1, x_2). Then the set of first components $\pi_1(S) = \{x_1 \mid (x_1, x_2) \in S\}$ has size $|\pi_1(S) = k^\varepsilon$ for some $\varepsilon \in (0, 1)$. By the pigeonhole principle there must exist a subset $A \subseteq \pi_1(S)$ of size at least $k^\varepsilon/2$ such that for every $a \in A$ we have that $|\{(a, x_2) \in S\}| \leq 2k^{1-\varepsilon}$, otherwise we would have more than k keys! Therefore, the function $\Gamma((x_1, x_2)) = \Gamma_1(x_1) \parallel \Gamma_2(x_2)$ is k-unique on the set S provided that Γ_1 is k^ε-super-majority-unique and Γ_2 is $2k^{1-\varepsilon}$-majority-unique. By forming these combinations for $O(\log k)$ choices of ε we can cover all possible structures of two-character keys, ensuring that Γ is k-unique while keeping the size of the right-hand side close to $O(k)$. For a given choice of ε we obtain Γ_1 and Γ_2 by componentwise concatenation of differently sized prefixes of the output a k-independent function. Based on this idea we can then build a divide-and-conquer recursion similar to the one behind Theorem 9.2. The technical details behind this idea is deferred to Appendix 9.5

9.4 Conclusion

We have presented new constructions of k-independent hash functions that come close to Siegel's lower bound on the space-time tradeoff for such functions. An interesting open problem is whether the gap to the lower bound can be closed. From the perspective of efficient expanders it would be very interesting to achieve space $o(k)$ while preserving computational efficiency. Of course, such a result is not possible via k-independence.

Acknowledgements

9.5   Appendix: Details behind the prefix technique

The following lemma shows how we can construct a k-unique function over U^2 from a set of k-super-majority-unique functions over U.

Lemma 9.4. *Let q be a positive integer. For $j = 1, 2, \ldots, q$ let $\Gamma_j : \langle \kappa \rangle^c \to \langle m_j \rangle^d$ be $\min(2k^{j/q}, k)$-super-majority-unique and set $m = \max_j(m_j + m_{q-j+1})$. Then the function $\Gamma : \langle \kappa \rangle^c \times \langle \kappa \rangle^c \to \langle m \rangle^{dq}$ defined for $(j, l) \in \{1, \ldots, q\} \times \{1, \ldots, d\}$ by*

$$\Gamma(x_1, x_2)_{(j-1)d+l} = (\Gamma_j(x_1)_l \,\|\, \Gamma_{q-j+1}(x_2)_l)[m]$$

is k-unique.

Proof. Consider a set of keys $S \subseteq \langle n \rangle^c \times \langle n \rangle^c$ with $|S| \leq k$. We will show that there exists an index $j \in \{1, \ldots, q\}$ and a key $x = (x_1, x_2) \in S$ such that x has a unique neighbor with respect to S and $\Gamma_j \,\|\, \Gamma_{q-i+1}$. Consider the set of first components of the set of keys $\pi_1(S)$. For some $j \in \{1, \ldots, q\}$ we must have that $k^{(j-1)/q} \leq |\pi_1(S)| \leq k^{j/q}$. By the super-majority-uniqueness properties of Γ_j there must exist more than $k^{(j-1)/q}/2$ first components $x_1 \in \pi_1(S)$ such that $\Gamma_j(x_1)$ has more than $d/2$ unique neighbors with respect to $\pi_1(S)$. Furthermore, because $|S| \leq k$, there exists at least one such x_1 that is a component of at most $\min(2k^{(q-j+1)/q}, k)$ keys. Following a similar argument to the proof of Lemma 9.3, by the majority-uniqueness properties of Γ_{q-j+1} there exists $x_2 \in S_{1,x_1}$ such that we get a unique neighbor. $\square$

In the following lemma we use a single k-independent function to represent a set of k-super-majority-unique functions such that the concatenated product of these functions is k-unique. The proof of the lemma is omitted since it follows from using the approach of Lemma 9.2 to obtain expansion $|\Gamma(S)| > (7/8)d|S|$, and then applying Lemma 9.4 to obtain the k-uniqueness property.

Lemma 9.5. *For every choice of positive integers c, q, κ, let $f : \langle \kappa \rangle^c \to \langle 2\kappa + 12 \rangle^{16cq}$ be a 2^κ-independent function. For $j = 1, \ldots, q$ define $\Gamma_j : \langle \kappa \rangle^c \to \langle \lceil ((j+1)/q)\kappa \rceil + 12 \rangle^{16cq}$ by*

$$\Gamma_j(x)_l = f(x)_l[\lceil ((j+1)/q)\kappa \rceil + 12] \text{ for } l \in \{1, \ldots, 16cq\}.$$

Let $m = \lceil (1 + 3/q)\kappa \rceil + 26$. Then the function $\Gamma : \langle \kappa \rangle^c \times \langle \kappa \rangle^c \to \langle m \rangle^{16cq^2}$ defined for $(j, l) \in \{1, \ldots, q\} \times \{1, \ldots, 16cq\}$ by

$$\Gamma(x_1, x_2)_{(j-1)16cq+l} = (\Gamma_j(x_1)_l \parallel \Gamma_{q-j+1}(x_2)_l)[m]$$

is k-unique with probability at least $1 - 2^{-2c\kappa}$.

We remind the reader that the notation $x[m]$ is used to denote the zero-padded m-bit prefix of x. Taking the prefix of the concatenated output characters of Γ_j and Γ_{q-j+1} is done with the sole purpose of padding the output characters of Γ to uniform length.

We now define a randomized recursive construction of a k-unique function similar to the one in Theorem 9.2. The parameters of the data structure are λ, κ, and q. The parameters λ and κ determine the size of the universe and the desired k-uniqueness. The parameter q controls the space-time tradeoff of the character tables used in the recursion. At the outer level of the recursion, for $i = 1, \ldots, \lambda$, we repeatedly square the size of the domain, constructing k-unique functions of the form $\Gamma_i : \langle \kappa \rangle^{2^i} \to \langle 2\kappa + 26 \rangle^{144 \cdot 2^i}$. At level i of the recursion, we obtain a k-independent function by composing Γ_i with a simple tabulation function $h_{i+1} : \langle 2\kappa + 26 \rangle^{144 \cdot 2^i} \to \langle 2\kappa + 12 \rangle^{48 \cdot 2^{i+1}}$. The output of this function is then used to construct Γ_{i+1}, following the approach of Lemma 9.5 with the parameter q set to 3. For $i = 1, 2, \ldots, \lambda$ the recursion is described by the following set of equations

$$\Gamma_i(x_1, x_2)_{(j-1)48 \cdot 2^i + l} = (\Gamma_{i,j}(x_1)_l \parallel \Gamma_{i,4-j}(x_2)_l)[2\kappa + 26]$$
$$\Gamma_{i,j}(x_s)_l = h_i(\Gamma_{i-1}(x_s))_l[\lceil ((j+1)/3)\kappa \rceil + 12]$$
$$\Gamma_0(x_s)_l = \mathrm{I}^{(48)}(x_s)_l[2\kappa + 26]$$

where the indices are $j \in \{1, 2, 3\}$, $l \in \{1, \ldots, 48 \cdot 2^i\}$, and $s \in \{1, 2\}$. We have defined Γ_0 by simply repeating the input 48 times, padded to length $2\kappa + 26$, to ensure that it fits into the recursion. In practice we only require $h_1 \circ \Gamma_0$ be be k-independent over domain $\langle \kappa \rangle$.

To further reduce the space usage we apply the technique from Lemma 9.5 to implement the character tables of h_i. Each character table has domain $\langle 2\kappa + 26 \rangle$. We view this domain as consisting of two characters of length $\kappa' = \kappa + 13$. We apply Lemma 9.5 with parameters $c = 1$, q, and $\kappa = \kappa'$ to construct a function $Y : \langle \kappa' \rangle^2 \to \langle \lceil (1 + 3/q)\kappa' \rceil + 26 \rangle^{16q^2}$ that is k-unique with probability at

least $1 - 2^{-2\kappa'}$. To facilitate fast evaluation we tabulate the k-independent function $f' : \langle \kappa' \rangle \to \langle \lceil (1 + 1/q)\kappa' \rceil + 12 \rangle^{16q}$ used to construct Y. The jth character table of h_i is constructed by composing Y with an appropriate simple tabulation function,

$$h_{i,j} = Y \circ g_{i,j},$$

where $g_{i,j} : \langle \lceil (1 + 1/q)\kappa' \rceil + 12 \rangle^{16q} \to \langle 2\kappa + 12 \rangle^{48 \cdot 2^i}$ is tabulated.

Theorem 9.4. *There exists a randomized data structure that takes as input positive integers λ, κ, q, and initializes a function $\Gamma : \langle \kappa \rangle^{2^\lambda} \to \langle 2\kappa + 26 \rangle^{48 \cdot 2^\lambda}$. In the word RAM model with word length w the data structure satisifes the following:*

- *The space usage is $O(2^{(1+3/q)\kappa + 2\lambda} q^2 \kappa / w)$.*

- *The evaluation time of Γ is $O(2^\lambda q^2 (\lambda + 2^\lambda \kappa / w))$.*

- *With probability at least $1 - 2^{-2(\kappa - 1)}$ we have that Γ is 2^κ-unique.*

Proof. The total space usage is dominated by the simple tabulation functions used to implement the character tables of h_λ. There are $O(2^\lambda)$ simple tabulation functions $g_{i,j}$. Each of these has $O(q^2)$ character tables with a domain of size $O(2^{(1+3/q)\kappa})$ that map to bit strings of length $O(2^\lambda \kappa)$. This gives a total space usage of $O(2^{(1+3/q)\kappa + 2\lambda} q^2 \kappa / w)$.

Let $T(i)$ denote the evaluation time of Γ_i. For $i = 1$ we can evaluate Γ_1 by performing a constant number of lookups into h_0 and combine prefixes of the output in $O(1)$ time. For $i > 1$ evaluating Γ_i takes two evaluations of Γ_{i-1} and an additional amount of work combining prefixes that is only a constant factor greater than the time required to read the output of $h_i \circ \Gamma_{i-1}$. Evaluating h_i is performed by $O(2^i)$ evaluations of character tables of the form $g_{i,j} \circ Y$. The degree of Y is $O(q^2)$ and it has an evaluation time that is proportional to the degree. We therefore perform $O(q^2)$ lookups into the character tables of $g_{i,j}$ where we read bit strings of length $O(2^i \kappa)$. The recurrence describing the evaluation time of Γ_i takes the form

$$T(i) \leq \begin{cases} 2T(i-1) + O(2^i q^2 (1 + 2^i \kappa / w)) & \text{if } i > 1 \\ O(1) & \text{if } i = 1. \end{cases}$$

The solution to the recurrence is $O(2^i q^2 (i + 2^i (n + \kappa) / w))$.

The construction fails if Y fails to be k-unique or if $\Gamma_1, \ldots, \Gamma_\lambda$ fails to be k-unique. According to Lemma 9.5 this happens with probability less than $2^{-2\kappa'} + \sum_{i=1}^{\lambda} 2^{-2^i \kappa} < 2^{-2(\kappa-1)}$ □

Corollary 9.4. *There exists a randomized data structure that takes as input positive integers u, $r = u^{O(1)}$, t, k and selects a family of functions $\mathcal{F}$ from $[u]$ to $[r]$. In the word RAM model with word length w the data structure satisfies the following:*

- *The space used to represent $\mathcal{F}$, as well as a function $f \in \mathcal{F}$, is $O(ku^{1/t}t^2 \log(k)/w)$ words.*

- *The evaluation time of f is $O(t^2(\log(k)/\log u)(\log(\log(u)/\log k) + \log(u)/w))$.*

- *$\mathcal{F}$ is k-independent with probability at least $1 - k^{-2}$.*

Proof. Assume without loss of generality that $k \le u$ and apply Theorem 9.4 with parameters $\lambda = \lceil \log(\log(u)/\log k) \rceil + 1$, $\kappa = \lceil \log k \rceil + 1$, and $q = \lceil 3t \log(k)/\log u \rceil$. This gives is a function Γ that is k-unique over $[u]$ with probability at least $1 - k^{-2}$. We compose Γ with a suitable simple tabulation function h that maps to elements of $[r]$. Implementing h using Y we get the same bounds on the space usage, evaluation time, and probability of failure as for the data structure used to represent Γ.   □

Remark 9.5. The construction in Corollary 9.4 presents an improvement in the case where we wish to minimize the space usage. For $w = \Theta(\log u)$ and $t = \lceil \log u \rceil$ we get a space usage of $O(k \log(u) \log(k))$ and an evaluation time of $O(\log(u) \log(k) \log(\log(u)/\log(k)))$. In comparison, for these parameters Corollary 9.1 gives a space usage of $O(k \log^2 u)$ and an evalution time of $O(\log^2(u) \log(k))$.

Bibliography

[1] ACM. ACM paris kanellakis theory and practice award. `https://awards.acm.org/award_winners/charikar_0308379`, 2012. [Online; accessed 26-April-2018].

[2] C. Aggarwal, D. A. Keim, and A. Hinneburg. On the surprising behaviour of distance metrics in high dimensional space. In *Proc. ICDT '01*, pages 420–434, 2001.

[3] A. Agresti. Bounds on the extinction time distribution of a branching process. *Advances in Applied Probability*, 6(2):322–335, 1974.

[4] T. D. Ahle, M. Aumüller, and R. Pagh. Parameter-free locality sensitive hashing for spherical range reporting. In *Proc. SODA '17*, pages 239–256, 2017.

[5] T. D. Ahle, R. Pagh, I. P. Razenshteyn, and F. Silvestri. On the complexity of inner product similarity join. In *Proc. PODS'16*, pages 151–164, 2016.

[6] J. Alman and R. Williams. Probabilistic polynomials and hamming nearest neighbors. In *Proc. FOCS '15*, pages 136–150, 2015.

[7] N. Alon and N. Asaf. k-wise independent random graphs. In *Proc. FOCS '08*, pages 813–822, 2008.

[8] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304, 1992.

[9] A. Andoni. *Nearest neighbor search: the old, the new, and the impossible*. PhD thesis, MIT, 2009.

[10] A. Andoni and P. Indyk. Efficient algorithms for substring near neighbor problem. In *Proc. SODA '06*, pages 1203–1212, 2006.

[11] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proc. FOCS '06*, pages 459–468, 2006.

[12] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[13] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal lsh for angular distance. In *Proc. NIPS '15*, pages 1225–1233, 2015.

[14] A. Andoni, P. Indyk, H. L. Nguyen, and I. P. Razenshteyn. Beyond locality-sensitive hashing. In *Proc. SODA '14*, pages 1018–1028, 2014.

[15] A. Andoni, P. Indyk, and M. Patrascu. On the optimality of the dimensionality reduction method. In *Proc. FOCS '06*, pages 449–458, 2006.

[16] A. Andoni, T. Laarhoven, I. P. Razenshteyn, and E. Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In *Proc. SODA '17*, pages 47–66, 2017.

[17] A. Andoni and I. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proc. STOC '15*, pages 793–801, 2015.

[18] A. Andoni, I. P. Razenshteyn, and N. S. Nosatzki. LSH forest: Practical algorithms made theoretical. In *Proc. SODA '17*, pages 67–78, 2017.

[19] A. Andoni and I. Razensteyn. Tight lower bounds for data-dependent locality-sensitive hashing. In *Proc. SoCG '16*, pages 9:1–9:11, 2016.

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

[21] N. Augsten and M. H. Böhlen. Similarity joins in relational database systems. *Synthesis Lectures on Data Management*, 5(5):1–124, 2013.

[22] M. Aumüller, T. Christiani, R. Pagh, and F. Silvestri. Distance-sensitive hashing. *CoRR*, abs/1703.07867, 2017.

[23] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *Proc. 16th World Wide Web Conference (WWW)*, pages 131–140, 2007.

[24] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proc. SODA '16*, pages 10–24, 2016.

[25] Y. Benyamini and J. Lindenstrauss. *Geometric nonlinear functional analysis*, volume 48. American Mathematical Soc., Providence, Rhode Island, 1998.

[26] M. Bhattacharya, R. Creutzburg, and J. Astola. Some historical notes on the number theoretic transform. In *Proc. 2004 Int. TICS Workshop on Spectral Methods and Multirate Signal Processing*, 2004.

[27] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complexity*, 21(4):420–446, 2005.

[28] J. Braun-Blanquet. *Plant sociology. The study of plant communities*. McGraw-Hill, 1932.

[29] M. Braverman. Polylogarithmic independence fools AC^0 circuits. *J. ACM*, 57(5):28:1–28:10, 2010.

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

[31] A. Z. Broder. Identifying and filtering near-duplicate documents. In *Proc. Symp. on Combinatorial Pattern Matching (CPM)*, pages 1–10. Springer, 2000.

[32] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proc. STOC '98*, pages 327–336, 1998.

[33] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[34] A. Z. Broder, A. M. Frieze, and E. Upfal. Static and dynamic path selection on expander graphs: a random walk approach. *Random Structures & Algorithms*, 14(1):87–109, 1999.

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

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

[37] H. Burhman, P. B. Miltersen, J. Radhakrishnan, and S. Venkatesh. Are bitvectors optimal? *SIAM J. Comput.*, 31(6):1723–1744, 2002.

[38] M. Capalbo. Explicit bounded-degree unique-neighbor concentrators. *Combinatorica*, 25(4):379–391, 2005.

[39] M. Capalbo, O. Reingold, S. Vadhan, and A. Wigderson. Randomness conductors and constant-degree lossless expanders. In *Proc. STOC '02*, pages 659–668, 2002.

[40] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Proc. STOC '77*, pages 106–112, 1977.

[41] J. L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[42] J. L. Carter and M. N. Wegman. New hash functions and their use in authentication and set equality. *J. Comput. System Sci.*, 22(3):265–279, 1981.

[43] A. Chakrabarti, V. Satuluri, A. Srivathsan, and S. Parthasarathy. Locality sensitive hashing for similarity search and estimation. `https://sites.google.com/site/lshallpairs/`, 2012. [Accessed 15-May-2017].

[44] A. Chakrabarti, V. Satuluri, A. Srivathsan, and S. Parthasarathy. A bayesian perspective on locality sensitive hashing with extensions for kernel methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 10(2):19, 2015.

[45] D. Chakraborty, E. Goldenberg, and M. Koucký. Streaming algorithms for embedding and computing edit distance in the low distance regime. In *Proc. STOC '16*, pages 712–725, 2016.

[46] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random variables. *Jour. Am. Stat. Assoc.*, 71(354):340–344, 1976.

[47] M. Charikar. Similarity estimation techniques from rounding algorithms. In *Proc. STOC '02*, pages 380–388, 2002.

[48] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *Proc. 22nd Conference on Data Engineering (ICDE)*, page 5, 2006.

[49] F. Chierichetti and R. Kumar. Lsh-preserving functions and their applications. *J. ACM*, 62(5):33, 2015.

[50] F. Chierichetti, R. Kumar, A. Panconesi, and E. Terolli. The distortion of locality sensitive hashing. In *Proc. ITCS '17*, pages 54:1–54:18, 2017.

[51] S. Choi, S. Cha, and C. C. Tappert. A survey of binary similarity and distance measures. *J. Syst. Cybern. Informatics*, 8(1):43–48, 2010.

[52] B. Chor, O. Goldreich, J. Hastad, J. Freidmann, S. Rudich, and R. Smolensky. The bit extration problem or t-resilient functions. In *Proc. FOCS '85*, pages 396–407, 1985.

[53] T. Christiani. Fast locality-sensitive hashing for approximate near neighbor search. *CoRR*, abs/1708.07586, 2017.

[54] T. Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In *Proc. SODA '17*, pages 31–46, 2017.

[55] T. Christiani and R. Pagh. Generating k-independent variables in constant time. In *Proc. FOCS '14*, pages 196–205, 2014.

[56] T. Christiani and R. Pagh. Set similarity search beyond minhash. In *Proc. 49th Symp. on Theory of Computing (STOC)*, pages 1094–1107, 2017.

[57] T. Christiani, R. Pagh, and J. Sivertsen. Scalable and robust set similarity join. *CoRR*, abs/1707.06814, 2017.

[58] T. Christiani, R. Pagh, and M. Thorup. From independence to expansion and back again. In *Proc. STOC '15*, pages 813–820, 2015.

[59] K. Chung, M. Mitzenmacher, and S. P. Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. *Theory of Computing*, 9:897–945, 2013.

[60] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comp. Syst. Sci.*, 55(3):441–453, 1997.

[61] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):64–78, 2001.

[62] E. Cohen and H. Kaplan. Leveraging discarded samples for tighter estimation of multiple-set aggregates. *ACM SIGMETRICS Performance Evaluation Review*, 37(1):251–262, 2009.

[63] J. Cohen and D. Kane. Bounds on the independence required for cuckoo hashing. `http://web.mit.edu/jscohen/www/cuckoo.pdf`, 2009. [Online; accessed 29-April-2018].

[64] S. Dahlgaard, M. B. T. Knudsen, and M. Thorup. Fast similarity sketching. In *Proc. FOCS '17*, pages 663–671, 2017.

[65] A. Dasgupta, R. Kumar, and T. Sarlós. Fast locality-sensitive hashing. In *Proc. SIGKDD '11*, pages 1073–1081, 2011.

[66] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. SOCG '04*, pages 253–262, 2004.

[67] R. David, Karthik C. S., and B. Laekhanukit. The curse of medium dimension for geometric problems in almost every norm. *CoRR*, abs/1608.03245, 2016.

[68] M. de Berg, M. van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational geometry*. Springer, Berlin, third edition, 2008.

[69] M. Dietzfelbinger. On randomness in hash functions (invited talk). In *Proc. STACS '12*, pages 25–28, 2012.

[70] M. Dietzfelbinger and F. Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Proc. ICALP '90*, pages 6–19, 1990.

[71] M. Dietzfelbinger and M. Rink. Applications of a splitting trick. In *Proc. ICALP '09*, pages 354–365, 2009.

[72] M. Dietzfelbinger and P. Woelfel. Almost random graphs with simple hash functions. In *Proc. STOC '03*, pages 629–638, 2003.

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

[74] P. Duhamel and M. Vetterli. Fast Fourier transforms: a tutorial review and state of the art. *Signal Processing*, 19(4):259–299, 1990.

[75] K. Eshghi and S. Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. In *Proc. KDD '08*, pages 221–229, 2008.

[76] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31(3):538–544, 1984.

[77] S. Gao and T. Mateer. Additive fast Fourier transforms over finite fields. *IEEE Trans. Inf. Theory*, 56(12):6265–6272, 2010.

[78] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. VLDB '99*, pages 518–529, 1999.

[79] O. Goldreich. *A primer on pseudorandom generators*, volume 55. American Math. Soc., Providence, RI, 2010.

[80] Y. Gong, S. Kumar, V. Verma, and S. Lazebnik. Angular quantization-based binary codes for fast similarity search. In *NIPS*, pages 1205–1213, 2012.

[81]   Google. sparse hash c++ implementation. `https://github.com/sparsehash/sparsehash`, 2017. [Online; accessed 1-May-2017].

[82]   T. Grandlund and P. L. Montgomery. Division by invariant integers using multiplication. *SIGPLAN Not.*, 29(6):61–72, 1994.

[83]   S. Gueron and M. E. Kounavis. Intel carry-less multiplication instruction and its usage for computing the CGM mode. Technical report, Intel Corporation, 2014.

[84]   V. Guruswami, C. Umans, and S. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *J. ACM*, 56(4):20:1–20:34, 2009.

[85]   T. Hagerup. Sorting and searching on the word RAM. In *Proc. STACS '98*, pages 366–398, 1998.

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

[87]   T. E. Harris. *The theory of branching processes*. Courier Corporation, 2002.

[88]   W. Hoeffding. Probability inequalities for sums of bounded random variables. *Jour. Am. Stat. Assoc.*, 58(301):13–30, 1963.

[89]   P. Indyk. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 877–892. Chapman and Hall/CRC, 2004.

[90]   P. Indyk and A. Gilbert. Sparse recovery using sparse matrices. *Proc. IEEE*, 98(6):937–947, 2010.

[91]   P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. STOC '98*, pages 604–613, 1998.

[92]   A. Joffe. On a set of almost deterministic k-independent random variables. *Ann. Prob.*, 2(1):161–162, 1974.

[93]   R. Kaas and J. M. Buhrman. Mean, median and mode in binomial distributions. *Statistica Neerlandica*, 34(1):13–18, 1980.

[94] J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions (extended abstract). In *Proc. FOCS '88*, pages 68–80, 1988.

[95] M. Kapralov. Smooth tradeoffs between insert and query complexity in nearest neighbor search. In *Proc. PODS '15*, pages 329–342, 2015.

[96] K. S. Kedlaya and C. Umans. Fast modular composition in any characteristic. In *Proc. FOCS '08*, pages 146–155, 2008.

[97] C. Kennedy and R. Ward. Fast cross-polytope locality-sensitive hashing. In *Proc. ITCS '17*, pages 53:1–53:16, 2017.

[98] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey, second edition, 1988.

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

[100] T. Laarhoven. Tradeoffs for nearest neighbors on the sphere. *CoRR*, abs/1511.07527, 2015.

[101] K. G. Larsen. Higher cell probe lower bounds for evaluating polynomials. In *Proc. FOCS '12*, pages 293–301, 2012.

[102] P. Lévy. *Calcul des probabilités*, volume 9. Gauthier-Villars, Paris, 1925.

[103] P. Li and A. C. König. Theory and applications of b-bit minwise hashing. *Communications of the ACM*, 54(8):101–109, 2011.

[104] D. Lu and W. V. Li. A note on multivariate gaussian estimates. *Journal of Mathematical Analysis and Applications*, 354(2):704–707, 2009.

[105] E. Lukacs. *Characteristic Functions*. Griffin, London, second edition, 1970.

[106] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proc. VLDB '07*, pages 950–961, 2007.

[107] W. Mann, N. Augsten, and P. Bouros. An empirical evaluation of set similarity join techniques. *PVLDB*, 9(9):636–647, 2016.

[108] J. Marcinkiewicz. Sur une propriete de la loi de gauss. *Mathematische Zeitschrift*, 44(1):612–618, 1939.

[109] P. B. Miltersen. Lower bounds for static dictionaries on RAMs with bit operations but no multiplications. In *Proc. ICALP '96*, pages 442–453, 1996.

[110] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[111] D. S. Mitrinović. *Analytic inequalities*. Springer-Verlag, Berlin, New York, 1970.

[112] M. Mitzenmacher, R. Pagh, and N. Pham. Efficient estimation for high similarities using odd sketches. In *Proc. WWW '14*, pages 109–118, 2014.

[113] M. Mitzenmacher and E. Upfal. *Probability and computing*. Cambridge University Press, New York, NY, 2005.

[114] R. Motwani, A. Naor, and R. Panigrahy. Lower bounds on locality sensitive hashing. *SIAM J. Discrete Math.*, 21(4):930–935, 2007.

[115] R. Motwani and P. Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[116] J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856, 1993.

[117] H. L. Nguyen. *Algorithms for High Dimensional Data*. PhD thesis, Princeton, 2014.

[118] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[119] R. O'Donnell. Open problems in analysis of boolean functions. *CoRR*, abs/1204.6447, 2012.

[120] R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

[121] R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

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

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

[124] A. Pagh and R. Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[125] A. Pagh, R. Pagh, and M. Ruzic. Linear probing with 5-wise independence. *SIAM Review*, 53(3):547–558, 2011.

[126] R. Pagh. Large-scale similarity joins with guarantees (invited talk). In *LIPIcs-Leibniz International Proc. in Informatics*, volume 31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[127] R. Pagh, N. Pham, F. Silvestri, and M. Stöckel. I/O-efficient similarity join. In *Proc. 23rd European Symp. on Algorithms (ESA)*, pages 941–952, 2015.

[128] R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[129] R. Pagh, M. Stöckel, and D. P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 109–120. ACM, 2014.

[130] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proc. SODA '06*, pages 1186–1195, 2006.

[131] R. Panigrahy, K. Talwar, and U. Wieder. A geometric approach to lower bounds for approximate near-neighbor search and partial match. In *Proc. FOCS '08*, pages 414–423, 2008.

[132] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion. In *Proc. FOCS '10*, pages 805–814, 2010.

[133] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion. In *Proc. FOCS '10*, pages 805–814, 2010.

[134] F. Parvaresh and A. Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *Proc. FOCS '05*, pages 285–294, 2005.

[135] M. Patrascu and M. Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14:1–14:50, 2012.

[136] G. Pólya. Remarks on characteristic functions. In *Proc. Berkeley Symposium on Mathematical Statistics and Probability 1945-1946*, pages 115–123, 1949.

[137] F. P. Preparata and D. V. Sarwate. Computational complexity of Fourier transforms over finite fields. *Mathematics of Computation*, 31(131):740–751, 1977.

[138] M. Pătraşcu and M. Thorup. Dynamic integer sets with optimal rank, select, and predecessor search. In *Proc. FOCS '14*, pages 166–175, 2014.

[139] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Proc. NIPS '07*, pages 1177–1184, 2007.

[140] A. Rubinstein. Hardness of approximate nearest neighbor search. *CoRR*, abs/1803.00904, 2018.

[141] W. Rudin. *Fourier Analysis on Groups*. Wiley, New York, 1990.

[142] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Proc. SIGMOD International Conference on Management of Data*, pages 743–754. ACM, 2004.

[143] I. R. Savage. Mill's ratio for multivariate normal distributions. *Jour. Res. NBS Math. Sci.*, 66(3):93–96, 1962.

[144] J. P. Schmidt, A. Siegel, and A. Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM J. Discrete Math.*, 8(2):223–250, 1995.

[145] B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, Massachusetts, 2002.

[146] G. Seroussi. Table of low-weight binary irreducible polynomials. Technical report, Hewlett-Packard Laboratories, 1998.

[147] V. Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, Cambridge, UK, second edition, 2009.

[148] A. Shrivastava. Simple and efficient weighted minwise hashing. In *NIPS*, pages 1498–1506, 2016.

[149] A. Shrivastava. Optimal densification for fast and accurate minwise hashing. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3154–3163. PMLR, 2017.

[150] A. Shrivastava and P. Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Proc. NIPS '14*, pages 2321–2329, 2014.

[151] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *Proceedings of the 24th International Conference on World Wide Web*, pages 981–991. ACM, 2015.

[152] A. Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, 2004.

[153] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *PVLDB*, 6(14):1930–1941, 2013.

[154] S. J. Szarek and E. Werner. A nonsymmetric correlation inequality for gaussian measure. *Journal of multivariate analysis*, 68(2):193–211, 1999.

[155] A. Ta-Shma, C. Umans, and D. Zuckerman. Lossless condensers, unbalanced expanders, and extractors. *Combinatorica*, 27(2):213–240, 2007.

[156] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

[157] K. Terasawa and Y. Tanaka. Spherical LSH for approximate nearest neighbor search on unit hypersphere. In *Proc. WADS '07*, pages 27–38, 2007.

[158] M. Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 371–380. ACM, 2013.

[159] M. Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. FOCS '13*, pages 90–99, 2013.

[160] M. Thorup. Fast and powerful hashing using tabulation. *Commun. ACM*, 60(7):94–101, 2017.

[161] F. Topsœ. *Some Bounds for the Logarithmic Function*, volume 4, pages 137–151. Nova Science, 2007.

[162] N. G. Ushakov. *Selected Topics in Characteristic Functions*. VSP, Utrecht, The Netherlands, 1999.

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

[164] S. Vijayanarasimhan, P. Jain, and K. Grauman. Hashing hyperplane queries to near points with applications to large-scale active learning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(2):276–288, 2014.

[165] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, Cambridge, UK, third edition, 2013.

[166] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[167] H. W. Watson and F. Galton. On the probability of the extinction of families. *The Journal of the Anthropological Institute of Great Britain and Ireland*, 4:138–144, 1875.

[168] R. Weber, H. Schek, and B. Stephen. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. VLDB '98*, pages 194–205, 1998.

[169] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760. Curran Associates, Inc., 2008.

[170] R. Williams. A new algorithm for optimal constraint satisfaction and its implications. In *Proc. ICALP '04*, pages 1227–1237, 2004.

[171] R. Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In *Proc. SODA '18*, pages 1207–1215, 2018.

[172] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Transactions on Database Systems (TODS)*, 36(3):15, 2011.

[173] H. Zhang and Q. Zhang. Embedjoin: Efficient edit similarity joins via embeddings. In *Proc. KDD '17*, pages 585–594, 2017.

[174] A. L. Zobrist. A new hashing method with application for game playing. *ICCA journal*, 13(2):69–73, 1970.

[175] V. M. Zolotarev. *One-dimensional stable distributions*, volume 65. American Mathematical Soc., 1986.