Optimal Set Similarity Data-structures Without False Negatives

IT University of Copenhagen

Thomas D. Ahle

Januar 1 2017

Abstract

We consider efficient combinatorial constructions, that allow us to partly derandomize data-structures using the locality sensitive framework of Indyk and Motwani (FOCS '98). In particular our constructions allow us to make Zero-Error Probabilistic Polynomial Time (ZPP) analogues of two state of the art algorithms for 'Approximate Set Similarity':

This data-structure problem deals with storing a collection X of sets such that given a query set q for which there exists $x \in P$ with $|q \cap x|/|q \cup x| \geq s_1$, the data structures return $x' \in P$ with $|q \cap x'|/|q \cup x'| \geq s_2$. The first algorithm by Broder et al. [?, ?] introduced the famous 'minhash' function, which in the locality sensitive framework yields an n^{ρ_b} time, $n^{1+\rho_b}$ space data structure for $\rho_b = (\log 1/s_1)/(\log 1/s_2)$. The second by Christiani et al. [?] gives an n^{ρ_c} time $n^{1+\rho_c}$ space data-structure for $\rho_c = (\log 2s_1/(1+s_1))/(\log 2s_2/(1+s_2))$.

Both algorithms use Monte Carlo randomization, but we show that this is not necessary, at least up to $n^{o(1)}$ factors. This settles an open problem from Arasu et al. [?] and Pagh [?] asking whether locality sensitive data-structures could be made *exact* or *without false negatives* other than for hamming distance, and whether a performance gap was needed in the exponent.

The main approach in the thesis is to replace the 'locality sensitive hash functions' or 'space partitions' with 'combinatorial design'. We show that many such designs can be constructed efficiently with the 'multi-splitters' introduced by Alon et al. [?]. We further show that careful constructions of such designs can be efficiently decoded.

We also investigate upper and lower bounds on combinatorial analogues of the minhash algorithm. This is related to the existence of small, approximate minwise hashing families under l_∞ distance.

Contents

Chapter 1

Introduction

Motivation Imagine you are building an iris database of every person in the world and who has ever lived. Because lighting and angles can be different, you store quite a lot of iris scans for each person. One way you might proceed is the following: Using a good theory of irides, and perhaps some statistics and machine learning, you find a set of a million different features that might be present in an iris scan. By Zipf's law [**?**], it is likely that most eyes have only a small subsets of these features; perhaps an average eye has just around a hundred different such distinguishing characteristics. This is great, since your feature computation algorithm only takes time proportional in the output.

Once you have build your database, you want to start connecting it to the world's public surveillance cameras. For each frame of each camera, you identify the iris and calculate the features. Now you need to find a good match in your database. Your first thought is to use high number of shared features as a good measure of similarity, however you quickly realize that this means irides with many features are likely to be a good match with nearly everything. Instead you decide for the Jaccard similarity, which for sets x and y is defined as $|x \cap y|/|x \cup y|$. That is, the number of shared features normalized by the total number of features in the two sets. Now you can run through the entries in your database, compute the similarity and output the best match.

You hook up your video feed to the database and it starts crunching. But something doesn't work! The amount of video frames you receive totally overwhelm your database. Computing the Jaccard similarity to hundreds of billions of sets of features takes maybe a minute!, and you get billions of frames a second. Of course you try to parallelize on multiple computers, but distributing the database is complicated and the amount of computers you need is astronomical.

You call your theory friend, but he tells you that the problem is OVP hard, and can't be solved better than what you're already doing. Instead the honorable doctor tells you, that you can consider approximation. Since your iris pictures are likely to be very similar to the potential matches, but not very similar to the rest of the data-set, you settle on the following model: All irides of the same person have similarity greater than s_1, why irides of different people have similarity less than s_2. Perhaps you have $s_1 = 7/8$ and $s_2 = 1/10$. Thus you can relax your requirements and look for approximate data-structures.

You look around, and find a nice one: Minhash LSH. It promises query time $n^{\frac{\log 1/s_1}{\log 1/s_2}} < n^{0.058}$. With $n = 10^9$, this value is less than 4! Great you can use this... except: The data-structure only gives probabilistic guarantees! No matter how you configure the data-structure, there is a chance, albeit small, that it won't return a match, even if there is one! This is a deal breaker: If an unknown person shows up at your door, you need to know exactly where she has been before. If the system suddenly has a one out of a thousand failure and decides it hasn't seen her before, you may be in great danger.

You ask your theory friends again, but they tell you that this is an open problem, and we don't know whether any algorithms exists that can solve nearest neighbor problems with exact guarantees (without false negatives), unless we are willing to suffer (sometimes a lot) in the performance.

Luckily today this changes. At least in theory. You probably don't want to implement what is presented in this thesis. Or maybe you would. I don't know, I am neither a practitioner nor an evil dictator.

Introduction During the last decade or so, it has become possible to collect and process very large sets of data. This change has been felt in most areas of computer science, such as image processing, databases, medical computing, machine learning and natural language processing. A big part of this progress has been driven by better algorithms, and in particular the emergence of high dimensional, approximate geometric algorithms, such as Locality Sensitive Hashing, LSH. These algorithm have allowed beating the 'curse of dimensionality', which is a phenomenon often present in precise data-structures such as KD-Trees, Ball-Trees, and the like. Locality sensitive algorithms all have one big downside compared to earlier data-structures: They only give probabilistic bounds.

Such algorithms are known as Monte Carlo, and belong to the complexity class RP of polynomial algorithms with one sided error. If the answer to the question 'is there a near point' is no, they always answer 'no', but if it is yes, they sometimes answer 'no' as well. In contrast, the complexity class ZPP describes problems solvable by polynomial (expected) time, also know as Las Vegas algorithms. It is not generally known whether RP = ZPP, but so far it seems that the use of randomization improves what is efficiently possible.

In this thesis we show that having an error probability is not needed for LSH on Jaccard similarity, and strongly indicate that it isn't needed for other known LSH or LSF (Locality Sensitive Filters) data-structures either. The first indication in this direction was [**?**].

We solve the problem by constructing strong, pseudo-random space partitions. Thus our solution may be applicable to other problems in high dimensional geometry, such as sphere packing.

1.1 The Set Similarity Problem

Note that we'll sometimes talk about the 'exact approximate set similarity' problem, which is the approximate set similarity problem without false negatives. The 'exact' problem thus shouldn't be confused with the 'precise' problem defined below, in which the word 'precise' refers to the lack of approximation. There can thus be precise algorithms that are not exact and vice versa.

Definition 1 (Precise similarity search)**.** *Let $X \subset U$ be a set of data sets (or points) with $|X| = n$. Let $sim : U \times U \to [0, 1]$ be a 'similarity measure' on U. A solution to the s-precise similarity search problem is a data-structure on X, such that given a query $q \in U$, it can return a point $x \in X$ with $sim(q, x) \geq s$ if one exists.*

The most common similarity measure we'll consider is the 'Jaccard Similarity'. This is defined between two sets $x, y \subseteq \{1, \dots, d\}$ as $\text{sim}(x, y) = |x \cap y|/|x \cup y| \in [0, 1]$. Set similarity over Jaccard Similarity is closely related to Containment Search [**?**] and Maximum Inner Product Search [**?**, **?**], in that those problem can be reduced to doing (at most $O(d)$) queries on a Jaccard similarity data structure.

In general for this thesis, we'll assume sets have cardinality t. Sometimes we'll need to solve the more general problem of varying cardinalities in order to solve the t-version, and then we'll do that.

1.1.1 Precise Algorithms

There are three common algorithms for set similarity, neither of which are very advanced, but all of which turns out to be optimal for certain ranges of parameters.

The first one is 'brute force' and consists of simply enumerating the data set X, calculating all similarities. This takes space $O(n)$ and time $O(n)$, considering space usage per set and similarity calculation time constant. We can also get time $O(1)$ by writing down the answers to all 2^d possible queries in space $O(2^d)$. Note that always $n \leq 2^d$, since otherwise we could just deduplicate the input.

The third algorithm we'll call 'meet in the middle':

Theorem 1. *For a set $X \subseteq \{1, \dots, d\}$, as in definition 1, let all $x \in X$ have $|x| = t$. There is an algorithm with query time $\binom{t}{a}$ and space $n\binom{t}{a}$, where $a = |x \cap y| = \frac{2s}{1+s}t$.*

Proof. We'll consider X as a subset of $\{0,1\}^d$ of size n.

Building the data structure: Given X, we create a single hash table T with keys from $\{0,1\}^d$ with hamming weight a. For each point $x \in X$, we store x in $T[x']$ for all subsets $x' \subseteq \binom{x}{a}$. Note that we allow multiple points to be stored in the same hash table bucket.

Querying the data structure: Given a point $q \subseteq \{1, \dots, d\}$, we look at each $T[q']$ for each $q' \subseteq \binom{q}{a}$. If any $T[q']$ is non-empty, we return a point from the bucket.

To see that the algorithm is correct, we need to show that for any x, q with $\mathrm{sim}(x,q) \geq s$, there is a bucket $T[\cdot]$ in which both are present. But this follows, $|x \cap q| \geq a$ and so there is a subset $s' \subseteq x \cap q$ of size a such that x and q are both in $T[s']$.

The query time is simply $\binom{t}{a}$, since we can find all subsets in time proportional to the output, and we only have to do constant work per subset. The space usage per point is also $\binom{t}{a}$, since that is the number of buckets each point is stored in. □

We may note, that this data structure has better query time than the brute force approach, whenever $a < \frac{\log n}{\log t}$. However for, say, $d = \sqrt{n}$ and $t = d/3$, it only gives good results for $a \leq 2$.

'Meet in the middle', like many of the algorithms we'll discuss, allow a trade-off between space usage and query time usage. However, for simplicity, we'll only discuss the 'balanced' case where space usage equals n times the query time. We'll also assume that the $n \log_2 \binom{d}{t}$ bits of memory required to store the data points is given for free, so that we may simply refer to points with unit memory pointers. Finally we'll assume that $\mathrm{sim}(x,y)$ can be computed in unit time, so the brute force algorithm from before becomes n space and n query time.

1.1.2 Hardness of Set Similarity

We may wonder if it's possible to do better than the two algorithms described above. Unfortunately it turns out, that unless some very widely hold conjectures are wrong, we won't be able to do any better than brute force, at least for $d, t, a = \omega(\log n)$.

Ahle et al. [?] showed a lower bound for data structures supporting maximum inner product search. They used the following conjecture, originally proposed in [?], who showed that it was at least as strong as the strong exponential time hypothesis.

Conjecture 1 (Orthogonal Vectors (Problem), OVP)**.** *For any $\epsilon > 0$ it's conjectured that there is no $n^{2-\epsilon}e^{o(d(n))}$ time algorithm (randomized or not) for the following problem: Given $X \subseteq \{0,1\}^d$ with $|X| = n$, determine if there are two orthogonal vectors in X. That is $x, y \in X$ with $\langle x, y \rangle = 0$.*

Ahle et al.'s proof doesn't directly apply to our case, since their vectors didn't necessarily have fixed weight t. We can however easily adapt their proof, and show the following:

Lemma 1 (OVP for Data-structures)**.** *For any $\epsilon > 0$, the Orthogonal Vector conjecture is false, if there exists a data structure for the data-structure problem: Given $X \subseteq \{0,1\}^d$, create a data structure in time $n^{O(1)}e^{o(d)}$ such that queries on the following form, can be answered in time $n^{1-\epsilon}e^{o(d)}$: For a point $q \in \{0,1\}^d$, determine if X contains a vector orthogonal to q. That is an $x \in X$ s.t. $\langle q, x\rangle = 0$.*

Note that the original theorem has $d^{O(1)}$, rather than $e^{o(d)}$, but this isn't necessary for the proof.

We'll show that a data structure for the set similarity problem gives a data structure for the orthogonal vectors problem with the same construction and query time. Hence any algorithm for the precise set similarity data-structure problem must use $n^{1-o(1)}$ query time, or have an exponential dependency on d. This means the 'brute force' and 'meet in the middle' algorithms we discussed are optimal.

To be formal:

Theorem 2 (OVP hardness)**.** *Assume the Orthogonal Vector conjecture and let $\epsilon > 0$. Then there is no data-structure for the precise similarity search problem with construction time $n^{O(1)}e^{o(d)}$ and query time $n^{1-\epsilon}e^{o(d)}$.*

Proof. Consider the following 'coordinate gadgets' from $\{0,1\}$ to $\{0,1\}^4$:

$$\hat{f}(1) := (1,1,0,0) \qquad \hat{g}(1) := (0,0,1,1)$$
$$\hat{f}(0) := (1,0,1,0) \qquad \hat{g}(0) := (0,1,1,0).$$

These have the property that for any $\hat{x}, \hat{y} \in \{0,1\}$, we have that $|\hat{f}(\hat{x})| = |\hat{g}(\hat{y})| = 2$ and $\langle \hat{f}(\hat{x}), \hat{g}(\hat{y})\rangle = 1 - \hat{x}\hat{y}$. By concatenating, we can create embeddings from $\{0,1\}^d$ to $\{0,1\}^{4d}$:

$$f(x) := \hat{f}(x_1)\hat{f}(x_2)\dots\hat{f}(x_d)$$
$$g(x) := \hat{g}(x_1)\hat{g}(x_2)\dots\hat{g}(x_d).$$

These have the property that for any $x, y \in \{0,1\}^d$ (of any sizes $|x|$ and $|y|$) we have that $|f(x)| = |g(y)| = 2d$ and $\langle f(x), g(y)\rangle = d - \langle x, y\rangle$. In particular for any orthogonal x, y, we get $\langle x, y\rangle = d$, while for any non-orthogonal x, y we get $\langle x, y\rangle \le d - 1$.

We can now use a data-structure for the precise similarity search problem, with parameters $d = 4d'$, $t = 2d'$ and $s = 1/2$, to create a data structure for the OVP problem. This is done by mapping the data points with f and the queries with g. Both of these operations take time linear in d, which is less than $e^{o(d)}$. □

1.1.3 Approximate Formulation

Hardness results similar to the above have forced researchers to consider weaker formulations of many problems in high dimensional geometry.

In particular we will consider the following adaption of the similarity search problem from definition 1:

Definition 2 (Approximate Similarity Search, ASS)**.** *Let $X \subset U$ be a set of data points with $|X| = n$. Let $sim : U \times U \to [0,1]$ be a 'similarity measure' on U. A solution to the (s_1, s_2)-approximate similarity search problem is a data-structure on X, such that given a query $q \in U$, it can return a point $x \in X$ with $sim(q,x) \ge s_2$ if one x' exists with $sim(q,x') \ge s_1$.*

Such a data-structure is allowed to always answer a query q with a set x, if $\text{sim}(x,q) \ge s_2$, but it doesn't have return anything at all, unless there is a point x' with $\text{sim}(x', x) \ge s_1$. Another way to look at this is as a promise problem: The data-structure is given the promise that there is a gap $s_1 - s_2$ between the points we're interested in, and the points we're not interested in.

An important note with regards to this thesis is the use of randomization. We will allow Las Vegas-type randomization, that is our running times and space bounds only have to be in expectation. Nearly all papers also allow Monte-Carlo type randomization, that is, the algorithm only has to succeed with some high probability, even when there is a data point with very high similarity to the query. In this thesis however, we want 100% success probability, when there is a point to return. That is, we don't want any false negatives.

1.2 Related Work

The most well studied problem in high dimensional data-structures is probably that of 'nearest neighbors'. Similarity problems as described above, can usually be recast as distance problems where near neighbors correspond to points with high similarity.

The algorithms of [?, ?, ?, ?] were the first to find algorithms without exponential dependency on the dimension. The key ingredient was to use 'locality sensitive' hash functions with the property that near points are likely to hash to the same value. In a different terminology, they used a 'grid like' high dimensional space partition, in which near points are likely to end up in the same regions. A key challenge has been to find the best space partitions for different distance and similarity measures.

A later approach has been making space partitions based on the input data set [?, ?, ?]. This is natural, when comparing with Voronoi diagrams, which are space partitions in which a query point is guaranteed to collide only with its nearest neighbor. We can't however use exactly the Voronoi partition, as it can't be decoded efficiently.

What all these methods have in common, however, is that they use randomization in the construction of their space partitions. They do this because it turns out to be quite hard to produce good point lattices in high dimensions. For example, the natural 'square grid' construction is far from optimal.

1.2.1 Near Neighbors without False Negatives

While Indyk et al. were the first to get provable performance guarantees on high dimensional near neighbor problems, Arasu et al. [?] were the first to do so with a Las Vegas algorithm. That is, all previous algorithms either had an exponential dependency on d, or they were only correct 'with high probability'.

Their algorithm was for the 'set-similarity join' problem, but we'll consider it as a data-structure problem for consistency with the thesis. In particular they wanted to find points with low hamming distance in $\{0,1\}^d$. For this problem a typical hash function is to map a vector to a small random subset of its coordinates. If the subset has size k, and the two vectors have distance r, the vectors collide with probability $\binom{d-r}{k}/\binom{d}{r} \approx (r/d)^k$.

Arasu et al.'s approach was to consider *all* subsets rather than a smaller set of random ones. This guarantee that a collision must happen at some point. In order to reduce the number of subsets needed to consider, they reduce the dimension to d/B, by partitioning the coordinates in B blocks. This guarantees that the distance between x and y is at least r/B in the of the blocks. The performance achieved is not close to the randomized $n^{1/c}$, but for $c > 7.5$ the authors claim performance of $O(r^{2.39})$.

Pagh greatly improved on this constructions by using Hadamard codes for constructing efficient covering designs for many sets of parameters. Such a design corresponds to a sets of k-sets, such that any r set is contained in a k set. Their construction is within a factor $\log 4 < 1.4$, in the exponent, from the optimal, and matches the optimal when $cr = (\log n)/k$ for some integer k.

It is possible to reduce Jaccard similarity to hamming distance, but only either incurring a loss, or using randomization, which removes the entire point what we are trying to do. On the other hand, our methods (though not our results) also enable us to solve the hamming distance problem within $n^{o(1)}$ factors of the optimal $n^{1/c}$.

1.2.2 The Minhash Algorithm

Broder et al. [**?**] introduced the minhash technique for the search engines at Alta Vista. The idea is to sample a uniformly random hash function $h : \{1, \dots, n\} \to [0, 1]$. A set $x \subseteq \{1, \dots, n\}$ can then by mapped to its smallest element under h: $\arg\min_{i \in x} h(i)$. The collision probability, for two distinct sets, under a random h is the Jaccard similarity $\Pr_h[\arg\min_{i\in x} h(i) = \arg\min_{i\in x} h(i)] = |x \cap y|/|x \cup y| = s$.

We can think of h as a space partition with n regions. It isn't however a completely well balanced partition, since the element that maximizes h will only be the hash value if it is the only element in the set hashed.

If we want to further decrease the probability that two non-similar sets collide, we can pick multiple hash functions h_i and map $x \subseteq \{1, \dots, n\}$ to the vector $v(x) = (\arg\min_{x\in X} h_1(x), \dots, \arg\min_{x\in X} h_k(x))$. We may notice, that this is also an embedding of Jaccard similarity into hamming distance. The distance between two vectors $v(x), v(y)$ is going to be closely concentrated around ks, where s is the Jaccard similarity between x and y. This embedding allows using various near neighbor data-structures, and things like SVM's for classification, as it was done in [**?**].

Using minhash in the framework of Indyk et al., produces a data-structure for the approximate set similarity problem with query time $O(n^\rho)$ and space usage $O(n^{1+\rho})$ where $\rho = \log 1/s_1 / \log 1/s_2$.

There has been work on making minhash more realistic, by reducing the amount of randomness needed when sampling the hash functions [**?**]. Unfortunately it appears [**?**, **?**, **?**, **?**, **?**, **?**, **?**] hard to get as good results as for normal hash functions. There are no known k-wise independent permutation families, for $k \geq 4$. In fact it is known that for $n \geq 25$ and $k \geq 4$ there are no non-trivial k-wise independent subgroups of S_n [**?**].

Instead people have looked at so called ϵ approximate, k-wise independent families. While these are useful for derandomization, they are still much too large for our purposes.

1.2.3 Optimal Similarity Data-Structure

The recent algorithm by Christiani and Pagh [**?**] showed that, surprisingly, the Minhash algorithm is not optimal for Approximate Set Similarity data-structures. They found an algorithm in the locality sensitive filter framework, which they proved to be optimal for data insensitive space partitioning algorithms, by reducing to the LSH lower bound by O'Donnel et al. [**?**].

Christiani and Pagh found some very interesting techniques to improve the practical performance of their algorithm. We will briefly sketch the 'raw' version, which doesn't include those techniques, since it is more similar to our final precise construction.

Given n data sets, $X \subseteq \{0, 1\}^d$, each of size t, and Jaccard similarity thresholds s_1 and s_2, we make the following variables:

$$b_1 = \tfrac{2s_1}{1+s_1}, b_2 = \tfrac{2s_2}{1+s_2}, \rho = \tfrac{\log 1/b_1}{\log 1/b_2}, s = \tfrac{\rho \log n}{\log d/(b_1 t)}, u = \tfrac{\log d/(b_1 t)}{\log 1/b_1} \text{ and } m = \left(\tfrac{d}{b_1 t}\right)^s (\log u + 1).$$

The later three variables must be integers, but we suppress the rounding in this presentation for easy of presentation.

We sample uniformly and independently u sets of m points from $\{0, 1\}^d$, $S_1, S_2, \dots, S_u$. For any point $x \in \{0, 1\}^d$, we define the function $\text{decode}(x) = \{s \in S_1 \mid x \in s\} \times \cdots \times \{s \in S_u \mid x \in s\}$. Note that $\text{decode}(x) \cap \text{decode}(y) = \text{decode}(x \cap y)$.

The data-structure consists of a hash-table, T, in which we store each $x \in X$ in every bucket $T[k]$ for $k \in \text{decode}(x)$. Note that each bucket may store multiple points. Querying the data-structure with a point $q \in \{0,1\}^d$ consist of simply looking at every point x in every bucket $T[k]$ for $k \in \text{decode}(q)$. As soon as $\text{sim}(q,x) \geq s_2$ we return x.

This data-structure solves the approximate set similarity problem, if we can guarantee that all pairs x, y with $\text{sim}(x,y) \geq s_1$ have $\text{decode}(x) \cap \text{decode}(y) = \text{decode}(x \cap y) \neq \emptyset$. We show that this is the case with high probability. Note that $|x \cap y| \geq 2s_1 t/(1+s_1) = b_1 t$ by definition of Jaccard similarity. Because the point sets $S_1, \dots, S_u$ are sampled uniformly and independently, we have

$$\Pr[\text{decode}(x,y) \neq \emptyset] = (1-(1-(b_1t/d)^s)^m)^u \geq 1 - ue^{-(b_1t/d)^s m} = 1 - e^{-1} \geq 5/8.$$

Which is sufficient for our purposes.

The processing time per query is (1) the time for evaluating decode (2) the time for reading each hash-table bucket from memory, and (3) the time for calculating the similarity to the points in the buckets. For (1) it takes time $O(m)$ to calculate each of the sets $\{s \in S_i \mid x \in s\}$. There are u such sets to be calculated, but after that, taking the product takes time proportional to the output size, which is (2). For (2) we calculate the expected number of buckets we have to look at. That is by linearity of expectation

$$E[|\text{decode}(x)|] = m^u(t/d)^{su} = \left(\frac{1}{b_1}\right)^{su} (\log u)^u = n^\rho (\log u + 1)^u.$$

Finally for (3), since we return as soon as we see a point x with $\text{sim}(q,x) \geq s_2$, we have to pay only for 'collisions' with points y such that $\text{sim}(q,y) < s_2$, that is $|q \cap y| < b_2 t$. Again by linearity of expectation, this is

$$nE[|\text{decode}(q \cap y)|] = nm^u(b_2t/d)^{su} = n^\rho(\log u + 1)^u.$$

The total query time (1)+(2)+(3) is thus bounded by $O(n^\rho(\log u+1)^u) \leq O(n^\rho(d/t)^{\log\log\log(d/t)})$. By sampling we can assume that d/t is logarithmic in n, such that the last factor is $n^{o(1)}$. Building the data-structure takes n times (1)+(2) which is $O(n^{1+\rho+o(1)})$.

1.2.4 Combinatorial Design and K-Restrictions

Our approach to derandomization is heavily inspired by the field of Combinatorial Design and the problem of k-restrictions. Thus we'll review a few important results.

Combinatorial Design concerns itself with properties of systems of finite sets, and more generally combinatorial objects, such as tournament scheduling, lotteries and group testing schemes. The first example of looking for such objects is the 'magic square', in which you are asked to put then numbers 1 to 9 in a 3 by 3 grid, such that each row and column sum to 15. The Handbook of Combinatorial Designs [?] surveys many different results, existential and constructive.

Many modern results are based on finite geometries, such as the Fano-plane, which is the sets $\{\{0,1,2\}, \{0,3,4\}, \{0,5,6\}, \{1,3,5\}, \{1,4,6\}, \{2,3,6\}, \{2,4,5\}\}$ interpreted as 'lines' in a two dimensional projective geometry. These have all the usual properties of lines (in projective geometry): Any two lines share a point; and any two points share a line. The Fano-plane is, among other things, the solution to Kirkman's Schoolgirl Problem, which states: "Fifteen schoolgirls walk each day in five groups of three. Arrange the girls' walk for a week so that in that time, each pair of girls walks together in a group just once." [?] Such geometries can also be used to construct Covering designs, which are used in work on exact bit-sampling LSH by Pagh [?].

Another type of general combinatorial construction, useful for derandomization, is the k-restriction problem:

Definition 3 (k-restriction problem). *k-restriction problems are as follows:*

1. *The input is an alphabet Σ of size $|\Sigma| = q$, a length m and a set of s possible demands $f_i : \Sigma^k \to \{0,1\}, 1 \leq i \leq s$. For every i there exists $x \in \Sigma^k$ so that $f_i(x) = 1$.*

2. *The task is to prepare a set $\mathcal{A} \subseteq \Sigma^m$ so that: For any choice of k indices $1 \leq i_1 < \cdots < i_k \leq m$, and a demand $j, 1 \leq j \leq s$, there is some $x \in \mathcal{A}$, such that $f_j(x_{i_1} \cdots x_{i_k}) = 1$.*

Given a probability distribution, $D : \Sigma^m \to [0,1]$, the density of a k-restriction problem with respect to D is $\epsilon := \min_{\substack{1 \leq i_1 < \cdots < i_k \leq m \\ 1 \leq j \leq s}} \Pr_{x \sim D}[f_j(x_{i_1} \cdots x_{i_k}) = 1]$.

This was first studied by Naor et al. [?] who found a "a fairly general method" for finding deterministic constructions for k-restrictions. They used their methods to construct (n,k)-universal sets (a collection of binary vectors of length n such that for any subset of size k of the indices, all 2^k configurations appear) and families of perfect hash functions.

The main approach by Naor et al., and later Alon et al, is to note that many solutions to k-restriction problem can be constructed efficiently by random sampling. Some can also be sampled from small sample spaces, which allows the general theorem:

Theorem 3 ([?], Theorem 1)**.** *Fix some efficiently approximateable probability distribution D. For any k-restriction problem with density ϵ with respect to D there is an algorithm, that given an instance of the problem and an accuracy parameter $0 < \delta < 1$, obtains a solution of size at most $\lceil \frac{k \log m + \log s}{(1-\delta)\epsilon} \rceil$ in time poly$(s, m^k, q^k, \epsilon^{-1}, \delta^{-1})$.*

Naor et al. further improved their constructions with what they called an '(n,l)-splitter'. The is the set of all surjective, non-decreasing functions $\pi : \{1,\ldots,n\} \to \{1,\ldots,l\}$. There are $\binom{\{n}{l-1\}}$ such function, as we can see by picking $l-1$ 'separators' and 'filling' the space in between with the numbers $1,\ldots,l$. We can notice that for any $K \subseteq \{1,\ldots,n\}$ of size k, there is a π in the splitter, such that $\lfloor k/l \rfloor \leq K \cap \pi^{-1}(i) \leq \lceil k/l \rceil$ for all $i \in \{1,\ldots,l\}$.

The notion of splitters was generalized by Alon et al. [?], who defined the multi-way splitter. The construction is similar to the 1-way splitter by Naor, but require a much more elaborate proof, through the topological necklace theorem by Alon.

Definition 4 (multi-way splitter, [?])**.** *A partition of m coordinates into b blocks is a function $\pi : [m] \to [b]$, assigning each coordinate the block that contains it. We say π splits a subset $I \subseteq [m]$ of the coordinates, if every block $1 \leq j \leq b$ sees the same number of coordinates up to rounding, i.e $\left\lfloor \frac{|I|}{b} \right\rfloor \leq |\pi^{-1}(j) \cap I| \leq \left\lceil \frac{|I|}{b} \right\rceil$.*

Let m, b be natural numbers. A t-way splitter *is a list A of partitions $\pi : [m] \to [b]$, s.t. for any series of t pairwise disjoint subsets of coordinates $S_1,\ldots,S_t \subseteq \{1,\ldots,m\}$, there exists a partition in A that splits each one of $S_1,\ldots,S_t$.*

Lemma 2 ([?])**.** *There exists an explicit construction of a t-way splitter that splits m indices into b blocks, and has size $b^{p+1}\binom{m}{p}$ where $p = (b-1)t$.*

In this thesis, we will use k-restrictions when we can, since they help derandomize our constructions, but the focus of the paper is not really derandomization, and we'll sometimes use large amounts of random bits.

The splitters and multi-way splitters on the other hand, are going to be of crucial importance.

1.3 Contributions

The main contribution of the thesis is the observation that multi-way splitters can be used to bridge the gap between $\log n/\log\log n$ size 'verifiable random constructions' and $\log n$ size constructions, which are needed for optimal, exact 'locality sensitive hashing' data structures.

This has been an important open problem since [?], but no non-trivial results were known until [?].

In addition we show that so called ‘locality sensitive filter’ data structures can also be optimally derandomized. This is a natural extension to the first problem, but requires more work, since ‘filter’ data structures can’t just enumerate a big space partition generated in advance. Filters tend to use space partitions that are so large, that an efficient ‘decoding’ algorithm must be supplied. We show that it is indeed possible to use splitters even with this restriction.

The thesis further introduces the problem of bottom-k designs. This is a combinatorial analog of the minhash data structures, and related to the important problem of finding small sample spaces with minwise independence. The thesis uses Ramsey theory to show the first (obviously) non-trivial lower bounds for these structures. The lower bounds are shown to be sharp up to $1+o(1)$ factors by constructing matching upper bounds based on incidence matrices of permutations.

Finally we show a very useful bound for the ratio $\binom{n}{k}/\binom{m}{k}$, which might be useful generally in computer science.

1.4 Acknowledgments

While this thesis has only been a short project, I would like to thank Rasmus Pagh for creating a great research environment at the ITU. I would also like to thank him for inspiring me to work on derandomizing LSH, as well as many good discussions on combinatorial design.

Further, I would like to thank Freja Elbro for early discussions of bottom-k designs, and the people of Nebbegårdsbakken 52 for letting me practice explaining the topic to a larger audience.

At ITU I am grateful to Tobias Christiani for sharing a preprint of their paper ‘Beyond Minhash for Similarity Search’, to Johan von Tangen Sivertsen for proofreading, to Matteo Dusefante and Christopher Lee for providing great music and atmosphere while writing this thesis, and to Martin Aumüller and the rest of the Scalable Similarity Search group for discussions and comments.

Chapter 2

Algorithms with Bottom-k Designs

We define a new kind of combinatorial design, which we will call a bottom-k design.

Definition 5 (Bottom-k design)**.** *Let $n \geq a, b$ and $a \geq k$. An (a, b, n)-bottom-k design is a set of permutations, Π, on $\{1, \dots, n\}$, such that for any disjoint $A, B \subset \{1, \dots, n\}$ of size $|A| = a$ and $|B| = b$, there is a permutation $\pi \in \Pi$ and a subset $K \subseteq A$ of size $|K| = k$, such that π orders every element of K before any element of B.*

We will think of $\pi \in \Pi$ as functions $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$, and use the notation $\pi(K) < \pi(B) \equiv \forall_{k \in K, b \in B} \pi(k) < \pi(b)$.

The bottom-k number, $B_k(a, b, n)$, is the smallest number of permutations in any (a, b, n) bottom-k design. A design is optimal, if it has the minimum possible number of permutations.

Example 1: The permutations $\{(1, 4, 2, 3), (2, 4, 1, 3), (3, 4, 1, 2)\}$ are a $(1, 2, 4)$-bottom-1 design. We can see that if $A = K$ is any of $\{1\}$, $\{2\}$, $\{3\}$, then one of the permutations order K before anything else. On the other hand, if $A = K = \{4\}$, then for any pair of $1, 2, 3$ there is a permutation where 4 is before both of the numbers.

Example 2: The permutations $\{(1, 2, 3, 4, 5, 6, 7), (7, 6, 5, 4, 3, 2, 1)\}$ are a $(3, 1, 7)$-bottom-2 design. Since B has only one element and is disjoint from A, there must to be two elements of A with a numerical value either higher or lower than that of B. Hence, by taking Π to be any permutation and its reverse, we get a design. Note that for any k and $n \geq 2k$, we can make a $(2k - 1, 1, n)$-bottom-k design using this construction.

We will later delve more on the exciting properties, constructions and limits of bottom-k designs, but first we will show how they may be used to solve the approximate set semilarity problem without false negatives.

2.1 A Non-constructive Algorithm

Recall the problem from definition 2: From the universe $U \subseteq \{1, \dots, d\}$, we are given a set of sets $X \subseteq U$ of size $|X| = n$. We are further given two Jaccard-similarity thresholds s_1 and s_2. Our goal is to create a data structure which, given a query $q \subseteq \{1, \dots, d\}$, returns a point x with $\text{sim}(q, x) \geq s_2$, although only if X contains some (possibly different) x' with $\text{sim}(q, x') \geq s_1$.

Theorem 4. *Assume data points $x \in X$ all have size t_x and all queries $q \in U$ have size t_q. Given a (a, b, d) bottom-k design, Π, where $a = \frac{s}{1+s}(t_x + t_q)$, $b = \frac{1-s}{1+s}(t_x + t_q)$ and $k = \lceil \log n / \log 1/s_2 \rceil$, there is a data-structure for the (s_1, s_2)-exact similarity search problem over $\{1, \dots, d\}$ with Jaccard similarity.*

The data-structure takes $O(|\Pi|)$ space and queries can be performed in time $O(|\Pi| n)$.

In the theorem we expect that s_1 and s_2 are chosen so a and b are integers. We'll usually assume that $s_1, s_2 = \Theta(1)$ such that $k = \omega(1)$ in which case the ceiling shouldn't make a big difference for most constructions. If we want to support more than one set of values t_q, t_x we can build d^2 data structures, which will be insignificant when $d = n^{o(1)}$.

Proof. The data structure works like this:

Building the data structure: We first pick a permutation $\sigma : U \to U$ uniformly at random. Then, for each π in Π we create a hash-table T_π. For each $\pi \in \Pi$ and $x \in X$, we store a pointer to x in $T_\pi[h_\pi(\sigma x)]$. Here $h_\pi : U \to \{1, \dots, d\}^k$ maps x to the set of its first k elements under π.

For example, if $k = 2$, $\pi = (5, 4, 1, 2, 3)$ and $x = \{1, 3, 5\}$, then $h_\pi(x) = \{1, 5\}$, since 5 and 1 are ordered before 3. If $\sigma = (2, 1, 3, 5, 4)$, then $h_\pi(\sigma x) = h_\pi(\{2, 3, 4\}) = \{2, 4\}$, since σ maps 1 to 2, 3 to 3 and 5 to 4.

The permutation σ has no impact on correctness, but is important in order to get provable performance guarantees against worst case input. Note that $h_\pi(\sigma x) = h_{\pi\sigma}(x)$, where $\pi\sigma$ is the composition of π and σ.

Querying the data structure: Given a set $q \subseteq \{1, \dots, d\}$, we look at $T_\pi[h_\pi(\sigma q)]$ for each π in Π. For each set $x \in T_\pi[h_\pi(\sigma q)]$, we calculate the similarity $\text{sim}(q, x)$, and if it's at least s_2, we return x. Any set x that shares a bucket with q is called a 'collision'.

Correctness To see that the algorithm is correct, we first note that the algorithm can never return a set x with $\text{sim}(q, x) < s_2$. Thus, to show correctness, we just need to show that if a point $x \in X$ has $\text{sim}(q, x) \geq s_1$, then it collides with q in one of the tables, and thus has the potential of being returned.

For this, note that if x has similarity exactly s_1 to q, then $|x \cap q| = a$ and the symmetric difference $|x \triangle q| = |x \cup q \setminus x \cap q| = t_x + t_y - 2a = b$. The permutation of x and q under σ doesn't change any of these sizes. By the definition of bottom-k designs (definition 5), this means that there is some subset $K \subseteq x \cup q$ with $|K| = k$ and $\pi \in \Pi$ such that $\pi(K) < \pi(x \triangle q)$. Because of the way h_π is chosen, this means that $h_\pi(q) = h_\pi(x)$, which is the collision we needed.

We also need to have a collision when the similarity $\text{sim}(q, x)$ is strictly greater than s_1. In these situations we have $|x \cap q| \geq a$ and $|x \triangle q| \leq b$. In the next section we'll show that any (a, b, n)-bottom-k design is also a $(a + 1, b, n)$ and $(a, b - 1, n)$ design, which solves the problem.

Analysis The algorithm is dominated by three parts: (1) Calculating h_π of various points, (2) accesses in the tables T_π, and (3) computing similarities with collisions.

As in the rest of the thesis, we assume calculating $\text{sim}(x, y)$ and $h_\pi(x)$ takes unit time. In practice, with a little preprocessing, both calculations are bounded by time t.

We calculate h_π exactly once for each table access, so (1) and (2) are both $O(|\Pi|)$. For (3) we calculate the expected number of collisions. We only have to care about collisions with low similarity points, since as soon as we see an x with $\text{sim}(q, x) \geq s_2$, we can safely return it. Using lemma 3 below, we can bound the expected collisions as follows:

$$\begin{aligned}
E_\sigma \sum_{\substack{\pi\in\Pi,\\ x\in X, \text{sim}(x,q)<s_2}} [h_\pi(\sigma q) = h_\pi(\sigma x)] &= \sum_{\substack{\pi\in\Pi,\\ x\in X, \text{sim}(x,q)<s_2}} \Pr_\sigma[h_{\pi\sigma}(q) = h_{\pi\sigma}(x)] \\
&= |\Pi| \sum_{x\in X, \text{sim}(x,q)<s_2} \Pr_\sigma[h_\sigma(q) = h_\sigma(x)] \\
&= |\Pi| \sum_{x\in X, \text{sim}(x,q)<s_2} \binom{|x \cap q|}{k} \Big/ \binom{|x \cup q|}{k} \\
&\leq |\Pi| n \binom{a}{k} \Big/ \binom{a+b}{k}
\end{aligned}$$

By theorem 14, the binomial ratio is bounded by $(\frac{a}{a+b})^k = s_2^k$. Thus the total expected time spent on (3) is at most $n|\Pi| s_2^{\lceil \log n / \log 1/s_2 \rceil} \leq |\Pi|$,

This shows that (1), (2) and (3) are all $O(|\Pi|)$, giving the query time in the theorem. When building the data-structure, steps (1) and (2) are repeated n times, storing Π pointers at each iteration. This gives the space usage and building time. □

Lemma 3. *Given sets $A, B \subseteq \{1, \ldots, n\}$ of size a and b respectively, the probability that a random permutation on $\{1, \ldots, n\}$ orders some size k subset of A before all of B is*

$$\Pr_{\pi}[\exists_{K \subseteq A, |K|=k} \pi(K) < \pi(B)] = \binom{a}{k} \Big/ \binom{a+b}{k}. \tag{2.1}$$

Proof. When calculating the probability, we can ignore all the elements of $\{1, \ldots, n\}$ not in A or B. That is, we can consider π to be a random permutation over $A \cup B$. Then the first k elements of π must be from A, which can be done in $\binom{a}{k}k!$ ways. In total, the first k elements of π may be chosen in $\binom{a+b}{k}k!$ different ways. Since π is uniformely random, it takes all of these prefixes with the same probability, which gives the lemma. □

2.2 Properties, Bounds and Constructions

As promised we'll note the following monotonicities on the optimal size of (a, b, n)-bottom-k designs, $B_k(a, b, n)$:

Proposition 1.

$$B_k(a, b, n) \geq B_k(a+1, b, n) \tag{2.2}$$
$$B_k(a, b, n) \leq B_k(a, b+1, n) \tag{2.3}$$
$$B_k(a, b, n) \leq B_k(a, b, n+1) \tag{2.4}$$
$$B_k(a, b, n) \leq B_{k+1}(a, b, n) \tag{2.5}$$

Proof. Inequality (2.2) follows, since a permutation $\pi \in \Pi$ that orders k elements from A before B will order the same k elements before B if an element is added to A. For (2.3) note that a permutation that works for some A and B will also work, when removing an element from B. (2.4) corresponds to removing an element from the universe. This means some pairs A, B will no longer exist, thus only removing restrictions for our design. Finally for (2.5), if π orders some $k+1$ elements of A before B, it also orders k elements. □

The most general bound is given by the probabilistic method:

Proposition 2.

$$B_k(a, b, n) \leq \left\lceil \binom{a+b}{k} \Big/ \binom{a}{k} \log\left(\binom{n}{a}\binom{n-a}{b}\right) \right\rceil \leq \left\lceil \left(\frac{a+b}{a}\right)^k e^{k^2/a}(a+b)\log n \right\rceil$$

Proof. Form Π by sampling m permutations independently, for m to be decided. For any disjoint subsets of $\{1, \ldots, n\}$, A and B of size a and b, the probability that there is no $\pi \in \Pi$ ordering k elements of A before B is $(1-p)^m$ where p is the probability from (2.1).

There are $\binom{n}{a}\binom{n-a}{b}$ ways to choose A and B, so by the union bound, the probability that there exists A, B for which Π doesn't work is at most

$$\binom{n}{a}\binom{n-a}{b}(1-p)^m \leq \exp\left(\log\left(\binom{n}{a}\binom{n-a}{b}\right) - pm\right).$$

Taking $m \geq \binom{a+b}{k} / \binom{a}{k} \log\left(\binom{n}{a}\binom{n-a}{b}\right)$ as in the theorem, this probability is less than 1, and so by the probabilistic argument, there must be a design of size m.

For the $(\frac{a+b}{a})^k \cdots$ bound, we have used theorem 14 and bounded $\binom{n}{a}\binom{n-a}{b} = n!/(a!b!(n-a-b)!) \leq n!/(n-a-b)! \leq n^{a+b}$. □

For large k, the probabilistic bound is basically as good as we may hope for. This is clear from the lower bounds below.

The problem with the probabilistic bound is that it's not constructive. That is, we can make a set of permutations that will nearly certainly be a bottom-k design, but we cannot guarantee it, which is the goal of this thesis. Note however that a construction can be checked against all possible pairs of subsets in time $n^{O(a+b)}$, which can be reduced to $(a+b)^{O(a+b)}$ by perfect hashing. Thus for $a+b$ small enough, we can indeed make a near optimal design this way. The approach uses Las Vegas randomization, but not Monte Carlo. In the section on k-restrictions later in this chapter, we completely derandomize this process, and add some more tricks, that allow us to increase the design size further.

There are however also other ways of constructing designs. Very simple constructions give us the following bounds:

Proposition 3.

$$B_k(a,0,n) = 1 \tag{2.6}$$

$$B_0(a,b,n) = 1 \tag{2.7}$$

$$B_k(2k-1,1,n) = 2 \tag{2.8}$$

$$B_1(1,b,n) \leq \frac{nb+1}{b+1} \leq \frac{b}{b+1}n \tag{2.9}$$

Proof. For (2.6) any single permutation will do. (2.7) could arguably be defined to 0, but for our later recursions it makes more sense to use 1. (2.8) can be done by taking any permutation and its reverse. This is a generalization of Example 2 from when we defined bottom-k designs.

Finally (2.9) can be constructed by making a $b+1$-ary tree, with a permutation starting at each leaf and going up the parents till it hits the root. After hitting the root, it visits the remaining elements in any order. To see that this works, we imagine coloring the elements of B black and the one element of A green. Because the tree is $b+1$-ary any element will have a path down to a leaf, without visiting a black node. Such a tree with l levels has size $1+(b+1)+(b+1)^2+\cdots+(b+1)^l = \frac{(b+1)^{l+1}-1}{b}$ and $(b+1)^l$ leaves. □

A set of results similar to proposition 1, but more advanced is

Proposition 4.

$$B_k(a,b,n) \geq B_{k-1}(a-1,b,n-1) \tag{2.10}$$

$$B_k(a,b,n) \leq B_k(a,b,n-1) + B_{k-1}(a-1,b,n-1) \tag{2.11}$$

$$B_k(a,b,n) \leq nB_k(a,b-1,n-1) \tag{2.12}$$

$$B_k(a,b,n) \leq \sum_{\substack{0\leq a'\leq \min(a,n/2)\\ 0\leq b'\leq \min(b,n/2)}} \min_{k'\leq k}\left[B_{k'}(a',b',n/2)\,\frac{n}{2}\,B_{k-k'}(a-a',b-b',n/2)\right] \tag{2.13}$$

Proof. (2.10): Take an (a,b,n) design and remove an element x from $\{1,\ldots,n\}$ and from all the permutations. With the new design, for any A, B, we can add an arbitrary element to A and the original design will order k of the elements before B. This means that it also orders $k-1$ of our remaining elements before B.

(2.11) corresponds to the construction of extending a $(a,b,n-1)$ design, Π_1 with one element, x, by adding it to the end of each permutation. Then we take another $(a-1,b,n-1)$ design,

Π_2 and put x in the beginning of each permutation. This is a (a, b, n) bottom-k design, because if a subset A does not include x, a $\pi \in \Pi_1$ will work, and if $x \in A$, then a $\pi \in \Pi_2$ will do.

(2.12): For each $x \in \{1, \ldots, n\}$, take an $(a, b-1, n-1)$ design on the remaining elements and add x to the end of each permutation in the design. The union of these n designs is an (a, b, n) design, since for any location of an element in B we can take the sub-design that ordered this element at the end.

(2.13): Partition $\{1, \ldots, n\}$ in two sets of size $n/2$, call them 'left' L and 'right' R. We want to create a design for each and combine them somehow, but we don't know how many elements of A and B went to each side. Thus, for each $0 \leq a' \leq \min(a, n/2), 0 \leq b' \leq min(b, n/2)$ and which ever k minimizes $B_{k'}(a', b', n/2)B_{k-k'}(a-a', b-b', n/2)$, we create a $(a', b', n/2)$ bottom-k' design Π_1 and a $(a-a', b-b', n/2)$ bottom-$(k-k')$ design. Assume for some A, B that $\pi_1 \in \Pi_1$ orders k' elements of $A \cap L$ before $B \cap L$, and $\pi_2 \in \Pi_2$ orders $k-k'$ elements of $A \cap R$ before $B \cap R$. Then we'd like our final design to have the combination $\pi_1 \to \pi_2$. However, if we simply append π_2 to the end of π_1, we run into the problem, that the result will order $B \cap L$ before all of $A \cap R$. To solve this, our final design uses each prefix from π_1, adding the extra factor $n/2$ in the bound. □

Another approach is relating bottom-k designs to other kinds of combinatorial designs. In particular we can make constructions based on Turán-designs, Perfect Hash Families and Group testing.

Proposition 5.

$$B_k(a, b, n) \leq \mathit{Turán}(n, a, k) \tag{2.14}$$

$$B_k(a, b, a+b) \geq \mathit{Turán}(a+b, a, k) \tag{2.15}$$

$$B_k(a, b, n) \leq B_k(a, b, n_2)\mathit{Perfect\text{-}hash\text{-}family}(n, n_2, a+b) \tag{2.16}$$

$$B_1(1, b, n) \leq \mathit{Group\text{-}test}(n, b) \tag{2.17}$$

Proof. (2.14): Turán designs are sets of subsets $R \subseteq \{1, \ldots, n\}$ of size r' such that for any $K \subseteq \{1, \ldots, n\}$ of size k', there is R such that $R \subseteq K$. We can make a bottom-k design by turning the R-sets into permutations. We do this by for each R making a π that orders these elements before all other elements of $\{1, \ldots, n\}$. The order internally in R and $\{1, \ldots, n\} \setminus R$ is arbitrary. For $k' = a$ and $r' = k$ and any b, this is a (a, b, n) bottom-k design, since for any A, B we can take the permutation made from $R \subseteq A$ which then orders k elements from A before B.

(2.15): Given a $(a, b, a+b)$ bottom-k design, we can make a Turán design, by making a set R from the first k elements of each π. This is a Turán $(a+b, a, k)$ design because for any $K \subseteq \{1, \ldots, n\}$ of size a, we can take $A = K$ and $B = \{1, \ldots, n\} \setminus K$, and take the π that orders k elements of A before B. Since all elements of $\{1, \ldots, n\}$ are either in A and B, the set R formed from π must have all elements from A and thus by K.

(2.16): A (n, n_2, t)-perfect hash family is a set of assignments $h : \{1, \ldots, n\} \to \{1, \ldots, n_2\}$, such that for any set $T \subseteq \{1, \ldots, n\}$ of t elements, there is an assignment h which assigns all elements of T to different elements of $\{1, \ldots, n_2\}$.

We can make a (a, b, n) bottom-k design from a (a, b, n_2) bottom-k design Π and a $(n, n_2, a+b)$ perfect hash family P: For each $\pi \in \Pi$ and $h \in P$ we define $\pi_h(x) < \pi_h(y) \equiv \pi(h(x)) < \pi(h(y))$ for $h(x) \neq h(y)$. If x and y are mapped to the same element, π_h order them arbitrarily. This works because for any $A, B \subseteq \{1, \ldots, n\}$, there is an assignment h mapping all $A \cup B$ elements to different values of $\{1, \ldots, n_2\}$. Thus if π ordered k elements of $h(A)$ before $h(B)$, π_h will order k elements of A before B.

(2.17): Group testing stems from doctors wanting to test soldiers' blood for illnesses without having to manage n samples of blood. Instead they would mix the blood and perform tests on the mixed samples. In particular a non-adaptive (n, b) group testing design, G, is a system of sets, such that for any subset $B \subseteq \{1, \ldots, n\}$ and $x \notin B$, there is a set $R \subseteq G$ such that $x \in R$, but $B \cap R = \emptyset$. Thus if a soldier was not infected, his blood would be in some sample without any infected blood and could be acquitted. For our purposes we create a permutation π for every R by ordering the elements of R before all elements of $\{1, \ldots, n\} \setminus R$. Hence for any A, B with $|A| = 1$, there is a permutation ordering $A \subseteq R \subseteq \{1, \ldots, n\} \setminus B$ before B.

□

From (2.14) and (2.15) we get that $B_k(a, b, a+b) = \text{Turan}(a+b, a, k)$. This shows that bottom-k designs can be seen as a kind of generalization of Turán designs.

Alon et. al. show in [?] that a (n, k^2, k) perfect hashing family of size $O(k^4 \log n)$ can be efficiently constructed. They also show that a (n, b) group testing scheme of size $(1+\delta)eb^2 \log n$ can be created in time polynomial in n^b for any $\delta > 0$. For all n, except n very close to b, this is much better than our bound (2.9).

2.2.1 Lower Bounds

Finally we have some lower bounds. We're mainly interested in determining how sharp the probabilistic bound is to the truth.

Proposition 6.

$$B_k(a, b, n) \geq \binom{a+b}{k} \Big/ \binom{a}{k} \geq \left(\frac{a+b}{a}\right)^k \tag{2.18}$$

$$B_k(a, b, n) \geq b + 1 \tag{2.19}$$

$$B_1(1, 2, n) \geq \lg \lg n \tag{2.20}$$

Note that the probabilistic bound had three factors: $\binom{a+b}{k}/\binom{a}{k}$, $(a+b)$ and $\log n$. Our lower bounds show that each part is reasonably close to the optimal values.

Proof. (2.18): This is shown using a volume bound. Given an ordering, in how many ways can we choose k out of a before anything from b? The answer is $\binom{n}{a+b}\binom{a+b-k}{a-k}$: First we choose the non-frees, then we make the first k candy, and choose $a-k$ candy from the remaining. If we divide the total number of configurations by this, we get the lower bound

$$\frac{\binom{n}{a+b}\binom{a+b}{a}}{\binom{n}{a+b}\binom{a+b-k}{a-k}} = \frac{\binom{a+b}{a}}{\binom{a+b-k}{a-k}} = \frac{\binom{a+b}{k}}{\binom{a}{k}}.$$

(2.19): This one is easy. If we had b permutations or less, we could form B by taking the first ordered element of each permutation. Then for no element outside of B would we be able to find a permutation ordering that element before B.

(2.20): This one is more tricky. Clearly for $b = 1$ we don't generally need the size of the design to depend on n. The construction 2.8 showed us that much. The obvious next choice becomes $b = 2$, for which we prove a non-trivial lower bound, which turns out to be optimal.

Given a $(1, 2, n)$ bottom-1 design, Π of size t, we show that there is a sequence of sets $s_t \subseteq s_{t-1} \subseteq \ldots s_0 = \{1, \ldots, n\}$ such that the first i permutations of Π all order s_i either strictly increasing or strictly decreasing. We know $|s_t| < 3$, because otherwise there are A and B placing the element of A between those of B, thus failing the condition for the design.

We'll prove that $|s_i| \geq n^{2^{-i}}$. Hence $n^{2^{-t}} \leq |s_t| < 3$ implying $t > \lg \lg n - \lg \lg 3 > \lg \lg n$.

The proof is by induction. $|s_0| = |\{1, \ldots, n\}| = n$ follows by the definition of the sequence. Now assume the statement for $i' < i$. We'll prove it for i.

A classic theorem by Erdös and Szekeres [?] say that any sequence of length $(n-1)^2 + 1 \leq n^2$ has either an increasing or decreasing sub-sequence of length n. Consider the ordering π_i on s_{i-1}. By Erdös and Szekeres there is either a subset of s_{i-1} that's increasing or decreasing of size $\lceil \sqrt{|s_{i-1}|} \rceil \geq \sqrt{n^{2^{-i+1}}} \geq n^{2^{-i}}$. We let s_i be the largest of those. □

We may worry if $\log\log n$ is the right lower bound, or $\log n$ from the probabilistic method is the right upper bound. The former turns out to be the case, as we see from the from the following construction.

Proposition 7.

$$B_1(1,2,n) \le \lg\lg n + O(\lg\lg\lg n) \tag{2.21}$$

Proof. We assume $n = 2^p$ for p even. Let s be a set of $|s| = p$ binary strings of length r, which starts with 0 and have an equal number of 1's and 0's. Clearly we need $\binom{r}{r/2}/2 \ge 2^r/\sqrt{2r} \ge p$ so we can choose $r = \lg p + O(\lg\lg p)$. r will be the size of our family. For example, for $p = 3$, we might have $s = (0011, 0101, 0110)$.

We construct the family π_i recursively from smaller $\pi_{i,j} : \{1, \dots, 2^j\} \to \{1, \dots, 2^j\}$ where $1 \le i \le r$ and $1 \le j \le p$. We define $\pi_{i,0} = 1 \mapsto 1$ and

$$\pi_{i,j}(x) = \begin{cases} \pi_{i,j-1}(x) & \text{if } x \le 2^{j-1} \text{ and } s_{i,j} = 0 \\ \pi_{i,j-1}(x) + 2^{j-1} & \text{if } x \le 2^{j-1} \text{ and } s_{i,j} = 1 \\ \pi_{i,j-1}(x - 2^{j-1}) & \text{if } 2^{j-1} < x \text{ and } s_{i,j} = 1 \\ \pi_{i,j-1}(x - 2^{j-1}) + 2^{j-1} & \text{if } 2^{j-1} < x \text{ and } s_{i,j} = 0 \end{cases}.$$

Intuitively this family is constructed from r binary trees, one for each bit in the strings of s. The tree has n leafs and p levels. The children of level j in the ith tree are swapped exactly if $s_{i,j} = 1$. For $s = (0011, 0101, 0110)$ we get the permutations $(1,2,3,4,5,6,7,8)$, $(4,3,2,1,8,7,6,5)$, $(6,5,8,7,2,1,4,3)$ and $(7,8,5,6,3,4,1,2)$.

To show that this is a $(1, 2, 2^p)$ bottom-1 design, we'll consider the incidence matrices of the permutations. That is matrices I_i, such that $I_{i,x,y} = 0$ if $\pi_i(x) < \pi_i(y)$ and 1 otherwise. For $s = (0011, 0101, 0110)$ we get the following incidence matrices I_1, I_2, I_3, I_4:

$$\begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & 1 & 1 & 1 & \cdot & \cdot \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & \cdot \end{bmatrix} \begin{bmatrix} \cdot & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & 1 & 1 & 1 & \cdot & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & \cdot & \cdot & 1 & 1 \\ 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & 1 \\ 1 & 1 & 1 & 1 & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & 1 & \cdot & \cdot & 1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 \\ 1 & 1 & \cdot & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & \cdot & \cdot & 1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot & \cdot & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \cdot & 1 & 1 & 1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & 1 & 1 & 1 \\ \cdot & \cdot & 1 & \cdot & 1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & 1 & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \end{bmatrix}.$$

If we 'stack' the matrices, we can consider them as just one matrix, $M = M_p$, with elements from s. We can write the matrix recursively as $M_0 = [\cdot]$ and

$$M_{j+1} = \begin{bmatrix} M_j & s_{j+1} \\ 1 - s_{j+1} & M_j \end{bmatrix}.$$

Note that the diagonal doesn't matter for incidence matrices, so we just leave it as $\cdot$. If we name the elements $0011 \mapsto 3$, $0101 \mapsto 2$, $0110 \mapsto 1$ and their binary opposites $1 - s_i$: $1100 \mapsto -3$, $1010 \mapsto -2$, $1001 \mapsto -1$, we can write the matrix as

$$M_3 = \begin{bmatrix} \cdot & 1 & 2 & 2 & 3 & 3 & 3 & 3 \\ -1 & \cdot & 2 & 2 & 3 & 3 & 3 & 3 \\ -2 & -2 & \cdot & 1 & 3 & 3 & 3 & 3 \\ -2 & -2 & -1 & \cdot & 3 & 3 & 3 & 3 \\ -3 & -3 & -3 & -3 & \cdot & 1 & 2 & 2 \\ -3 & -3 & -3 & -3 & -1 & \cdot & 2 & 2 \\ -3 & -3 & -3 & -3 & -2 & -2 & \cdot & 1 \\ -3 & -3 & -3 & -3 & -2 & -2 & -1 & \cdot \end{bmatrix}.$$

The important property in M is that no row contains both x and $-x$. This follows from induction: For M_0 there is only one element. Assume it holds for $j' < j$ then, together with the fact that M_j contains neither s_{j+1} nor $1 - s_{j+1}$, it also holds for M_{j+1}.

We need to prove that for all x, y, z there is an i such that $\pi_i(x) < \pi_i(y)$ and $\pi_i(x) < \pi_i(z)$. In matrix notation this means for all rows x and pairs of columns y, z there is an I_i such that $I_{i,x,y} = 0$ and $I_{i,x,z} = 0$.

By construction, this is equivalent to saying that there is an i such that $(M_{x,y})_i$ and $(M_{x,z})_i$ are both zero. Since these vectors all have the same number of 1s and 0s, if for any i $(M_{x,y})_i = (M_{x,z})_i = 1$, then there is also an i where both are zero. Alternatively, if they don't share any position, then $M_{x,y} = 1 - M_{x,z}$, but we know that M has no such rows, which proves the construction. □

While we managed to prove a number of results, we still haven't found a general, practical construction. In the next section we will get one.

2.2.2 Using K-restrictions

Using theorem 3 on k-restrictions (Definition 3) we will show the following theorem:

Theorem 5. *For $a, b \leq n$ and $k \leq a$, there is an (a, b, n)-bottom-k design, Π, of size*

$$|\Pi| = O\left(\left(\frac{a+b}{a}\right)^k e^{k^2/a}(a+b)\log n\right)$$

and can be constructed in time $n^{O(a+b)}$.

Note that sampling uniform permutations and checking that they are indeed a correct bottom-k design gives the same result. The k-restriction method however shows that we don't have to use any randomness at all.

Then with the help of splitters (definition 4 and lemma 2) we will improve it to:

Theorem 6. *For some integers $a, b \leq n \leq m$ and $k \leq a$ with $a, b = o\left(\frac{\log m}{\log\log m}\right)^2$, we can construct an (a, b, n)-bottom-k design, Π, of size*

$$|\Pi| = O\left(\left(\frac{a+b}{a}\right)^k e^{k^2/a} m^{o(1)}\right)$$

in time $|\Pi| m^{o(1)}$.

Proof of theorem 5. When formulating the (a, b, n)-bottom-k as an $(a + b)$-restriction problem, it is most convenient to work over some space with independent coordinates. For this purpose, we first define an $(a + b)$-restriction problem over $\Sigma^n := \{1, \dots, (a + b)^2\}^n$ and then show how to turn the solution into a bottom-k design.

Following the notation from definition 3, we let $q = |\Sigma| = (a + b)^2$, $m := n$, $k := a + b$. For all partitions $A, B \subseteq \{1, \dots, a + b\}$ with $|A| = a$ and $|B| = b$, we have define restrictions $f_{A,B} : \Sigma^{a+b} \to \{0, 1\}$ by $f_{A,B}(x) = [x \text{ has all distinct coordinates, the smallest } k\ x_i \text{ have } i \in A]$. Thus the number of restrictions is $s := \binom{n}{a+b}\binom{a+b}{a} \leq n^{a+b}$.

We choose D to be the uniform distribution over Σ^n. This is efficiently approximateable by [**?**]. Since our restrictions are symmetric in A, B and permutation of x, the 'density', ϵ is the probability that an $x \in \Sigma^{a+b}$ uniformely sampled, satisfies some demand. This is easy to calculate as

$$\begin{aligned}\Pr_{x\sim\Sigma^{a+b}}[f_{A,B}(x)] &= \Pr_x[x \text{ has all distinct coordinates}]\\ &\quad \Pr_x[\text{the smallest } k\ x_i \text{ has } i \in A \mid x \text{ distinct}]\\ &\geq \frac{1}{2}\binom{a}{k}\Big/\binom{a+b}{k},\end{aligned}$$

where the second part is lemma 3 and the first part is
$\Pr_x[x \text{ distrinct}] = q^{\underline{a+b}}/q^{a+b} = \prod_{0\leq i<a+b} 1 - i/q \geq 1 - \sum_{0\leq i<a+b} i/q \geq 1-(a+b)^2/(2q) = 1/2$.

Finally, we don't care much about δ. Setting it to 1/2 suffices. The theorem then shows that we can construct a set $\mathcal{A} \subseteq \Sigma^n$ such that for any choice of $a+b$ indices $1 \leq i_1 < \cdots < i_{a+b} \leq n$, and a demand $f_{A,B}$, there is some $x \in \mathcal{A}$, such that $f_{A,B}(x_{i_1}\cdots x_{i_k}) = 1$. This $\mathcal{A}$ has size

$$|\mathcal{A}| \leq \left\lceil \frac{k\log m + \log s}{(1-\delta)\epsilon} \right\rceil = \left\lceil \frac{(a+b)\log n + \log\binom{n}{a+b}\binom{a+b}{a}}{(1/2)\binom{a}{k}/\binom{a+b}{k}(1/2)} \right\rceil = O\left(\left(\frac{a+b}{a}\right)^k e^{k^2/a}(a+b)(\log n)\right)$$

and can be constructed in time $\operatorname{poly}(s, m^k, q^k, \epsilon^{-1}, \delta^{-1}) = n^{O(a+b)}$.

We convert $\mathcal{A}$ into a bottom-k design, Π, like this: For any $x \in \mathcal{A}$, let define π_x by $\pi_x(i) < \pi_x(j) \equiv x_i < x_j$ if $x_i \neq x_j$ and $\pi_x(i) < \pi_x(j) \equiv i < j$ otherwise. This is a total order, so it can be represented as a permutaiton $\pi : \{1,\dots,n\} \to \{1,\dots,n\}$.

To show that Π is an (a,b,n)-bottom-k design, consider subsets $A, B \subseteq \{1,\dots,n\}$ and $x \in \mathcal{A}$ which satisfies $f_{A,B}$. Then x has distinct values over $A \cup B$ and for the k smallest s_i, $i \in A$. Thus π_x orders k elements of A before B. Since Π has size equal to $\mathcal{A}$, it proves theorem 5. □

While it is nice, that theorem 5 can construct near optimal designs, without using randomization of any kind, the construction time $n^{O(a+b)}$ is quite restrictive. Theorem 6 allows reducing this to much more manageable levels.

Proof of theorem 6. Let $a+b = o\left(\frac{\log m}{\log\log m}\right)^2$, $B = \sqrt{t} = o\left(\frac{\log m}{\log(a+b)}\right)$.

We first user perfect hashing, as for (2.16), so we can assume that $n \leq (a+b)^2$. This costs us a factor $(a+b)^4\log n = m^{o(1)}$ in the design size, and allows us to assume $n^{O(B)} = (a+b)^{o(\log m/\log(a+b))} = m^{o(1)}$ and $n^{(a+b)/B} = m^{o(1)}$.

We proceed by three steps:

1. Construct a 2-way n-wise splitter S, such that for any disjoint subsets $S_1, S_2 \subseteq \{1,\dots,n\}$, there is a $s : \{1,\dots,n\} \to \{1,\dots,B\}$ such that $\lfloor |S_i|/B \rfloor \leq |s^{-1}(j) \cap S_i| \leq \lceil |S_i|/B \rceil$ for both $1 \leq i \leq 2$ and all $1 \leq j \leq B$. By lemma 2, such a splitter can be made with size $|S| = B^B n^{2B} \leq n^{3B} = m^{o(1)}$.

2. Construct an $(\lfloor a/B \rfloor, \lceil b/B \rceil, n)$-bottom-$\lceil k/B \rceil$ design, Π, of size $|\Pi| = O\left(\left(\frac{a+b}{a}\right)^{k/B} e^{k^2/(Ba)}(a+b)\log n\right)$ using theorem 5 in time $n^{O((a+b)/B)} = m^{o(1)}$.

3. For every splitter partition $s \in S : \{1,\dots,n\} \to \{1,\dots,B\}$ and prefix π_i of every $\pi \in \Pi$, we use the product construction from (2.13). This gives a final design of size $|S||\Pi|n^B = |\Pi|m^{o(1)}$.

□

For sets of size $o\left(\frac{\log n}{\log\log n}\right)^2$, this means that we can create optimal designs to use with the algorithm from theorem 4 in time sub-linear in the number of sets in the data-structure.

In the next section we show how to generalize this to work with much larger sets.

2.3 The Complete Algorithm

Theorem 7. *Assume data points $x \in X$ and queries $q \in U$ all have size t. There is an exact data-structure for the (s_1, s_2)-approximate similarity search problem over $\{1, \dots, d\}$ with Jaccard similarity with space usage $O(n^{1+\rho+o(1)})$ and query time $O(n^{\rho+o(1)})$, where*

$$\rho = \frac{\log 1/s_1}{\log 1/s_2}.$$

This matches the best non-exact algorithms up to lower order terms.

Theorem 7 extends theorem 4 to large sets by the simple idea of randomly partitioning the coordinates into B smaller 'blocks'. Let $h : \{1, \dots, d\} \to \{1, \dots, B\}$ be a partition, then we write $x_{h^{-1}(j)}$ for the elements of x that are placed in the jth block. In contrast with the splitter idea from the previous section, we only make a single partitioning, we only get probabilistic guarantees on the split. This idea is present in all previous work on exact LSH algorithms[**?**, **?**], but in our case it'll be more complicated. Much like Alon improved over splitters with multi-way splitters, our probabilistic partitioning needs to take care of two sets of elements A and B concurrently.

Intuitively this is similar to dimensionality reduction by sampling, but more exact, since no coordinates are thrown away. We need to prove a number of things: (1) If $\text{sim}(q, x) \geq s_1$ then in one of the blocks, j, we have $\text{sim}(q_{h^{-1}(j)}, x_{h^{-1}(j)}) \geq s_1$. (2) The number of collisions with less-similar sets should not significantly increase in any of the blocks. (3) The size of the sets in the blocks shouldn't end up too small: We should never get $|(q \cap x)_{h^{-1}(j)}| < k$ for x and q with $\text{sim}(x, q) \geq s_1$. (4) The size of the sets in the blocks shouldn't end up too large: We can only efficiently create designs for $d \approx (\log n)^2$.

We'll start by explaining the new data structure, which will show how we intend to handle (3) and (4). We'll assume $t \geq (\log n)^2$. If not, we can either use the exact algorithm from the introduction, or replication the coordinates as in the first algorithm of this chapter. The algorithm will depend on parameters $B = \lceil t(\log n)^{-3/2} \rceil$ and $m = t\lceil \sqrt{\frac{\log n}{2dB}} \rceil = \lceil \frac{(\log n)^{5/4}}{\sqrt{2d}} \rceil$. We will assume d/B is integer, padding the universe with dummy elements if needed.

Building the data structure: Given $X' \subseteq \{1, \dots, d\}$ of size n, we then take a random partition of $\{1, \dots, d\}$ into B blocks of size d/B. More precisely we form the blocks by repeated sampling without replacement. We then build the following data-structure over each projection of X' on a block:

Given $X \subseteq \{1, \dots, d/B\}$ we build $(2m + 1)^2$ instances of the data structure from theorem 7 for (s_1,s_2)-similarity with set sizes t_x, t_q where $t/B - m \leq t_x, t_q \leq t/B + m$. In addition, we create a simple linked-list call 'the stash'.

For each x in the block, if the size of x is in the range $t/B \pm m$, we add it to each data-structure with $t_x = |x|$ and $t_q \in \{t/B - m, \dots, t/B + m\}$. If $|x|$ is not in the range, we add it to the stash.

Querying the data structure: Given a set $q \subseteq \{1, \dots, d\}$, we take each projection over the B blocks and query the data-structures as so:

If $|q| > t/B + m$ or $|q| < t/B - m$, we 'brute force' compute $\text{sim}(q, x)$ for every $x \in X$, returning x if the similarity is greater than s_1. Otherwise we first 'brute force' those x that are in the stash, and then query the $2m + 1$ data structures of the type from theorem 7 where $t_q = |q|$ and $t/B - m \leq t_x \leq t/B + m$.

The inner-data structures are amended such that any points x they find are checked, such that if the similarity $\text{sim}(q, x)$ with the original, large x and q is not greater than s_2, the search is continued.

See figure 2.1 for an illustration of the construction.

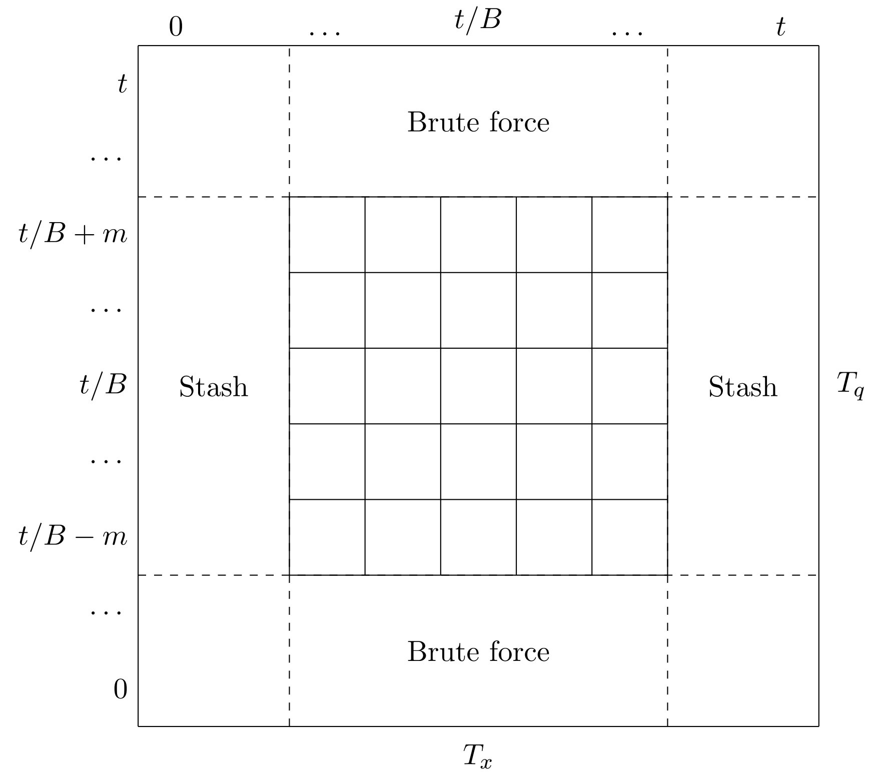

Figure 2.1: For each of $B \approx t(\log n)^{-3/2}$ partitions, we build $(2m+1)^2 \approx (\log n)^{5/2}/d$ data structures plus a stash. All data points which have size outside the $t/B \pm m$ range go in the Stash and must be searched by every query. All queries that go outside the normal range are changed to brute force queries over all points in the partition.

Proof of theorem 7. We first show that the algorithm is correct. The first step of this is handling (1) which we do by the following lemma 4:

Lemma 4. *Let $A_1, \dots, A_n$ and $B_1, \dots, B_n$ be positive numbers. Then there is an $i \in [1, n]$ such that*

$$\frac{A_i}{B_i} \geq \frac{A_1 + \cdots + A_n}{B_1 + \cdots + B_n}$$

Proof. Let i be an index such that for all $j \in [1, n]$, $A_i/B_i \geq A_j/B_j$. We can 'pull' this largest term out of the combined fraction:

$$\frac{A_1 + \cdots + A_n}{B_1 + \cdots + B_n} = \frac{A_i}{B_i} \frac{A_1/A_i + \cdots + A_n/A_i}{B_1/B_i + \cdots + B_n/B_i}$$

Since $A_i/B_i \geq A_j/B_j$, and the numbers are positive, we have $B_j/B_i \geq A_j/A_i$ and so the terms in the numerator are dominated term-wise by the denominator, which shows that the second fraction is at most 1. □

This shows that, assuming the construction on each block is sound, the hole data-structure is. Inside a block, if t_q is not in the range $t/B \pm m$, we do a brute force search, which is clearly correct. If t_x is not in the range $t/B \pm m$ then x will be in the stash, which we always brute force, so x is again certain to be found. Finally if both t_q and t_x are in the 'normal' range, we query the inner data structure, which works by theorem 4.

This shows that all 'good' points still *could* be found. Now, because we never return a point with similarity less than s_2 on the non-projected points, this proves that the data-structure is correct.

Analysis The first thing we show is that only $O(1/n)$ sets end up having sizes outside of the $t/B \pm m$ range, such that the expected time spent on brute force and 'the stash' will be $O(1)$ by linearity of expectation.

Let T be a random variable describing the size of sets after partitioning. Since we sample with replacement, T is hyper-geometrically distributed as drawing d/B balls from a population of size d with t 'successes'. By (A.12) a simple Hoeffding bound still holds in this situation, even though draws are not independent. In particular the bounds says T is larger than $t/B+m$ with probability $\Pr[T \geq t/B+m] \leq \exp(-2dBm^2)$. Thus taking $m \geq \sqrt{\frac{t^2 \log n}{2dB}}$ we get

$$\Pr[|T - t/B| > m] \leq 2/n.$$

Ideally the analysis of the inner constructions would just be taking $B(2m+1)$ times theorem 4. However we have amended the algorithm with extra distance checking, so we need to make sure that we don't get drowned in new projection induced collisions. Or alternatively, the complexity bounds of theorem 4 assumes that there are no points with similarity between s_1 and s_2. We can no longer guarantee this.

Instead we open the black box and show that the probability for two points x, y to collide under a random permutation *decreases* when sampled as in our algorithm. This is not obvious, since there's a reasonable probability that $A/(A+B)$ doubles or more, where $A/(A+B)$ is the similarity under projection, A is the intersection and B is the symmetric difference. Given that A and B are hype-geometrically distributed, we could compute the collision probability $p = E[\binom{A}{k}/\binom{A+B}{k}\Pr[A]\Pr[B]]$ directly, however there seems to be no nice form for this, and indeed there is a nicer argument.

We'll show that $p \leq \binom{a}{k}/\binom{a+b}{k}$. For this, consider a random permutation π over the non-partitioned $\{1,\dots,d\}$. The probability that this permutation orders k elements of a set C before some set D is $\binom{|C|}{k}/\binom{|C|+|D|}{k}$ as always. Now consider π truncated to its first d' positions. We'll say that such a truncate permutation orders x before y only if $\pi(x) < \pi(y)$ and both elements are in the considered prefix. Since strictly fewer elements are 'ordered before others', such a truncated permutation has a strictly smaller chance of ordering 'k elements of C before D'. Sampling (with replacement) and then taking a random permutation, is exactly equivalent to taking a random permutation and then restricting it to some subset of its elements. This proves that the collision probability strictly decreases in our blocks. Hence we can use the running time bounds from theorem 4 without worries.

All there is left to do summarizing the complexity and deciding no the value for B.

By theorem 6, we can create (a,b,d)-bottom-k designs efficiently of size $O(s_1^{-k}e^{k^2/a}n^{o(1)})$. The bound varies slightly for different t_x and t_q, since $a = \frac{s_1}{1+s_1}(t_q+t_x)$. We bound it by the worst value, which is $a = 2\frac{s_1}{1+s_1}(t/B-m)$. However

$$t/B - m = (\log n)^{3/2} - \sqrt{t/d}(\log n)^{5/4} = \omega(\log n)$$

and $k = O(\log n)$, so $e^{k^2/a} = e^{o(\log n)} = n^{o(1)}$. The total complexity per point thus becomes $B(2m+1)^2 s_1^{-k} n^{o(1)} = n^{\frac{\log 1/s_1}{\log 1/s_2}+o(1)} = n^{\rho+o(1)}$ query time, and $n^{1+\rho+o(1)}$ space and construction time, which is the theorem. □

2.4 Conclusion

Perhaps surprisingly, we were able to utilize multiway-splitters to get complete performance parity with minhash-lsh, up to lower order terms.

They key ingredients were a combinatorial interpretation of bottom-k minhashing as permutations. Simply making an analog of 'bottom-1' minhash and taking the kth product, as is normally done in lsh would not have worked, since each construction would have had size at least $\log d$, and $(\log d)^k = n^{O(\log \log d)}$ which would entirely dominate our algorithm. That said, the permutation interpretation did give us certain problems, since it doesn't have perfect correspondence with the Jaccard similarity the way normal minhash has. The problem was overcome by making sure sets were always reasonably large, so the two more or less coincided, but perhaps a better combinatorial construction can work even for a and b of constant size.

We also note, that the tight ratio analysis of theorem 14 was key to not suffering a factor $n^{1/\log 1/s_2}$ in the performance. This analysis may also be of separate interest in the computer science community, since binomial coefficients come up in so many different contexts.

The work in this section was, due to the large lower order terms, mostly of theoretical interest. It thus leaves many open questions for further research. We list a few of them here:

1. The algorithm as described, used a uniformly random partition, and the sub algorithms each use a uniformly random permutation. This is needed because the bottom-k permutations only filter out most far points, but not all. An adversary could design a data-set to take advantage of this to create many more collisions than we can survive. While it might not be possible to entirely eliminate some randomness here, it would be interesting to see if we could get away with, say, 2-independent hash functions.

2. In order to make the algorithm more usable, it would be interesting to remove some of the lower order terms. It's not clear that this can be done within the k-restriction framework, since we have to pay for things like splitters and perfect hash functions. However maybe there are large, useful families of bottom-k designs, that can be directly constructed, similar to how covering designs are constructed using finite geometries in covering LSH.

3. On the topic of direct constructions, it's interesting whether we can generalize the $(1, 2, n)$ bottom-1 construction of size $\lg \lg n$. In matrix notation a (a, b, n) bottom-1 design, would correspond to a set of incidence matrices, such that for all combinatorial a, b-rectangles there is a matrix in which all elements are 0.

4. On the other side, it would be interesting to find some parameters (a, b, k) which permits a $\log n$ lower bound. Right now it's open whether a $(1, 3, n)$ bottom-1 design needs $\log n$ or just $\log \log n$ permutations.

5. Another way to get more knowledge about the designs, would be finding more relations to other types of design. We showed in proposition 5 that $B_k(a, b, a+b) = \text{Turán}(a+b, a, k)$, but it doesn't give us any information on the dependency on n.

6. It's also likely that the upper bound from the probabilistic method (and hence k-restrictions) could be improved somewhat. In particular it feels strange, that we have a factor $(a + b)$ rather than just b or $(k + b)$.

7. The recursions (2.10), (2.11), (2.12) all seem too weak. Similar identities have given good results for other types of design, but in our case they are somehow not taking enough advantage of the permutations. In particular we haven't found any non-trivial lower bounds based on the recursions.

8. In the appendix we note a number of embeddings, which allow solving various generalizations of Jaccard similarity exact. However they do come with a certain loss of precision if

multiplicities are allowed to be very large. Minhash is sometimes generalized to 'weighted minhash'. In that spirit, it would be interesting to find exact versions of this as well.

Chapter 3

Algorithms with Filters and Turan Designs

In the previous chapter we showed how to make the classical minhash-LSH data-structure exact. We managed to match the efficiency of that algorithm, up to lower order terms, so in that sense our algorithm could be considered optimal.

There are however other senses of optimally we might consider. The strongest sense would of course be to have an approximate set similarity algorithm provably optimal with respect to the cell probe model. This is however outside of what we currently know how to do. The strongest result in this model is [**?**] for a two probe data structure.

Instead we'll try to match the very recent algorithm by Christiani and Pagh [**?**] which they show is optimal in the model of space partitions. We will again relate the problem to a type of combinatorial design, however the algorithm using it will be quite different from the previous chapter.

Definition 6 (Turán Design [**?**])**.** *A Turán (n, k, r)-design $(n \geq k \geq r)$ is a collection of r-element blocks of an n element set X_n such that every k element subset of X_n contains at least one of the blocks.*

The Turán number $T(n, k, r)$ is the minimum number of blocks in a Turán (n, k, r)-system.

Example 1: Recall the Fano-plane: $\{\{0,1,2\}$, $\{0,3,4\}$, $\{0,5,6\}$, $\{1,3,5\}$, $\{1,4,6\}$, $\{2,3,6\}$, $\{2,4,5\}\}$. Since every pair is a member of exactly one set in the Fano-plane, the complement of the Fano-plane forms a Turán$(7,5,4)$-design of size 7. In other words, any complement of a pair contains the complement of a line.

In general geometries can be used to create designs with quite high k and r, but they don't generally allow $k \leq d/2$.

Example 2: The set of all r-subsets of $\{1,\ldots,d\}$, that is $\binom{\{1,\ldots,d\}}{r}$, is a Turán$(d,r,r)$-design of size $\binom{d}{r}$. Since every r-subset is included in the design, any r subset of $\{1,\ldots,d\}$ must have a subset in the design (itself).

We won't show as many properties of Turán designs as we did of bottom-k designs in the previous chapter. Instead we refer to the survey of Sidrenko [**?**] and more recently, but less specifically Keevash [**?**].

The designs are closely related to the covering designs used by Pagh [**?**]. Relation to other designs [**?**]

Proposition 8.

$$T(n, k, r) = \mathit{Covering}(n, n-r, n-k)$$

Here Covering(n, m, p) is the minimum number of size m sets, such that every p subset of $\{1, \dots, n\}$ is a subset of an m-set.

We'll also note the trivial probabilistic upper bounds and volume lower bounds:

Proposition 9.

$$\binom{n}{r}\Big/\binom{k}{r} \le T(n,k,r) \le \left\lceil \binom{n}{r}\Big/\binom{k}{r} k \log n \right\rceil$$

The random construction in particular comes from uniformly sampling random size r sets. A particular size k set has probability $\binom{k}{r}/\binom{n}{r}$ of containing such an r set. Thus by the union bound, the number stated in the proposition suffices.

In the previous chapter, we could use any designed stored on our disk, however for the algorithms in this chapter, we will need one more property:

Definition 7 (Efficiently Decodable Turán Design)**.** *We say that a Turán (n, k, r)-design is 'efficiently decodable' if there is an algorithm, which given any subset $K \subseteq \{1, \dots, n\}$ of size $\ge k$ reports all blocks R, which are a subset of K. The algorithm has 'overhead' τ, if it runs in time proportional to the output size plus τ.*

Among our examples, example 2 is efficiently decodable, since given any set K, we know that it is also a block, and the only one at that. Hence we can simply return it. Example 1 is also efficiently decodable, but only because it has finite size. In general it is not obvious how to efficiently decode geometrical designs. Note that the decoding requirement makes it harder to create designs using the 'greedy' k-restriction approach from the previous chapter.

3.1 Using an Efficiently Decodable Turán Design

The algorithm by Christiani and Pagh [?] can be seen as a generalization of the exact 'meet in the middle' algorithm from theorem 1. Recall that the algorithm stored every data-set $\binom{t}{a}$ times, index by their every a subset. Here t is the size of the data-sets and $a = \frac{2s}{1+s}t$ is the size of the intersection between to points with Jaccard similarity s. The algorithm in [?] generalizes this concept by indexing data-points only at carefully chosen subsets, smaller than a. This is more efficient, since there are much fewer such sets, and because in the approximate case, dissimilar sets have a very little overlap, we do not get many collisions even with this smaller choice of index sets.

Again, we solve the problem from definition 2: From the universe $U = \binom{\{1,\dots,d\}}{t}$ of t element sets, we are given a set of sets $X \subseteq U$ of size n. We are further given two Jaccard-similarity thresholds s_1 and s_2. Our goal is to create a data structure which, given a query $q \in \binom{\{1,\dots,d\}}{t}$, returns a point x with $\mathrm{sim}(q, x) \ge s_2$, although only if X contains some (possibly different) x' with $\mathrm{sim}(q, x') \ge s_1$.

Theorem 8. *Assume there is a Turán (d, a, r) design S, where $a = \frac{2s_1}{1+s_1}t$ and $r = \frac{\log n}{\log(1+s_2)/(2s_2)}$, and an efficient decoding algorithm A.*

Then there is an algorithm solving the approximate similarity search problem with $O(|S|n^\eta)$ time per query and $O(|T|n^{1+\eta})$ space, where $\eta = \frac{\log d/t}{\log(1+s_2)/(2s_2)}$, and $|S|$ is the size of the design.

Proof. We'll first explain the algorithm, and then show that it solves the problem correctly and efficiently.

Building the data structure We first pick a permutation $\sigma : U \to U$ uniformly at random. Then we create a single hash-table T, and for each data point $x \in X$, we decode $A(\sigma x) = \{r_1, r_2, \dots\}$ and store x in each bucket $T[r_i]$. As always we allow multiple points to be stored in the same hash-table bucket.

Querying the data structure Given a point $q \subseteq \{1, \dots, d\}$, we decode $A(\sigma q) = \{r_1, r_2, \dots\}$. We then look at all points x in buckets $T[r_i]$ and compute their similarity to q, returning the point if $\text{sim}(q, x) \geq s_2$.

To see that the algorithm is correct, we just need to show that if a point $x \in X$ has $\text{sim}(q, x) \geq s_1$, then it collides with q in one of the tables.

For this, note that if x has similarity more or equal to s_1 with q, then $|x \cap q| \geq a$. By the definition of Turán designs (definition 6), this means that there is some subset $R \subseteq x \cup q$ which is a block in S. Since R is also a subset of x and q, the two points will collide in bucket $T[R]$.

Next we analyze the complexity of the algorithm. The algorithm is dominated by three parts: (1) Decoding subsets, (2) accesses to the tables $T[R]$, and (3) computing similarities with collisions.

By assumption that the design is efficient, (1) and (2) both take time proportional to the number of blocks we get when decoding a set. Because we shuffle the elements of U with the permutation σ, the number of such blocks is random. By linearity of expectation, the expected number of such blocks is $|S|$ times the probability that a random r-subset of $\{1, \dots, n\}$ is also a subset of a t-set. Thus we have, in expectation, $(1) + (2) = O(|S|\binom{t}{r}/\binom{d}{r})$.

For (3) we likewise calculate the expected number of collisions, which is at most n times the probability that two points with similarity $\leq s_2$ collide. Such points collide exactly when there is a block in their intersection. Since $|x \cap q| \leq \frac{2s_2}{1+s_2}t$ for such sets, this happens at most $\binom{\frac{2s_2}{1+s_2}t}{r}$ times out of $\binom{d}{r}$.

Using theorem 14 (1)+(2) is bounded by $|S|(t/d)^r$ and (3) is bounded by $n|S|(\frac{2s_2}{1+s_2}t/d)$. To balance, we set $r = \frac{\log n}{\log(1+s_2)/(2s_2)}$. Then (1)+(2)+(3) is bounded by $O(|S|n^\eta)$ as in the theorem. □

3.2 An efficiently decodable Turán construction

Corollary 1. *There is a constructive algorithm for the approximate similarity search problem with $O(n^\rho)$ time per query and $O(n^{1+\rho})$ space, where $\rho = \frac{1+\log(1+s_1)/(2s_1)}{\log(1+s_2)/(2s_2)}$.*

We can now prove the corollary using a simple, constructive Turán design construction. That is, to construct a (d, a, r)-design, we arbitrarily partition the coordinates into a/r parts of size dr/a. For each such set of coordinates, we take all of the r-subsets as blocks, giving us $\frac{a}{r}\binom{dr/a}{r}$ blocks in total. Any a-set must have at least $a/(a/r) = r$ elements in one of the sets, and so one of the r-subsets of this set is a block in the a-set.

This construction gives the corollary:

Proof. Assume k/r is an integer b and d/b is integer as well. The decoding algorithm can work like this: For a point $x \in \{1, \dots, d\}$:

1. Sort the values of x according to the natural order on $\{1, \dots, d\}$.

2. For each partition $P_i = \{ib + 1, ib + 2, \dots, (i + 1)b\}$ $(0 \leq i < d/b)$, take the intersection $I = P_i \cap x$ and calculate all subsets of I using an efficient algorithm.

After taking time $t \log t$ for sorting, the intersections can be done fast, using a sweep over the elements. In total the decoding time is thus $t \log t$ plus the size of the output.

Finally we note that

$$\begin{aligned}\log|S| + \log n\frac{\log d/t}{\log(1+s_2)/(2s_2)} &\le r\log\left(\frac{ed}{a}\right) + \log n\frac{\log d/t}{\log(1+s_2)/(2s_2)}\\ &= \log n\frac{\log et/a}{\log(1+s_2)/(2s_2)}\\ &= \log n\frac{1+\log(1+s_1)/(2s_1)}{\log(1+s_2)/(2s_2)}\end{aligned}$$

which is the corollary. □

We note that the algorithm is only a factor of $n^{1/(\log(1+s_2)/(2s_2))}$ from the optimal. For small s_2 we may be surprised how well this simple methods works. However for moderately small $s_2 = 1/10$, the overhead is still more than a factor $\sqrt{n}$.

In the next sections we will see how to get the overhead down to $n^{o(1)}$ for any s_1, s_2.

3.3 Using K-restrictions or just Randomness?

Theorem 9. *For $n < m$ and $k = o(\log m/\log\log m)^2$, we can construct a Turán (n,k,r) design of size $(n/k)^r e^{r^2/k} m^{o(1)}$ in time proportional to its size times $m^{o(1)}$.*

We first note how not to prove the above theorem, using k-restrictions:

Let D be a product distribution over $\Sigma^n = \{0,1\}^n$ with probability r/n for picking 1 and $1 - r/n$ for picking 0 at each coordinate. If we consider R sampled from D as a subset of $\{1,\dots,n\}$, the probability that it has size r is $\binom{n}{r}(r/n)^r(1-r/n)^{n-r} \ge \binom{n}{n/2}2^{-n} \ge 1/\sqrt{4n}$. Conditioned on R having size r, the probability that it is a subset of a particular size k subset is $\binom{k}{r}/\binom{n}{r}$. Hence the density is $\epsilon = \binom{k}{r}/\binom{n}{r}/\sqrt{4n}$. We make $s = \binom{n}{k}$ demands $f_K(a_1,\dots,a_n) = [R = \{i \mid a_i = 1\} \subseteq K \text{ and } |R| = r]$.

By [?] such product distributions are efficiently approximateable, and can thus be used with the k-restriction theorem 3. A design of size $2\sqrt{4n}(n\log n + \log\binom{n}{k})\binom{n}{r}/\binom{k}{r} = O(\binom{n}{r}/\binom{k}{r}n^{3/2}\log n)$ can thus be produced in time $n^{O(n)}$.

While this gives us a derandomized construction, the $n^{O(n)}$ time requirement is just too large. Note that we got basically $n^{O(t)}$ when we got when we constructed bottom-k designs. The problem is that it's not clear how to sample subsets from a randomness source without full independence. However, since our algorithms use full randomness anyway, in terms of the permutations and partitions used, we might also use another process:

Proof. We sample $m = \lceil\binom{n}{r}/\binom{k}{r}(k\log n + 1)\rceil$ R-sets independently, uniformly at random, and check that every K-subset of $\{1,\dots,n\}$ contain an R-set. By the union bound, this succeeds with probability at least

$$1 - \binom{n}{k}\left(1 - \binom{k}{r}\bigg/\binom{n}{r}\right)^m \ge 1 - n^k e^{-k\log n - 1} \ge 1 - e^{-1} \ge 1/2.$$

Thus if we fail, we can simply restart, and expectation we'll have to try only twice. Each try takes time $n^{O(k)}$ and so we get a Turán (n,k,r) design of size m in expected time $2n^{O(k)}$. Note that $m = O((n/k)^r e^{r^2/k} k\log n)$.

As in the previous chapter, we improve on the simple construction, by using a splitter. Where for bottom-k designs we needed a 2-way splitter because we had to split both the intersection and the difference, for Turán design we only really have to split the intersection. However because Turán designs grow rapidly with the size of the universe, we need to partition the coordinates in equal size sets. A simple way to do this is to again use a 2-way splitter, splitting X as well as $\{1,\dots,n\} \setminus X$. After splitting, both parts will have size most $\lceil |X|/B \rceil + \lceil (n-|X|)/B \rceil \le \lceil n/B \rceil + 1$. Any partition with a set of size larger than this can then be ignored, since it can never satisfy a pair X, $\{1,\dots,n\} \setminus X$.

For some $m > n$, let $k = o(\log m/\log\log m)^2$ and $B = \sqrt{k}$.

1. Construct a 2-way splitter S, such that for any disjoint $S_1 \cup S_2 = \{1,\dots,n\}$, there is a partition of $\{1,\dots,n\}$ in B sets X_i, such that for each i, $|S_1 \cap X_i| = |S_1|/B$ and $|S_2 \cap X_i| = |S_2|/B$ up to rounding.

2. Construct a $(\lceil n/B \rceil + 1, \lfloor k/B \rfloor, \lceil r/B \rceil)$ Turán design, T, of size $O((n/k)^{r/B} e^{r^2/(Bk)} k/B \log(n/B))$ in time $n^{O(k/B)} = m^{o(1)}$.

3. Let $f_X : \{1,\dots,n\} \to \{1,\dots,\lceil n/B \rceil + 1\}$ be the function that gives the ordinal position of $i \in X$, that is $f_X(i) = |\{j \in X \mid j \le i\}|$. For each partition $X_1,\dots,X_B$ and each combination of B (possibly equal) blocks $R_1,\dots,R_B \in T$, output $\bigcup_i f_{X_i}^{-1}(R_i)$ where $f_X^{-1}(R) = \{i \in \{1,\dots,n\} \mid f_{X_i}(i) \in R\}$. The output has size $|S||T|^B$ and is constructed in time proportional in the output.

The construction gives a (n,k,r) Turán design. We see this by considering any set $K \subseteq \{1,\dots,n\}$ of size k. By construction, there will be a split of K and $\{1,\dots,n\} \setminus K$ into sets $X_1,\dots,X_B$ of size most $\lceil n/B \rceil + 1$. Each part will contain at least $\lfloor k/B \rfloor$ elements of K, and thus the inner Turán design T will have an R of size at least $\lceil r/B \rceil$ contained in those elements. The union of these R-sets, projected on the splits, will have size at least $B\lceil r/B \rceil \ge r$ and be contained in K.

The size of the design is

$$\begin{aligned}
|S||T|^B &= n^{O(B)}((n/k)^{r/B} e^{r^2/(Bk)} k/B \log(n/B))^B \\
&= m^{o(1)}(n/k)^r e^{r^2/k} (k/B \log(n/B))^B \\
&= (n/k)^r e^{r^2/k} m^{o(1)}
\end{aligned}$$

And was constructed in time proportional in this plus $m^{o(1)}$. □

3.4 Making Designs Decodable with Necklace Tensoring

Theorem 10. *For any positive integer c such that $ck \le n$, assume we have an (n,k,r) design T that can decode size t sets in time $\tau(t)$. Then we can make an (n,ck,cr) design of expected size $|T|^c c^{-cr}(en)^c$, which can decode size t sets in time $\tau'(t) = (en)^c \tau(t) + m$ where m is the size of the output.*

Proof. We make a c-splitter using the splitter theorem, and shuffle it over $\{1,\dots,n\}$.

The decoding algorithm works like this: Given a set $X \subseteq \{1,\dots,n\}$, we enumerate the splitter S, and for each $\pi : [n] \to [c]$ we decode each of $\pi^{-1}(1) \cap X, \pi^{-1}(2) \cap X, \dots$ into sets of blocks, $\{R_{1,1}, R_{1,2}, \dots\}, \{R_{2,1}, \dots\}, \dots$. We return the union of each tuple of blocks in the product of the c sets.

The entire design is simply what the above algorithm returns, when decoding the entire $\{1,\dots,n\}$.

This is an (n,ck,cr) splitter, since for any set of size at least ck, there is a π partitioning the set in set of size exactly k. Each of these must have at least one r-set in the inner design. The union of these have size cr, since the k-sets were disjoint, and so the ck set contains a cr block.

It's hard to say how big exactly our set will be, but we can give an expectation, which will be enough for our purposes. If we decode a size t set, each partition splits it into some sizes

$t_1, t_2, \ldots, t_c$. Then the output has expected size

$$\prod_i \binom{n_i}{r} \frac{|T|}{\binom{n}{r}} \le (n_1 n_2 \cdots n_c)^r \frac{|T|^c}{n^{cr}} \le \frac{|T|^c}{c^{cr}}$$

times the size of the splitter, $|S| = (c-1)^c \binom{n}{c-1} \le n^{c-1}(c-1)e^{c-2} \le (en)^c$. □

Note that the more classical approach to tensoring would give a size $\binom{|T|}{c} \le |T|^c$ design. That approach would take every c (disjoint) blocks and combine them. The c^{cr} factor, however, turns out to be very significant for getting all the way to an optimal algorithm.

3.5 Algorithm with Partitioning

In this section we show how a chain of random construction + multi-way splitters + perfect hashing + tensoring + partitioning can combine to give an optimal exact solution to the approximate set similarity problem.

Theorem 11. *Assume data points $x \in X$ and queries $q \in U$ all have size t, there is a data-structure for the (s_1, s_2)-exact similarity search problem over $\{1, \ldots, d\}$ with Jaccard similarity with space usage $O(n^{1+\rho+o(1)})$ and query time $O(n^{\rho+o(1)})$, where*

$$\rho = \frac{\log 2s_1/(1+s_1)}{\log 2s_2/(1+s_2)}.$$

This matches the best non-exact algorithms up to lower order terms.

Building the data structure: Given $X' \subseteq ds$ of size n, we then take a random partition of $\{1, \ldots, d\}$ into B segments of size d/B by repeated sampling without replacement. We build a Turán design with theorem 9, and for each segment we build the data structure from theorem 8.

Querying the data structure: Given a point $q \subseteq \{1, \ldots, d\}$, we take each projection over the B segments and query the inner data-structure as in theorem 8. We modify the inner data structure, so if it would return a point x, we calculate $\mathrm{sim}(q, x)$, and the if the value is larger than s_2, the search is continued.

Let $b_1 = \frac{2s_1}{1+s_1}$, Let $b_2 = \frac{2s_2}{1+s_2}$, Assume $b_1 t > (\log n)^{3/2}$, otherwise duplicate the coordinates.

Let $c = \left\lceil \frac{\log \frac{d}{b_1 t \log 1/b_1} + \log\log \frac{d}{b_1 t \log 1/b_1} + 1}{\log 1/b_1} \right\rceil = O(\log d/t)$. then by lemma 6 we have $c \ge \log \frac{dc}{b_1 t} / \log \frac{1}{b_1}$. With perfect hashing, we can assume that $d = t^{O(1)}$, thus $c = O(1)$. And $B = \lceil \frac{bt}{c} (\log n)^{-3/2} \rceil$. And $r = \lceil \frac{\log n}{\log 1/b_2} \rceil$. And $\rho = \frac{\log 1/b_1}{\log 1/b_2}$.

Then

$$\frac{b_1 t}{cB} = (\log n)^{3/2} = o\left(\frac{\log n}{\log\log n}\right)^2$$

1. Make a $(d/B, b_1 t/(cB), r/c)$ Turán design, T, using theorem 9. We get

$$|T| \le \left(\frac{dc}{b_1 t}\right)^{r/c} e^{r^2 B/(b_1 tc)} n^{o(1)} \tag{3.1}$$

$$= e^{\frac{\log n}{\log 1/b_2} \frac{\log(dc)/(b_1 t)}{c}} e^{r^2/(c^2 (\log n)^{3/2})} n^{o(1)} \tag{3.2}$$

$$= n^{\rho+o(1)} \tag{3.3}$$

And the construction time proportional in $|T|$ times $n^{o(1)}$.

2. Next use theorem 10 to convert the design into a $(d/B, b_1t/B, r)$ design. The original design could be decoded in time $n^{\rho+o(1)}$, so the new one can decode size s sets with overhead $n^{\rho+o(1)}$. The size is $(d/(b_1t))^r d^c n^{o(1)} = (d/(b_1t))^r n^{o(1)}$.

Our algorithm is correct, because one of the segments have $\pi(x) \cap \pi(q) \geq b_1t/B$, and so it can be returned by the inner construction.

For performance, we have the same three parts as always: (1) Decoding time, (2) Looking in the buckets and (3) Collisions.

Let S be a random variable, describing the size of a size t set projected to a segment. This is hyper-geometrically distributed as d/B draws from d with t successes. The Turán designs have size $n^{c\rho+o(1)} = \binom{dc}{b_1t} n^{o(1)}$ out of $\binom{d/B}{r}$ possible blocks. By linearity of expectation we are going to 'catch'

$$E_S\binom{S}{r}\left(\frac{d}{b_1t}\right)^r n^{o(1)} \bigg/ \binom{d/B}{r} \leq \left(\frac{1}{b_1}\right)^r n^{o(1)}$$

3.6 Conclusion

We have shown how the optimal algorithm for set similarity search could be made to have no false negatives. Doing so required extra work compared to the previous chapter, since we had to consider decoding time for our designs. The successes here indicate good possibilities for making more near neighbor data structures exact in the future.

There are two main open problems:

1. Our methods easily build covering designs, which can be used to solve the approximate hamming distance problem. It seems likely that they can also imitate the 'cap carving' algorithms for this problem, which obtain better exponents for large distances. Most interesting is that such a construction would likely allow an optimal exact solution to the approximate l_2 distance problem. (Please don't work on this ;-))

2. It would be interesting to make an efficient decodable Turán designs using a more direct approach. One idea, that works better than corollary 1 for most parameters, is using a hash function to map coordinates to F_2^{bt-1} and using the fact that any sets of bt vectors in $bt - 1$ dimensions are linearly dependent.

Bibliography

[1] Thomas D Ahle, Rasmus Pagh, Ilya Razenshteyn, and Francesco Silvestri. On the complexity of inner product similarity join. *arXiv preprint arXiv:1510.02824*, 2015.

[2] Noga Alon and Shachar Lovett. Almost k-wise vs. k-wise independent permutations, and uniformity for general group actions. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 350–361. Springer, 2012.

[3] Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms (TALG)*, 2(2):153–177, 2006.

[4] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1018–1028. Society for Industrial and Applied Mathematics, 2014.

[5] Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. *arXiv preprint arXiv:1608.03580*, 2016.

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

[7] Alexandr Andoni and Ilya Razenshteyn. Tight lower bounds for data-dependent locality-sensitive hashing. *arXiv preprint arXiv:1507.04299*, 2015.

[8] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *Proceedings of the 32nd international conference on Very large data bases*, pages 918–929. VLDB Endowment, 2006.

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

[10] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.

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

[12] Arthur Cayley. Iv. on the triadic arrangements of seven and fifteen things. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 37(247):50–53, 1850.

[13] Chen Chen, Xifeng Yan, Philip S Yu, Jiawei Han, Dong-Qing Zhang, and Xiaohui Gu. Towards graph containment search and indexing. In *Proceedings of the 33rd international conference on Very large data bases*, pages 926–937. VLDB Endowment, 2007.

[14] Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. *arXiv preprint arXiv:1605.02687*, 2016.

[15] Vašek Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.

[16] Charles J Colbourn and Jeffrey H Dinitz. *Handbook of combinatorial designs*. CRC press, 2006.

[17] Paul Erdös and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[18] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

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

[20] Piotr Indyk. A small approximately min-wise independent family of hash functions. *Journal of Algorithms*, 38(1):84–90, 2001.

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

[22] Toshiya Itoh, Yoshinori Takei, and Jun Tarui. On permutations with limited independence. In *Symposium on Discrete Algorithms: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, volume 9, pages 137–146. Citeseer, 2000.

[23] Toshiya Itoh, Yoshinori Takei, and Jun Tarui. On the sample size of k-restricted min-wise independent permutations and other k-wise distributions. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 710–719. ACM, 2003.

[24] Peter Keevash. Hypergraph turán problems. *Surveys in combinatorics*, 392:83–140, 2011.

[25] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[26] Jiří Matoušek and Miloš Stojaković. On restricted min-wise independence of permutations. *Random Structures & Algorithms*, 23(4):397–408, 2003.

[27] Moni Naor, Leonard J Schulman, and Aravind Srinivasan. Splitters and near-optimal derandomization. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 182–191. IEEE, 1995.

[28] Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[29] Serguei Aleksandrovich Norin. A polynomial lower bound for the size of any k-min-wise independent set of permutations. *Zapiski Nauchnykh Seminarov POMI*, 277:104–116, 2001.

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

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

[32] Rasmus Pagh and Tobias Lybecker Christiani. Beyond minhash for similarity search. personal communication, Slack.

[33] Michael Saks, Aravind Srinivasan, Shiyu Zhou, and David Zuckerman. Low discrepancy sets yield approximate min-wise independent permutation families. *Information Processing Letters*, 73(1):29–32, 2000.

[34] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014.

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

[36] Mikkel Thorup Søren Dahlgaard, Christian Igel. Nearest neighbor classification using bottom-k sketches. In *Big Data, 2013 IEEE International Conference on*, pages 28–34. IEEE, 2013.

[37] Mikkel 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.

[38] Flemming Topsok. Some bounds for the logarithmic function. *Inequality theory and applications*, 4:137, 2006.

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

Appendix A

A.1 Embeddings for Multi-sets

We can use embeddings as a way to generalize our results to different similarity measures. Since the host space is discrete, we have to round values to integers. E.g. if we want to work over vectors from $[0,1]^d$, we might scale by a factor 100 and map from $\{0,\dots,100\}^d$ instead, incurring errors of order 1/100. This way, we can loosely see the maximum value, k, in the following embeddings, as ϵ^{-1} where ϵ is the error we want.

We present two simple embeddings of multi-sets into simple sets, which preserve different generalizations of Jaccard similarity.

Theorem 12. *There is a mapping $f : \{0,\dots,k\}^d \to \{0,1\}^{kd}$, such that sim$(f(x), f(y)) = \frac{\sum_i \min(x_i,y_i)}{\sum_i \max(x_i,y_i)}$.*

Proof. Define $\hat{f}(n) = (\underbrace{1,\dots,1}_{n \text{ times}}, \underbrace{0,\dots,0}_{k-n \text{ times}})$ and let f be the concatenation $f(x) = \hat{f}(x_1)\dots\hat{f}(x_d)$.

It's easy to check that we get

$$\begin{aligned} \langle f(x), f(y)\rangle &= \sum_i \langle \hat{f}(x_i), \hat{f}(y_i)\rangle = \sum_i \min(x_i, y_i), \\ |f(x)| &= \sum_i x_i, \\ \text{sim}(f(x), f(y)) &= \frac{\sum_i \min(x_i, y_i)}{\sum_i x_i + \sum_i y_i - \sum_i \min(x_i, y_i)} = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)}. \end{aligned}$$

□

Another generalization of Jaccard similarity is sometimes called Tanimoto distance. This similarity between vectors x and y is $\frac{\langle x,y\rangle}{\|x\|_2^2 + \|y\|_2^2 - \langle x,y\rangle}$. We don't quite have an exact way to embed this, but we can get the following:

Theorem 13. *There is an asymmetric mapping $f, g : \{0,\dots,k\}^d \to \{0,1\}^{k^2 d}$, such that sim$(f(x), g(y)) = \frac{\langle x,y\rangle}{k\|x\|_1 + k\|y\|_1 - \langle x,y\rangle}$.*

Proof. Define

$$\hat{f}(n) = \left(\left.\begin{matrix} \overbrace{\begin{matrix} 1 & \dots & 1 \end{matrix}}^{n} & \overbrace{\begin{matrix} 0 & \dots & 0 \end{matrix}}^{k-n} \\ \begin{matrix} \vdots & & \vdots \end{matrix} & \begin{matrix} \vdots & & \vdots \end{matrix} \\ \begin{matrix} 1 & \dots & 1 \end{matrix} & \begin{matrix} 0 & \dots & 0 \end{matrix} \end{matrix}\right\} k \right), \quad \hat{g}(n) = \left(\begin{matrix} \left.\begin{matrix} \overbrace{\begin{matrix} 1 & \dots & 1 \end{matrix}}^{k} \\ \begin{matrix} \vdots & & \vdots \end{matrix} \\ \begin{matrix} 1 & \dots & 1 \end{matrix} \end{matrix}\right\} n \\ \left.\begin{matrix} 0 & \dots & 0 \\ \vdots & & \vdots \\ 0 & \dots & 0 \end{matrix}\right\} k-n \end{matrix} \right),$$

where the matrices are read as vectors in some order.

Here $|\hat{f}(n)| = |\hat{g}(n)| = kn$ and $\langle \hat{f}(n), \hat{g}(m)\rangle = nm$. Thus we can define f and g by concatenation, $f(x) = \hat{f}(x_1) \cdots \hat{f}(x_d)$, $g(x) = \hat{g}(x_1) \cdots \hat{g}(x_d)$.

This gives us

$$\begin{aligned} \langle f(x), g(y)\rangle &= \langle x, y\rangle, \\ |f(x)| = |g(x)| &= k \sum x_i, \\ \operatorname{sim}(f(x), g(y)) &= \frac{\langle x, y\rangle}{k\|x\|_1 + k\|y\|_1 - \langle x, y\rangle}. \end{aligned}$$

□

A.2 The Ratio of Two Binomial Coefficients

Classical bounds for the binomial coefficient: $(n/k)^k \le \binom{n}{k} \le (en/k)^k$ give us simple bounds for binomial ratios, when $n \ge m$: $(n/em)^k \le \binom{n}{k}/\binom{m}{k} \le (en/m)^k$. The factor e on both sides can often be a nuisance.

Luckily tighter analysis show, that they can nearly always be either removed or reduced. Using the fact that $\frac{n-i}{m-i}$ is increasing in i for $n \ge m$, we can show $\binom{n}{k}/\binom{m}{k} = \prod_{i=0}^{k-1} \frac{n-i}{m-i} \ge \prod_{i=0}^{k-1} \frac{n}{m} = \left(\frac{n}{m}\right)^k$. This is often sharp enough, but on the upper bound side, we need to work harder to get results.

Let $\mathrm{H}(x) = x \log 1/x + (1-x) \log 1/(1-x)$ be the binary entropy function,

Theorem 14. *For $n \ge m \ge k \ge 0$ we have the following bounds:*

$$\left(\frac{n}{m}\right)^k \le \left(\frac{n}{m}\right)^k \exp\left(\frac{n-m}{nm}\frac{k(k-1)}{2}\right) \le \binom{n}{k} \Big/ \binom{m}{k} \le \frac{\exp\left(n\,\mathrm{H}(k/n)\right)}{\exp\left(m\,\mathrm{H}(k/m)\right)} \le \left(\frac{n}{m}\right)^k e^{k^2/m}$$

If $m \ge n$ we can simply flip the inequalities and swap n for m. Note that $(n/em)^k \le (n/m)^k$ and $e^{k^2/m} \le e^k$, so the bounds strictly tighten the simple bounds states above.

Especially the entropy bound is quite sharp, since we can also show: $\frac{\exp((n+1)\,\mathrm{H}(k/(n+1)))}{\exp((m+1)\,\mathrm{H}(k/(m+1)))} \le \binom{n}{k}/\binom{m}{k}$, though for very small values of k, the lower bound in the theorem is actually even better. We can also get a feeling for the sharpness of the bounds, by considering the series expansion of the entropy bound at $k/m \to 0$: $\frac{\exp(n\,\mathrm{H}(k/n))}{\exp(m\,\mathrm{H}(k/m))} = \left(\frac{n}{m}\right)^k \exp(\frac{n-m}{nm}\frac{k^2}{2} + O(k^3/m^2))$.

For the proofs, we'll use some inequalities on the logarithmic function from [**?**]:

$$\log(1+x) \ge x/(1+x) \tag{A.1}$$

$$\log(1+x) \ge 2x/(2+x) \text{ for } x \ge 0 \tag{A.2}$$

$$\log(1+x) \le x(2+x)/(2+2x) \text{ for } x \ge 0. \tag{A.3}$$

In particular (A.2) and (A.3) imply the following bounds for the entropy function:

$$\mathrm{H}(x) \le x \log 1/x + x(2-x)/2 \tag{A.4}$$

$$\mathrm{H}(x) \ge x \log 1/x + 2x(1-x)/(2-x), \tag{A.5}$$

which are quite good for small x.

We'll prove theorem 14 one inequality at a time, starting from the left most:

Proof. The first inequality follows simply from $\frac{n-m}{nm}\frac{k(k-1)}{2} \ge 0$, which is clear from the conditions on $n \ge m \ge k$.

The second inequality we prove by using (A.1), which implies $1+x \geq \exp(x/(1+x))$, to turn the product into a sum:

$$\begin{aligned}
\binom{n}{k}\Big/\binom{m}{k} &= \prod_{i=0}^{k-1} \frac{n-i}{m-i} \\
&= \left(\frac{n}{m}\right)^k \prod_{i=0}^{k-1} \frac{1-i/n}{1-i/m} \\
&= \left(\frac{n}{m}\right)^k \prod_{i=0}^{k-1} \left(1 + \frac{i/m - i/n}{1-i/m}\right) \\
&\geq \left(\frac{n}{m}\right)^k \exp\left(\sum_{i=0}^{k-1} \frac{i(n-m)}{(n-i)m}\right) \\
&\geq \left(\frac{n}{m}\right)^k \exp\left(\sum_{i=0}^{k-1} i\frac{n-m}{nm}\right) \\
&= \left(\frac{n}{m}\right)^k \exp\left(\frac{k(k-1)}{2}\frac{n-m}{nm}\right).
\end{aligned}$$

For the entropy upper bound we will use an integration bound, integrating $\log(n-i)/(m-i)$ by parts:

$$\begin{aligned}
\binom{n}{k}\Big/\binom{m}{k} &= \prod_{i=0}^{k-1} \frac{n-i}{m-i} \\
&= \exp\left(\sum_{i=0}^{k-1} \log\frac{n-i}{m-i}\right) \\
&\leq \exp\left(\int_0^k 1 \log\frac{n-x}{m-x} dx\right) \\
&= \exp\left(x\log\frac{n-x}{m-x}\bigg|_0^k - \int_0^k x\left(\frac{1}{m-x} - \frac{1}{n-x}\right)dx\right) \\
&= \exp\left(k\log\frac{n-k}{m-k} + \int_0^k \left(\frac{m}{m-x} - \frac{n}{n-x}\right)dx\right) \\
&= \exp\left(k\log\frac{n-k}{m-k} - \bigg|m\log\frac{1}{m-x} - n\log\frac{1}{n-x}\bigg|_0^k\right) \\
&= \exp\left(n\,\mathrm{H}(k/n) - m\,\mathrm{H}(k/m)\right).
\end{aligned}$$

The integral bound holds because $\log\frac{n-i}{m-i}$ is increasing in i, and so $\log\frac{n-i}{m-i} \leq \int_i^{i+1}\log\frac{n-x}{m-x}dx$. We see that $\frac{n-i}{m-i}$ is increasing by observing $\frac{n-i}{m-i} = \frac{n}{m} + \frac{in/m-i}{m-i}$ where the numerator and denominator of the last fraction are both positive. The entropy lower bound, mentioned in the discussion after the theorem, follows similarly from integration, using $\log\frac{n-i}{m-i} \geq \int_{i-1}^{i}\log\frac{n-x}{m-x}dx$.

For the final upper bound, we use the bounds (A.4) and (A.5) on $\mathrm{H}(k/n)$ and $\mathrm{H}(k/m)$ respectively:

$$\frac{\exp\left(n\,\mathrm{H}(k/n)\right)}{\exp\left(m\,\mathrm{H}(k/m)\right)} \leq \left(\frac{n}{m}\right)^k \exp\left(\frac{k^2}{2}\left(\frac{1}{m-k/2} - \frac{1}{n}\right)\right) \leq \left(\frac{n}{m}\right)^k \exp\left(\frac{k^2}{m}\right).$$

□

A.3 Moments and Tail-bounds of the Hyper-geometric Distribution

Many of our algorithms use sampling *without* repetitions. In the field of algorithms, this is usually avoided, since sampling *with* repetitions is often more simple to study, and the results tend to be basically the same. However, sampling with repetitions always run the low probability risk of sampling the same items repeatedly, and other such degenerate behavior. Thus it is vital that we have precise tools for working with such probability distributions.

We say that a random variable X is hyper-geometrically distributed with parameters N, K, n, when it counts the number of 'successes' drawn from a population of N items, where K are 'successful' and $N-K$ are 'bad'.

Lemma 5.

$$E\left[\binom{X}{a}\right] = \binom{n}{a}\binom{K}{a}\Big/\binom{N}{a} \qquad \le \binom{n}{a}\left(\frac{K}{N}\right)^a \tag{A.6}$$

$$E\left[s^X\right] \le (1 + \tfrac{K}{N}(s-1))^n \qquad \le e^{n\frac{K}{N}(s-1)} \tag{A.7}$$

$$\Pr[X \ge t] \le \exp(-n\,\mathrm{D}(\tfrac{t}{n} \parallel \tfrac{K}{N})) \qquad \le \exp(-2n(\tfrac{t}{n} - \tfrac{K}{N})^2) \tag{A.8}$$

We'll give a version of a proof by Vašek Chvátal [?], which we adapt to also give us bounds for the binomial and polynomial moments of X. It is interesting to notice that $E\binom{X}{a}$, for constant a and X hyper-geometrically distributed, can be computed exactly:

$$\begin{aligned}
E\left[\binom{X}{a}\right] &= \sum_k \binom{k}{a}\Pr[X=k] \\
&= \sum_k \binom{k}{a}\binom{K}{k}\binom{N-K}{n-k}\Big/\binom{N}{n} \\
&= \sum_k \binom{K-a}{k-a}\binom{N-K}{n-k}\binom{K}{a}\Big/\binom{N}{n} \\
&= \binom{N-a}{n-a}\binom{K}{a}\Big/\binom{N}{n} \\
&= \binom{n}{a}\binom{K}{a}\Big/\binom{N}{a} \qquad \text{(A.9)} \\
&\le \binom{n}{a}\left(\frac{K}{N}\right)^a.
\end{aligned}$$

Here we first used trinomial revision to reduce the number of k's present in the summant, and then Vandermonde convolution to complete the sum.

Unfortunately the moment generating function of X doesn't appear to have a closed form. However, once we have (A.9), we can use the binomial theorem to get a bound for Es^X.

$$\begin{aligned}
E\left[s^X\right] &= E\left[\sum_a \binom{X}{a}(s-1)^a\right] \\
&= \sum_a E\left[\binom{X}{a}\right](s-1)^a \\
&= \sum_a \binom{n}{a}\binom{K}{a}\Big/\binom{N}{a}(s-1)^a \\
&\le \sum_a \binom{n}{a}\left(\frac{K}{N}\right)^a (s-1)^a \\
&= (1 + \tfrac{K}{N}(s-1))^n. \qquad \text{(A.10)}
\end{aligned}$$

To proceed, we had to apply the inequality $\binom{K}{a}/\binom{N}{a} \le (K/N)^a$. This is the only loss we get compared to if X was binomial (as shown below).

We can compare this result to the lower bound from Jensen's inequality: $E[s^X] \ge s^{E[X]} = s^{nK/N}$. The bound is close to the exact ${}_2F_1(-K, -n; -N; 1-s)$ when n is small. If n is close to N, it is better to swap the roles of green and drawn marbles: $K, n \mapsto n, K$, which gives $(1 + \frac{n}{N}(s-1))^{N-K}$.

Finally we can use the 'Chernoff trick' of applying Markov's inequality to the moment generating function, to get a bound for the tail:

$$\begin{aligned}
\Pr[X \ge t] &= \Pr[s^X \ge s^t] \\
&\le E[s^X]/s^t \\
&= (1 + \tfrac{K}{N}(s-1))^n/s^t \\
&= \exp(-n\,\mathrm{D}(\tfrac{t}{n} \parallel \tfrac{K}{N})) && \text{(A.11)} \\
&\le \exp(-2n(\tfrac{t}{n} - \tfrac{K}{N})^2) && \text{(A.12)}
\end{aligned}$$

Here we substituted $s = \frac{N/M-1}{n/t-1}$, which is at least 1 when $t/n \ge K/N$.

It is interesting to compare the bounds with what we get for a binomial distributed $X \sim \mathrm{B}(n, p)$. Here it is possible to evaluate $E\binom{X}{a}$ as well as Es^X exactly:

$$\begin{aligned}
E\left[\binom{X}{a}\right] &= \sum_k \binom{k}{a}\binom{n}{k} p^k (1-p)^{n-k} \\
&= \binom{n}{a} \sum_k \binom{n-a}{k-a} p^k (1-p)^{n-k} \\
&= \binom{n}{a} p^a \sum_k \binom{n-a}{k} p^k (1-p)^{n-k-a} \\
&= \binom{n}{a} p^a
\end{aligned}$$

$$\begin{aligned}
E\left[s^X\right] &= E\left[s^{\sum_{i=1}^n X_i}\right] \\
&= \prod_{i=1}^n E\left[s^{X_i}\right] \\
&= (ps^1 + (1-p)s^0)^n \\
&= (1 + p(s-1))^n
\end{aligned}$$

Only for the tail bound do we incur a bit of loss:

$$\begin{aligned}
\Pr[X \ge t] &= \Pr[s^X \ge s^t] \\
&\le E[s^X]/s^t \\
&= (1 + p(s-1))^n/s^t \\
&= \exp(-n\,\mathrm{D}(\tfrac{t}{n} \parallel p)) && \text{(A.13)} \\
&\le \exp(-2n(\tfrac{t}{n} - p)^2) && \text{(A.14)}
\end{aligned}$$

A.4 Tail bounds for Jaccard Similarity

Say we have two sets $x, y \subseteq \{1, \dots, d\}$ with inner product a and symmetric difference b. That is, they have Jaccard similarity $s = a/(a+b)$. Now we sample B coordinates from d, independently with repetitions. What is the probability that the Jaccard similarity ends up much larger/smaller than s in the sampled coordinates? If we let $f = s + \epsilon$, then we get the following tail-bound:

$$\begin{aligned}
\Pr[X \geq f] &= \sum_{\substack{k,m \\ \frac{k}{k+m} \geq f}} \binom{B}{k,\, m,\, B-k-m} \left(\frac{a}{d}\right)^k \left(\frac{b}{d}\right)^m \left(\frac{d-a-b}{d}\right)^{B-k-m} \\
&\leq \sum_{k,m} \binom{B}{k,\, m,\, B-k-m} \left(\frac{a}{d}\right)^k \left(\frac{b}{d}\right)^m \left(\frac{d-a-b}{d}\right)^{B-k-m} x^{(1-f)k-fm} \\
&= \left(ax^{1-f} + bx^{-f} + d - a - b\right)^B d^{-B} \\
&\leq \left(a\left(\frac{b}{a}\frac{f}{1-f}\right)^{1-f} + b\left(\frac{b}{a}\frac{f}{1-f}\right)^{-f} + d - a - b\right)^B d^{-B} \qquad \text{(A.15)} \\
&= \exp\left[-B\left(\frac{(a+b)^3}{2abd}\epsilon^2 + O(\epsilon^3)\right)\right] \\
&\leq \exp\left[-B\left(\frac{2(a+b)}{d}\epsilon^2 + O(\epsilon^3)\right)\right]
\end{aligned}$$

Here we used $[k/(k+m) \geq f] = [(1-f)k - fm \geq 0] \leq x^{(1-f)k-fm}$ where the inequality holds for any $x > 1$. We choose the value $x = \frac{b}{a}\frac{f}{1-f}$ which is greater than 1 whenever $f > \frac{a}{a+b}$.

We see that setting $B = \log n \frac{d\epsilon^{-2}}{2(a+b)}$ gives us high probability of preserving the Jaccard similarity. Perhaps expectedly, for small $(a+b)$ we need a large B.

Unfortunately we need the $O(\epsilon^3)$ term in the simplest bound. It would be interesting to make a uniform upper bound for (A.15).

A.5 Polylog

Lemma 6. *Assume $b \geq e$. then*

$$\begin{aligned}
&\textit{if } c = \log(b) + \log\log(b) + 1 &&\textit{then } c \geq \log(bc) \\
&\textit{if } c = \log(b) + \log\log(b) &&\textit{then } c \leq \log(bc)
\end{aligned}$$

Proof. From the assumptions, we have $\log\log(b) \geq 0$. By the classical $1 + x \leq e^x$ we get $\frac{\log\log(b)+1}{\log(b)} \leq 1$.

For the upper bound, let $c = \log(b) + \log\log(b) + 1$ then

$$\begin{aligned}
\log(bc) &= \log(b(\log(b) + \log\log(b) + 1)) \\
&= \log(b) + \log(\log(b) + \log\log(b) + 1) \\
&= \log(b) + \log\log(b) + \log(1 + \frac{\log\log(b)}{\log(b)}) \\
&\leq \log(b) + \log\log(b) + \log(2) \\
&\leq c
\end{aligned}$$

If $c = \log(b) + \log\log(b)$ then

$$\begin{aligned}
\log(bc) &= \log(b(\log(b) + \log\log(b))) \\
&= \log(b) + \log(\log(b) + \log\log(b)) \\
&\geq \log(b) + \log\log(b) \\
&= c
\end{aligned}$$

□