IT UNIVERSITY OF COPENHAGEN

Randomized Primitives for Big Data Processing

by

Morten Stöckel

A thesis submitted in partial fulfillment for the degree of PhD

in the
Theoretical Computer Science Section

July 2015

IT UNIVERSITY OF COPENHAGEN

Abstract

by Morten Stöckel

The growth in information technology during the last decade has brought a great increase in the number of users that have access to computers or mobile phones, as well as an increase in the number of data-based services offered to users. For instance, the number of web servers almost doubled from 70 to 135 million during 2005-2007. The growth in users, combined with the growth in services, means that the amount of total data to manage is exploding.

An important query in the field of algorithms asks how much two data sets intersect, that is, the "overlap" between the pieces of data. Such a query is fundamental in applications such as recommender systems, where the answer would be used to measure similarity over shopping patterns and, based on that, recommend items to the user.

In this dissertation we examine the problem of computing intersection sizes among data sets in several applications and in the context of the information explosion. That is, we consider that the data in our applications is too large to be stored entirely or too large to fit in the main memory of the computer. The main contribution of the dissertation is improvement of several fundamental applications of such data intersection computations, such as approximating the set intersection size and multiplying two matrices. The improvements over the current state of the art methods are either in the form of less space required or less time needed to process the data to compute the answer to the query.

IT UNIVERSITY OF COPENHAGEN

Abstract, Danish

by Morten Stöckel

Den voksende udbredelse af informationsteknologi over de sidste årtier har medført en betydelig stigning i antallet af brugere, der har adgang til computere eller mobiltelefoner. Ligeledes har vi set en stigning i antallet af data-baserede services, som bliver udbudt til brugerne. Som et eksempel er antallet af web-servers næsten fordoblet fra 70 til 135 millioner servere i tidsperioden $2005 - 2007$. Denne stigning af brugere, kombineret med stigningen af udbudte services, har resulteret i, at den samlede mængde data der skal håndteres er eksploderet.

En vigtig type forespørgsel i algoritmik søger svar på hvor stort overlappet er mellem to eller flere mængder af data. En sådanne forespørgsel er fundamental for anvendelser såsom rekommandations-systemer, hvori man søger at anbefale en bruger eksempelvis varer at købe. Denne anbefaling kan være baseret på forbrugsmønstre, hvori overlap mellem tidligere indløb indgår til at foretage en klassificering af brugeren.

I denne afhandling tager vi udgangspunkt i flere algoritmiske problemstillinger og anvendelser, der har det til fælles, at udregningen af overlap mellem mængder er nødvendig. Vi undersøger disse problemer i konteksten af den førnævnte "informations-eksplosion", idet vi undersøger scenarier hvori datamængderne er store. Hovedbidraget af afhandlingen er forbedringer til adskillige fundamentale problemer i algoritmik, der relaterer til overlap-udregninger, eksempelvis udregning af matrix-produkter og approximation af mængde-overlap. Forbedringerne i afhandlingen tager form af enten mindre plads brugt, eller et lavere antal operationer nødvendigt for at foretage en udregning, sammenlignet med hidtil kendt arbejde på samme problemstillinger.

Acknowledgements

I would like to start by thanking my advisor Rasmus Pagh for his support throughout the years. I am overall very grateful to have had the opportunity to work with such a talented researcher and positive person as him. I would hence also like to thank my master's thesis advisor Philip Bille for making me more interested in algorithms as well as introducing me to Rasmus.

The algorithms group at ITU is pleasant company — in particular I would like to thank Thore Husfeldt and Jesper Larsson for the interesting lunch conversations, as well as all my previous office mates; Ninh Pham, Konstantin Kutzkov, Tobias Christiani and Johan Sivertsen for the non-technical chats throughout the years.

I had the opportunity to work with some great co-authors besides my advisor — Søren Bøg, Hjalte Wedel Vildhøj and David Woodruff – all of whom I am very thankful to. I further had the opportunity to stay at ETH Zürich with Riko Jacob during the summer 2014, where Riko made the stay a pleasant experience.

I would also like to thank the people in general of the Copenhagen-based algorithms community — in particular over the past year I have enjoyed immensely looking at too difficult problems with the people from DTU and DIKU, sometimes followed by celebratory bar trips.

In terms of writing the thesis I would like to give a special thanks to my advisor Rasmus Pagh, but also to Tobias Christiani, Matthew Skala and Mathias Bæk Tejs Knudsen for many helpful comments.

Finally, I would like to send the biggest thanks to my family and friends for keeping me sane through rough periods — in particular my parents Mikael and Lis, for the immense amount of support, both practical and emotional, and my girlfriend Marie-Louise for calming me down during the final period of writing and preparing this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The amount of data in the world is expanding at a rapid pace. This is a consequence of a global desire to save *information*: when a person performs an action anywhere, the world as a whole has an interest in storing information about that action in order to be able to learn from that experience. Say we could save such information for all actions performed. Then, in theory, a mistake should never be made twice, and if a question has been posed before it could always be answered quickly. However, technical issues arise in such a "store everything" paradigm:

1. Space. Since storing information requires an amount of storage capacity and there are many actions performed in the world, we simply cannot store everything.
2. Time. Even if storage of "everything" were possible, the time required to navigate the information might be infeasible. Methods to quickly answer queries on massive amounts of information would be needed.

Of course, everything is not stored now, but a lot is. In particular, performing actions on the Internet will likely trigger the storage of information about what actions were performed and their results. The facts that the number of users on the Internet is steadily growing,[1] and that the amount of published information is growing as well, equates to the much-discussed problem of *information explosion*. Soon, all books ever written — estimated to be around 130 million [2] — will be available online; and the number of

[1]`http://www.internetlivestats.com/internet-users/`

[2]`http://booksearch.blogspot.dk/2010/08/books-of-world-stand-up-and-be-counted.html`

search engine searches in a single month of 2014 is beyond 17 billion[3]. In short, there is a massive number of users asking questions whose answers are based on a massive amount of information - usually called *data* in the scope of computers. Consider then the following fundamental question:
What is the intersection between two data sets?
Such a question (or query) has well-founded applications, e.g. the answer can be used as a measure of similarity between data as the size of the intersection, informally speaking, states the overlap between the pieces of data. Consider, as a motivating example, *recommender systems*. In this setting the data consists of shopping baskets containing items. The intersection size of two shopping baskets is the number of items the baskets have in common. In this thesis, the aim is to answer such a question with the two main problems (space and time) from above in mind. The main results of the thesis can, broadly speaking, be grouped in three groups:

(a) Data can (often) be "summarized" in order to save space, and yet the question can still be answered based on such summaries. In the recommender systems setting this means that space usage can be reduced by only storing a few cleverly picked items and intersection sizes on the original baskets can still be approximately answered.

(b) Intersection sizes can in certain settings be predicted well. In recommender systems this is essential: predicting the probability of a customer making a specific purchase and then recommending items to buy based on that probability.

(c) If data can be represented sparsely, i.e. it contains little information, then the question can be answered quickly. Likewise, if the answer can be represented sparsely then the answer can be computed quickly. This is generally the case for real life data such as shopping baskets. The number of items in a basket is typically much smaller than the total number of items offered by the shop, which motivates algorithms that uses computational resources proportional to the input size rather than the universe size.

This thesis, in short, aims to provide solutions to a small part of the information explosion issue related to computing intersection among data.

[3]`http://searchengineland.com/google-search-share-stable-bing-continues-cannibalize-yahoo-187124`

1.2 Organization

In general the level of the technical writing in this thesis is such that it should be possible to understand the main ideas as an undergraduate and the technical details as a computer science graduate.

The thesis is organized as follows. We start by stating the models of computation in Section 1.3. Then in Section 1.4 we introduce the technical problems which the thesis provides insight to as well as stating the results achieved and the closest related work. In Section 1.10 we provide a short summary of the fundamental technical tools and notation which are used throughout the thesis.

In Chapter 2 we show how to summarize arbitrary sets into small size such that intersection size queries can still be answered approximately. in Chapter 3 we go into the problems of input- and output-sparse matrix multiplication, which can be seen as many exact intersection queries. Following this, in Chapter 4 we show how the maximum entropy estimate can be used to predict intersection size of three sets in certain settings and in Chapter 5 we consider the problem of reporting point sets that "approximately intersect" in high dimensions. Motivated by the fact that almost all our solutions use hash functions, we examine in Chapter 6 the power of hash functions of limited independence in a fundamental load balancing setting. Finally, in Chapter 7 we end the thesis by summarizing all results presented and propose related open problems and future research directions.

1.3 Models of computation

The goal of theoretical computer science in general is to show upper bounds, i.e., algorithms and data structures to perform some computation and lower bounds which give formal guarantees on how low the upper bound can go. To carry out such arguments we need precise mathematical models in which we can analyze an algorithm or a data structure. The common struggle for such computational models is how well they predict actual (real life) performance vs how hard they are to work with. For upper bounds in this thesis, i.e. to show existence of algorithms and data structures, we use the *word-RAM* model and the *external memory* model. For a specific space lower bound we use the *communication complexity* model. All three models have been extensively researched and the related work discussed in this thesis will constrained to only be closely related problems or techniques.

The word-RAM model is the standard model to use when designing new upper bounds and has been shown to have good predictive power. In short, the cost an algorithm/data structure in this model is the number of "C-style" operations it performs, i.e. each such operation has a price of one in this model. See Section 1.3.1 for full definitions.

When the input data is so large that it does not fit in internal memory of the computer then the number of CPU operations (as counted approximately by the word-RAM model) does not apply. Since disks are slow and the input data resides on a disk then the bottleneck of a computation is now the number of times the disk is read from or written to. This case is captured in the external memory model by Aggarwal and Vitter [1]. In this model the cost of an algorithm or a data structure does not depend on the number of CPU operations but only on how many times the disk is accessed. This model has shown good predictive power for massive data sets and has been widely applied. See Section 1.3.2 for full definitions.

To show lower bounds on space usage we wish to be independent of the instruction set and memory parameters of the machine, that is, it would be preferable to argue that no matter such parameters then we still have a lower bound. Such arguments for lower bounds on space usage can be carried out using communication complexity [2, 3]. The model can be used to make arguments of the following form. If two players, Bob and Alice, hold input x and y respectively and they wish to compute function $f(x, y)$, then how much do they *need* to communicate in order to compute the function? The application of such a setting to space lower bounds of data structures is non-obvious and will be elaborated upon in Chapter 2 and the model itself is discussed further in Section 1.3.3

1.3.1 The word-RAM model

The most used model of computation is the (unit cost) word-RAM model due to its intuitiveness and predictive power. Algorithms and data structures in this model have a random access memory that holds a (theoretically infinite) number of words. Each word consists of w bits and so the address space is $[2^w] = \{0, \dots, 2^w - 1\}$. It assumed that a word has enough bits to store a pointer to any other word and a common assumption is that $w = \Omega(\log n)$ where n is the input size. This assumption is reasonable in the sense that for $w = o(\log n)$ then one cannot even write the number of input elements in a word.

The time cost of an algorithm or data structure in the word-RAM model is the number of word operations performed during computation. The instructions denoted as word operations are the standard "C-style" instructions such as OR, AND, XOR, shifts as well as integer operations such as addition, subtraction, multiplication and division and also comparison operations such as equality check, greater than and smaller than. Further note that it is assumed that a memory access takes constant time. The space cost in this model is simply the largest memory address used during computation as this denotes the maximum number of words that were in use.

1.3.2 The external memory model

When the size of the input data is much larger than the main memory of the computer then the word-RAM from the previous section no longer predicts computation time on real life machines. The reason being that since the input can not reside in the fast main memory it needs to reside in slow external memory (such as a harddisk). Hence the time spent accessing the slow external memory dominates the time spent on CPU instructions. On the other hand, the (latency wise) slow disks are very fast at reading consecutive disk addresses. This is captured in the external memory model, also called the I/O model, by Aggarwal and Vitter [1].

In the model we have a disk which consists of a (theoretically infinite) number of words, where words are usually assumed to be big enough to contain one input element and optionally a pointer or an address. Then connected to the CPU is a main memory that can hold at most M words. The CPU can only do CPU instructions on words that reside in the main memory and these CPU instructions are *free*. Words are transferred from the disk to the main memory in blocks of B consecutive words. One such block transfer is called an I/O. The time complexity of an algorithm or data structure in this model is then the number of I/Os performed during computation and the space complexity is the maximum number of words residing on disk during computation.

We make a distinction between *cache-aware* algorithms, that explicitly use model parameters B or M, and *cache-oblivious* [4] algorithms, that do not use the model parameters. The latter is the most desirable property as it implies optimality on all levels of the memory hierarchy and does not require parameter tuning when executed on different physical machines. The cache-oblivious model assumes that the cache is *ideal* in the sense that it has optimal cache-replacement policy that can evict the block that is used the farthest in the future, and also that a block can be placed anywhere in the cache (full associativity). The model is justified by the fact that such an optimal cache loses only a constant factor in terms of I/Os when converted to a LRU or FIFO cache with limited associativity, [4, Lemmas 12 and 16].

The main tool to show lower bounds on I/O complexity is by Hong and Kung [5] who proved I/O lower bounds e.g. for dense matrix multiplication, n-point fast fourier transform. In the thesis we shall apply an extension of their technique to achieve a lower bound on sparse matrix multiplication, see Section 3.2.6.

1.3.3 Communication complexity

The theory of communication complexity [2, 3] is a well-studied category of complexity theory. Lower bounds for communication complexity have in recent years been the primary tool to prove space lower bounds for streaming algorithms and has been widely applied to show data structure lower bounds as is the case in this thesis, see Section 2.4.

Communication complexity deals with the following scenario. Two players, Alice and Bob, are given input vectors $x \in \{0,1\}^d$ and $y \in \{0,1\}^d$ respectively. Neither players know each others input vector and the goal is to collaborate to compute $f(x, y)$ where the function $f : \{0,1\}^d \times \{0,1\}^d \mapsto \{0,1\}$ is known to both players. The strategy of communication, denoted a *protocol*, is however fixed in advance by the players. The cost of a protocol is then the number of *bits* communicated for worst case inputs x, y. There are several variations of the scenario - in this thesis we will restrict ourselves to that of *one-way* communication: The inputs arrive, Alice sends one message to Bob and then Bob must output $f(x, y)$.

At a high level, lower bounds for bits communicated by any protocol can be used to prove data structure lower bounds due to reductions: Say a solution to the data structure problem implies computing $f(x, y)$, and computing $f(x, y)$ must use $\Omega(g(n))$ bits. Then if there was a solution to the data structure problem that used $o(g(n))$ bits Alice could just run that solution, send the working memory to Bob who would then output $f(x, y)$ using only $o(g(n))$ bits communication. Hence $\Omega(g(n))$ can not be a lower bound on computing $f(x, y)$ and thus a contradiction is reached.

1.4 Overview of problems and results

This section is meant as a brief overview of the problems worked on and new results presented in this thesis. The articles that make up the thesis [6–11] are listed below (in order of appearance).

1. Rasmus Pagh, Morten Stöckel, and David P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2014
   Chapter 2

2. Rasmus Pagh and Morten Stöckel. The input/output complexity of sparse matrix multiplication. In *Algorithms - ESA*, 2014
   Chapter 3

3. Riko Jacob and Morten Stöckel. Fast output-sensitive matrix multiplication. In *Algorithms - ESA*, 2015
   Chapter 3

4. Rasmus Pagh and Morten Stöckel. Association rule mining using maximum entropy. *CoRR*, abs/1501.02143, 2015
   Chapter 4

5. Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. I/O-efficient similarity join. In *Algorithms - ESA*, 2015
   Chapter 5

6. Mathias Bæk Tejs Knudsen and Morten Stöckel. Quicksort, largest bucket, and min-wise hashing with limited independence. In *Algorithms - ESA*, 2015
   Chapter 6

We will define the fundamental problem of computing *intersection sizes*, which is the general umbrella under which the problems dealt with in this thesis fall under. After the general discussion we will introduce the specific problems which the thesis contains improvements to as well as go through the most basic solutions in order to offer some intuition on the problems, followed by stating the most closely related state of the art solutions. The following Section 1.4.1 can be skipped by readers familiar with the basic relation between dot products and set intersection sizes.

1.4.1 Intersections and applications

Perhaps the most fundamental type of data in computer science is that of *binary data*. A binary variable is one that can take only two values: 0 or 1. From a statistical point of view a binary variable is the outcome of a *Bernoulli trial*, which is a (probabilistic) experiment with only two possible outcomes. We say that binary data or a binary data set is a collection of such binary variables. Binary data is fundamental also due to ease of representation - a binary data set can on a computer be seen as just a bit string/vector,

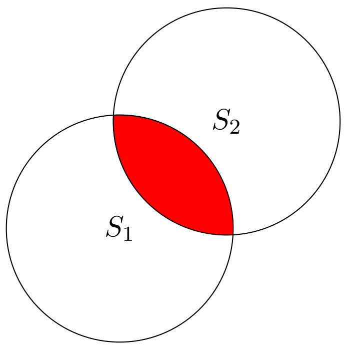


FIGURE 1.1: The binary strings a and b as sets S_1 and S_2. The intersection size $S_1 \cap S_2$ (colored red) denotes the dot product $a \cdot b$.

which in turn can be manipulated by *bitwise operations* that are known to run very fast in practice.

Consider the following simple relation between binary data and sets. Say that we have universe $\mathcal{U} = \{0,1\}^5$ and let $a, b \in \mathcal{U}$ be bit strings from $\mathcal{U}$. For the two binary strings a and b consider then the *dot product* of a and b:

$$a \cdot b = \sum_{i=0}^{4} a_i b_i.$$

This fundamental quantity describes exactly the overlap the two bit vectors have. Now consider constructing the sets S_1 and S_2 from a and b: Let sets S_1 and S_2 hold item i if and only if $a_i = 1$ and $b_i = 1$ respectively. Clearly S_1 and S_2 are now a collection of items hence they are sets. The intersection of S_1 and S_2 is then

$$S_1 \cap S_2 = \{x \mid x \in S_1 \wedge x \in S_2\}.$$

We then have that the size of the set intersection of sets S_1 and S_2 is exactly the dot product of bit strings a and b since on a position in the bit vectors there is a distinct item in the set if and only if the position has value 1. See Figure 1.1 for a view of the setting. Clearly, the word-RAM complexity of computing the dot product and hence the set intersection size of two bit vectors of length n is $\mathcal{O}(n)$ as this is just a sum of n terms each using $\mathcal{O}(1)$ operations. In the I/O model it can be done trivially in $\mathcal{O}(n/B)$ I/Os as this is just the time it takes to scan both bit strings.

In this thesis, we deal with questions related to dot products and set intersection which are not as trivial to compute as above.

In Section 1.5 we answer the question: *If we have m sets and wish to return set intersection sizes of arbitrary subsets of those sets then how many bits do we need to store per set to be approximately correct?*

In Section 1.6 we consider matrix multiplication when either the input or output matrices have few nonzero entries. Notice that a matrix product of two $U \times U$ matrices can be seen as U^2 dot products between length U vectors.

In Section 1.7 we consider how to predict future behavior of the intersection size of three sets based on previous observations of the set sizes and the pair-wise intersection sizes.

Closely related to the above is that of similarity in high dimensions. Note that set intersection and dot products can be seen as a *distance measure.* In Section 1.8 we consider two collections of points from the same metric space and we wish to find the pairs with one element from each collection that are "close" by the distance metric provided. Note that this can simply be seen as regular set intersection except being a member of the intersection requires "closeness" instead of equality.

Finally in Section 1.9 we consider a problem related to the techniques used in the previous sections: Almost all our solutions in this thesis use random hash functions. Typically for analysis one will assume full independence to ease the math, however in practice there exists only efficient hash functions that provide *limited independence* (see Definition 1.9). In light of this, we ask and answer the following fundamental question: *If n balls are thrown into $2n$ bins using a l-independent hash function then what is the maximum number of balls in a single bin?* This question has obvious connections to load balancing and many applications of hash functions require the largest bin to not be "too large".

1.5 Multiple set intersection

In this part of the thesis we will consider the problem of estimating set intersection sizes of many sets. In particular we deal with the issue of storing as little as possible information from each set and still being able to provide a good estimate of the intersection size. Estimates of multiple set intersection have direct applications in approximation algorithms but can be used to optimize performance for exact computation as well. For example, conjunctive queries in databases essentially use running time proportional to the intersection between two relations and for such a sequence of relations the order in which the intersections are computed has a large impact on performance - having a good estimate in short time can be used by an algorithm to optimize this order of computation.

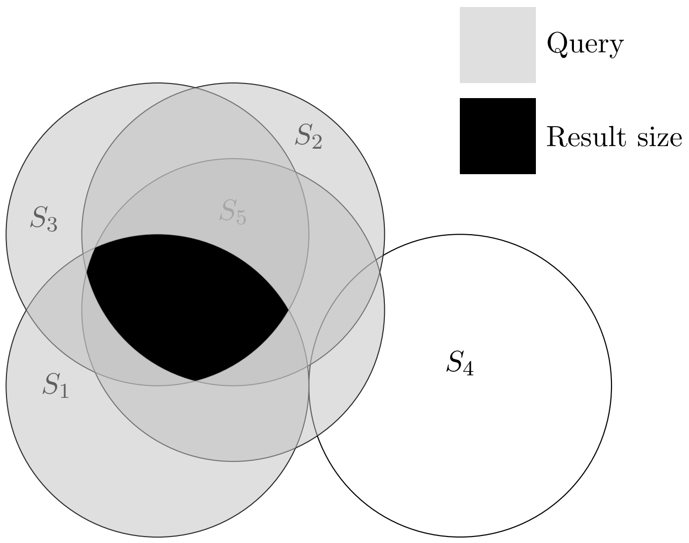


FIGURE 1.2: The setting. We have multiple sets $S_1, \dots, S_5$ with the sets S_1, S_2, S_3, S_5 being the query sets (grey color) and the return value is the size of the black region

The setting is as follows (see Figure 1.2). We have $m \geq 2$ sets $S_1, \dots, S_m$ where each $S_i \subseteq [u]$ and $|S_i| = n$. A query is then of them form: For a particular subset of the m sets what is their intersection size? Formally, let the query q be the set of indices of the sets of interest, then we are interested in the quantity

$$t = |\bigcap_{j \in q} S_j|.$$

If one stores the entire sets $S_1, \dots, S_m$ then one can easily compute the intersection size t: Simply make a pass over the sets and count the elements they have in common using $\mathcal{O}(mn)$ operations in the word-RAM model. On the contrary we consider computing a *summary* of each set and then answering the query approximately based on those summaries alone. The summaries are to be computed independently of each other, i.e., one cannot store intermediate intersection sizes or information about other sets. The need for such an approach comes from the growth of data sizes while many applications still require fast query times (see Section 2.1 for further discussion). First recall the definition of an (ε, δ)-estimate:

Definition 1.8. Let $X \in \mathbb{R}$ and let $\hat{X}$ be a random variable. We say that $\hat{X}$ is an (ε, δ)-*estimate* of X if $\mathbf{Pr}\left[\left|\hat{X} - X\right| \geq \varepsilon X\right] \leq \delta$. We use ε-estimate as shorthand for $(\varepsilon, 1/3)$-estimate.

We address the question: *How many bits is it necessary to allocate to each set summary in order to get a ε-estimate?*

1.5.1 Our results

The models used here are the word-RAM for the upper bounds (stated in bits) and the communication complexity model is used to derive the lower bound.

Informally this result is about how much information one can get from storing the elements that have the smallest hash values - such a summary is called a "MinHash". The state-of-the-art technique for minimizing the summary size, for any desired estimation error, is b-bit min-wise hashing due to Li and König (Communications of the ACM, 2011). Say we wish to store k elements from each set. Then the approach of Li and König is to use k random hash functions to, for each hash function, determine the element from the set with the smallest hash value and then store that element. Our approach is to use a single random hash function and then store the k elements with the k smallest hash values. Another point of view is from the angle of random permutations: They permute the set k times and store the first element of each permutation while we permute the elements once and store the k first elements. The effect we gain by using the one-permutation approach is similar to that of sampling with replacement vs sampling without replacement. Further we show a lower bound which our new approach nearly reaches when there are many sets. Our results can be summarized as follows.

1. Using information complexity arguments, we show that b-bit min-wise hashing is *space optimal* for $m = 2$ sets in the sense that the estimator's variance is within a constant factor of the smallest possible among all summaries with the given space usage. But for $m > 2$ sets we show that the performance of b-bit min-wise hashing (and more generally any method based on "k-permutation" min-hash) deteriorates as m grows.
2. We describe a new summary that nearly matches our lower bound for $m \geq 2$. It asymptotically outperforms all k-permutation schemes (by around a factor $\Omega(m/\log m)$), as well as methods based on subsampling (by a factor $\Omega(\log n_{\max})$, where $n_{\max}$ is the maximum set size).

The specific space usage in bits of the new upper and lower bounds are highlighted in Table 1.1. We will postpone the discussion of the other related techniques present in the table until Chapter 2.

1.6 Matrix multiplication

We will now consider the problem of computing the product of two matrices, typically denoted as the problem of *matrix multiplication*. Matrix multiplication is interesting as

Method	Required space (bits)	Time
Inclusion-exclusion	$s \geq \varepsilon^{-2}(mn/t)^2 + \log n$	2^m
Subsampling	$s \geq \varepsilon^{-2}(n/t)\log m \log^2 n$	sm
b-bit min-wise hashing	$s \geq \varepsilon^{-2}(mn/t)$	sm
New upper bound	$s \leq \varepsilon^{-2}(n/t)\log(m)\log(n/\varepsilon t)$	sm
New general lower bound	$s \geq \varepsilon^{-2}(n/t)$	-

Table 1.1: Comparison of estimators of intersection size t for relative error ε and constant error probability, with m sets of maximum size n. Bounds on the summary size s ignore constant factors.

it sees massive use both in theory and in practice. In practice it is used in computer graphics as well as computational algebra and statistics and in theory many algorithmic problems reduce to matrix multiplication. We refer to Section 3.2 for example applications and discussion.

A $U \times U$ matrix A can be seen as U row vectors each containing U values. The matrix multiplication problem can be stated as follows: Let the *input matrices* be A and C both of size $U \times U$. The task is then to multiply A and C to create the *output matrix* AC which is also of dimensions $U \times U$. If the values of the *entries* of A and C can only take values 0 and 1 we call it boolean matrix multiplication. Letting $[U]$ denote the set $\{0, \dots, U-1\}$, the output matrix AC is defined as

$$\forall i, j \in [U] \ : \ (AC)_{i,j} = \sum_{k=0}^{U-1} A_{i,k} C_{k,j}. \tag{1.1}$$

In this section and in parts of our results the input matrices will be assumed to be quadratic - see Figure 1.3 for an illustration of the general case. Notice that such a matrix product can simply be seen as U^2 dot products between size U row and column vectors. In such a dot product between a row and column vector we say that the product *cancels* if there are nonzero summands but the sum - hence the output entry - is zero. Cancellation of terms proves to be a challenge to achieve good matrix multiplication algorithms: One would like to spend number of operations proportional to the size of the output matrix, but terms that cancel does not count towards the output size but must still be computed.

The standard solution in the word-RAM model is simply to compute Equation (1.1) directly using $\mathcal{O}(U^3)$ operations. In the I/O model the basic solution is to divide the matrices into size $c\sqrt{M} \times c\sqrt{M}$ blocks where c is picked such that three $c\sqrt{M} \times c\sqrt{M}$ fit into internal memory of size M. This reduces the problem to $\mathcal{O}((U/\sqrt{M})^3)$ matrix products that fit in main memory, costing $\mathcal{O}(M/B)$ I/Os each, and hence $\mathcal{O}(U^3/B\sqrt{M})$ in total [12]. Hong and Kung [5] also provided a tight lower bound $\Omega(U^3/B\sqrt{M})$ that

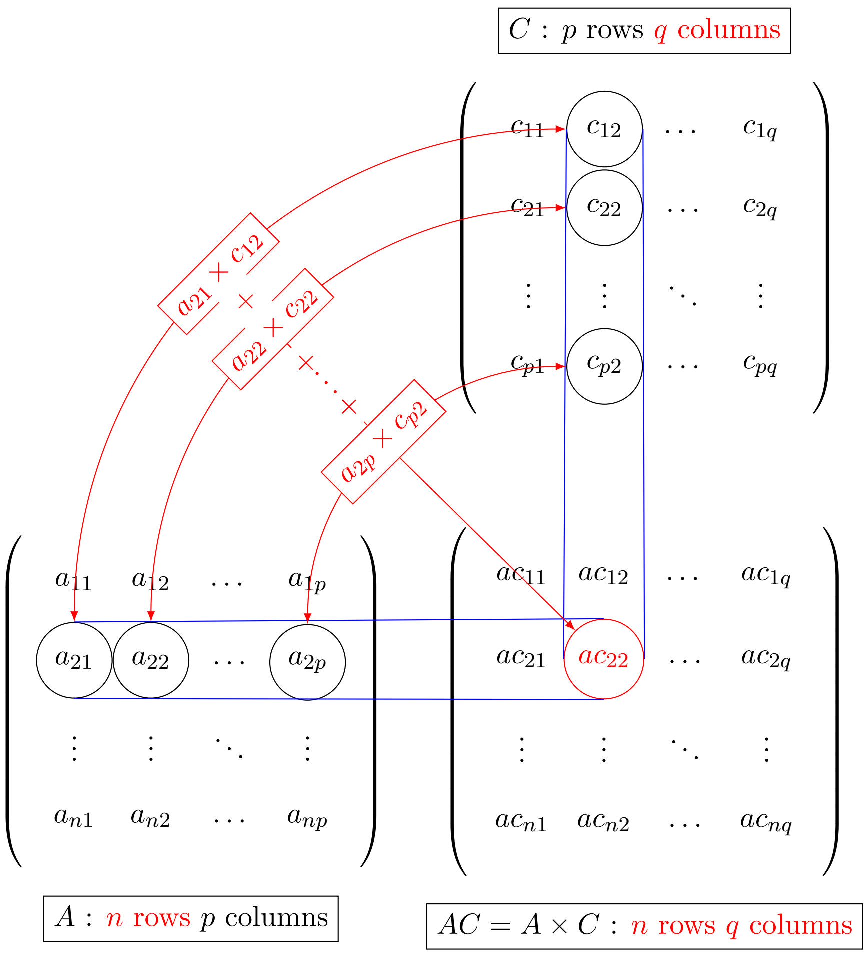

C : p rows q columns
c_{11} c_{12} $\dots$ c_{1q}
c_{21} c_{22} $\dots$ c_{2q}
c_{p1} c_{p2} $\dots$ c_{pq}
$a_{21} \times c_{12}$
$\times$
$a_{22} \times c_{22}$
$\times \dots \times$
$a_{2p} \times c_{p2}$
a_{11} a_{12} $\dots$ a_{1p}
a_{21} a_{22} $\dots$ a_{2p}
a_{n1} a_{n2} $\dots$ a_{np}
ac_{11} ac_{12} $\dots$ ac_{1q}
ac_{21} ac_{22} $\dots$ ac_{2q}
ac_{n1} ac_{n2} $\dots$ ac_{nq}
A : n rows p columns
$AC = A \times C$: n rows q columns

FIGURE 1.3: Matrix A of size $n \times p$ is multiplied with matrix C of size $p \times q$ to create matrix AC of size $n \times q$. Drawing credit to `http://www.texample.net/tikz/examples/matrix-multiplication/`

holds for algorithms that work over a semiring. In the word-RAM model, the naive $\mathcal{O}(U^3)$ bound was improved by Strassen [13] who showed a $\mathcal{O}(U^\omega)$ algorithm for $\omega = \log 7$ by exploiting clever cancellations. The algorithms that use cancellations to its advantage are called “fast matrix multiplication” algorithms or “Strassen-like” algorithms.

We consider the problem of *sparse* matrix multiplication. We say that the input or output matrices are sparse if the number of nonzero entries in them is low. Say N is the number of nonzero entries in the input and Z is the number of nonzero entries in the output. In this thesis we present two new matrix multiplication algorithms, in the I/O and word-RAM model respectively. They share that their running time is proportional to the sparsity of the matrices, namely N and Z instead of the matrix dimension U. This is a natural parametrization of the problem and is justified by the fact that in many applications there are sparse matrices involved.

1.6.1 Our results

For this problem we have new results that improve upon the state of the art in both the I/O model and the word-RAM model. Our algorithms for this problem are all Monte Carlo - they succeed with high probability.

In the I/O model we consider the natural case where we have N nonzero entries in the input matrices and Z nonzero entries in the output matrix and the task is then to get an I/O complexity that depends on those parameters instead of the matrix dimension U. The previous state of the art algorithm for this setting was by Amossen and Pagh [14], which had I/O complexity $\tilde{\mathcal{O}}(N\sqrt{\texttt{nnz}(AC)}/BM^{1/8})$. We improve this bound by a factor of $M^{3/8}$. Contrary to the previous algorithm we also handle cancellations. Our results summarized:

1. We show that using $\tilde{\mathcal{O}}\left(\frac{N}{B}\min\left(\sqrt{\frac{Z}{M}},\frac{N}{M}\right)\right)$ I/Os, AC can be computed with high probability. This is tight (up to polylogarithmic factors) when only semiring operations are allowed, even for dense rectangular matrices:
2. We show a lower bound of $\Omega\left(\frac{N}{B}\min\left(\sqrt{\frac{Z}{M}},\frac{N}{M}\right)\right)$ I/Os, hence closing the gap between lower and upper bounds for this problem.

In the word-RAM model we consider the case of exploiting fast matrix multiplication to get a speedup when the output matrix is sparse. In fact for any matrix multiplication algorithm that can multiply two $U \times U$ matrices in $\mathcal{O}(U^\omega)$ for $2 \leq \omega \leq 3$ we can use that directly in our new algorithm. For matrix size dimension U, number of nonzeros in input N and number of nonzeros in output Z our new algorithm uses $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + Z + N\right)$ operations. In Table 1.2 we show our algorithm against the most related results. We postpone discussion of the related results to Section 3.3.2 but note that we improve upon the start of the art for $Z = \omega(U)$ (when Z is asymptotically larger than U) and we support cancellations.

1.7 Association Rule Mining using Maximum Entropy

In this chapter we consider *association rules* on triples of random variables. For example: Suppose that a customer belongs to categories A and B, each of which is known to have positive correlation with buying product C, how do we estimate the probability that she will buy product C? Making such predictions has practical applications in recommender systems, query completion and more. We consider the case of *low support*: When there is a low (or no) number of observations that correlates A,B and C. Notice the relation

Method	Time	Assumptions
General dense	$\mathcal{O}(U^{\omega})$	
Lingas	$\tilde{\mathcal{O}}\left(U^2 Z^{\omega/2-1}\right)$	Boolean matrices.
Iwen-Spencer, Le Gall	$\mathcal{O}\left(U^{2+\varepsilon}\right)$	$\mathcal{O}\left(U^{0.3}\right)$ nonzeros per column.
Williams-Yu, Pagh	$\tilde{\mathcal{O}}\left(U^2 + UZ\right)$	
This thesis, Theorem 3.15	$\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}} + Z + N\right)$	Balanced rows and columns.
This thesis, Theorem 3.16	$\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + Z + N\right)$	

Table 1.2: Comparison of matrix multiplication algorithms of two $U \times U$ in the RAM model. N denotes the number of nonzeros in the input matrices, Z the number of nonzeros in the output matrix and ω is the currently lowest matrix multiplication exponent.

to set intersection and dot products: We are here trying to predict intersection size of three sets based on the observed set sizes and their pairwise intersection sizes.

For a big data set $\mathcal{D}$ we consider a sample $D \subset \mathcal{D}$. Given such D we wish to estimate event frequencies of $\mathcal{D}$ also in the difficult cases where the events do not occur in D. In particular we will focus on triples: Let I be the set of possible items, $|I| = n$, and D be an $m \times n$ binary data set where each of the m rows D_i encodes a transaction $D_i \subseteq I$. For all singleton- and pair-subsets of I we assume that we know the number of transactions which they occur in, i.e., all singleton and pair frequencies are known. For each $X \subset I$, $|X| = 3$ where the frequency θ_X in D is 0 we then wish to estimate θ_X in $\mathcal{D}$.

The standard approach would be that of *extrapolation.* If you have observed an event X to happen $\text{occ}(X)$ times in data set D then the extrapolation estimate for the probability of X in $\mathcal{D}$ is $\text{occ}(X)/|D|$. However since $\text{occ}(X)$ is low or zero then one would expect this estimate to be imprecise at best. Another approach is to assume independence. Say event X is the occurrence of events X_1,X_2,X_3 at the same time and we know the probabilities of these three events in D then the *independence estimate* is simply those three probabilities multiplied. This has the downside that real life data typically does not behave independently.

1.7.1 Our results

We propose instead to use the *Maximum entropy estimate* which is the probability according to the possible distribution with the highest entropy. In other words the distribution with the least bias possible based on the given observations, which in this case are the single and pair probabilities. Our new results in this can be summarized as:

1. It has been an open problem to provide an explicit formula for the maximum entropy estimate. We partially answer this by providing an explicit formula that

under realistic assumptions gives an approximation to the maximum entropy estimate for triples.

2. Through an experimental evaluation on several real life data sets we show that the maximum entropy estimate gives meaningful estimates where the independence and extrapolation estimates do not.

1.8 Similarity joins in high dimensions

For two sets we consider the problem of computing not the intersection but the subset of points from the two sets that are "close" to being equal, where the measure of closeness is given by the metric space of the input sets. The problem has applications such as web deduplication, document clustering and click fraud detection.

For a space $\mathcal{U}$ and a distance function $d : \mathcal{U} \times \mathcal{U} \to \mathbf{R}$. The *similarity join* of sets $R, S \subseteq \mathcal{U}$ is the following: Given a radius r, compute the set

$$R \underset{\leq r}{\bowtie} S = \{(x, y) \in R \times S \mid d(x, y) \leq r\} \ .$$

Let the total number of points be $N = |R| + |S|$. Since there are $\mathcal{O}(N^2)$ number of pairs the naive approach is to just perform this number of comparisons and so the trivial time bounds for this problem depend on this number of pairs. We go below this barrier by using a special kind of hash function called a *locality sensitive hash function*(LSH): Informally such a hash function will guarantee that two close points hash to the same value with probability p_1 and that two far away points hash to the same value with probability p_2 where $p_1 > p_2$. The rough idea is that since close points hash to the same value often and far away points hash to the same value rarely, then by examining only pairs that hash to the same value we avoid looking at many of the far away pairs.

1.8.1 Our results

We consider the I/O model only and we use the LSH idea as described above in order to avoid comparing far away pairs. Ideally one would like an algorithm that depends on the output size $|R \bowtie_{\leq r} S|$ and a subquadratic dependency in the number of points N. However it turns out that this requires that the "c-near" join $R \bowtie_{\leq cr} S$, for a constant c, has few pairs in it. Our main result is the first I/O-efficient (in fact cache-oblivious) algorithm for similarity join that has provably sub-quadratic dependency on the data size and at the same time inverse polynomial dependency on M. The I/O complexity of our algorithm is

$$\tilde{O}\left(p_1^{-1}\left(\frac{N}{M}\right)^{\rho}\left(\frac{N}{B}+\frac{|R \bowtie_{\leq r} S|}{MB}\right)+\frac{|R \bowtie_{\leq cr} S|}{MB}\right) \text{ I/Os.}$$

Here $\rho \in (0;1)$ is a parameter of the LSH and p_1 is the lower bound on the probability given by the hash function on points that are within distance r hashing to the same value. Where previous methods have an overhead factor of either N/M or $(N/B)^{\rho}$ we obtain an overhead of $(N/M)^{\rho}$, strictly improving both.

1.9 Load Balancing with Limited Independence

Finally we consider a problem that concerns strength of the hash functions used. In both the set intersection, the matrix multiplication and the similarity join results, hash functions are a central part in achieving the stated complexities. For ease of analysis it is convenient to assume that the hash function is fully independent (Definition 1.9). However in practice efficient hash functions exist only for low independence and the lower independence that is needed the faster hash function can be used, roughly speaking.

Alon et al. [15] posed and answered the question "*Is linear hashing good?*" by among other things showing the following. Let $\mathcal{F}$ be a finite field and let $\mathcal{H}$ be the family of all linear transformations between two vector spaces over $\mathcal{F}$. Then they show that there exists a $\mathcal{F}$ with a bad set $B \subset \mathcal{F}^2$ of size n such that a randomly chosen linear transformation from $\mathcal{H}$ on B has constant probability of mapping $\Omega(\sqrt{n})$ elements to the same value. That is, in hashing terms, the largest bucket has size $\Omega(\sqrt{n})$ when hashing n keys to $2n$ buckets using a linear transformation as a hash function.

Since linear transformations are known to be 2-wise independent functions then the natural question arises if the result generalises, namely it would be ideal to examine the question *Is polynomial hashing good?* This would include considering if hashing using a degree $l-1$ polynomial, which are known to be l-wise independent, implies that the largest hash bucket is of size $\Omega(n^{1/l})$ with constant probability.

1.9.1 Our results

We partially answer this question: We examine if l-wise independence in itself is enough to guarantee small hash buckets. Our main result is a family of l-wise independent hash functions, that when used to throw n balls into $2n$ bins, there is a constant probability of the largest bin being of size $\Omega(n^{1/l})$. Such a bound was unknown previously for all

Symbol	Meaning
S	Set
n	Set size
m	Number of sets
t	Intersection size
k	Summary size
l	l-wise independence
u	Universe size
c	Constant
N	Input size
U	Dimension
M	Internal memory size
B	Block size

Table 1.3: Recurring symbols used in the thesis and their meaning

values of l except 2, which is implied by the bound of Alon et al. Further, an interesting property of our hash function is that it is "almost" degree $l-1$ polynomial hashing that is used, hinting that perhaps the true generalization of Alon et al. can be achieved.

1.10 Technical preliminaries

1.10.1 Notation

We state notation that is used throughout the thesis.

Unless explicitly stated otherwise then for an integer n we use $[n]$ as shorthand for the set $\{0, 1, \dots, n-1\}$. For finite sets S we will use $|S|$ to denote the cardinality of S and S_i to denote the i'th element of S. For a finite set S we denote by $x \in S^d$ that x is length d vector with a member of S in each position.

We will let $\texttt{nnz}(\cdot)$ denote the number of nonzero entries of a matrix or vector.

For a constant c we let $\tilde{\mathcal{O}}(f(n))$ denote $\mathcal{O}(f(n)\log^c f(n))$ unless stated otherwise, e.g. in Chapter 3 we let $\tilde{\mathcal{O}}$ also hide polylog factors in the dimension size U.

We refer to Table 1.3 for a list of recurring symbols and what they denote. For symbols not in the list we will allow them to denote different things in each chapter, and in specific sections we will allow symbols from the table to be used not as stated, e.g. k as summand variable, when it is clear from the context that the symbol doesn't represent the meaning stated in the table.

1.10.2 Probability and randomized analysis

We state the probability theoretical tools and definitions that are widely used in the technical chapters of the thesis. See e.g. [16, 17] for more details on the probability theoretical tools used to analyze randomized algorithms. Readers familiar with analysis of randomized algorithms and data structures can skip this section.

Definition 1.1. A random variable $X : E \mapsto V$ is a random function from space E to outcome space V. If the range of V is finite then X is called a *discrete* random variable

All random variables in this thesis should be considered discrete unless explicitly stated otherwise.

Definition 1.2. For a random variable $X : E \mapsto V$ we say that its *expectation* is the value given by

$$\mathbb{E}(X) = \sum_{x \in V} x\mathbf{Pr}(X = x).$$

Definition 1.3. Let $X_1, \ldots, X_n$ be random variables with outcome ranges $A_1, \ldots, A_n$. The random variables are *independent* if for every $a_i \in A_i$

$$\mathbf{Pr}(X_1 = a_1) \wedge \mathbf{Pr}(X_2 = a_2) \wedge \ldots \wedge \mathbf{Pr}(X_n = a_n) = \prod_j \mathbf{Pr}(X_j = a_j).$$

When the above definition holds, the variables are also said to be *fully* independent. Such full independence is often not needed and thus the below definition is needed.

Definition 1.4. Let $X_1, \ldots, X_n$ be random variables with outcome ranges $A_1, \ldots, A_n$. The random variables are *l-wise independent* if for any distinct $1 \leq i_1 \leq i_2 \leq \ldots \leq i_l$ and $a_{i_1} \in A_{i_1}, \ldots, a_{i_l} \in A_{i_l}$ it holds that

$$\mathbf{Pr}(X_{i_1} = a_{i_1}) \wedge \mathbf{Pr}(X_{i_2} = a_{i_2}) \wedge \ldots \wedge \mathbf{Pr}(X_{i_l} = a_{i_l}) = \prod_{j=1}^{l} \mathbf{Pr}(X_{i_j} = a_{i_j}).$$

Note that for $l = n$ we have full independence.

The algorithms and data structures in this thesis are mostly randomized and to analyze their behavior some fundamental tools are used throughout, namely Markov's inequality, Chernoff's inequality and Chebychev's inequality. When steps of an analysis are attributed as "by standard application of Markov/Chernoff/Chebychev", then the three results below or slight variations of them are used.

Fact 1.5. (Markov's inequality) For a nonnegative random variable X and a scalar $\lambda > 1$ then it holds

$$\mathbf{Pr}[X \geq \lambda \mathbb{E}[X]] \leq \frac{1}{\lambda}.$$

Fact 1.6. (Chernoff's inequality) For a random variable X with $\mathbb{E}[X] = \mu$ and scalar $\varepsilon > 0$ then it holds

$$\mathbf{Pr}\left[|X - \mu| > \varepsilon\mu\right] \leq 2e^{-\varepsilon^2\mu/2}.$$

Fact 1.7. (Chebychev's inequality) For a random variable X with variance σ^2 and a scalar $\lambda > 0$ then it holds

$$\mathbf{Pr}[|X - \mathbb{E}[X]| \geq \lambda\sigma] \leq \frac{1}{\lambda^2}.$$

A common property of some randomized algorithms is the notion of an *estimate*, that is, the algorithm will as a result output not the true answer but instead a quantity close to the true answer. Sometimes such an estimate is close to the true answer with a probability - the setting is captured by the definition below.

Definition 1.8. Let $X \in \mathbb{R}$ and let $\hat{X}$ be a random variable. We say that $\hat{X}$ is an (ε, δ)*-estimate* of X if $\mathbf{Pr}\left[\left|\hat{X} - X\right| \geq \varepsilon X\right] \leq \delta$. We use ε-estimate as shorthand for $(\varepsilon, 1/3)$-estimate.

Throughout the thesis the fundamental notion of *hash functions* will be used repeatedly. A hash function $h : \mathcal{U} \mapsto [b]$ is a random function that maps a universe $\mathcal{U}$ to a (typically smaller) set $\{0, \ldots, b-1\}$. Such hash functions will broadly speaking be used to divide data into smaller parts that can then be processed. The optimal hash function would divide U independently and uniformly over $[b]$ but there are no practical implementations of such functions [18]. To achieve practicality we re-introduce the notion of l-wise independence onto hash functions.

Definition 1.9. Let $h : \mathcal{U} \mapsto V$ be a random hash function, $l \in \mathbb{N}$ and let $u_1, \ldots, u_l$ be any distinct l elements from $\mathcal{U}$ and $v_1, \ldots, v_l$ be any l elements from V.
Then h is l-wise independent if it holds that

$$\mathbf{Pr}\left[h(u_1) = v_1 \wedge \ldots h(u_l) = v_l\right] = \frac{1}{|V|^l}.$$

The above definition is interesting since often times full ($l = |U|$) is not needed and there are practical implementations for lower l, see e.g. [19, 20] for very fast families of hash functions for $l = 2$ and $l = 3$ respectively and [21, 22] for families that are practical for $l = \mathcal{O}(\log |U|)$ for universe U.

1.10.3 Algebra

The thesis contains algorithms where it is crucial to distinguish between the input allowed. In particular, the matrix multiplication results of Chapter 3 rely on the input being from a ring, semiring or a field.

Definition 1.10. A semiring R is a set S and two binary operators $+$ and $*$ where for all $a, b, c \in S$ it holds:

(a) $(a+b)+c = a+(b+c)$.

(b) $a+b = b+a$.

(c) $(a*b)*c = a*(b*c)$.

(d) $a*(b+c) = a*b + a*c$ and $(b+c)*a = b*a + b*c$.

Definition 1.11. A ring R is a semiring for which it additionally holds:

(a) There exists an additive identity $0 \in S$ s.t. for all $a \in S$, $0+a = a+b = a$.

(b) There exists an additive inverse for every $a \in S$ denoted $-a \in S$ s.t. $a+(-a) = (-a)+a = 0$.

Further the following two conditions *may* hold optionally:

1. There exists an multiplicative identity $1 \in S$ s.t. for all $a \neq 0 \in S$, $1*a = a*1 = a$.
2. There exists an multiplicative inverse for every $a \neq 0 \in S$ denoted $a^{-1} \in S$ s.t. $a*a^{-1} = a^{-1}*a = 1$.

Definition 1.12. A field $\mathbb{F}$ is a set of elements that satisfy associativity, commutativity, distributivity, identity and the existence of inverses for both addition and multiplication.

Chapter 2

Multiple Set Intersection

2.1 Introduction

Many basic information processing problems can be expressed in terms of intersection sizes within a preprocessed collection of sets. For example, in databases and data analytics, aggregation queries often use a conjunction of several simple conditions such as *"How many sales occurred in June 2013, in Sweden, where the sold object is a car?"* In this chapter we consider the problem of quickly *estimating* the size of the intersection of several sets, where a succinct precomputed summary of s bits is available for each set. Specifically, we answer the question:

How many bits is it necessary to allocate to each summary in order to get an estimate with $1 \pm \varepsilon$ *relative error?*

Note that we require the summaries to be *independently* computed, which for example prevents solutions based on precomputing all answers. This restriction is motivated by yielding scalable and flexible methods for estimating intersection sizes, with no need for a centralized data structure.

2.1.1 Motivation

Estimates of intersection size can be used directly as part of algorithms with approximation guarantees, but are also useful for exact computation. For example, when evaluating conjunctive database queries the order in which intersections are computed can have a large impact on performance. Good estimates of intersection sizes allow a query optimizer to make a choice that is near-optimal. In other settings, estimates of intersection sizes can be used as a filter to skip parts of an exact computation that would not influence the output (e.g., we might only be interested in a particular sales figure if it exceeds some threshold).

In data warehouses it is common to perform extensive precomputation of answer sets and summaries of simple queries, so that these can be combined to answer more complex queries quickly (see e.g. [23, 24]). At PODS 2011 Wei and Yi [25] showed that a number of different summaries of sets fulfilling a range condition can be efficiently extracted from augmented B-tree indexes. The number of I/Os for creating a summary of all data in a given range is close to the number of I/Os needed for reading a precomputed summary of the same size. That is, the efficiency is determined by the size of each summary, which motivates the question of how small a summary can be. Though Wei and Yi do not consider this explicitly, it is easy to see that (at least when efficient updates of data is not needed) their ideas apply to the kind of summaries, based on min-wise hashing, that we consider in the upper bounds of this chapter.

2.1.2 Brief history

Motivated by document similarity problems encountered in AltaVista, Broder [26] pioneered algorithms for estimating set intersection sizes based on independently precomputed "summaries" of sets. More specifically he presented a summary technique called "min-wise hashing" where, given summaries $k_{\min}(S_1)$ and $k_{\min}(S_2)$ of sets S_1 and S_2, it is possible to compute a low-variance, (asymptotically) unbiased estimator of the *Jaccard similarity* $J(S_1, S_2) = |S_1 \cap S_2|/|S_1 \cup S_2|$. Assuming that $|S_1|$ and $|S_2|$ are known, an estimate of $J(S_1, S_2)$ with small relative error can be used to compute a good estimate of $|S_1 \cap S_2|$, and vice versa. In fact, we state many of our results in terms of the ratio between the size of the intersection and the largest set, which is $\Theta(J)$.

Li and König [27] presented "b-bit min-wise hashing", a refinement of Broder's approach that reduces the summary representation size by storing a vector of b-bit hash values of elements from $k_{\min}(X)$. Even though the resulting hash collisions introduce noise in the estimator, this can be compensated for by a small increase in the size of $k_{\min}(X)$, yielding a significantly smaller summary with the same variance. Specifically, with $b = 1$ and using s bits of space, the variance is $2(1 - J)/s$.[1] In order to get an estimation error of at most εJ with probability (say) 1/2, by Chebychev's inequality it suffices that $(1 - J)/s < (\varepsilon J)^2$, i.e., $s > (1 - J)/(\varepsilon J)^2$. It is not hard to show that the estimator is

[1]The variance bound stated in [27] is more complex, since it deals with min-wise hashing based on permutations, which introduces correlations. By replacing this with full independence one arrives at the stated variance.

well-concentrated, and this bound is tight up to constant factors. Increasing the value of b (while keeping the space usage fixed) does not improve the variance.

2.1.3 Our contribution

First, we show that the variance of *any* estimator for Jaccard similarity based on summaries of s bits must be $\Omega(1/s)$ for fixed J between 0 and 1. More specifically, there exists a distribution of input sets such that with constant probability any such estimator makes an error of $\Omega(\sqrt{1/s})$ with constant probability. This means that b-bit min-wise hashing cannot be substantially improved when it comes to estimating intersection size (or Jaccard similarity) of two sets, except perhaps when J is asymptotically close to 0 or 1.

Second, we show that it *is* possible to improve existing estimators for the intersection size of $m = \omega(1)$ pre-processed sets. In fact, we show that estimators (such as b-bit min-wise hashing) that are based on many permutations are inherently less precise than their one-permutation counterpart when considering the intersection of many sets. We then show that a suitable approximate encoding of one-permutation min-wise hashing summaries is always competitive with b-bit min-wise hashing, while reducing the space required for accurately estimating the intersection size of many sets.

2.2 Related work

Problem definition. Let $S_1, S_2, \ldots$ be sets of size $n_i = |S_i|$ where all $S_i \subseteq [u]$ and the largest set is $n_{\max} = \max n_i$. A query is a subset $I \subseteq \mathbb{N}$ of the set indices and the output to the query is the intersection size $|\cap_{i\in I} S_i|$. For ease of notation we assume that the query is $I = \{1, \ldots, m\}$ and intersection size to estimate is then $t = |S_1 \cap \ldots \cap S_m|$.

In this chapter we consider estimators for the intersection size t. As previously noted we focus on the setting where the sets $S_1, S_2, \ldots, S_m$ are available for *individual* pre-processing. Storing only a small summary of each set, which requires not even approximate knowledge of t, we provide an estimator for the intersection size t. Note that in this model, we allow ourselves only to pre-process the sets independently of each other, i.e., intersection sizes or other information that rely on more than the set currently being pre-processed cannot be stored. See [28] for work on (exact) set intersection in the model where information about all sets can be used in the pre-processing phase.

For the applications, we seek to obtain bounds that are parameterized on the size of the summary required of each set as a function of largest set size $n_{\max}$, the intersection size t, and the relative error ε. Further, let s denote the space in bits stored per set and k the number of permutations or number values taken from one permutation for k-permutation and one-permutation min-wise hashing respectively.

2.2.1 Lower bounds

Several well-known problems in communication complexity imply lower bounds for special cases of the set intersection problem:

In the Index problem Alice is given a subset of $\{1, \ldots, n\}$, and Bob is given a set of size 1. The task is to determine whether the intersection size is 0 or 1. It is known that even for randomized protocols with error probability 1/3, the one-way communication complexity of this problem is $\Omega(n)$ bits (see [29]). Informally, this shows that the cost of estimating set intersection grows with the ratio between the intersection size t and the size $n_{\max}$ of the largest set.

In the GapAnd problem Alice and Bob are both given subsets of $\{1, \ldots, n\}$, and the task is to determine if the intersection size is below $n/4 - \sqrt{n}$ or above $n/4 + \sqrt{n}$ (if it is in-between, any result is okay). This is a variant of the well-studied GapHamming problem, for which the randomized one-way communication complexity is $\Omega(n)$ bits [30, 31]. In fact, the randomized two-way communication complexity for this problem is also $\Omega(n)$ bits [32], though in our application of first preprocessing the sets in order to then answer queries, we will only need the result for one-way communication. Informally, this lower bound means that the cost of estimating set intersection is inversely proportional with the square of the relative error.

Informally, our lower bound shows that these results generalize and compose, such that the lower bound is the *product* of the cost due to Index and the cost due to GapAnd, each with constant error probability. That is, our lower bound will be $\Omega(n_{\max}\varepsilon^{-2}/t)$, which we can use to bound the variance of any estimator for Jaccard similarity. The intuitive idea behind the lower bound is to compose the two problems such that each "bit" of GapAnd is encoded as the result of an Index problem. Unlike typical arguments in information complexity, see, e.g., the PODS 2010 tutorial by Jayram [33], we instead measure the information a protocol reveals about *intermediate bits* in Claim A.4, rather than about the inputs themselves. See the beginning of Section 2.3 for a more detailed intuition.

We note that using the output bits of multiple instances of one problem as the input bits to another problem was also used in [34], though not for our choice of problems, which are specific to and arguably very useful for one-way communication given the widespread usage of Index and GapAnd problems in proving encoding size or "sketching" lower bounds. We note that our problems may become trivial for 2-way communication, if e.g., one set has size $n_{\max}$ while the other set has size 1, while the lower bounds for the problems considered in [34] are qualitatively different, remaining hard even for 2-way communication.

2.2.2 Min-wise hashing techniques

Min-wise hashing was first considered by Broder [26] as a technique for estimating the similarity of web pages. For completeness, below we define min-wise independence along with the standard algorithm to compute an unbiased estimator for resemblance.

Definition 2.1 ([35, Eq. 4])**.** Let S_n be the set of all permutations on $[n]$. Then a family $\mathcal{F} \subseteq S_n$ is *min-wise independent* if for any set $X \subseteq [n]$ and any $x \in X$, when permutation $\pi \in \mathcal{F}$ is chosen at random we have

$$\mathbf{Pr}\left[\min \pi(X) = x\right] = 1/|X| \ .$$

In particular, for two sets $X, Y \subseteq [n]$ and a randomly chosen permutation $\pi \in \mathcal{F}$ we have

$$\mathbf{Pr}\left(\min \pi(X) = \min \pi(Y)\right) = J = \frac{|X \cap Y|}{|X \cup Y|}.$$

This can be used to compute an estimate of the Jaccard similarity. Specifically, given k independent min-wise permutations $\pi_1, \dots, \pi_k$ then

$$\hat{J} = \frac{1}{k}\sum_{i=1}^{k}\left[\min \pi_i(X) = \min \pi_i(Y)\right]$$

is an unbiased estimator of J (where $[\alpha]$ is Iverson Notation for the event α) with variance $\mathrm{Var}(\hat{J}) = J\left(1 - J\right)/k$.

In both theory and practice it is often easier to use a hash function with a large range (e.g. size u^3) instead of a random permutation. The idea is that the probability of a collision among the elements of a given set should be negligible, meaning that with high probability the order of the hash values induces a random permutation on the set. We will thus use the (slightly misleading) term "one-permutation" to describe methods using a single hash value on each set element.

Min-wise summaries. For a given set X the *k-permutation* min-wise summary of size k is the vector

$$(\min \pi_1(X), \dots, \min \pi_k(X)).$$

The *one-permutation* min-wise summary (sometimes called bottom-k sketch) of size k for a permutation π is the set $k_{\min}(X) = \{\pi(x) \,|\, x \in X, \pi(x) < \tau\}$, where τ is the $k+1$'th largest permutation rank (hash value) of the elements in X. That is, intuitively k-permutation summaries store the single smallest value independently for each of k permutations, while one-permutation summaries store the k smallest values for one permutation. It is not hard to show that $|k_{\min}(X \cup Y) \cap k_{\min}(X) \cap k_{\min}(Y)|/k$ is a good estimator for J, where $k_{\min}(X \cup Y)$ can be computed from $k_{\min}(X)$ and $k_{\min}(Y)$. For k-permutation min-wise summaries, if $\pi_1, \ldots, \pi_k$ are independent min-wise permutations then

$$\frac{1}{k}\sum_{1}^{k} |\min \pi_i(X) \cap \min \pi_i(Y)|$$

is analogously an estimator for J.

2.2.3 Previous results on set intersection

For m sets $S_1, \ldots S_m$ let the *generalized* Jaccard similarity be $J = |\cap_i S_i|/|\cup_i S_i|$. If we multiply an estimate of the generalized Jaccard similarity of several sets and an estimate of the size of the union of the sets, we obtain an estimate of the intersection size. Using existing summaries for distinct element estimation (also based on hashing, e.g. [36, 37]) we get that previous work on (generalized) Jaccard similarity implies results on intersection estimation [26, 35, 38, 39]. Recently, b-bit variations of min-wise hashing were proposed [40] but so far it is not clear how they can be used to estimate Jaccard similarity of more than three sets [41]. See Section 2.5.2 for further discussion.

The problem of computing aggregate functions (such as set intersection) over sets when hash functions are used to sample set elements has been widely studied [42–44]. In the general case of arbitrary aggregate functions, Cohen and Kaplan [44] characterizes for a given aggregate function f if an unbiased estimator for f with finite variance can be achieved using one- or k-permutation summaries. For the specific case of set intersection, *RC* (Rank Conditioning) estimators [42, 43] have been shown to provide an unbiased estimator based on both one- and k-permutation summaries and these can be extended to work with limited precision, analogous to b-bit min-wise hashing. Further, experimental work show that estimators based on one-permutation summaries outperform those based on k-permutation summaries [42] on the data sets used.

In contrast, this chapter provides an explicit worst-case analysis of the space requirement needed to achieve ε error with error probability at most δ for set intersection using one-permutation summaries, where signatures (2.5.2) are used to shave off a logarithmic factor for the upper bound, making the bound close to being tight.

Method	Required space (bits)	Time
Inclusion-exclusion	$s \geq \varepsilon^{-2}(mn/t)^2 + \log n$	2^m
Subsampling	$s \geq \varepsilon^{-2}(n/t)\log m \log^2 n$	sm
b-bit min-wise hashing*	$s \geq \varepsilon^{-2}(mn/t)$	sm
New upper bound	$s \leq \varepsilon^{-2}(n/t)\log(m)\log(n/\varepsilon t)$	sm
General lower bound	$s \geq \varepsilon^{-2}(n/t)$	-

Table 2.1: Comparison of estimators of intersection size t for relative error ε and constant error probability, with m sets of maximum size n. Bounds on the summary size s ignore constant factors. The subsampling bound assumes that no knowledge of t is available, and thus $\log n$ levels of subsampling are needed. *The bound for b-bit min-wise hashing assumes that the number of hash functions needed in the analysis of min-wise summaries is optimal, see Section 2.7.

Table 2.1 shows the performance of different algorithms along with our estimator based on one-permutation min-wise hashing. The methods are compared by time/space used to achieve an (ε, δ)-estimate (Definition 1.8) of the intersection size t of m sets of maximum size n for constant δ.

The Jaccard estimator computed using k-permutation min-wise hashing, as described in Section 2.2.2, can trivially be used to estimate intersection when cardinality estimate of the union of the sets is given (by simply multiplying by the union estimate). However, there are instances of sets where J can be as low as $t/(t+m(n-t))$ for a "sunflower", i.e., m sets of n elements that are disjoint except for the t intersection elements. Following from Chernoff bounds, such an instance requires to store $\frac{1}{J\varepsilon^2}$ elements to get an ε-estimation of J with constant probability. See Section 2.7 for a discussion of the bound for "b-bit min-wise hashing" in Table 2.1.

In contrast, the one-permutation approach described in this chapter stores $s \geq \frac{n}{t}\frac{\log m \log u}{\varepsilon^2}$ bits for m sets of maximum size n, while maintaining estimation time sm. Recent work investigated a *different* way of doing min-wise hashing using just one permutation [45], but this method seems to have the same problem as k-permutation min-wise hashing for the purpose of m-way set intersection. Intersection estimation can also be done by applying inclusion-exclusion to union size estimates of all subset unions of the m sets. To achieve error εt then by Chernoff bounds for sampling without replacement we need sample size $s > (\sum_i n_i/(\varepsilon t))^2$. As there are $2^m - 1$ estimates to do for m sets this yields time $2^m s$. Bloom filters [46] also support set intersection operations, and cardinality estimation, but to work well need the assumption that the sets have similar size. Therefore we will not discuss them further.

2.3 Our results

We show a lower bound for the size of a summary for two-way set intersection by a reduction from one-way communication complexity. More specifically, any summary that allows a $(1+\varepsilon)$-approximation of the intersection size implies a one-way communication protocol for a problem we call $\mathsf{GapAndIndex}$, which we think of as the composition of the Index and GapAnd communication problems. Namely, Alice has $r = \Theta(1/\varepsilon^2)$ d-bit strings $x^1, \ldots, x^r$, while Bob has r indices $i_1, \ldots, i_r \in [d]$, together with bits $b_1, \ldots, b_r$. Bob's goal is to decide if the input falls into one of the following cases, for a constant $C > 0$:

a For at least $\frac{r}{4} + C\varepsilon r$ of the $j \in [r]$, we have $x^j_{i_j} \wedge b_j = 1$.

b For at most $\frac{r}{4} - C\varepsilon r$ of the $j \in [r]$, we have $x^j_{i_j} \wedge b_j = 1$.

If neither case occurs, Bob's output may be arbitrary (i.e., this is a *promise* problem).

A straightforward reduction shows that if you have an algorithm that can $(1+\varepsilon)$-approximate the set intersection size $|S_1 \cap S_2|$ for sets S_1 and S_2, then you can solve $\mathsf{GapAndIndex}$ with the parameter d roughly equal to $|S_1|/t$ and $|S_2| \leq 4t$. Let the randomized communication complexity $R^{1-way}_{1/3}(\mathsf{f})$ of problem f be the minimal communication cost (maximum message transcript) of any protocol computing f with error probability at most 1/3.

The crux of our lower bound argument is to show:

Theorem 2.2. *For $r = \Theta(1/\varepsilon^2)$, $d = n_{max}/t$,*

$$R^{1-way}_{1/3}(\mathsf{GapAndIndex}) = \Omega(dr).$$

In terms of the parameters of the original set intersection problem, the space lower bound is proportional to the ratio d between the largest set size and the intersection size multiplied by ε^{-2}. Since $d = \Theta(1/J)$ this is $\Omega(\varepsilon^{-2}/J)$, which is a lower bound on the space needed for a $1 \pm \varepsilon$ approximation of J. If we consider the problem of estimating J with *additive* error $\leq \varepsilon_{\text{add}}$ with probability 2/3, observe that in this case $\varepsilon = \Theta(\varepsilon_{\text{add}}/J)$, so the lower bound becomes $\Omega(J/\varepsilon^2_{\text{add}})$. Conversely, for fixed $J > 0$ and space usage s we get $\varepsilon_{\text{add}} = \Omega(1/\sqrt{s})$ with probability 1/3 so the variance is $\Omega(1/s)$.

Our second result is a simple estimator for set intersection of an arbitrary number of sets, based on one-permutation min-wise hashing. The intuition behind our result is that when using k-permutation min-wise hashing, the probability of sampling intersection elements relies on the size of the union, while in contrast our one-permutation approach depends on the maximum set size, hence we save almost a factor of the number of input sets in terms of space. We show the following:

Theorem 2.3. *Let sets $S_1, \ldots, S_m \subseteq [u]$ be given and let $n_{max} = \max_i |S_i|$, $t = |S_1 \cap \ldots \cap S_m|$ and k be the summary size $|k_{\min}(S_i)|$. For $0 < \varepsilon < 1/4$, $0 < \delta < 1/\sqrt{k}$, consider the estimator*

$$X = \frac{\left|\bigcap_{i\in[m]} k_{\min}(S_i)\right| n_{\max}}{k} .$$

With probability at least $1 - \delta\sqrt{k}$:

$$t \in \begin{cases} [X/(1+\varepsilon); X/(1-\varepsilon)] & \text{if } X > 3n_{\max}\log(2m/\delta)/k\varepsilon^2 \\ [0; 4n_{\max}\log(2m/\delta)/k\varepsilon^2] & \text{otherwise} \end{cases}$$

That is, we either get an (ε, δ)-estimate or an upper bound on t. Whenever $k \geq \frac{4n_{\max}\log(2m/\delta)}{\varepsilon^2 t}$ we are in the first case with high probability. We note that the lower and upper bounds presented are parameterized on the estimand t, i.e., the bounds depend on the size of what we are estimating. This means that the error bound ε will depend of t, so the relative error is smaller for larger t.

Theorem 2.3 follows from two main arguments: First we show that if the summary of each set is constructed by selecting elements independently using a hash function then we get a good estimate with high probability. As our summaries are of fixed size, there is a dependence between the variables denoting whether an element is picked for a summary or not. The main technical hurdle is then to bound the error introduced by the dependence.

We then extend the use of signatures, which are well-known to reduce space for k-permutation min-wise hashing, to one-permutation min-wise hashing as used in our estimator. This reduces the number of bits s by a logarithmic factor. Section 2.5.2 discusses this further.

Result structure. We will start by showing the general space lower bound in Section 2.4. Following this we show the upper bound - a new estimator for set intersection size - in Section 2.5. We end by discussing the amount of independence the hash functions used in our estimator uses in Section 2.6 and in Section 2.7 we provide the bad instance for our (in spirit) nearest related estimator, the k-permutation min-wise estimator.

2.4 Lower Bound

2.4.1 Preliminaries

We summarize terms and definitions from communication complexity that are used in the lower bound proof.

Communication model. We consider two-player one-way communication protocols: Alice is given input x, Bob is given input y and they need to compute function $f(x, y)$. Each player has his/her own private randomness, as well as a shared uniformly distributed public coin $\mathbf{W}$ of some finite length. Since the protocol is 1-way, the *transcript* of protocol Π consists of Alice's single message to Bob, together with Bob's output bits. For a protocol Π, the maximum transcript length in bits over all inputs is called the *communication cost* of Π. The *communication complexity* $R_\delta(f)$ of function f is the minimal communication cost of a protocol that computes f with probability at least $1 - \delta$.

High level proof strategy. We briefly summarize the general approach used to prove the space lower bound of Theorem 2.2 and how communication complexity applies to space lower bounds. In general, say Alice gets input string x and Bob gets input string y and the communication problem that they need to compute is the function $f(x, y)$. Now assume the set intersection computation corresponds to computing $f(x, y)$ via a reduction argument and let $f(x, y)$ have communication complexity $\Omega(g(n))$ bits. Then the set intersection computation must have space complexity $\Omega(g(n))$ bits. Otherwise, assume there is an algorithm $\mathcal{A}$ that computes intersection using $o(g(n))$ bits. Then in the communication problem, simply run $\mathcal{A}$ on Alice's input, send the entire working memory to Bob who then finishes the computation. This protocol is a $o(g(n))$ bit protocol for $f(x, y)$, contradicting that $\Omega(g(n))$ was a lower bound for $f(x, y)$.

Mutual information. For random variables X and Y with support $\mathcal{X}$ and $\mathcal{Y}$ and let $p(x, y), p(x), p(y)$ be the joint and marginal distributions respectively. The *entropy* and *conditional entropy* are defined as:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x)$$
$$H(X \mid Y) = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x, y) \log \frac{p(y)}{p(x, y)}$$

The *mutual information* is given as:

$$I(X; Y) = H(X) - H(X \mid Y) = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

We make use of the following rule:

Fact 2.4. (Chain Rule) For discrete random variables
X, Y, Z it holds that $I(X, Y; Z) = I(X; Z) + I(X; Y \mid Z)$.

For a protocol Π that uses public random coins $\mathbf{W}$ and has transcript $\Pi(X, Y, Z)$ for random variables $X, Y, Z \sim \mu$, the *conditional information cost* of Π with respect to distribution μ is $I(X, Y, Z; \Pi(X, Y, Z) \mid \mathbf{W})$. For function f we have that the *conditional information complexity* $CIC_\delta^\mu(f)$ is the minimal conditional information cost of any δ-error protocol Π with respect to distribution μ.

Fano's inequality. We make use of Fano's equality which intuitively relates the error probability of a function between random variables to the conditional entropy between them.

Definition 2.5. Given domains $\mathcal{X}$ and $\mathcal{Y}$ and random variables X, Y on these domains with distribution μ, we say a function $g : \mathcal{Y} \to \mathcal{X}$ has error δ_g if

$$\mathbf{Pr}_{X,Y\sim\mu}[g(Y) = X] \geq 1 - \delta_g.$$

Fact 2.6. Let X and Y be a random variables chosen from domains $\mathcal{X}$ and $\mathcal{Y}$ respectively according to distribution μ. There is a deterministic function $g : \mathcal{Y} \to \mathcal{X}$ with error δ_g, where $\delta_g \leq 1 - \frac{1}{2^{H(X \mid Y)}}$.

Fact 2.7. (Fano's inequality.) Let X and Y be a random variables chosen from domains $\mathcal{X}$ and $\mathcal{Y}$ respectively according to distribution μ. For any reconstruction function $g : \mathcal{Y} \to \mathcal{X}$ with error δ_g,

$$H_b(\delta_g) + \delta_g \log(|\mathcal{X}| - 1) \geq H(X \mid Y).$$

2.4.2 A Communication Problem and its Application to Set Intersection

Let $r = \Theta(1/\varepsilon^2)$, and $d = n_{max}/t$. We consider a two-party one-way communication problem:

Definition 2.8. In the GapAndIndex problem, Alice has bit vectors $x^1, \ldots, x^r \in \{0,1\}^d$ while Bob has indices $i^1, \ldots, i^r \in [d]$, where $[d] = \{1, 2, \ldots, d\}$, together with bits $b^1, \ldots, b^r \in \{0,1\}$. Let $\mathbf{x} = (x^1, \ldots, x^r)$, $\mathbf{i} = (i^1, \ldots, i^r)$, and $\mathbf{b} = (b^1, \ldots, b^r)$ and

$C > 0$ be a fixed constant. The output of $\mathsf{GapAndIndex}(\mathbf{x}, \mathbf{i}, \mathbf{b})$ is:

$$1 \text{ if } \sum_{j=1}^{r} (x^j_{i^j} \wedge b^j) \geq \frac{r}{4} + C\varepsilon r$$
$$0 \text{ if } \sum_{j=1}^{r} (x^j_{i^j} \wedge b^j) \leq \frac{r}{4} - C\varepsilon r.$$

This is a promise problem, and if neither case occurs, the output can be arbitrary.

If the input $(\mathbf{x}, \mathbf{i}, \mathbf{b})$ is in either of the two cases we say the input *satisfies the promise.*

We say a one-way randomized protocol Π for $\mathsf{GapAndIndex}$ is δ-error if

$$\forall \mathbf{x}, \mathbf{i}, \mathbf{b} \text{ satisfying the promise :}$$
$$\mathbf{Pr}[\Pi(\mathbf{x}, \mathbf{i}, \mathbf{b}) = \mathsf{GapAndIndex}(\mathbf{x}, \mathbf{i}, \mathbf{b})] \geq 1 - \delta,$$

where the probability is over the public and private randomness of Π.

Let κ be the set of randomized one-way δ-error protocols Π. We note that κ is finite for any problem with finite input, as we can always have one player send his/her entire input to the other player.

Then,

$$R_\delta^{1-way}(\mathsf{GapAndIndex}) = \min_{\Pi \in \kappa} \max_{\substack{\mathbf{x},\mathbf{i},\mathbf{b}, \\ \text{randomness of } \Pi}} |\Pi(\mathbf{x}, \mathbf{i}, \mathbf{b})|,$$

where $|\Pi(\mathbf{x}, \mathbf{i}, \mathbf{b})|$ denotes the length of the transcript with these inputs. Since the protocol is 1-way, we can write this length as $|M(\mathbf{x})| + 1$, where $M(\mathbf{x})$ is Alice's message function in the protocol Π given her input $\mathbf{x}$, and we add 1 for Bob's output bit. Here, implicitly M also depends on the private randomness of Alice, as well as the public coin $\mathbf{W}$.

Let μ be the uniform distribution on $\mathbf{x} \in (\{0,1\}^d)^r$. We use the capital letter $\mathbf{X}$ to denote random $\mathbf{x}$ distributed according to μ. We introduce a distribution on inputs solely for measuring the following notion of information cost of the protocol; we still require that the protocol is correct on *every* input satisfying the promise with probability $1 - \delta$ over its public and private randomness (for a sufficiently small constant $\delta > 0$).

For a uniformly distributed public coin $\mathbf{W}$, let

$$CIC_\delta^{\mu, 1-way}(\mathsf{GapAndIndex}) = \min_{\Pi \in \kappa} I(M(\mathbf{X}); \mathbf{X} \mid \mathbf{W}),$$

where for random variables Y, Z and W, $I(Y; Z \mid W) = H(Y \mid W) - H(Y \mid Z, W)$ is the conditional mutual information. Recall that the conditional entropy $H(Y \mid W) =$

$\sum_w H(Y \mid W = w) \cdot \mathbf{Pr}[W = w]$, where w ranges over all values in the support of W. For any protocol Π,

$$\begin{aligned}\max_{\mathbf{x},\mathbf{i},\mathbf{b}} |\Pi(\mathbf{x}, \mathbf{i}, \mathbf{b})| &= \max_{\mathbf{x}} |M(\mathbf{x})| + 1 \\ &> \max_{\mathbf{x}} |M(\mathbf{x})| \\ &\geq H(M(\mathbf{X}) \mid \mathbf{W}) \\ &\geq I(M(\mathbf{X}); \mathbf{X} \mid \mathbf{W}),\end{aligned}$$

which implies that

$$R_\delta^{1-way}(\mathsf{GapAndIndex}) \geq CIC_\delta^{\mu,1-way}(\mathsf{GapAndIndex}).$$

We now consider the application to set intersection. Let $r = 10/\varepsilon^2$ and d be the desired ratio between the intersection size t and the largest set. The idea is to give Alice a subset of elements from $[dr]$, where the characteristic vector of her subset is $x^1, \ldots, x^r$. Also, for each $j \in [r]$, Bob is given the element $d \cdot (j-1) + i^j$ if and only if $b^j = 1$. If Alice and Bob's sets are constructed in this way then the intersection is either of size at most $r/4 - C\varepsilon r$, or of size at least $r/4 + C\varepsilon r$. Hence, a 1-way protocol for approximating the intersection size up to a relative $(1 + \Theta(\varepsilon))$-factor can be used to distinguish these two cases and therefore solve the GapAndIndex promise problem.

To get intersection size t without changing the problem, we duplicate each item $4t/r$ times, which means that the problem becomes distinguishing intersection size at most $t(1 - \Theta(\varepsilon))$ and at least $t(1 + \Theta(\varepsilon))$. By rescaling ε by a constant factor, a 1-way protocol for $(1+\varepsilon)$-approximating the intersection of Alice and Bob's sets with constant probability can be used to solve GapAndIndex with constant probability. Hence, its space complexity is $\geq CIC_{1/3}^{\mu,1-way}(\mathsf{GapAndIndex})$. This holds for any distribution μ for measuring information, though we shall use our choice of μ above.

Hence, the final step is showing that the stated conditional information complexity of GapAndIndex is $\Omega(dr)$ for $d = n_{\max}/t$ and $r = \Theta(1/\varepsilon^2)$. The proof details of the argument can be seen in Appendix A.

2.5 Upper bound

Recall that sets $S_1, \ldots, S_m \subseteq [u]$ of sizes $n_1, \ldots, n_m$ where $n_{\max} = \max_i n_i$ are given, and we wish to obtain an (ε, δ)-estimate of $t = |S_1 \cap \ldots \cap S_m|$ using one-permutation k-min summaries as described in Section 2.2.2. Theorem 2.3 defines an estimator (see Figure 2.2 for pseudocode). In our proof of Theorem 2.3 we will assume that the hash function used to construct the summaries is random and fully independent. In many applications it will be possible to achieve this by simply maintaining a hash table of values during the construction phase. However, Section 2.6.1 shows how to replace the full randomness assumption with concrete hash functions in case the number of different hash values is too large to store. The space needed to store a summary that gives an (ε, δ)-estimate is $\mathcal{O}\left(\frac{n_{\max} \log(m/\delta) \log u}{t\varepsilon^2}\right)$ bits. In Section 2.5.2 we show that this can be reduced almost by a factor $\log u$ by use of signatures. The time to compute the estimation of t of m sets is sm.

For an intersection query on m sets the main insight is that our estimator relies only on the maximum set size $n_{\max}$ in contrast to the known k-permutation estimator that depends on the size of the union, making it less accurate given the same space (see Table 2.1). The difference is illustrated in Figure 2.1: We have two sets, $S_{\max}$ and S_2, one is big and is of size $n_{\max}$, the other is smaller of size n and the intersection size is t. The intuitive argument can be made by considering *sampling*: If we sample an element from the intersection with probability p then we must store $1/p$ elements in order to in expectation have one intersection element stored. Roughly speaking, for k-permutation summaries there is probability around the intersection size divided by the union size per permutation to have one of the t intersection elements being the sampled one. It follows from this observation that an input that maximizes space usage k-permutation summaries is a set instance with large union size and small intersection size - see Section 2.7 for the argument on such an instance. For one-permutation summaries as used by our estimator the space saving comes from the fact that for sufficient summary size k, when an element from the intersection is sampled from $S_{\max}$ then it is also sampled from S_2 with high probability, meaning roughly sampling probability $t/n_{\max}$. This comes from the fact that if elements of the intersection hashes among the k smallest hash values in the big set, then this is likely the case for the smaller set as well, since we use the same hash function to sample.

2.5.1 Proof of Theorem 2.3

Recall that $k_{\min}(S_i)$ denotes the size-k one-permutation min-wise summary of S_i and the indicator variable $\hat{X}_j^{(i)}$ denotes the event that item j is chosen for the size-k one-permutation min-wise summary of S_i as defined below: $\hat{X}_j^{(i)} = 1$ if $j \in k_{\min}(S_i)$ and $\hat{X}_j^{(i)} = 0$ otherwise.

High-level proof strategy. Observe that $\mathbf{Pr}[\hat{X}_j^{(i)} = 1] = k/n_i$. Our algorithm uses size k summaries so for each set S_i we have $\sum_{j=1}^{n_i} \hat{X}_j^{(i)} = k$, which causes negative dependence between the indicator variables [47], i.e., when an item is in the summary

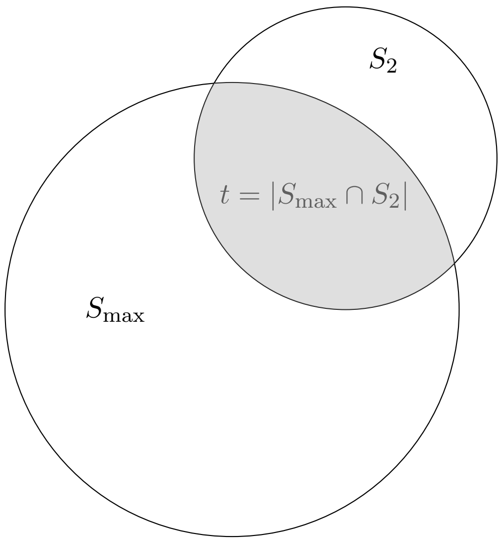


FIGURE 2.1: k-perm vs one-perm intuition. For k-perm we have probability around $t/|S_{\max} \cup S_2|$ to sample an intersection element. For one-perm and high enough k the big set dictates and so we get probability roughly $t/|S_{\max}|$. Higher sampling probability implies lower space usage.

```
Input: Sets $S_1, S_2, \ldots \subseteq [u]$
Output: $k$-min summaries for all $S_i$
1 h $\longleftarrow$ fully independent random hash function
2 foreach $S_i$ do
3     $k_i \leftarrow$ the $k$th smallest $h(x)$ for $x \in S_i$
4     $k_{\min}(S_i) \leftarrow \{x \mid x \in S_i \wedge h(x) \leq k_i\}$
```

(A) Pre-processing the sets.

```
Input: $k$-min summaries and set sizes $k_{\min}(S_1), n_1 = |S_1|, \ldots$ and query set $M \subseteq \mathbf{N}$
Output: $X$: An $(\varepsilon, \delta)$-estimation of $t = \left|\bigcap_{i \in M} S_i\right|$
1 $n_{\max} \longleftarrow \max_{i \in M} n_i$
2 $X \longleftarrow \left|\bigcap_{i \in M} k_{\min}(S_i)\right| n_{\max}/k$
```

(B) Computing the estimator. The output is an (ε, δ)-estimator whenever $X > 3n_{\max} \log(1/\delta)/k\varepsilon^2$ (See Theorem 2.3).

FIGURE 2.2: Pseudocode for performing pre-processing and computing the estimator.

of a set then the other items have smaller probability of being in the that summary. The main technical hurdle is showing that even with such a dependence one can use the intersection size between the summaries to estimate the intersection size of the sets.

To do this we analyze the case where for each S_i, the variables $X_1^{(i)}, X_2^{(i)}, \ldots, X_{n_i}^{(i)}$ are independent random variables:

$$X_j^{(i)} = \begin{cases} 1 & \text{if } h(j) \leq k/n_i \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $h : u \mapsto [0, 1]$ is a fully random hash function. Let the setting with negative dependence be called *the dependent case* and the case using (2.1) be *the independent case*. The independent case conditioned on the sum of the variables being k is identically distributed as the dependent case. Therefore the final step is to bound the additional error probability of going from the independent case to the dependent one.

First we bound the probability of sampling k specific items given the number of sampled items is k. Let $i \in [m]$ and $\tilde{S}_i = \{x \in S_i \,|h(x) \leq k/n_i\}$ be a sample of $S_i \subseteq [u]$ picked according to (2.1). An important consequence of picking elements to be in summaries is that of *consistent* sampling: If the hash value of an element from the intersection is one of the k smallest hash values computed, it will be guaranteed to be sampled in all sets. The following lemma shows that any specific outcome of a sample has equal probability given we restrict a sample to be size k.

Lemma 2.9. *If $S_i \subseteq [u]$ and $\{i_1, i_2, \ldots, i_k\}$ is a specific size k outcome then*

$$\mathbf{Pr}\big[\tilde{S}_i = \{i_1, i_2, \ldots, i_k\} \mid \sum_{j=1}^{n_i} X_j^{(i)} = k\big] = \frac{1}{\binom{n_i}{k}}$$

Proof. We have:

$$\mathbf{Pr}\left[\tilde{S}_i = \{i_1, i_2, \ldots, i_k\}\right] = \left(\frac{k}{n_i}\right)^k \left(1 - \frac{k}{n_i}\right)^{n_i - k}$$

$$\mathbf{Pr}\left[\sum_{j=1}^{n_i} X_j^{(i)} = k\right] = \binom{n_i}{k} \left(\frac{k}{n_i}\right)^k \left(1 - \frac{k}{n_i}\right)^{n_i - k}$$

The final step of the lemma follows from Bayes theorem:

$$\mathbf{Pr}\big[\tilde{S}_i = \{i_1, i_2, \ldots, i_k\} \mid \sum_{j=1}^{n_i} X_j^{(i)} = k\big] = \frac{\mathbf{Pr}\left[\tilde{S}_i = \{i_1, i_2, \ldots, i_k\}\right]}{\mathbf{Pr}\left[\sum_{j=1}^{n_i} X_j^{(i)} = k\right]} = \frac{1}{\binom{n_i}{k}} \ .$$

■

We show the lower bound on the probability of the size of any $\tilde{S}_i$ being equal to its expectation k:

Lemma 2.10. *For a sample $\tilde{S}_i$ of S_i we have*

$$\mathbf{Pr}\left[|\tilde{S}_i| = k\right] = \Omega\left(\frac{1}{\sqrt{k}}\right) \ . \tag{2.2}$$

Proof. The mean μ of $|\tilde{S}_i|$ is the most likely outcome, i.e,

$$\mathbf{Pr}\left[|\tilde{S}_i| = k\right] \geq Pr\left[|\tilde{S}_i| = j\right] \text{ for } 1 \leq j \leq u$$

holds due to $\mathbb{E}\left[\sum_{j=1}^{n_i} X_j^{(i)}\right] = k$ and the mode of binomial distributions [48]. Next step is showing that $|\tilde{S}_i|$ is within $2\sqrt{k}$ of the mean $\mu = k$ with probability at least 1/2, that is:

$$\mathbf{Pr}\left[|\sum_{j=1}^{n_i} X_j^{(i)} - k| \geq 2\sqrt{k}\right] \leq \frac{1}{2}.$$

This follows from the Chernoff bounds on the sum $\sum_{j=1}^{n_i} X_j^{(i)}$:

$$\begin{aligned}\mathbf{Pr}\left[|\sum_{j=1}^{n_i} X_j^{(i)} - k| \geq 2\sqrt{k}\right] &\leq 2\exp\left(\frac{-k\left(\frac{2\sqrt{k}}{k}\right)^2}{2}\right)\\ &\leq \frac{1}{2}\ \forall k, i > 0\ .\end{aligned}$$

■

Let S be the elements of the size-t intersection and $\tilde{S}_{\max}$ be the sample of the largest set $S_{\max}$. We show that if the summary size k satisfies

$$k \geq \frac{2n_{\max}\log\left({}^{2m}/_{\delta}\right)}{\varepsilon^2 t} \tag{2.3}$$

then properties 1 and 2 below are satisfied.

Property 1. $\left|\tilde{S}_{\max} \cap S\right|$ is an $(\varepsilon, {}^{\delta}/_{2})$-estimate of $t\frac{k}{n_{\max}}$.

Property 2. $\forall_i \left|\tilde{S}_i \cap S\right| \geq t(1-\varepsilon)\frac{k}{n_{\max}}$ with probability at least $1 - {}^{\delta}/_{2m}$.

We show that the given properties hold for sufficiently large k, given by (2.3).

Lemma 2.11. *If (2.3) holds and* $0 \leq \varepsilon, \delta \leq 1$ *then properties 1 and 2 hold.*

Proof. We show that property 1 holds when (2.3) holds. This follows from Chernoff bounds on $\sum_{j=1}^{t} X_j^{(max)}$:

$$\begin{aligned}\gamma_1 &= \mathbf{Pr}\left[|\tilde{S}_i \cap S| \notin \left[t\frac{k}{n_{\max}}(1-\varepsilon), t\frac{k}{n_{\max}}(1+\varepsilon)\right]\right]\\ &= \mathbf{Pr}\left[\sum_{j=1}^{t} X_j^{(max)} \leq t\frac{k}{n_{\max}}(1-\varepsilon)\right] + \mathbf{Pr}\left[\sum_{j=1}^{t} X_j^{(max)} \geq t\frac{k}{n_{\max}}(1+\varepsilon)\right]\\ &< 2\exp\left(-\frac{\varepsilon^2 tk}{3n_{\max}}\right)\ .\end{aligned}$$

Since $k \geq \frac{2n_{\max}\log(2m/\delta)}{\varepsilon^2 t}$ the error probability is $\gamma_1 \leq \frac{\delta}{2}$, thus property 1 holds.

Now we are to show that the given k implies property 2 holds, i.e., the size of the intersection between any single $\tilde{S}_i$ sample and intersection S is at least the expected size of the intersection between the sample of the largest set, $\tilde{S}_{\max}$ and S. The intersection of any sample $\tilde{S}_i$ and S has expectation $\mu = \mathbb{E}\left[|\tilde{S}_i \cap S|\right] = t\frac{k}{n_i}$. Since $n_{\max} \geq n_i$, it holds that $\forall_i t\frac{k}{n_{\max}} \leq t\frac{k}{n_i}$ and thus we bound error γ_2:

$$\begin{aligned}\gamma_2 &= \mathbf{Pr}\left[\sum_{j=1}^{t} X_j^{(max)} < (1-\varepsilon)t\frac{k}{n_{\max}}\right] \\ &\leq \mathbf{Pr}\left[\sum_{j=1}^{t} X_j^{(i)} < (1-\varepsilon)t\frac{k}{|S_i|}\right] \leq \exp\left(-\frac{\varepsilon^2 tk}{2n_{\max}}\right) \quad .\end{aligned}$$

Since $k \geq \frac{2n_{\max}\log(2m/\delta)}{\varepsilon^2 t}$ the error probability is $\gamma_2 \leq \frac{\delta}{2m}$, thus property 2 holds. ■

We will now show that the independent case provides an estimator with the desired guarantees.

Lemma 2.12. *If (2.3) holds and we have* $0 \leq \varepsilon, \delta \leq 1$ *then* $\frac{|\bigcap_{i\in[m]}\tilde{S}_i| n_{\max}}{k}$ *is an* (ε,δ)-*estimate of* t.

Proof. First we need that $\left|\tilde{S}_{\max} \cap S\right| \leq (1+\varepsilon)t\frac{k}{n_{\max}}$ with probability $\geq 1-\delta$. By Lemma 2.11 this holds, as property 1 holds since k satisfies (2.3). We now argue:

$$\left|\bigcap_{i\in[m]} \tilde{S}_i\right| \geq (1-\varepsilon)\,t\frac{k}{n_{\max}} \text{ with probability } \geq 1-\delta \ . \tag{2.4}$$

Let $z = (1-\varepsilon)\,t\frac{k}{n_{\max}}$. By Lemma 2.11 we have that for each set S_i its sample $\tilde{S}_i$ contains at least z items from the intersection set S with probability $\geq 1-\delta/2m$. Also from Lemma 2.11 we have that the biggest set has at least z items from S in its sample $\tilde{S}_{\max}$. By the definition of how we sample, if the sampled values from the largest set are present in a smaller set then they will be sampled as well, which holds here since the z are in the intersection S.

To show that

$$\left|\bigcap_{i\in[m]} \tilde{S}_i\right| \leq (1+\varepsilon)\,t\frac{k}{n_{\max}} \text{ with probability } \geq 1-\delta \tag{2.5}$$

holds we need that $\left|\tilde{S}_{\max} \cap S\right| \leq (1+\varepsilon)t\frac{k}{n_{\max}}$ with probability $\geq 1-\delta$. This follows directly from property 1 holding since k satisfies (2.3) as shown in Lemma 2.11.

We now show that our estimator computes an (ε,δ)-estimate, i.e., it holds that,

$$\frac{\left|\bigcap_{i\in[m]} \tilde{S}_i\right| n_{\max}}{k} \in [(1-\varepsilon)\,t, (1+\varepsilon)\,t]$$

with probability at least

$$1 - \left(\frac{\delta}{2} + m\frac{\delta}{2m}\right) \geq 1 - \delta \ .$$

By (2.5) and (2.4) we have the relative error of at most ε as required. To bound the error probability we apply the union bound on the error probabilities given by Lemma 2.11. As we have error probability $\delta/2$ on property 1 and error probability $\delta/2m$ on property 2, by the union bound we get $\leq (\delta/2 + m\delta/2m) = \delta$ where the factor m on the second term comes from the union bound over all m sets. ■

For each set S_i let B_i denote the set of samples where property 1 or 2 does *not* hold. We have probability $\mathbf{Pr}\left[\tilde{S}_i \in B_i\right]$ of the estimator based on samples $\tilde{S}_i$ being bad. We now relate the independent case where a sample has expected size k to the case where k-min summaries are used and thus we have samples of strictly size k.

Lemma 2.13. *If (2.3) holds and* $0 \leq \varepsilon, \delta \leq 1$ *then* $\mathbf{Pr}\left[\tilde{S}_i \in B_i \mid |\tilde{S}_i| = k\right] \leq \delta\sqrt{k}$.

For a specific itemset $I = \{i_1, i_2, \ldots, i_k\}$ *we have*

$$\mathbf{Pr}\left[\tilde{S}_i = I \mid \sum_{j=1}^{n_i} X_j^{(i)} = k\right] = \mathbf{Pr}\left[k_{\min}(S_i) = I\right] = \frac{1}{\binom{n_i}{k}} \tag{2.6}$$

Proof. An upper bound of the conditional probability can be obtained through Bayes theorem:

$$\mathbf{Pr}\left[\tilde{S}_i \in B \mid |\tilde{S}_i| = k\right] \leq \frac{\mathbf{Pr}\left[\tilde{S}_i \in B\right]}{\mathbf{Pr}\left[|\tilde{S}_i| = k\right]} \ .$$

The probability of the sample being of size k was bounded in (2.2) and by union bound on the error probabilities found in Lemma 2.11 we get

$$\mathbf{Pr}\left[\tilde{S}_i \in B \mid |\tilde{S}_i| = k\right] \leq \frac{\mathbf{Pr}\left[\tilde{S}_i \in B\right]}{\mathbf{Pr}\left[|\tilde{S}_i| = k\right]} \leq \delta/\tfrac{1}{\sqrt{k}} = \delta\sqrt{k} \ .$$

Now we argue that (2.6) holds, i.e., that the conditional distribution of any sample $|\tilde{S}_i| = k$ is the same as that of $k_{\min}(S_i)$. This follows directly from Lemma 2.9 and from

$$\mathbf{Pr}\left[k_{\min}(S_i) = I\right] = \frac{1}{\binom{n_i}{k}}.$$

■

Proof. (Theorem 2.3.) By Lemma 2.12 we have that X is an (ε, δ)-estimate of t in the independent case whenever the expected number of elements k in our summaries satisfy (2.3). Lemma 2.13 relates the independent case to the dependent case with fixed summary size, showing that X is an $(\varepsilon, \delta\sqrt{k})$-estimate when (2.3) holds. To show Theorem 2.3 we consider two cases for t.

1. If $t \geq 2n_{\max} \log(2m/\delta)/k\varepsilon^2$ then (2.3) is satisfied, so X is an $(\varepsilon, \delta\sqrt{k})$-estimate of t. Since $\varepsilon < 1/4$ we get that $X < 3n_{\max} \log(2m/\delta)/k\varepsilon^2$ implies

$$X/(1-\varepsilon) < 4n_{\max} \log(2m/\delta)/k\varepsilon^2 \ .$$

   So as long as $t \in [X/(1+\varepsilon); X/(1-\varepsilon)]$, which happens with probability $1 - \delta\sqrt{k}$, we get a true answer regardless of whether the first or second answer is returned.

2. If $t < 2n_{\max} \log(2m/\delta)/k\varepsilon^2$ then the probability that $X > 3n_{\max} \log(2m/\delta)/k\varepsilon^2)$ is at most $\delta\sqrt{k}$. This is because X is dominated by an estimator X' derived from X by artificially increasing the intersection size to that required by (2.3). This means that with probability $1 - \delta\sqrt{k}$ the algorithm correctly reports that t is in the interval $[0; 4n_{\max} \log(2m/\delta)/k\varepsilon^2]$.

$\blacksquare$

2.5.2 Use of signatures for the upper bound

An advantage of k-permutation min-wise hashing is that it can easily be combined with signatures to decrease space usage, i.e., elements from u in the min-hash can be replaced with hash values using significantly fewer bit. As shown by Li and König [40], using b-bit signatures, where b is a small integer, allows us to increase k by a factor $\log(u)/b$ without increasing the space usage. With a suitable estimator that takes the signature collisions into account, the net result is an increase in precision for a given space usage. It is a nontrivial matter to extend the estimator to work for the intersection of more than two sets when b is small. The case of three sets was investigated in [41].

It seems to be less well known that one-permutation hashing allows a similar space saving. The idea is to consider signatures of $\log(k)+b$ bits, and store the *set* of signatures for each set $k_{\min}(X)$. By using an appropriate encoding of the signature set the space usage becomes roughly $k(b + \log e)$ bits, see e.g. [49]. There even exist methods that use word-level parallelism to compute the set of signatures that are in common between two such encodings [50, Lemma 3], meaning that there is a speedup in comparing two summaries that is similar to the factor saved in space usage. At least in theory, this means that the difference between the efficiency of k-permutation and one-permutation schemes compressed using signatures is not so large.

We now argue that if we choose a signature hash function $h : [u] \to \{0,1\}^b$ where $b \geq \log(2k^2/\delta)$, a signature collision that affects the estimate will occur with probability at most $\delta/2$, independent of the number of sets considered. Recall that k is the size of a min-hash, and consider a specific set of min-hashes $k_{\min}(S_j)$, $j = 1, \ldots, m$. If we replace $k_{\min}(S_j)$ by the set $h(k_{\min}(S_j))$ of signatures there is a chance that $|\cap_j h(k_{\min}(S_j))|$ is different from $|\cap_j k_{\min}(S_j)|$ because of collision of elements in some set I with at least one element in each min-hash. We define an *i-cover* as a set I where $|I| = i$ and $\forall j : I \cap k_{\min}(S_j) \neq \emptyset$, i.e., an i-cover is a set of i elements that includes an element from every minhash. We now argue that there is a low probability that there exists an i-cover with $i > 1$ for which all elements have the same signature under h. For now we assume that h is fully random, which means that the probability a particular i-cover colliding is at most

$$(2^{-b})^{i-1} = \left(\frac{\delta}{2k^2}\right)^{i-1} .$$

For $i \leq m$ we have at most k^i possible i-covers, so by a union bound the probability of any colliding i-cover occurring is at most

$$\sum_{i=2}^{m} k^i \left(\frac{\delta}{2k^2}\right)^{i-1} \leq \delta/2 .$$

We conclude that with probability at least $1 - \delta/2$ we end up with exactly $|\bigcap_i k_{\min}(S_i)|$ signatures in the intersection, meaning that the result is the same as when storing the elements of $k_{\min}(S_1), \ldots, k_{\min}(S_m)$. Hence one can simply think of the sets $k_{\min}(S_i)$, with the understanding that they can be replaced by a representation of size roughly $k \log(e2k^2/\delta)$ bits using a suitable encoding of signatures.

2.6 Hash functions of limited independence

Until now we have assumed to have access to a fully random hash function on the sets. In this section we show that there are realizable hash functions of limited independence such that our results hold. Thorup [51] recently showed that for Jaccard similarity (and hence intersection size) estimation with one-permutation min-wise summaries it suffices to use a pairwise independent hash function. However, this does not extend to the setting where we seek the intersection size of many sets (see Theorem 2.14).

We argue that k-wise independence is sufficient for the hash function used to construct the one-permutation min-wise summaries and that m-wise independence is sufficient for the hash functions used to create signatures as described in Section 2.5.2.

2.6.1 Hash functions for one-permutation min-wise summaries

We will argue that k-wise independent hash functions are sufficient for the hash function used to create the summaries.

For n variables $X_1, \dots X_n$, $X = \sum_i^n X_i$, $\mu = \mathbb{E}[X]$ and $\delta > 0$ then by [52] we have that if the variables $X_1, \dots X_n$ are $\lceil \frac{\mu\delta}{1-\mu/n} \rceil$-wise independent, the Chernoff tail bounds hold. Examining the tail bounds used in Section 2.5.1 we see that if we impose the additional constraint $\delta \leq 1 - k/n$ then $\lceil \frac{\mu\delta}{1-\mu/n} \rceil \leq k$ and hence k-wise independence is sufficient for the construction of our summaries.

2.6.2 Hash functions for signatures

We will now argue that m-wise independent hash functions are sufficient to obtain error probability $\leq \delta$ when being used to create signatures. This follows directly from the fact that we consider collisions in terms of i-covers for $i \leq m$ and apply a summation of m terms to bound the error probability to be $\leq \delta/2$. For the family of hash functions we will use the construction of Siegel [22]. This construction gives a RAM data structure of space $\mathcal{O}\left(u^{\sqrt{\lg k/\lg u + \varepsilon}} \lg v\right)$ bits when hashing from $\{0, \dots, u-1\}$ to $\{0, \dots, v-1\}$. A function from the family can be evaluated in constant worst-case time and it is k-wise independent with high probability. In particular, for $m = k = u^{\mathcal{O}(1)}$ we have space usage $\mathcal{O}(u^{\varepsilon} \lg v)$ for some constant $\varepsilon > 0$.

2.6.3 Lower bound for l-wise independent hash functions

Motivated by recent work by Thorup [51] showing 2-wise independent hash functions to work well for Jaccard estimation we will now consider an instance where any estimator based on the k smallest hash values of a $\mathcal{O}(1)$-wise independent hash function will not be unbiased. In particular the argument follows from the existence of small families of hash functions.

Theorem 2.14. *Let $[u]$ be the universe of elements and $h : [u] \mapsto 0, \dots, v-1$ be any l-wise independent hash function for $l = \mathcal{O}(1)$. There exists an instance on v^2 sets $S_1, \dots, S_{v^2}$ with intersection size $t = |\cap_i S_i| = n - k$. For any estimator $\tilde{t}$ for t that guarantees a relative error bound and is based on k size min-wise summaries constructed using h it holds that $\hat{t}$ is not unbiased.*

For Theorem 2.14 we construct an instance on v^l sets where one of the k one-permutation min-wise summaries will hold no elements from S with high probability.

Proof. Let $h : [u] \mapsto \{0, \dots, v-1\}$ be a l-independent hash function where $l < \log_v m$. We will consider an instance on $m > v^l$ sets that has large intersection S, but where an unbiased estimator of the intersection size $|S|$ using the smallest k hash values is not possible with high probability.

For any h there exists a set B_z of size k where $h(B_z) = \{0, \dots, k-1\}$, i.e., the k elements of B_z map to the k smallest possible hash values. Let $S_i = B_i \cup S$ for $0 \leq i < v^l$ be n-sized sets where S is the intersecting elements to be specified later. We have $h(S_z) = h(B_z) \cup h(S) = \{0, \dots, k-1\} \cup h(S)$ and $z \in \{0, \dots, v^l\}$, i.e., by the existence of size v^l families of hash function there is a hash function that hashes k elements from a particular set S_z to the k smallest possible hash values. It follows that if $\forall j \in h(S) : j \geq k$ then the set of the k smallest hash values will contain no elements from S, even though we have size $|S| = n - k$. For a uniformly random $n-k$-sized set S we have $\mathbf{Pr}\left[\forall j \in h(S) : j \geq k\right] = \left(1 - \frac{k}{v}\right)^{n-k}$ which is ≈ 1 for $k \ll n$.

Hence if we consider the intersection S of all $m > v^l$ sets S_i it will hold with high probability that this instance will have intersection size $|S| = n - k$ but no elements from S in the set of the k smallest hash values. Consider the case of there being no elements from S in the set of the k smallest hash values and let $\tilde{t}$ be an estimate of $|S|$. Any estimate $\tilde{t}$ of $|S|$ with relative bounded error that is based on v^l min-wise summaries will be unable to distinguish the case of $|S| = 0$ from $|S| = n - k$ when there are no elements from S in the set of the k smallest hash values. Thus when presented with such a set the estimate will always be that $\tilde{t} = 0$. Let ϕ be the probability of there being no elements in from S in the set of the k smallest hash values. Then let the outcome of the random variable X be the estimate $\tilde{t}$. We have $\mathbb{E}[X] \leq \phi 0 + (1-\phi)n$ where $\phi = \left(1 - \frac{k}{v}\right)^{n-k}$.

To obtain an unbiased estimator $\mathbb{E}[X] = n - k$ for this instance we need $(1-\phi)n \geq n-k$ hence $\phi < k/n$. Now assuming n and v are polynomial is sufficient: if $v = k^3$ and $n = k^2$ then we have $\phi = \left(1 - \frac{k}{v}\right)^{n-k} = \left(1 - \frac{}{k^2}\right)^{k^2-k} > 1/e$. Since we have $k/n = 1/k$ then assuming $k > e$ implies that that $\phi > k/n$. Thus when v and n are polynomial in k it is not possible to create an unbiased estimator on the intersection size. ■

2.7 Space of k-permutation min-wise summaries on sunflower sets

Sunflower sets. In this final section we give details on the hard instance that gives the bound for k-permutation min-wise hashing of Table 2.1. For m sets $S_1 \dots S_m$ each of size n, let $t = |\cap_i S_i|$ be the intersection size of all sets. Then a sunflower instance has

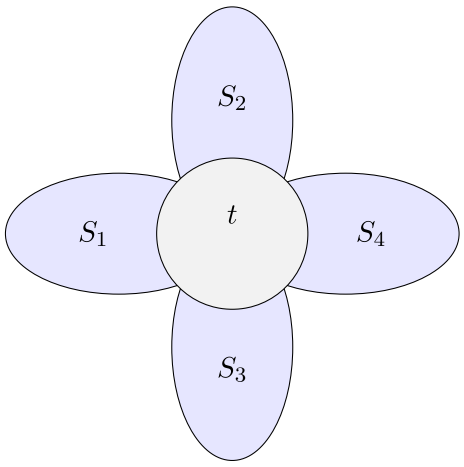


FIGURE 2.3: Hard instance for k-permutation summaries for set intersection. Must store number of elements inversely proportional to the sampling probability which is t over the union size. A sunflower instance of size m (shown for $m = 4$) has union size $t + m(n - t)$.

the property $\forall_{i \neq j} |S_i \cap S_j| = t$, i.e., the m sets are disjoint except for the t intersection elements. The union size of such an instance is $|\cup_i S_i| = t + mn - mt = t + m(n - t)$ as there are t elements in the intersection and each of the m sets hold additional $n - t$ elements. It follows that the Jaccard similarity for a sunflower instance is $t/(t+m(n-t))$. The hard instance is illustrated in Figure 2.3.

As we have sampling probability t over the union size to get an element from the intersection it is easy to show that to get a good estimate we need to store roughly inversely proportional to the sampling rate. Note that Lemma 2.15 below is not a true lower bound in the sense that it is in fact an upper bound argument with the conjecture that it should not be possible to do better than sampling independently.

Lemma 2.15. *Given m sets of size n with intersection size t. To obtain an $(\epsilon, \mathcal{O}(1))$-estimate of t using k-permutation min-wise hashing one needs to store $\mathcal{O}\left(\frac{mn}{t\epsilon^2}\right)$ elements from each set.*

Proof. The upper bound for k-permutation min-wise hashing of Table 2.1 is derived as follows. Let $X_1 \dots X_r$ be independent Bernoulli trials where $\mathbf{Pr}[X_i] = J$ and let $X = \sum_{i=1}^{r} X_i$ and $\mu = \mathbb{E}[X] = rJ$. There exists a r for which there is constant probability of the event that the outcome of X is a relative factor ε from $\mathbb{E}[X]$. This can be bounded applying a Chernoff-Hoeffding bound on X as follows.

$$\begin{aligned}\mathbf{Pr}\left[|X - \mathbb{E}[X]| \geq (1+\varepsilon)\mathbb{E}[X]\right] &= \mathbf{Pr}\left[|X - rJ| \geq (1+\varepsilon)rJ\right] \\ &= \delta \geq 2e^{(-rJ\varepsilon^2)/3}\end{aligned}$$

Then isolating c we have $r \geq \frac{3\log(2/\delta)}{J\varepsilon^2}$, which for $\delta = \mathcal{O}(1)$ is $\mathcal{O}\left(\frac{1}{J\varepsilon^2}\right) = \mathcal{O}\left(\frac{t+m(n-t)}{t\varepsilon^2}\right)$ following from the Jaccard similarity of the sunflower instance above. For $t < n/2$ we have $r = \mathcal{O}\left(\frac{mn}{t\varepsilon^2}\right)$, the sample size required for k-permutation min-wise summaries. ■

We note that the argument presented is of an upper bound, but conjecture that it is tight as well.

Chapter 3

Matrix Multiplication

3.1 Chapter overview

This chapter will deal with the topic of computing matrix products where either the input matrices or the output matrix contains few nonzero entries. For both cases we present algorithms that in the I/O model and word-RAM model respectively improve upon current state of the art algorithms for significant parts of the parameter space. Specifically, in Section 3.2 we present an algorithm that is optimal in the number of I/Os when parameterized on the number of nonzero entries in the input N and the number of nonzero entries in the output Z. Followed by this we present in Section 3.3 an algorithm that when the input matrices are dense and the output matrix is sparse improves upon state of the art methods. The two algorithms relate to each other only by dealing with the same problem in different models as they use different techniques to achieve their complexity. However a common trait of them is that they are both Monte Carlo and hence can fail with probability polynomially small in the input size. The complexity of the two algorithms is summarized here, see Sections 3.2.2 and 3.3.2 for a discussion on the most related algorithms and their complexities.

The two new algorithms summarized. We use N as the number of nonzero entries in the input, Z as the number of nonzero entries in the output, U as the matrix dimensions, M as the internal memory size, B as the block size and ω is the exponent for which it is possible to multiply two $U \times U$ matrices in time $\mathcal{O}(U^\omega)$.

1. In the I/O model we achieve a $\tilde{\mathcal{O}}\big(N\sqrt{Z}/(B\sqrt{M})\big)$ I/Os upper and lower bound.
2. In the word-RAM model we achieve a $\tilde{\mathcal{O}}\big(U^2(Z/U)^{\omega-2}\big)$ RAM operations upper bound.

3.2 Sparse matrix multiplication in the I/O model

First we consider the fundamental problem of multiplying matrices that are *sparse*, that is, the number of nonzero entries in the input matrices (but not necessarily the output matrix) is much smaller than the number of entries. Matrix multiplication is a fundamental operation in computer science and mathematics, due to the wide range of applications and reductions to it — e.g. computing the determinant and inverse of a matrix, or Gaussian elimination. Matrix multiplication has also seen lots of use in non-obvious applications such as bioinformatics [53], computing matchings [54, 55] and algebraic reasoning about graphs, e.g. cycle counting [56, 57].

Matrix multiplication in the general case has been widely applied and studied in a pure math context for decades. In an algorithmic context matrix multiplication is known to be computable using $\mathcal{O}(U^\omega)$ ring operations, for some constant ω between 2 and 3. The first improvement over the trivial cubic algorithm was achieved in 1969 in the seminal work of Strassen [13] showing $\omega \leq \log_2 7$ and most recently Vassilevska Williams [58] improved this to $\omega < 2.373$.

Matrix multiplication over a semiring, where additive inverses cannot be used, is better understood. In the I/O model introduced by Aggarwal and Vitter [59] the optimal matrix multiplication algorithm for the dense case already existed (see Section 4.1.3) and since then sparse-dense and sparse-sparse combinations of vector and matrix products have been studied, e.g. in [60–62].

The main contribution of this chapter to the sparse setting is a tight bound for matrix multiplication over a semiring in terms of the number of nonzero entries in the input and output matrices, generalizing the classical result of Hong and Kung on dense matrices [5] to the sparse case.

3.2.1 Preliminaries

Let A and C be matrices of U rows and U columns and let every entry $(A)_{i,j}, (C)_{i',j'} \in R$ for semiring R. Further for matrix A let A_{i*} denote row i of A and let A_{*j} denote column j of A. The matrix product AC, where each entry $(AC)_{i,j}$, $i, j \in [U]$ is given as $(AC)_{i,j} = \sum_k (A)_{i,k}(C)_{k,j}$. A nonzero term $(A)_{i,k}(C)_{k,j}$ is referred to as an *elementary product*. We say that there is *no cancellation* of terms when $(AC)_{i,j} = 0$ implies that $(A)_{i,k}(C)_{k,j} = 0$ for all k. For *sparse* semiring matrix multiplication, the number of entry pairs with nonzero product measures the number of operations performed up to a constant factor assuming optimal representation of the matrices. Specifically, let $\sum_{k=1}^{n} |\{j \mid (A)_{j,k} \neq 0\}||\{i \mid (C)_{k,i} \neq 0\}|$ be the number of such nonzero pairs of matrix entries. Finally let $\texttt{nnz}(A) = |\{i, j \mid (A)_{i,j} \neq 0\}|$ denote the number of nonzero entries of matrix A. When no explicit base is stated, logarithms in this chapter are base 2.

Semiring I/O model. The model of computation used for this result is the standard I/O model [59] introduced in Section 1.3.2 extended slightly. Here we will also use $\tilde{\mathcal{O}}(\cdot)$-notation to suppress polylogarithmic factor in input size N and matrix dimension U.

We assume that a word is big enough to hold a matrix element from a semiring as well as the matrix coordinates of that element, i.e., a block holds B matrix elements. We restrict attention to algorithms that work with semiring elements as an abstract type, and can only copy them, and combine them using semiring operations. We refer to this restriction as the *semiring I/O model.* Our upper and lower bounds use a slight extension of this model in which equality check is allowed, which allows us to take advantage of *cancellations*, i.e., inner products in the matrix product that are zero in spite of nonzero elementary products. The lower bound holds for an even more general setting where an algorithm uses P internal memory units each connected to a processing unit and thus it matches the model of the upper bound for $P = 1$.

The problem we solve. Given matrices A and C of dimension $U \times U$, containing $\texttt{nnz}(A)$ and $\texttt{nnz}(C)$ non-zero semiring elements from semiring R, respectively, we wish to output a sparse representation of the matrix product AC in the external memory model. We are dealing with sparse matrices represented as a list of tuples of the form $(i, j, (A)_{i,j})$, where $(A)_{i,j} \in R$ is a (nonzero) matrix entry. To produce output we must call a function $\texttt{emit}(e)$ for every nonzero entry $e = (i, j, (AC)_{i,j})$ of AC. We only allow $\texttt{emit}(\cdot)$ to be called once on each output element, but impose no particular order on the sequence of outputs.

We note that the algorithm could be altered to write the entire output before termination by, instead of calling $\texttt{emit}(\cdot)$, simply writing the output element to a disk buffer, outputting all $\texttt{nnz}(AC)$ elements using $\mathcal{O}(\texttt{nnz}(AC)/B)$ additional I/Os. However, in some applications such as database systems (see [14]) there may not be a need to materialize the matrix product on disk, so we prefer the more general method of generating output.

3.2.2 Related work

The external memory model was introduced by Aggarwal and Vitter in their seminal paper [59], where they provide tight bounds for a collection of central problems.

An I/O-optimal matrix multiplication algorithm for dense semiring matrices was shown by Hong and Kung [5]: Group the matrices into $c\sqrt{M} \times c\sqrt{M}$ submatrices where constant c is picked such that three $\sqrt{M} \times \sqrt{M}$ matrices fit into internal memory. This reduces the problem to $\mathcal{O}((U^3/\sqrt{M})^3)$ matrix products that fit in main memory, costing $\mathcal{O}(M/B)$ I/Os each, and hence $\mathcal{O}(U^3/B\sqrt{M})$ in total [12]. Hong and Kung also provided a tight lower bound $\Omega(U^3/B\sqrt{M})$ that holds for algorithms that work over a semiring. (It does not apply to algorithms that make use of subtraction, such as fast matrix multiplication methods, for which the blocking method described above yields an I/O complexity of $U^\omega/(M^{\omega/2-1}B)$ I/Os.)

For sparse matrix multiplication the previously best upper bound [14], shown for Boolean matrix products but claimed for any semiring, is $\tilde{\mathcal{O}}(N\sqrt{\texttt{nnz}(AC)}/BM^{1/8})$. However this bound requires "no cancellation of terms" (or more specifically, the output sensitivity is with respect to the number of output entries that have a nonzero elementary product). Our new upper bound of this section improves upon this: The Monte Carlo algorithm of Theorem 3.1 has strictly lower I/O complexity for the entire parameter space and makes no assumptions about cancellation.

An important subroutine in our algorithm is dense-vector sparse matrix multiplication: For a vector y and sparse matrix A we can compute their product using optimal $\tilde{\mathcal{O}}((\texttt{nnz}(A) + \texttt{nnz}(y))/B)$ I/Os [60] - this holds for arbitrary layouts of the vector and matrix on disk.

Our algorithm has an interesting similarity to Williams and Yu's recent output sensitive matrix multiplication algorithm [63, Section 6]. Their algorithm works by splitting the matrix product into 4 submatrices of equal dimension, running a randomized test to determine which of these subproblems contain a nonzero entry. Recursing on the non-zero submatrices, they arrive at an output sensitive algorithm. We perform a similar recursion, but the splitting is computed differently in order to recurse in a balanced manner, such that each subproblem at a given level of the recursion outputs approximately the same number of entries in the matrix product.

Size estimation of the number of nonzeros in matrix products was used by Cohen [64, 65] to compute the order of multiplying several matrices to minimize the total number of operations. For constant error probability this algorithm uses $\mathcal{O}(\varepsilon^{-2}N)$ operations in the RAM model to perform the size estimation. For $\varepsilon > 4/\texttt{nnz}(AC)^{1/4}$ Amossen et al. [66] improved the running time to be expected $\mathcal{O}(N)$ in the RAM model and expected $\mathcal{O}(\text{sort}(N))$ in the I/O model. Contrary to the approaches of [64–66] our new size estimation algorithm presented in Section 3.2.4 is able to deal with cancellation of terms, and it uses $\tilde{\mathcal{O}}(\varepsilon^{-3}N/B)$ I/Os. Informally, the main idea of our size estimation algorithm is to multiply a sequence of vectors x with certain properties onto AC but in the computationally inexpensive order $(xA)B$, in order to produce linear sketches of the rows (columns) of AC.

3.2.3 Our results

We present a new upper bound in the I/O model for sparse matrix multiplication over semirings. Our I/O complexity is at least a factor of roughly $M^{3/8}$ better than that of [14]. We show the following theorem:

Theorem 3.1. *Let A and C be dimension $U \times U$ matrices with entries from a semiring R, and let $N = \texttt{nnz}(A) + \texttt{nnz}(C)$, $Z = \texttt{nnz}(AC)$. There exist algorithms (a) and (b) such that:*

(a) emits the set of nonzero entries of AC with probability at least $1 - 1/U$, using $\tilde{\mathcal{O}}\left(N\sqrt{Z}/(B\sqrt{M})\right)$ I/Os.

(b) emits the set of nonzero entries of AC, and uses $\mathcal{O}\left(N^2/(MB)\right)$ I/Os.

For every A and C, using $\tilde{\mathcal{O}}(N/B)$ I/Os we can determine with probability at least $1 - 1/U$ if one of the two I/O bounds is significantly lower, i.e., distinguish between $N\sqrt{Z}/(B\sqrt{M}) > 2N^2/(MB)$ and $2N\sqrt{Z}/(B\sqrt{M}) < N^2/(MB)$.

The above theorem makes no assumptions about cancellation of terms. In particular, $\texttt{nnz}(AC)$ can be smaller than the number of output entries that have nonzero elementary products.

Our second main contribution is a new lower bound on sparse matrix multiplication in the parallel semiring I/O model.

Theorem 3.2. *For all positive integers N and $Z < N^2$ there exist matrices A and C with $\texttt{nnz}(A), \texttt{nnz}(C) \le N$, $\texttt{nnz}(AC) \le Z$, such that an algorithm computing AC in the parallel semiring I/O model requires $\Omega\left(\min\left(\frac{N^2}{PBM}, \frac{N\sqrt{Z}}{PB\sqrt{M}}\right)\right)$ I/Os.*

We note that our lower bound holds in the more general parallel setting with P processing units with internal memories. Since we can determine and run the algorithm satisfying the minimum complexity of the lower bound (for $P = 1$), our bounds are tight.

Result structure. Section 3.2.4 describes a new size estimation algorithm which we will use as a subprocedure for our sparse matrix multiplication algorithm. The new size estimation algorithm may be of independent interest since to the knowledge of the authors there are no published size estimation procedures that handle cancellation of terms.

Section 3.2.5 first describes a simple output insensitive algorithm in Section 3.2.5.1, algorithm (b) of Theorem 3.1. Then we describe how algorithm (a) of Theorem 3.1 works: In Section 3.2.5.4 we describe how to divide the sparse matrix product into small enough subproblems (with respect to output size), and Section 3.2.5.3 describes how a version of Pagh's "compressed matrix multiplication" algorithm yields an I/O efficient algorithm for subproblems with a small output. Finally in Section 3.2.6 we show the new tight lower bound of Theorem 3.2.

3.2.4 Matrix output size estimation

We present a method to estimate column/row sizes of a matrix product AC, represented as a sparse matrix. In particular, for a column C_{*k} (or analogously row A_{k*}) we are interested in estimating the number of nonzeros $\texttt{nnz}(A(C)_{*k})$ ($\texttt{nnz}((A)_{k*}B)$). We note that there are no assumptions about (absence of) cancellation of terms in the following. We show the existence of the following algorithm.

Lemma 3.3. *Let A and C be dimension $U \times U$ matrices with entries from semiring R, $N = \texttt{nnz}(A) + \texttt{nnz}(C)$ and let $0 < \varepsilon, \delta \leq 1$. We can compute estimates $z_1, \ldots, z_U$ using $\tilde{\mathcal{O}}(\varepsilon^{-3}N/B)$ I/Os and $\mathcal{O}(\varepsilon^{-3}N \log(U/\delta) \log U)$ RAM operations such that with probability at least $1-\delta$ it holds that $(1-\varepsilon)\,\texttt{nnz}((AC)_{*k}) \leq z_k \leq (1+\varepsilon)\,\texttt{nnz}((AC)_{*k})$ for all $1 \leq k \leq U$.*

We note that Lemma 3.3 by symmetry can give the same guarantees for rows of the matrix product, which is done analogously by applying the algorithm to the product $(AC)^T = C^T A^T$. Further, from Lemma 3.3 we have, following from combining of all column estimates, an estimate of $\texttt{nnz}(AC)$.

Corollary 3.4. *Let A and C be dimension $U \times U$ matrices with entries from semiring R, $N = \texttt{nnz}(A) + \texttt{nnz}(C)$ and let $0 < \varepsilon, \delta \leq 1$. We can compute $\hat{Z}$ in $\tilde{\mathcal{O}}(\varepsilon^{-3}N/B)$ I/Os and $\mathcal{O}(\varepsilon^{-3}N \log(U/\delta) \log U)$ RAM operations such that with probability at least $1-\delta$ it holds that $(1-\varepsilon)\,\texttt{nnz}(AC) \leq \hat{Z} \leq (1+\varepsilon)\,\texttt{nnz}(AC)$.*

At a high level, the algorithm is similar in spirit to Cohen [64, 65], but uses linear F_0 sketches (see e.g. [36, 67]) that serve the purpose of capturing cancellation of terms.

We will make use of a well-known F_0-sketching method [67, 68], where $F_0(f)$ denotes the number of non-zero entries in a vector f. Consider a data stream of items of the form $((i,j), r)$, where $(i,j) \in U \times U$ and $r \in R$. The stream defines a vector indexed by $U \times U$ (which can also be thought of as a matrix), where entry (i,j) is the sum of all ring elements r that occurred with index (i,j) in the stream.

For a matrix A of dimension $U \times U$ the number of distinct indices is the sum of distinct indices over all column vectors $F_0(A) = \sum_{i \in [U]} F_0(A_{i*})$. One can compute in space $\mathcal{O}(\varepsilon^{-3} \log U \log \delta^{-1})$ [67, 68] a *linear* sketch over a length U vector that can output a number $\hat{z}$, where

$$(1-\varepsilon)F_0 \leq \hat{z} \leq (1+\varepsilon)F_0$$

holds with at least constant probability.

High-level algorithm description. We compute a linear sketch F followed by the matrix product $v = FAC$. From v for a given T we can distinguish between a column having more than $(1+\varepsilon)T$ and less than $(1-\varepsilon)T$ nonzero entries - we repeat this procedure for suitable values of T to achieve the final estimate. We use the following distinguishability result:

Fact 3.5. ([68], Section 2.1) There exists a projection matrix $P' \in \{0,1\}^{n \times d}$ such that for each frequency vector $f \in R^{1 \times n}$ we can be estimate $F_0(f)$ from fP'. In particular, for fixed $T' > 0$, $0 < \varepsilon', \delta' \leq 1$ with probability $1 - \delta'$ we can distinguish the cases $F_0(f) > (1+\varepsilon')T'$ and $F_0(f) < (1-\varepsilon')T'$ using space $d = \mathcal{O}(\varepsilon'^{-2} \log \delta'^{-1})$.

We will apply this distinguishability sketch to the columns of the product AC, since $F_0(AC) > (1+\varepsilon)T$ implies $\texttt{nnz}(AC) > (1+\varepsilon)T$ and analogously for the second case. This follows trivially from the definition of F_0 and the number of nonzeros in a matrix product. From Fact 3.5 we have a sketch $F \in \{0,1\}^{d \times U}$ which multiplied with a matrix A we can for the columns $[FA]_{*k}$ distinguish $\texttt{nnz}(A_{*k}) > (1+\varepsilon)T$ from $\texttt{nnz}(A_{*k}) < (1-\varepsilon)T$ with probability $1 - \delta$.

Proof. (Lemma 3.3) Let $F \in \{0,1\}^{d \times U}$ be a F_0-distinguishability sketch as described in Fact 3.5. To ensure that for every of the U columns in $v = FAC$ we can distinguish the two cases with probability at least $1 - \delta$ it is sufficient to invoke the algorithm from Fact 3.5 with $\delta' = \delta/U$. By the union bound over the error probabilities we have $\sum_{1 \leq i \leq U} \delta/U = \delta$. By linearity of F we have that from v we can for all columns $k \in [U]$ distinguish the cases $[AC]_{*k} < (1-\varepsilon)T$ and $[AC]_{*k} > (1+\varepsilon)T$.

Also by linearity, the order of operations in the computation of v is chosen to be $v = (FA)C$, hence the computation of v can be seen as $2d$ dense-vector sparse-matrix multiplications of dimension $1 \times U \times U$.

Remember that dense vector $y^{1 \times U}$ and sparse matrix A of dimension $U \times U$ we can compute yA in

$$\mathcal{O}((\texttt{nnz}(A)/B) \log_{M/B}(U/M))\text{I/Os [60].}$$

Letting $N = \texttt{nnz}(A) + \texttt{nnz}(C)$ computing v for a value of T has I/O complexity

$$\begin{aligned}\mathcal{O}(2d(N/B)\log_{M/B}(U/M) &= \mathcal{O}(\varepsilon^{-2}(N/B)\log_{M/B}(U/M)\log(U/\delta)\\ &= \tilde{\mathcal{O}}(\varepsilon^{-2}N/B). \end{aligned} \tag{3.1}$$

We note that for sparse matrices this bound is $\tilde{\mathcal{O}}(\text{sort}(U))$. Analogously, the number of RAM operations needed to compute v for a specific T is $\mathcal{O}(\varepsilon^{-2}N\log(U/\delta))$.

Since for a given T we can now using I/Os given in (3.1) distinguish $[AC]_{*k} < (1-\varepsilon)T$ and $[AC]_{*k} > (1+\varepsilon)T$ we simply repeat this procedure for $\mathcal{O}(\varepsilon^{-1}\log U)$ values $T = 1, (1+\varepsilon), (1+\varepsilon)^2, \ldots, \mathcal{O}(U)$, which yields a $1 \pm \varepsilon$ estimate of the number of nonzeros in each column, from which we have the desired estimate of the total number of nonzeros.

■

We note that the algorithm of Lemma 3.3 can be obtained using any linear F_0 sketch in I/O complexity $\mathcal{O}(\xi(N/B)\log_{M/B}(U/M))$, where ξ is the space complexity of the sketch used. From Lemma 3.3 we get Corollary 3.4 by combining all column estimates, an estimate of $\texttt{nnz}(AC)$. This size estimation algorithm will be used both in our I/O efficient coloring scheme of Section 3.2.5.4 as well as in partitioning the problem into approximately dense subproblems in our RAM-model result of Section 3.3.6.

3.2.5 Cache-aware upper bound

As in the previous section let A and C be dimension $U \times U$ matrices with entries from a semiring R, and let $N = \texttt{nnz}(A) + \texttt{nnz}(C)$ be the input size. We will start by describing a sample "output insensitive" algorithm in Section 3.2.5.1 followed by the main algorithm, an output sensitive Monte Carlo algorithm. The reason for describing the simple output insensitive algorithm is that it is needed to a tight upper bound for the entire parameter space.

3.2.5.1 Output insensitive algorithm

We first describe algorithm (a) of Theorem 3.1, which is insensitive to the number of output entries $\texttt{nnz}(AC)$. It works as follows: First put the entries of C in column-major order by lexicographic sorting. For every row a_i of A with more than $M/2$ nonzeros, compute the vector-matrix product a_iC in time $\tilde{\mathcal{O}}(N/B)$ using the algorithm of [60]. There can be at most $2N/M$ such rows, so the total time spent on this is $\tilde{\mathcal{O}}(N^2/(MB))$. The remaining rows of A are then gathered in groups with between $M/2$ and M nonzero entries per group. In a single scan of C (using column-major order) we can compute the

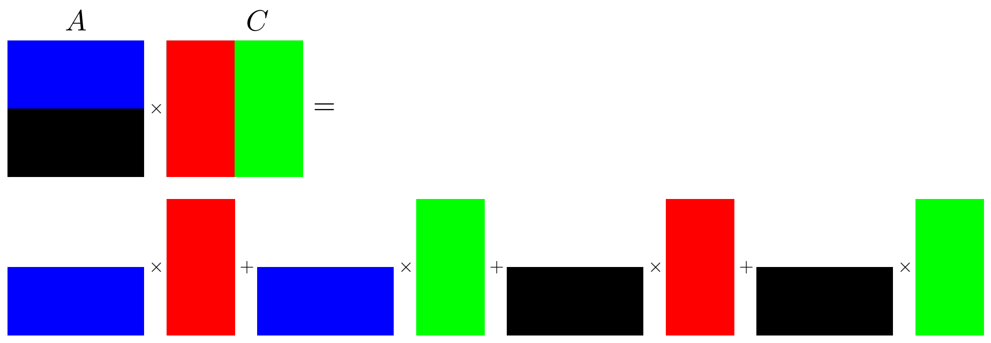


FIGURE 3.1: Partitioning of the input matrices A and C. The matrix product can be rewritten as a sum of smaller matrix products. Unfortunately such a simple partitioning scheme as shown is not possible in general.

product of each such row with the matrix C. The number of I/Os is $O(N/B)$ for each of the at most $2N/M$ groups, so the total complexity is $\tilde{O}(N^2/(MB))$.

3.2.5.2 Monte Carlo algorithm overview

We next describe algorithm (b) of Theorem 3.1 The algorithm works by first performing a step of coloring, the purpose of which is to partition the matrix product into submatrices, each of which can be computed efficiently. Consider this basic fact about matrix multiplication of matrices A and C: If you partition the rows of A and the columns of C then the matrix product can be written as a sum of products of the partitions. See illustration in Figure 3.1. Intuitively, if there was such a partitioning scheme for φ colors as shown in Figure 3.1 for $\varphi = 2$ where each row-column partition combination has few output entries, then we would be able to compute this efficiently in the I/O model. However, there are matrices where such a simple scheme would fail. Instead we employ a different coloring scheme: We create φ partitions that together form the matrix product and each of the φ partitions only consist of subproblems that are small enough to fit in main memory.

The overall idea is to color matrix rows A using φ colors and for each of the φ sets of colored rows we color matrix C also using φ colors, such that every combination of colored rows from A and colored columns from C yields a low number of non-zero output entries. The situation can be seen in Figure 3.2 for $\varphi = 2$. The main point is that every partition of rows from A has its own partitioning of the columns of C, meaning φ different partitions in total.

If there was no cancellation of terms possible, this would be the algorithm in full. However, the difficulty of handling cancellations come from the subtlety that even though the number of outputs generated from all our subproblems are small enough to fit in memory, the intermediate number of nonzero elementary products can only be bounded

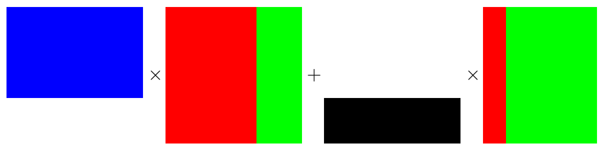


FIGURE 3.2: How we partition. We create φ different partitions (shown for $\varphi = 2$) that make up the result. It turns out that for arbitrary matrices there is such a scheme where each subproblem is small enough to fit in memory.

by the dimension of the subproblem. Informally, the situation is that we need to compute a big number of elementary products to know which of them cancel out to produce the output but on the other hand we do not have enough I/Os to spare, since we want our algorithm to run proportional to the number of output entries Z and the number of input entries N. To solve this, we use a "compressed" matrix multiplication algorithm (described by Lemma 3.6) to compute the output entries of every such combination. The number φ of colors needed to achieve the algorithm, to be specified later, depends on an estimate of $\texttt{nnz}(AC)$, found using Corollary 3.4.

A technical hurdle is that there might be rows of A and columns of C that we cannot color because they generate too many entries in the output. However, it turns out that we can afford to handle such rows/columns in a direct way using vector-matrix multiplication.

3.2.5.3 Compressed matrix multiplication in the I/O model

Let $\gamma > 0$ be a suitably small constant, and define $r = 4\gamma M/\log U$. We now describe an I/O-efficient algorithm for matrix products AC with $\texttt{nnz}(AC) \leq \gamma M/\log U = r/4$ nonzeros. If A is stored in column-major order and C is stored in row-major order, the algorithm makes just a single scan over the matrices.

The algorithm is a variation of the one found in [62], adapted to the semigroup I/O model. Specifically, for some constant ℓ and $d = 1, \ldots, \ell \log U$ let $h_d, h'_d : [U] \to [r]$ be pairwise independent hash functions. The algorithm computes the following $\ell \log U$ polynomials of degree at most $2r$:

$$p_d(x) = \sum_{k=1}^{U} \left(\sum_{i=1}^{U} A_{i,k} x^{h_d(i)} \right) \left(\sum_{j=1}^{U} C_{k,j} x^{h'_d(j)} \right) .$$

It is not hard to see that the polynomial $\sum_{i=1}^{U} A_{i,k} x^{h_d(i)}$ can be computed in a single scan over column i of A, using space r. Similarly, we can compute the polynomial $\sum_{j=1}^{U} C_{k,j} x^{h'_d(j)}$ in space r by scanning row j of C. As soon as both polynomials have been computed, we multiply them and add the result to the sum of products that will eventually be equal to $p_d(x)$. This requires additional space $2r$, for a total space usage of $4r$.

Though a computationally less expensive approach is described in [62], we present a simple method that (without using any I/Os) uses the polynomials $p_d(x)$, $d = 1, \ldots, \ell \log U$, to compute the set of entries in AC with probability $1 - U^{-3}$. For every i and j, to compute the value of $(AC)_{i,j}$ consider the coefficient of $x^{h_d(i)+h'_d(j)}$ in p_d, for $d = 1, \ldots, \ell \log U$. For suitably chosen c, with probability $1 - U^{-5}$ the value $(AC)_{i,j}$ is found in the majority of these coefficients. The majority coefficient can be computed using just equality checks among semigroup elements [69]. The analysis in [62] gives us, for a suitable choice of γ and ℓ, the following:

Lemma 3.6. *Suppose matrix A is stored in column-major order, and C is stored in row-major order. There exists an algorithm in the semiring I/O model augmented with equality test, and an absolute constant $\gamma > 0$, such that if $\texttt{nnz}(AC) < \gamma M / \log U$ the algorithm outputs the nonzero entries of AC with probability $1 - U^{-3}$, using just a single scan over the input matrices.*

3.2.5.4 Computing a balanced coloring

Let color set S_i contain rows A_{k*} that are assigned color i, and for each color i assigned to rows of A let color set $S_j^{(i)}$ contain columns C_{*k} that are assigned color j. Also, let $A|S_i$ be the input matrix A restricted to contain only elements in rows from S_i (and analogously for C and $S_j^{(i)}$).

The goal of the coloring step is to assign the colors such that for every pair of color sets $(S_i, S_j^{(i)})$, $1 \le i, j \le \varphi$ it holds that $\texttt{nnz}((A|S_i)(C|S_j^{(i)})) < \gamma M / \log U$. This can be seen as coloring the rows of A once and the columns of C φ times, each time with φ colors, creating φ^2 subproblems whose output fits in memory.

Lemma 3.7. *Let A and C be dimension $U \times U$ matrices with $N = \texttt{nnz}(A) + \texttt{nnz}(C)$ nonzero entries.*
Using $\tilde{\mathcal{O}}\left(\frac{N\sqrt{\texttt{nnz}(AC)}}{B\sqrt{M}}\right)$ I/Os a coloring with $\varphi = \sqrt{\frac{\texttt{nnz}(AC)\log U}{M}} + \mathcal{O}(1)$ colors can be computed that assigns a color to rows of A and for each such color i, assigns colors to columns of C such that:

1. *For every $i, j \in [\varphi]$ it holds that $\texttt{nnz}\left((A|S_i)(C|S_j^{(i)})\right) < M/\log U$.*
2. *Rows from A and columns from C that are not in some color sets S_i and $S_j^{(i)}$ has had their nonzero output entries emitted.*

Proof. At a high level, the coloring will be computed by recursively splitting the matrix rows in two disjoint parts to form matrices A_1 and A_2 where A_1 contains the nonzeros from the first $d-1$ rows, for some d, and A_2 contains the nonzeros from the last $U-d$ rows. Row number d, the "splitting row", will be removed from consideration by generating the corresponding part of the output using I/O-efficient vector-matrix multiplication. We wish to choose d such that:

I $\mathtt{nnz}(A_1C) \in \left[(1-\log^{-1} U)\,\mathtt{nnz}(AC)/2; (1+\log^{-1} U)\,\mathtt{nnz}(AC)/2\right]$.

II $\mathtt{nnz}(A_2C) \in \left[(1-\log^{-1} U)\,\mathtt{nnz}(AC)/2; (1+\log^{-1} U)\,\mathtt{nnz}(AC)/2\right]$.

And after $\log\varphi + \mathcal{O}(1)$ recursive levels of such splits, we will have $\mathcal{O}(\varphi)$ disjoint sets of rows from A. For each such set we then compute disjoint column sets of C in the same manner, and we argue below that this gives us subproblems with output size $\mathtt{nnz}(AC)/\varphi^2 = M/\log U$, where each subproblem corresponds exactly to a pair of color sets as described above.

In order to compute the row number d around which to perform the split, we invoke the estimation algorithm from Corollary 3.4 with $\varepsilon = \log^{-1} U$ such that for every row in $(AC)_{k*}$ we have access to an estimate $\hat{z}_k$ where it holds with probability at least $1-U^{-c}$ (for fixed $c > 0$ chosen to get sufficiently low error probability):

$$\hat{z}_k \in \left[(1-\log^{-1} U)\,\mathtt{nnz}((AC)_{k*})/2; (1+\log^{-1} U)\,\mathtt{nnz}((AC)_{k*})/2\right]. \tag{3.2}$$

In particular for any set of rows r we have that

$$(1-\log^{-1} U)\,\mathtt{nnz}\left(\sum_{i\in r}(AC)_{i*}\right) \le \sum_{i\in r}\hat{z}_i \le (1+\log^{-1} U)\,\mathtt{nnz}\left(\sum_{i\in r}(AC)_{i*}\right). \tag{3.3}$$

We will now argue that if we can create a split of the rows such that (I) and (II) hold, then when the splitting procedure terminates after $\log\varphi + \mathcal{O}(1)$ recursive levels, we have that for each pair of colors it is the case that $(A|S_i)(C|S_j^{(i)}) < M/\log N$. Consider the case where each split is done with the maximum positive error possible, i.e., on recursive level q we have divided the $\mathtt{nnz}(AC)$ nonzeros into subproblems where each are of size at most $\mathtt{nnz}(AC)(1/2 + 1/(2\log U))^q$. After $\log\varphi + \mathcal{O}(1)$ recursive levels we have subproblem size:

$$\begin{aligned}
\mathtt{nnz}(AC)\left(\frac{1}{2}+\frac{1}{2\log U}\right)^{\log\varphi^2} &= \mathtt{nnz}(AC)2^{-\log\varphi^2}\left(1+\frac{1}{\log U}\right)^{\log\varphi^2} \\
&\le \mathtt{nnz}(AC)2^{-\log\varphi^2}e^{\frac{\log\varphi^2}{\log U}} \qquad (3.4)\\
&\le \mathtt{nnz}(AC)\mathcal{O}(1)/\varphi^2 = \mathcal{O}(M/\log U) \qquad (3.5)
\end{aligned}$$

The main observation to see that we get the right subproblem size as in (3.5) is that for each recursion we decrease the output size by a factor $\Omega(\varphi)$. For (3.4) we use $(1+1/x)^y \leq e^{y/x}$, and (3.5) follows from $\texttt{nnz}(AC) \leq U^2$ and the definition of φ. The analysis for the case where each split is done with the maximum negative error possible is analogous and thus omitted.

We will now argue that with access to the $\hat{z}_i$ estimates as in (3.2) we can always construct a split such that (I) and (II) hold. Let partitions 1 and 2 be denoted P_1 and P_2 and $\hat{z} = \sum_i \hat{z}_i$ be the estimate of the total number of outputs for the current subproblem. Create P_1 by examining rows $(A)_{k*}$ one at a time. If the estimated number of nonzeros of $P_1 \cup (A)_{k*}$ is less than $\hat{z}/2$ then add $(A)_{k*}$ to P_1. Otherwise perform dense-vector sparse-matrix multiplication $(A)_{k*}C$ using $\tilde{\mathcal{O}}(\texttt{nnz}(C)/B)$ I/Os [60] and emit every nonzero of that product - this eliminates the row vector $(A)_{k*}$ from matrix A as all outputs generated by row $(A)_{k*}$ has now been emitted. Because of Equation (3.3) we have that the remaining rows of A can now be placed in partition P_2 and the sum of their outputs will be at most $(1+\log^{-1} U)\,\texttt{nnz}(AC)/2$. The procedure and analysis is equivalent for the case of columns. From Equation (3.5) we had that even with splits of $\texttt{nnz}(AC)(1/2+\log(U)/2)$ nonzeros then the subproblem size is the desired $\mathcal{O}(M/\log U)$ after all $\log\varphi^2$ splits are done.

In terms of I/O complexity consider first the coloring of all rows in A. First we perform the size estimates of Corollary 3.4 in $\tilde{\mathcal{O}}(N/B)$ such that we know where to split. Then we perform φ splits and each split also emits the output entries for a specific row using dense-vector sparse-matrix multiplication, hence this split takes $\tilde{\mathcal{O}}\left(\varphi N/B\right)$ I/Os. Finally for each of the φ sets of rows of A we partition columns of C in the same manner, first by invoking φ size estimations taking $\tilde{\mathcal{O}}(N/B)$ due to the sum of the nonzeros in the φ subproblems being at most N. Then for each of the φ row sets we perform φ splits and output a column from C. This step takes time $\tilde{\mathcal{O}}(\varphi N/B)$ and hence in total we use

$$\tilde{\mathcal{O}}(\varphi N/B + N/B) = \tilde{\mathcal{O}}\left(\frac{N\sqrt{\texttt{nnz}(AC)}}{B\sqrt{M}}\right).$$

■

3.2.5.5 I/O Complexity Analysis

Next, we will use Lemma 3.7 for the algorithm that shows part (b) of Theorem 3.1.

We summarize the steps taken and their cost in the external memory model.

Proof. (Theorem 3.1, part (b)) The algorithm first estimates $\texttt{nnz}(AC)$ with parameters $\varepsilon = 1/\log N$ and $\delta = 1/U$ which by Corollary 3.4 uses $\tilde{\mathcal{O}}(N/B)$ I/Os. We then perform the coloring, outputting some entries of AC and dividing the remaining entries into φ^2 balanced sets for $\varphi = \sqrt{\frac{\texttt{nnz}(AC)\log U}{M}} + \mathcal{O}(1)$. By Lemma 3.7 this uses $\tilde{\mathcal{O}}\left(\frac{N\sqrt{\texttt{nnz}(AC)}}{B\sqrt{M}}\right)$ I/Os. Finally we invoke the compressed matrix multiplication algorithm from Lemma 3.6 on each subproblem. This is possible since each subproblem has at most $\gamma M/\log U$ nonzeros entries in the output (for small constant γ). The total cost of this is $\mathcal{O}(\varphi N/B)$ I/Os, since each nonzero entry in A and C is part of at most φ products, and the cost of each product is simply the cost of scanning the input. ■

3.2.6 Lower bound

Our lower bound generalizes that of Hong and Kung [5] on the I/O complexity of dense matrix multiplication. We extend the technique of [5] while taking inspiration from lower bounds in [70–73]. A related previous work is the lower bound in [72] on the I/O complexity of triangle enumeration in a graph and machinery from [60] is used to argue that allowing comparisons of semiring elements and constants does not give any computational power.

Our lower bound holds in a parallel version of Hong and Kung [5], namely in a *parallel semiring model* where:

- Computation is done in P parallel internal memory units with a processing unit for each of them.
- A memory block holds up to B matrix entries (from the semiring), and internal memory can hold M/B memory blocks.
- Semiring elements can be multiplied and added, resulting in new semiring elements.
- Semiring elements can be compared to constants and other semiring elements.
- No other operations on semiring elements are allowed (e.g. subtraction, division)

Constants are here defined as being expressions involving semiring operations on 1 and 0. The model allows us to store sparse matrices by listing just non-zero matrix entries and their indices. We note that our algorithm respects the constraints of the semiring model and thus the model matches the one used for the upper bound I/O complexity.

We require the algorithm to work for every semiring, and in particular over fields of infinite size such as the real numbers, and for arbitrary values of nonzero entries in A

and C. Note that our model doesn't prohibit the use of randomization, i.e. the lower bound holds for randomized algorithms such as the one shown in this chapter.

Proof structure. We will in Lemma 3.12 show that a program (Definition 3.10) must use $\Omega\left(\frac{N}{PB}\min\left(\sqrt{\frac{Z}{M}}, \frac{N}{M}\right)\right)$ I/Os in the parallel semiring I/O model described above. We then apply Lemma 3.13 to argue that comparisons doesn't add computational power to this model and hence an algorithm (Definition 3.8) that uses comparisons must use at least the same number of I/Os.

Definition 3.8 (Algorithm)**.** An algorithm $\mathcal{A}$ is allowed to perform instructions: Read, write, comparison to a constant, emit, compute. In one parallel I/O an algorithm can perform P read/write operations. For an algorithm the conformation of the input defines the layout (to be e.g. column-major).

An algorithm $\mathcal{A}$ consists of a tree $T_{\mathcal{A}}$ that has tertiary decision nodes and unary operation nodes. The tree $T_{\mathcal{A}}$ can have infinitely many nodes, but for all valid inputs it must hold that:

1. A leaf node is reached.
2. On the root to leaf path `emit`(.) has been invoked on every correct output element (e.g. entries of AC) once and not otherwise.

The cost of algorithm $\mathcal{A}$ on an input is the number of read/write nodes on the root to leaf path in $T_{\mathcal{A}}$, since at every such node one parallel I/O is used.

The tertiary comparison nodes of an I/O tree are due to working with an ordered semiring, where if the semiring is unordered the comparison nodes are binary. We will use the following general definition of a computation. For the definition of computation we will as in the upper bound restrict `emit`(.) to only be invoked on values that are in internal memory.

Definition 3.9 (Computation)**.** We say that algorithm $\mathcal{A}$ computes a function on semirings $f : \mathbb{S}^d \mapsto \mathbb{S}^e$ if for all conformations of input there is a layout s.t. every for all semirings the output of function f is emitted once.

In the parallel semiring I/O model the external memory is shared between the P processors. We allow concurrent read reads from a cell but need not concurrent writes. It follows that no read/write conflicts occur. We define a program to be an algorithm without comparison nodes as follows.

Definition 3.10 (Program)**.** A program is a root to leaf path of $T_{\mathcal{A},f}$: it is a finite sequence of unary nodes that has no read/write conflicts between the P parallel processing units.

We will first argue that for every N and Z there exist matrices A and C with $\texttt{nnz}(A) + \texttt{nnz}(C) = \Theta(N)$ and $\texttt{nnz}(AC) = \Theta(Z)$, for which every execution of a program in the parallel semiring I/O model must use $\Omega\left(\frac{N}{PB}\min\left(\sqrt{\frac{Z}{M}}, \frac{N}{M}\right)\right)$ I/Os. The argument will depend crucially on the fact that we are in the semiring model: By the Schwartz-Zippel theorem [17, Theorem 7.2] we know that two polynomials agree on all inputs if and only if they are identical. Let $S \subseteq [N]$ be the set of elementary products that contribute to a nonzero output entry $\sum_{k\in S} A_{i,k}C_{k,j}$, i.e. we have that $\sum_{k\in[N]\setminus S} A_{i,k}C_{k,j} = 0$. Since we are working in the parallel semiring model, the only way to get the term $A_{i,k}C_{k,j}$ in an output polynomial is to directly multiply these input entries. That means that to compute an output entry $(AC)_{i,j}$ we need to compute a polynomial that is identical to the sum of elementary products $\sum_{k\in S} A_{i,k}C_{k,j}$. The hard instance for this lower bound is a rectangular dense matrix product, which maximizes the number of elementary products. In particular, since we ignore constant factors, we may assume that $\sqrt{Z}$ and $N/\sqrt{Z}$ are integers. Let A be a $(\sqrt{Z})$-by-$(N/\sqrt{Z})$ dense matrix, and let C be a $(N/\sqrt{Z})$-by-$(\sqrt{Z})$ dense matrix. Without loss of generality, every semiring element that is stored during the computation is either:

1. An input entry, or
2. Part of a sum that will eventually be emitted as the value of a unique nonzero element $(AC)_{i,j}$.

This is because these are the only values that can be used to compute an output entry (making use of the fact that additive and multiplicative inverses do not exist). This implies that every output entry can be traced through the computation, and it is possible to pinpoint the time in the execution where an elementary product is computed and stored in internal memory.

To upper bound the number of elementary products that can be computed in internal memory M we use the following lemma.

Lemma 3.11 ([70])**.** *In space M the number of elementary products that can be computed and stored is at most $M^{3/2}$.*

Following [72], observe that any execution of an I/O efficient algorithm can be split into *phases* of M/B parallel I/Os. For a phase that uses M/B parallel I/Os there are at most $2M$ entries that can be used by the computation of the phase: M from external memory and M that were residing in memory from the previous phase. Analogously a phase can compute at most $2M$ entries to be used in the next phases. For every phase we can therefore identify the set of at most $2M$ input and output entries that are used or computed in the phase. The lower bound will follow from bounding the number of rounds needed to compute all elementary products in the parallel setting. We are ready to show the following lemma.

Lemma 3.12. *Let A be a dimension $\alpha \times \beta$ matrix and C a dimension $\beta \times \gamma$ matrix, $\alpha, \beta, \gamma > 1$, over semiring $\mathbb{S}$.*
Then any correct program must use $\Omega\left(\frac{N}{PB}\min\left(\sqrt{\frac{Z}{M}}, \frac{N}{M}\right)\right)$ I/Os to compute AC.

Proof. To argue a lower bound for parallel Hung-Kung rounds we use Lemma 2.4 of [73]. It states that a parallel program on P processors, Φ being a potential function describing how many computations are performed ($\Phi(h)$ and $\Phi(0)$ denoting potential at the end and the beginning respectively) and $\Delta(2M)$ being an upper bound on the change in potential in a round involving $2M$ input and output entries then a lower bound is given by (3.6) below.

$$\left\lceil \frac{\Phi(h) - \Phi(0)}{P\Delta(2M)} - 1 \right\rceil \frac{M}{B} \text{ I/Os.} \tag{3.6}$$

We will pick the potential function Φ to be defined as the number of elementary products that have been computed and will be used in an output, i.e. we have

$$\Phi(h) = N\sqrt{Z}$$
$$\Phi(0) = 0$$

Next we wish to upper bound $\Delta(2M)$. We split the emit cases in two groups: direct and indirect outputs. If all values needed for emitting a particular output entry are present in a phase there may not be any storage location that can be associated with it. We first account for *direct* outputs: Each direct output requires two vectors of length $N/\sqrt{Z}$ to be stored in main memory. In each parallel phase we can store at most $M\sqrt{Z}/N$ such vectors, resulting in at most $(2M)^2 Z/N^2$ output pairs. An upper bound on the number of elementary products in a parallel phase with only direct inputs is thus $((2M)^2 Z/N^2)(N/\sqrt{Z}) = ((2M)^2\sqrt{Z})/N$.

Next, we focus on *indirect* output entries for which a partial sum is written to disk in some phase. Again we wish to upper bound the number of elementary products that can be computed. By Lemma 3.11 the number of elementary products computed and stored in internal memory of size $2M$ is at most $(2M)^{3/2}$.

We now have the lower bound due to (3.6).

$$\begin{aligned}\left\lceil \frac{\Phi(h)-\Phi(0)}{P\Delta(2M)} - 1 \right\rceil \frac{M}{B} &\geq \frac{N\sqrt{Z}-0}{P\left(\frac{M^2\sqrt{Z}}{N} + (2M)^{3/2}\right)} \frac{M}{B} \\ &\geq \frac{1}{2}\min\left(\frac{N^2}{4M^2}, \frac{N\sqrt{Z}}{(2M)^{3/2}}\right)\frac{M}{PB} \\ &= \Omega\left(\tfrac{N}{PB}\min\left(\sqrt{\tfrac{Z}{M}}, \tfrac{N}{M}\right)\right)\end{aligned}$$

■

Matrix multiplication over a semiring can be seen as a mapping $\mathbb{R}^d \mapsto \mathbb{R}^e$ between real input vector $x \in \mathbb{R}^d$ and output vector $y \in \mathbb{R}^e$, since the real numbers are a ring (and hence a semiring) and we require our bound to hold for any semiring.

It is easy to see that by the definition of matrix multiplication, the output is a polynomial over the input and hence the computation can be seen as computing a polynomial over nonzero entries of the input matrices. Due to the lower bound of programs computing the matrix product AC and the fact that the output of matrix multiplication is a polynomial over the input, we can now relate the program lower bound of Lemma 3.12 to that of an algorithm. We use the following lemma from [60].

Lemma 3.13 ([60], Lemma 9.4)**.** *For a polynomial $p : \mathbb{R}^d \mapsto \mathbb{R}^e$ let $\mathcal{A}$ be an algorithm that computes p in the parallel semiring I/O model using r I/Os. Then there exists a program that computes p in the parallel semiring I/O model using at most r I/Os.*

By the lower bound on programs from Lemma 3.12 and the fact that using comparisons give no computational power to the model as stated in Lemma 3.13 we get the bound of Theorem 3.2 in the parallel semiring I/O model with comparison against constants allowed and we note that for $P = 1$ it matches our upper bound complexity.

3.3 Fast output-sparse matrix multiplication in the RAM model

In this section of the chapter we consider computing the matrix product AC of two matrices A and C in the case where the number of nonzero entries of the output AC is sparse. The case of sparse output is well-motivated by real-world applications such as computation of covariance matrices in statistical analysis. See the introduction in Section 3.2 for more examples of applications.

The main result of this section is a new output sensitive Monte Carlo algorithm, that for number of nonzero entries Z of AC uses $\tilde{\mathcal{O}}\left(U^2\left(\frac{Z}{U}\right)^{\omega-2}+Z+N\right)$ operations in the word-RAM model and outputs the set of nonzero entries of AC with high probability. This time bound is strictly better than all state of the art methods for $Z=\omega(U)$ and is never worse than $\mathcal{O}(U^\omega)$. Additionally, our algorithm works with arbitrary cancellations according to the field.

3.3.1 Preliminaries

Let A be a $U_1 \times U_3$ matrix and C be a $U_3 \times U_2$ matrix over any field $\mathbb{F}$, then remember that $A_{i,j}$ is the entry of A located in the i'th row and j'th column and $A_{i,*}$ will be used as shorthand for the entire i'th row (likewise for column). Remember that the matrix product is given as $(AC)_{i,j}=\sum_{k=1}^{U_3} A_{i,k}C_{k,j}$ and that we say that a sum of elementary products *cancel* if the sum over the group equals zero and there are nonzero entries in the group. As we assume the matrix entries can come from any field $\mathbb{F}$ we make no assumptions about cancellation occurring or not, i.e., "fast" Strassen-like methods are allowed. We assume wlog that $\log U$ (where $U \in \{U_1, U_2, U_3\}$) is integer.

We will use the following easy fact about the number of arithmetic operations needed to multiply an $m \times n$ matrix with a $n \times p$ matrix.

Fact 3.14. Let $\mathcal{O}(n^\omega)$ be the number of arithmetic required to multiply two $n \times n$ matrices. Then an $m \times n$ matrix can be multiplied with an $n \times p$ matrix using $\mathcal{O}\left(\alpha\beta^{\omega-2}\right)$ arithmetic operations where $\beta=\min(m,n,p)$ and $\alpha = mnp/\beta$.

Proof. Assume wlog that β divides α. Since α is the smallest dimension we can divide the matrices into α/β^2 submatrices of size $\beta \times \beta$, which can each be solved in $\mathcal{O}\left(\beta^\omega\right)$ operations. ■

3.3.2 Related work

Two $U \times U$ matrices can trivially be multiplied in $\mathcal{O}(U^3)$ arithmetic operations by U^2 inner products of length U vectors. The first to improve upon the $\mathcal{O}(U^3)$ barrier was Strassen [13] who for $\omega = \log_2 7$ showed an $\mathcal{O}(U^\omega)$ algorithm by exploiting clever cancellations. Since then there has been numerous advances on ω, e.g. [58, 74, 75] and as of this writing most recently $\omega < 2.3728639$ was shown due to Le Gall [76].

The closest algorithm in spirit to our general algorithm of Theorem 3.16 is due to Williams and Yu [63]. They recursively, using time $\tilde{\mathcal{O}}\left(U^2\right)$, with high probability compute all positions of nonzero output entries and following this can then compute each output value in time $\mathcal{O}(UZ)$ for a total number of $\tilde{\mathcal{O}}\left(U^2 + UZ\right)$ operations. This matches the exact case of Paghs compressed matrix multiplication result [62], which is significantly more involved but also gives a stronger guarantee: Using time $\tilde{\mathcal{O}}\left(U^2 + Ub\right)$ it gives the exact answer with high probability if $Z > b$, otherwise it outputs a matrix where each entry is close to AC in terms of the Frobenius norm. Contrary to this we use only polylog(Z) time to compute the position of each output entry after having done an estimation output size that takes time $\tilde{\mathcal{O}}(N)$ in general and then we can partition and compute the product using time $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2}\right)$ by Theorem 3.16, which for $Z > U$ matches the above complexity of Williams-Yu and Pagh, and specifically for $Z = \omega(U)$ (any function asymptotically greater than U) we improve upon this bound.

Iwen and Spencer [77] showed that if every column of the output matrix has $\mathcal{O}(U^{0.29462})$ nonzero entries, then the matrix product can be computed using time $\mathcal{O}(U^{2+\varepsilon})$ for constant $\varepsilon > 0$. Recently due to Le Gall [76] this result now holds for output matrices with columns of at most $\mathcal{O}(U^{0.3})$ nonzeros. In this case our balanced matrix multiplication algorithm of Theorem 3.15 uses time $\tilde{\mathcal{O}}\left(U^{2.19}\right)$ (for $\omega = 2.3728639$), which is asymptotically worse, but our method applies to general balanced matrices.

For boolean input matrices, the output sensitive algorithm of Lingas [78] runs in time $\tilde{\mathcal{O}}\left(U^2 Z^{\omega/2-1}\right)$, which we improve on for $1 \leq Z < U^2$ by a relative factor of $Z^{1-\omega/2}$ and match when $Z = U^2$. Additionally our algorithm works on any field. The author however shows a partial derandomization that achieves the same bound using only $\mathcal{O}(\log^2 U)$ random bits, which is a direction not pursued in this thesis.

The general case, i.e. dense input and output, for multiplying two boolean matrices has time complexity $\mathcal{O}\left(U^3 \left(\log\log U\right)^3 / \log^3 U\right)$ due to Chans recent combinatorial algorithm [79]. His algorithm is en essence an enhancement of the old Four Russians speedup trick which works roughly as: 1) divide the matrix into small $t \times t$ blocks 2) pre-compute results of all $t \times t$ blocks and store them in dictionary. Typically $t = \mathcal{O}(\log U)$ and the gain is that we now work with $(U/t)^2 = U^2/\log^2 U$ blocks instead of U^2 cells.

In the case of sparse *input* matrices, Yuster and Zwick [80] showed how to exploit sparseness of input using an elegant and simple partitioning method. Their result was extended to be both input and output sensitive by Amossen and Pagh [14], leading to a time bound of $\tilde{\mathcal{O}}\left(N^{2/3}Z^{2/3} + N^{0.862}Z^{0.408}\right)$ based on current ω. In our (non-input sensitive) case where $N = U^2$ we are strictly better for all $U > 1$ and $Z > U$. We note that the algorithm of Amossen and Pagh is presented for boolean input and claimed to work over any ring, however both the result of Yuster-Zwick and Amossen-Pagh do not support cancellations, i.e. their bounds are in terms of the number of vector pairs of the input that have nonzero elementary products. It remains as an fundamental open problem to achieve the optimal input- and output-sensitive complexity tradeoff for matrix multiplication.

Comparison summary. The general algorithm of Theorem 3.16 works over any field (and supports cancellation), is never worse than $\mathcal{O}(U^\omega)$ and improves upon the current state of the art methods algorithms when $Z = \omega(U)$.

3.3.3 Our results

We show the following theorem, that provides an output sensitive fast matrix multiplication algorithm granted that the output is balanced.

Theorem 3.15. *Let A and C be $U \times U$ matrices over the field $\mathbb{F}$ that contain at most N nonzero entries and the product AC contains at most Z nonzero entries in total and at most $\Theta(Z/U)$ per row and column. Then there exists an algorithm for which it holds:*

(a) The algorithm uses time $\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}} + Z + N\right)$ time in the RAM model.

(b) With probability at least $1-1/U^2$ the algorithm outputs the nonzero entries of AC.

We then show the main theorem, a fast matrix multiplication algorithm that works on any input and is sensitive to the average number of nonzero entries in the rows and columns of the output.

Theorem 3.16. *Let A and C be $U \times U$ matrices over field $\mathbb{F}$ that contain at most N nonzero entries and the product AC contains at most Z nonzero entries in total. Then there exists an algorithm for which it holds:*

(a) The algorithm uses time $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + Z + N\right)$ time in the RAM model.

(b) With probability at least $1-1/U^2$ the algorithm outputs the nonzero entries of AC.

Method	Time	Notes
General dense	$\mathcal{O}\left(U^{\omega}\right)$	
Lingas	$\tilde{\mathcal{O}}\left(U^2 Z^{\omega/2-1}\right)$	Requires boolean matrices.
Iwen-Spencer, Le Gall	$\mathcal{O}\left(U^{2+\varepsilon}\right)$	Requires $\mathcal{O}\left(n^{0.3}\right)$ nonzeros per column.
Williams-Yu, Pagh	$\tilde{\mathcal{O}}\left(U^2 + UZ\right)$	
This thesis, Thm. 1	$\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}} + Z + N\right)$	Requires balanced rows and columns.
This thesis, Thm. 2	$\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + Z + N\right)$	

Table 3.1: Comparison of matrix multiplication algorithms of two $U \times U$ in the RAM model. N denotes the number of nonzeros in the input matrices, Z the number of nonzeros in the output matrix and ω is the currently lowest matrix multiplication exponent.

The algorithm of Theorem 3.15 has asymptotically lower running time compared to that of Theorem 3.16 for $1 < Z < U^2$ (for current ω) and for $Z = U^2$ they both match $\tilde{\mathcal{O}}(U^{\omega})$. However, the algorithm from Theorem 3.15 requires balanced rows and columns and in fact the algorithm from Theorem 3.16, which works in the general case, is based on calling it on balanced partitions. We note that Theorem 3.16 is restricted to square input for sake of simplicity of presentation, as the algorithm applies to the general rectangular case. We summarize the results of this section and the closest related results in the table below.

Result structure. The result is split into three parts: compressing the output, recovering output entries from compressed matrices and partitioning the input. Intuitively, the compression step of Section 3.3.4 uses that many row-column vector products of the input yields no output entry as we only have Z output entries and thus we can "‘collapse” rows and columns of the input. This makes the matrices smaller and is what gives the speedup. In the recovery step of Section 3.3.5 we show how to in polylog locate the position of a nonzero output entry and compute the entry value. The two first parts make up the algorithm of Theorem 3.15 as the compression step assumes balanced rows/columns of the output. Finally in Section 3.3.6 we show how to partition the matrix product into smaller balanced products, such that the algorithm that requires balance can be invoked - Theorem 3.16 follows from this step.

3.3.4 Compressing the output

We describe a procedure to compress the computation of a matrix product AC that in its output matrix has bounded number of nonzero entries per row and column. The overall idea is simply to collapse rows of matrix A and columns of matrix C such that the size of output matrix of the compressed matrix product is around the same size as the (upper bound of) the number of nonzero entries of AC. The high level perspective is then that we subdivide the original matrix multiplication problem into several of such balanced subproblems and invoke the method described here on each of them.

Lemma 3.17. *Let A be an $U_1 \times U_3$ matrix and C an $U_3 \times U_2$ matrix over a field $\mathbb{F}$ and let $Z = \texttt{nnz}(AC)$. Let $d_1/8, d_2/8$, where $d_1 = \Theta(Z/U_1)$, $d_2 = \Theta(Z/U_2)$, be an upper bound on the number of nonzero entries of each row and column respectively in AC. Then there exist a compression matrix G s.t. each nonzero entry of AC maps to an entry of G and*

(a) G is of size $\sqrt{U_1 d_1} \times \sqrt{U_2 d_2}$.

(b) For each entry $(AC)_{i,j}$: with probability at least $5/8$ there is a corresponding entry of G with the value $(AC)_{i,j}$.

(c) G can be computed by three matrix multiplications using time $\mathcal{O}\left(U_3 Z^{\frac{\omega-1}{2}}\right)$ in the RAM model.

Proof. The construction follows similar in spirit to that of count-min sketches [81]: we hash the entries of AC to G using two d_1- and d_2-wise independent hash functions $h_i : U_i \mapsto \sqrt{U_i d_i}$, $i \in \{1, 2\}$ where h_1 maps rows of AC to rows of G and h_2 maps columns of AC to columns of G. We will argue that a constant fraction of the entries are collision free. The intuition is simply to collapse rows of A into each other and the same for columns of C. The amount of "collapsing" done is proportional to (an upper bound of) the output size Z such that when we multiply the collapsed matrices onto each other afterwards to form our matrix G then each entry of AC is in G with constant probability.

Consider a specific entry $G_{i,j}$. We will bound the probability p_c of nothing colliding with that entry. Let p_1 denote the probability of all entries from row i not colliding with i, j. We assume without loss of generality that $U_1 < U_2$ (The converse case $U_1 \geq U_2$ follows analogously). Since h_1 is d_1-wise independent and random we have:

$$p_1 \geq \left(1 - \frac{1}{\sqrt{U_1 d_1}}\right)^{d_1/8} \geq \left(1 - \frac{1}{d_1}\right)^{d_1/8} = e^{-1/8}.$$

Equivalently, the probability p_2 of no entry from column j mapping to i, j is at most $e^{-1/8}$, where we use that there are at most $d_2/8$ nonzero entries per column. Let p_3 be the probability of an entry not from row i or column j mapping to i, j. As there are $U_1 d_1$ possible entries of matrix G and at most $U_1 d_1/8$ nonzero entries of AC we have:

$$p_3 \geq \left(1 - \frac{1}{U_1 d_1}\right)^{U_1 d_1/8} = e^{-1/8}.$$

Now, by the union bound over the three collision probabilities we have a lower bound for p_c:

$$p_c \geq 1 - ((1-p_1) + (1-p_2) + (1-p_3)) > 5/8.$$

Remember that $Z = \texttt{nnz}(AC)$. Hash functions h_1 and h_2 used for compressing can themselves be represented as matrices R and K of size $\sqrt{Z} \times U_1$ and $U_2 \times \sqrt{Z}$ respectively. However since the matices are very sparse the time spent on compressing is linear. The dominating term is the computation $(RA) \times (CK)$ which is a $\sqrt{Z} \times U_3$ matrix multiplied with a $U_3 \times \sqrt{Z}$ matrix. This can be done in time $\mathcal{O}\left(U_3 Z^{\frac{\omega-1}{2}}\right)$ using Fact 3.14.

■

We will from here denote matrices of the above form as *compressed matrices*. The following corollary states that the entries of interest, the nonzero entries of AC, can be constructed from $\mathcal{O}(\log U)$ compressed matrices G_i as above by for each entry taking the median over all matrices G_i. Note that we will not construct the compressed matrices and do a vote directly as below, but will do a "recovering" process before, which gives the positions of the nonzero entries of AC.

Corollary 3.18. *Let $G_1, \dots, G_{5\log U}$ be compressed matrices from Lemma 3.17 of AC where for each G_i we pick the hash functions h_1, h_2 independently at random. Then, with probability at least $1 - 1/U^3$, we can from $G_1, \dots, G_{5\log U}$ compute every nonzero entry of AC.*

Proof. Each entry of AC appears correctly in G_i with probability at least 5/8. By a standard application of Chernoff bounds over the median of all $5 \log U$ entries of $G_1, \dots, G_{5\log U}$ we have that the probability of there being more than $5/2 \log U$ correct entries is at least $1 - \exp(5 \log U) \geq 1 - 1/U^5$. By the union bound over at most U^2 values we get the desired error probability $1 - 1/U^3$. ■

3.3.5 Recovering nonzero entry in polylog time

We wish to with high probability recover the nonzero entries of AC from compressed matrices as in Lemma 3.17, $G_1, \dots, G_{5\log U}$, in time polylog(U) per nonzero entry after performing some preprocessing. We will make use of the following easy lemma, that shows that a vector from a given field can be summarized by a single value, such that with probability at least 1/2 we can determine if the vector has nonzero elements or not.

Lemma 3.19. *Let g be a length d vector over a field $\mathbb{F}$ and let r be a vector where each entry r_i is chosen uniformly at random from $\mathbb{F}$. Let be a mapping $v : \mathbb{F}^d \mapsto \mathbb{F}$ defined as $v(g) = r \cdot g$. Over the random choice of r it holds for $v(g)$ that:*

(a) if $v(g) \neq 0$ then $\texttt{nnz}(g) > 0$.

(b) if $v(g) = 0$ then with probability at least $1/2$ $\texttt{nnz}(g) = 0$.

Proof. Let r be a vector where each entry of r is picked independently and uniformly at random from field $\mathbb{F}$. For simplicity consider the mapping v to be a mapping to 1 or 0 as follows.

$$v(g) = \begin{cases} 1 & \text{if } r \cdot g \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Note that if $\texttt{nnz}(g) = 0$ then multiplying onto it any vector will yield 0, hence this holds for random vector r as well. For a g with $\texttt{nnz}(g) > 0$ then with probability at least $1/2$ we have $v(g) = 1$, as the probability is lower bounded by the case of one nonzero entry, say at position g_i, and the field being GF_2. Since for any $\mathbb{F}$ g has one nonzero entry $g_i \neq 0$ the random entry r_i has probability $1/|\mathbb{F}|$ to make $r \cdot g = 0$ and hence $v(g) = 0$. Thus if entry $g_i \neq 0$ then

$$r \cdot g = 0 \iff r_i = -\frac{r \setminus r_i \cdot g \setminus g_i}{g_i} = 0.$$

Since $|\text{GF}_2| = 2$ and only one of the values from the field satisfies the above, we get the claimed probability lower bound of $1/2$ over the random choice of the vector r. ■

The compressed matrices $G_1, \ldots, G_{5 \log U}$ contains the set of nonzero entries of AC, which is the goal to compute. The algorithm of the following lemma, denoted the *recovery* algorithm, will instead of doing the simple majority vote as in Corollary 3.18 use a "compressed" version that uses Lemma 3.19 to test for nonzero entries. This will allow us to compute the set of nonzero entries of AC with high probability, using $\mathcal{O}(Z\text{polylog}(U))$ operations instead of $\mathcal{O}(ZU)$ operations.

Lemma 3.20. *Let $G_1, \ldots, G_{5 \log U}$ be compressed matrices where in each G_i an entry of matrix product AC over $\mathbb{F}$ appears correctly with at least probability $5/8$, let $Z = \texttt{nnz}(AC)$ and let U be the biggest dimension of AC.*
There exists an algorithm which uses $\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}}\right)$ preprocessing time, after which in time $\tilde{\mathcal{O}}(Z)$ with probability at least $1 - 1/U^3$ the set of nonzero entries of AC is computed.

Proof. Consider the matrices $G_1, \dots, G_{5\log U}$ as in Corollary 3.18, which with probability at least $1 - 1/U^3$ can be used to create the output. We will make use of the random mapping of Lemma 3.19 and we will for the sake of simplicity of presentation assume the binary definition from Equation (3.7).

The recovering algorithm will at a high level consider, for each row of the output matrix AC, a binary tree of depth $\log U$ where each node corresponds to a group of positions of the row. A node (group) is represented by value either 1 or 0: A leaf has value 0 if the vector it represents has no nonzero entries and 1 otherwise. An internal node has value 0 if and only if all its children have value 0 and 1 otherwise. The preprocessing step constructs this tree, and afterwards the position of a nonzero entry can be found by walking the root to leaf paths of only 1s. When a position can be computed we do majority vote over the $5\log U$ compressed matrices.

Construction of the tree. The value $v(g_i)$ of either 1 or 0 is computed by Lemma 3.19. Due to the fact that the mapping v from Lemma 3.19 is just a dot product we can define a matrix P_j as follows. Let P_j be a $U_2 \times U_1$ matrix (remember that AC is a $U_1 \times U_2$ matrix) with its nonzero entries picked independently at random from $\mathbb{F}$. Consider a specific row $AC_{x,*}$ of AC. We wish to store values representing if there are nonzero entries in intervals of $AC_{x,*}$. Specifically, for matrix P_j we consider consecutive intervals of size 2^j of each row, i.e., we partition each row into $U_2/2^j$ intervals and summarize each interval by value 1 or 0 using Lemma 3.19. Matrix P_j is then defined as having random elements of $\mathbb{F}$ in positions such that ACP_j gives a matrix of summarized values. Note that P_j is linear hence we can pick the cheap order $A(CP_j)$ of computation. We repeat this for every $j \in [5\log U]$ and thus have matrices $P_1, \dots, P_{5\log U}$ and the corresponding $5\log U$ matrix products $A(CP_j)$. For each row of AC we now have values as in Lemma 3.19 for $U_2/2^j$ intervals of size 2^j. Since for each matrix P_j we have that each summary value of the matrix $A(CP_j)$ has false negatives with probability at most $1/2$ we repeat each matrix P_j and hence matrix product $A(CP_j)$ $5\log U$ times with random vectors drawn independently at random each time. Then the probability p_4 of a summary value being wrong is

$$p_4 \le 1/2^{5\log U} = 1/U^5.$$

By the union bound over all at most U^2 summary values of a P_j we have that the summary values and hence summary products $A(CP_j)$ are correct with probability at least $1 - 1/n^3$.

Extraction from the tree. For every row in $AC_{i,*}$ we now implicitly have a tree T_i, which has $\log U$ levels (with each level being repeated $5\log U$ times) and on level j we have the nodes corresponding to summary values of $A(CP_j)$. Remember that a summary value of 1 indicates that there is at least none nonzero entry in the summarized interval and 0 indicates no nonzero entries. Extracting a nonzero entry from row $AC_{i,*}$ then corresponds to following a root to leaf path in the tree where every node, corresponding to an interval of halving size, is 1. This can be done by directly accessing the summary values of $A(CP_j)$. When a path has been followed from root to leaf, the position of a nonzero entry has been located by construction of the summary values. As we now have the position of a nonzero entry, by Corollary 3.18 we can then do a majority vote over all $5 \log U$ matrices $G_1, \ldots, G_{5 \log U}$ to get with probability at least $1 - 1/U^5$ the nonzero entry on that position.

RAM Complexity. The number of matrix multiplications needed for the preprocessing, i.e., creation of the implicit tree is $\text{polylog}(U)$ since the tree has depth $\log U$ and each level of the tree is repeated $5 \log U$ times. The dimensions of the matrices is $\sqrt{Z} \times U$, since the preprocessing is done in compressed space using Lemma 3.17 and so computing a level of the tree corresponds to $\text{polylog}(U)$ repetitions of a matrix product of the form $(RA)C(P_jK)$ where R and K are size at most $U \times \sqrt{Z}$ compression matrices as in Lemma 3.17, P_j is a matrix of random field elements and AC is the matrix product of interest. Since we have dimensions at most $\sqrt{Z} \times U$ on the matrices this can be done in time $\mathcal{O}\left(UZ^{(\omega-1)/2}\right)$ using Fact 3.14. Hence in total we use $\mathcal{O}\left(UZ^{(\omega-1)/2} \log^2 U\right) = \tilde{\mathcal{O}}\left(UZ^{(\omega-1)/2}\right)$ RAM operations used for the preprocessing. To perform the extraction of one nonzero entry of AC we traverse the tree of depth $\log U$, for each level check the values of all $5 \log U$ repetitions of the level. Every such check is a constant time direct access and hence the extraction is $\text{polylog}(U)$ per position containing a nonzero element. Finally we do the majority vote over the $5 \log U$ positions in the compressed matrices, yielding in total $\text{polylog}(U)$ time per nonzero entry from AC. ■

We have the two ingredients for Theorem 3.15 since we can compress the output for the balanced case and afterwards extract the nonzero entries of AC.

Proof. (Theorem 3.15) We consider first the RAM complexity. For a square matrix the compression step of Lemma 3.17 takes time $\tilde{\mathcal{O}}\left(UZ^{\frac{\omega-1}{2}}\right)$ to get $5 \log U$ compressed matrices. After this the recovery step from Lemma 3.20 uses time $\tilde{\mathcal{O}}\left(UZ^{(\omega-1)/2} + Z\right)$ and we need an additional $\mathcal{O}(N)$ to read the input.

The probability bound holds trivially since by a union bound over error probabilities of compression and recovery we have that we compute the nonzero entries of AC with probability at least $1 - (2/U^3) > 1 - 1/U^2$ as stated by the theorem.

■

3.3.6 Subdivision

By Theorem 3.15 we have that a matrix product with Z nonzero entries distributed evenly in the output can be computed in time $\tilde{\mathcal{O}}(UZ^{(\omega-1)/2} + Z)$. We will now show that an arbitrary matrix product can be divided into subproblems of that form.

We shall use the following algorithm that given arbitrary matrices can output an estimate of the number of nonzeros of each column (or row) of the matrix product. We will later use this estimate to perform subdivisions of the input.

Fact 3.21 (word-RAM part of Lemma 3.3). Let A and C be $U \times U$ matrices with entries of field $\mathbb{F}$, $N = \texttt{nnz}(A) + \texttt{nnz}(C)$ and let $0 < \varepsilon, \delta \leq 1$.
We can compute estimates $z_1, \ldots, z_U$ using $\mathcal{O}(\varepsilon^{-3} N \log(U/\delta) \log U)$ RAM operations such that with probability at least $1 - \delta$ it holds that $(1-\varepsilon)\,\texttt{nnz}([AC]_{*k}) \leq z_k \leq (1+\varepsilon)\,\texttt{nnz}([AC]_{*k})$ for all $1 \leq k \leq U$.

We note that Fact 3.21 makes no assumptions about cancellation and is applicable for matrix products over arbitrary fields. For certain matrices it is possible to perform size estimation faster, which can then yield a faster total running time for the methods presented here. Consider $U_1 \times U$ matrix A and $U \times U_2$ matrix C, where the N nonzero locations are random and independent: The expected value of $\texttt{nnz}(AC)$ is N/U and the expected row and column size is $N/(U_1 U)$ and $N/(U_2 U)$, hence an upper bound can be computed in $\text{polylog}(\max(U_1, U, U_2))$ time that holds with high probability.

Lemma 3.22. *Let AC be a matrix product of two $U_1 \times U_3$ and $U_3 \times U_2$ matrices over field $\mathbb{F}$. Let $Z = \texttt{nnz}(AC)$ and $U = \max(U_1, U_2, U_3)$.*
There is a division procedure that partitions the matrix product into $\text{polylog}(U)$ *smaller matrix products where with probability at least $1 - 1/U^5$ it holds:*

(a) A specific matrix product with Z' nonzero entries in its output has d_1 and d_2 number of nonzeros in rows and columns in its output and further $d_1 = \Theta(Z'/U)$ and $d_2 = \Theta(Z'/U)$.

(b) The partitioning takes time $\tilde{\mathcal{O}}(N + U)$ to compute.

Proof. We will argue how to perform a partition of the input such that the conditions of the lemma is satisfied. The partitioning will be done by invoking the size estimation algorithm of Fact 3.21 with parameters $\varepsilon = \mathcal{O}(\log U)$ and $\delta = 1/U^5$. We now have access to U_1 values $z_1, \ldots, z_{U_1}$ where it holds that z_k is an $\log U$-estimate of the number of nonzero entries of row k. We then sort the output rows by their z-values and divide the rows into $\log U$ groups where it holds that group i consists of $s_i = \frac{1}{2^i} \sum_j z_j$ nonzero entries for $i \in [\log U]$. By error bounds of the size estimation [7] we have that for group size s_i it holds that $(1-\log U)\frac{1}{2^i}\texttt{nnz}(AC) \leq s_i \leq (1+\log U)\frac{1}{2^i}\texttt{nnz}(AC)$. We now have $\log U$ disjoint subproblems that are approximately balanced, i.e. the heaviest output rows are grouped together. The guarantees of at most d_1 entries per row and d_2 per column can with probability at least $1-1/U^5$ be read directly from the z_i values. The time complexity of computing this subdivision is the time complexity of size estimation plus sorting, which in general is $\tilde{\mathcal{O}}(N)+\mathcal{O}(U\log U)=\tilde{\mathcal{O}}(N+U)$. ■

We now show the main theorem, which gives the time bound for arbitrary input matrices over any field.

Proof. (Theorem 3.16) From Lemma 3.22 we have a subdivision into $\log U$ dense subproblems on which the compression and aggregation algorithms of Lemmas 3.17 and 3.20 will be invoked on. Let x_i and y_i be the dimensions of the output of subproblem i. If $x_i < y_i$, then by Fact 3.14 we can compute problem i using $\mathcal{O}\left(Uy_ix_i^{\omega-2}\right)$ arithmetic operations. However since the dimension varies in size the total cost of this step is upper bounded by

$$U\left(\sum_{i=1}^{\log U} x_iy_i^{\omega-2}+\sum_{i=1}^{\log U} x_i^{\omega-2}y_i\right). \tag{3.8}$$

To bound 3.8 note that we have $\sum_i x_i = U$ and $\sum_i x_iy_i \leq 2Z$. We relax the last inequality and assume $x_iy_i = Z$ for every i. It follows that $y_i = U/2^i$ and $x_i = (Z/U)2^i$ and that 3.9 below is an upper bound for 3.8.

$$U\left(\sum_{i=1}^{\log U} \frac{Z}{U}2^i(U/2^i)^{\omega-2}+\sum_{i=1}^{\log U} (\frac{Z}{U}2^i)^{\omega-2}U/2^i\right). \tag{3.9}$$

We will bound 3.8 by the sum of the terms in the limit values for 2^i, namely 1 and U, this will provide and upper bound for the equation up to logarithmic factors. By insertion in 3.9 and by using that there can be at most U nonzero entries in a row/column we get an upper bound of

$$U\left(ZU^{\omega-3}+U+U(Z/U)^{\omega-2}+Z^{\omega-2}\right).$$

The third term dominates asymptotically and hence an upper bound on the total cost is given by

$$\tilde{\mathcal{O}}\left(U^2\left(\frac{Z}{U}\right)^{\omega-2}\right).$$

The error probability follows from a standard union bound argument. ■

Chapter 4

Association Rule Mining using Maximum Entropy

4.1 Introduction

Recommender systems that try to assess probabilities, e.g. for estimating probabilities based on the context of a particular user, may be faced with ambiguous statistical evidence. For example, consider the task: *Customer is known to belong to categories S_1 and S_2, each of which is known to increase the probability of buying product S_3 by* 50%*, how do we estimate the probability that she will buy product S_3? Is it increased by* 50%*,* 100%*, or perhaps* 125%*?*

Of course we *may* have enough data on S_1, S_2, and S_3 to make this assessment by computing the observed probability. But for rare terms or products there may not be enough data to directly produce such an estimate — perhaps we never directly observed a connection between S_1, S_2, and S_3. In the extreme case, what can we do when there is *no support* for estimating the probability by simply computing the observed frequency? Most likely, even the number of observations of proper subsets of S_1, S_2, and S_3 will then be small enough that there is nonnegligible uncertainty about the pairwise correlations.

The difficulty of estimating probabilities of events occurring clearly depends on the distribution of the input, and on how much information we have about this distribution. So rather than a classical approach that considers worst-case data, we should consider ideas from statistical analysis. The Maximum Entropy (maxent) Model is a method of statistical inference that based on partial knowledge of a distribution provides a *maximum entropy estimate.* Informally, it provides a probability prediction based on the distribution that has "the least bias possible" based on the given observations. In this chapter we consider the use of maximum entropy estimates in information retrieval contexts where estimates are sought of the probability that an item or term is of interest to a user.

4.1.1 Motivating examples

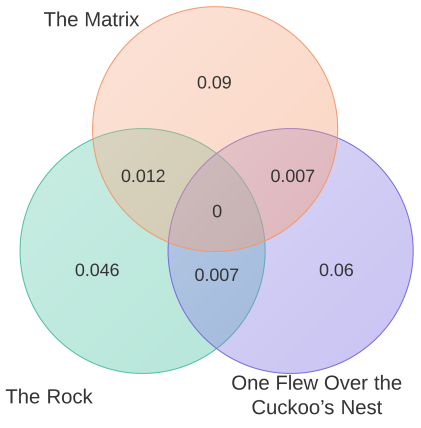
Movie recommendation. Suppose you know that a user loved "The Rock" and "The Matrix". What is the probability that he already saw and gave 5 stars to "One Flew Over the Cuckoo's Nest"? We will try to answer this question using the smallest possible sample of the MovieLens data set, which is examined further in Section 4.3. The difficulty is that only about 1 in 1000 people has seen all three movies and given them 5 stars. This means that to get a statistically significant answer (in absence of other information) we need to ask a very large set of people. Obviously, the less mainstream films you consider, the bigger this problem will become. See Figure 4.1 for visualization of the setting.

However, it is considerably easier to obtain information on pairs of movies. About 2.5% of people will have seen at least two of these movies and given them 5 stars. This means that we can reliably estimate conditional probabilities based on significantly less data. Using the information that your user loved "The Rock" gives a probability of about 11% that he loves "One Flew Over the Cuckoo's Nest". However, if we instead use the information that he loved "The Matrix" we get an estimate of about 7%. It seems that both these movies make it more likely that he will love "One Flew Over the Cuckoo's Nest", but how do we combine these pieces of information? It seems that anywhere in the range 11-18% is a reasonable guess.

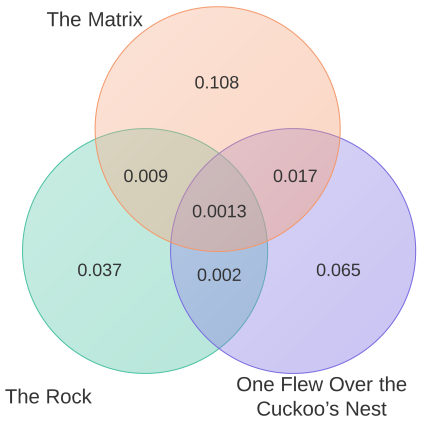
To resolve this ambiguity we again use a maximum entropy estimate based on subset frequencies. This estimate takes the correlation of "The Rock" and "The Matrix" into account, and arrives at an estimate of 14% based on a data set of 673 people in which nobody has given all three movies 5 stars. When we consider a data set 100 times larger it is possible to see how well this estimate fares: In the larger data set, 15% of those who gave 5 stars to "The Rock" and "The Matrix" also gave 5 stars to "One Flew Over the Cuckoo's Nest". We generally find that maximum entropy estimates are surprisingly accurate across a wide range of data sets from different areas. $\triangle$

Query completion. Consider the case where a search engine user types the words "`jordan air`" (followed by a space). What words should be suggested to complete the query?

We consider the simple method of ignoring the order of words and relying on association rules, in our case obtained from a set of 2.1 million queries from a major US search engine.



(A) Venn diagram of distribution based on the sample.



(B) Venn diagram of distribution for the whole data set.

FIGURE 4.1: Two distributions of movie watchers who love three selected movies. Figure 4.1a shows the observed probabilities in a sample of 1% of the data set, from which we want to approximate $S_1 \cap S_2 \cap S_3$. This is consistent with S_1, S_2, and S_3 never occurring together. Figure 4.1b shows the probabilities in the whole data set, and indeed it is not the case that loving two of the movies precludes loving the third one. Our findings are that a maximum entropy estimate in such a case is well-concentrated as opposed to independence or extrapolation estimators. In this example an independence assumption yields an estimate of 36 occurrences, our maxent estimate yields 82, while the true number of occurrences is 90.

Suppose we have two competing suggestions, "`force`" and "`wholesale`" (occurring in around 0.03% and 0.06% of queries, respectively). Around 9% of the queries that contain the word "`air`" also contain the word "`force`". On the other hand, less than 0.00003% of past queries contain "`jordan`" *and* "`force`", which means that the maximum entropy estimate for the probability of completing with "`force`" becomes less than 1%. For comparison, both "`air`" and "`jordan`" significantly increase the probability that the word "`wholesale`" occurs, to around 0.3% and 1%, respectively. Figure 4.2 summarizes the association rules involving two words.

If the combination of words had been just slightly more rare, we might have had no past queries containing them. Thus, for "long tail" queries we need to rely on other methods for estimating the likelihood of a particular completion.

A maximum entropy estimate, or more precisely the approximation formula of (4.1), shows that the user has a 5% probability of completing with "`wholesale`". This is

Assoc. rule	**Confidence**
`air => force`	0.091
`jordan => wholesale`	0.010
`air => wholesale`	0.0030
`jordan => force`	0.00097

Figure 4.2: Association rules for the words `force` and `wholesale` in the query set `jordan` and `air`, respectively. The probabilities are sorted in decreasing order. In the whole data set, `wholesale` occurs in a fraction 0.00064 of queries, and `force` in a fraction 0.00028. By the first association rule, presence of the word `air` increases the probability of the word `force` hundreds of times, to over 9%. However, a maximum entropy estimate correctly predicts less than 1% probability of seeing `force` when both `air` and `jordan` are present. On the other hand, the presence of `air` and `jordan` yield a maximum entropy probability for `wholesale` of 3%, close to the observed frequency of 5%.

consistent with the data, which contains 31 queries including {`jordan`, `air`, `wholesale`} out of a total of 575 queries containing "`jordan`" and "`air`".

We will argue by experimental evidence that in scenarios such as the one above, the maximum entropy estimate will give a better prediction than both extrapolation and the independence model (see Section 4.2.2 for definitions), while still being efficiently computable.

4.1.2 Our results

Problem definition. We consider the problem of estimating probabilities of conjunctions of boolean random variables, where each such conjunction occurs a statistically insignificant number of times in a data set of samples from the joint distribution (e.g., given by a complete data set). For some big data set $\mathcal{D}$ we consider a sample $D \subset \mathcal{D}$. Given such D we wish to estimate event frequencies of $\mathcal{D}$ also in the difficult cases where the events do not occur in D. In particular we will focus on triples: Let I be the set of possible items, $|I| = n$, and D be an $m \times n$ binary data set where each of the m rows D_i encodes a transaction $D_i \subseteq I$. For all singleton- and pair-subsets of I we assume that we know the number of transactions which they occur in, i.e., all singleton and pair frequencies are known. For each $X \subset I$, $|X| = 3$ where the frequency θ_X in D is 0 we then wish to estimate θ_X in $\mathcal{D}$.

Our contribution. We consider triple frequency estimation based on the principle of maximum entropy. Our main theoretical result is that a maximum entropy estimate based on a sample, which implies that the frequencies used as input to the estimator will have some relative error ε, will yield an estimate close to the true triple frequency under the maximum entropy assumption. We show this through a surprisingly simple estimator

$\tilde{p}$ that approximates the maximum entropy estimate well when triple frequencies are small. For a boolean random X we let $\overline{X}$ denote the event $X = 0$.

Theorem 4.1. *Consider the scenario where X, Y, W are boolean random variables and $\mathbf{Pr}(X)$, $\mathbf{Pr}(Y)$, $\mathbf{Pr}(W)$, $\mathbf{Pr}(XY)$, $\mathbf{Pr}(XW)$ and $\mathbf{Pr}(YW)$ are given. Assume that the maximum entropy distribution consistent with these probabilities satisfies*

$$\begin{aligned}\mathbf{Pr}(XYW) \leq \varepsilon \min(&\mathbf{Pr}(\overline{X}YW), \mathbf{Pr}(X\overline{Y}W), \mathbf{Pr}(XY\overline{W}), \mathbf{Pr}(\overline{XY}W), \\ &\mathbf{Pr}(\overline{X}Y\overline{W}), \mathbf{Pr}(X\overline{YW}), \mathbf{Pr}(\overline{XYW}))\ .\end{aligned}$$

Then given probability estimates p... such that

$$\begin{aligned}&\frac{p_{\overline{X}YW}}{\mathbf{Pr}(\overline{X}YW)}, \frac{p_{X\overline{Y}W}}{\mathbf{Pr}(X\overline{Y}W)}, \frac{p_{XY\overline{W}}}{\mathbf{Pr}(XY\overline{W})}, \frac{p_{\overline{XY}W}}{\mathbf{Pr}(\overline{XY}W)}, \frac{p_{\overline{X}Y\overline{W}}}{\mathbf{Pr}(\overline{X}Y\overline{W})}, \frac{p_{X\overline{YW}}}{\mathbf{Pr}(X\overline{YW})}, \frac{p_{\overline{XYW}}}{\mathbf{Pr}(\overline{XYW})} \\ &\in [(1-\varepsilon), (1+\varepsilon)],\end{aligned}$$

it holds that

$$\begin{aligned}\tilde{p} = &\frac{p_{XY\overline{W}} p_{X\overline{Y}W} p_{\overline{X}YW} p_{\overline{XYW}}}{p_{X\overline{YW}} p_{\overline{X}Y\overline{W}} p_{\overline{XY}W}} \\ &\in [(1-\mathcal{O}(\varepsilon))\mathbf{Pr}(XYW), (1+\mathcal{O}(\varepsilon))\mathbf{Pr}(XYW)]\end{aligned}$$

It follows from Theorem 4.1 that a) using sampled data to perform maximum entropy estimates of probabilities in the bigger data is theoretically well-founded b) there is a simple explicit estimator, $\tilde{p}$, that approximates the maximum entropy estimate in the interesting case where the triple frequency is significantly smaller than the pair frequencies.

It is instructive to consider a less precise, but even simpler estimator for the case where, informally, there is no strong positive correlation among X, Y, and W, and $\mathbf{Pr}(\overline{XYW})$ is close to 1. Then $p_{XY\overline{W}}/p_{p_{X\overline{YW}}} \approx p_{Y|X}$, the observed probability of Y given X, and similarly $p_{X\overline{Y}W}/p_{p_{\overline{XY}W}} \approx p_{X|W}$ and $p_{\overline{X}YW}/p_{p_{\overline{X}Y\overline{W}}} \approx p_{W|Y}$, so we can approximate the triple frequency by:

$$p^* = p_{Y|X} p_{X|W} p_{W|Y} \tag{4.1}$$

Applying (4.1) to estimate $\mathbf{Pr}(W|XY)$, we get the estimator

$$p^{\#} = p_{Y|X} p_{X|W} p_{W|Y} / p_{XY} = p_{W|X} p_{W|Y} / p_W;$$

that is, the factors by which conditioning on X and Y influence the probability of W get multiplied.

Empirical study. Our experimental evaluation on real data sets shows that maximum entropy estimates give meaningful, and often quite precise, frequency predictions also in cases where the independence estimate θ^i_X and the extrapolation estimate θ^e_X do not. The error of the estimator is well modeled by the assumption that transactions are independently sampled from a distribution having the estimated subset probabilities.

Result structure. In Section 4.2.1 we introduce basic terms and notation followed by a description in Section 4.2.3 of how the maximum entropy estimate of an item set is computed. We then prove in Section 4.2.3.2 that the maxent estimate is not too sensitive to error on the input distribution. For the experimental evaluation we first show in Section 4.3.1 that the maximum entropy estimate achieves better concentration in general and then in Section 4.3.2 we discuss results on item sets of low statistically insignificant support.

4.1.3 Related work

The principle of maximum entropy dates far back, but was introduced to information theory in a seminal work by E. T. Jaynes [82]. It has since seen applications in a large number of areas.

The maximum entropy distribution of n random variables is known to be computable in time exponential in n using the well-known Iterative Scaling algorithm [83]. The running time is due to the fact that for n variables, there are 2^n subsets of variables. In the general case, that is with no knowledge of the distribution, Tatti has proved that querying the model is PP-hard [84], which is (believed to be) harder than NP.

Association rule mining [85, 86] is a well-known and extensively studied problem, where a rule has the form $X \implies Y$ with X, Y being disjoint subsets of random variables. In transactional data sets association rule mining traditionally relies on finding *frequent itemsets*, i.e., for some set of items I and a set of transactions D over I then one wishes to report back the sets $X \subseteq I$ that are contained in more than s transactions, for a fixed threshold s.

The maxent distribution has been used as a model to measure how significant an itemset is, in the framework of frequent itemset mining, e.g. [87, 88]. The general approach is to compute the maximum entropy distribution (via the Iterative Scaling algorithm) and then compute the Kullback-Leibler divergence with the empirical distribution, from which a p-value can be found that is used to rank the item sets. Our approach is that we observe some sample of the subsets of the set of interest and then use these subsets to efficiently compute the maxent estimate.

For the case where all frequencies of the strict subsets are known, the maximum entropy model has been used by R. Meo [89] by comparing the probability estimate under maximum entropy to the empirical probability in order to achieve a measure of ranking an itemset. One of the main open problems of [89] was determining the existence of a closed form formula for a maxent estimate given the subset probabilities. This was partially resolved in [90], where the author provides a formula for the 1-dimensional search space in which the maxent estimate lies, which is then traversed by binary search that is shown to converge to the maxent estimate.

Comparisons. Our estimator takes as parameter all single and pair-frequencies. The hypothesis present implicitly in our model is that data generally has weak third-order dependencies. This is also the reason Chow-Liu trees[91], which model only first and second-order dependencies, are known to be a good approximations of many observed distributions. The maximum entropy estimate used can be seen as an application of [89], where singleton and pair frequencies are used to efficiently compute a maxent estimate in the interesting case where the estimand has no observed occurrences. One of the main open problems of [89] is an explicit formula for the maxent estimate - Theorem 4.1 in this chapter shows an explicit formula for an approximation of the maxent estimate under certain conditions. As the $1D$ search of the maximum entropy estimate θ_X^m is determined when having all subset frequencies of X, we can compute a good maximum entropy estimate using a small constant number of iterations of binary search as opposed to computing the full maximum entropy distribution. We give a proof of this that is similar to that of Meo [90], with the distinction that where she shows that there exists constants such that the maxent estimate can be classified by setting particular equations to be equal to each other, in our proof the constants are explicitly stated.

4.2 Frequency estimates of itemsets

4.2.1 Preliminaries

We provide some definitions and notation that will be used throughout the chapter.

Remember that a boolean random variable X is a variable with values in $\{0, 1\}$. For a boolean random variable let X let $\overline{X}$ be the event of $X = 0$.

A binary data set D of observations of boolean random variables is an $m \times n$ matrix consisting of m binary n-sized vectors, index $0 \leq i \leq n-1$ corresponding to the outcome of boolean random variable X_i. For a particular subset of the boolean variables $S \subseteq [n]$ a binary vector ω *covers* S iff $X_i = 1$ implies ω_i for every $i \in S$. The *frequency* θ_S of a set of boolean variables is the proportion of the m row vectors in D that covers S. A *distribution* p over data D is mapping $p : \{0,1\}^n \mapsto [0,1]$ s.t. $\sum_{\omega \in \{0,1\}^n} p(\omega) = 1$. For a distribution p and a vector of 1s v we denote by $p(S = v) = p(S = 1) = \theta_S$ the probability $\mathbf{Pr}(\omega \text{ covers } S)$.

The *empirical distribution* over data set D is given as

$$q_D(a_1 = v_1, \ldots, a_n = v_n) = \frac{|\{t \in D | t = v\}|}{m} \tag{4.2}$$

We will denote by *empirical frequency* the frequency according to the empirical distribution

Let a family of random variable sets $\mathcal{F}$ be *satisfied* by a distribution $p : \{0,1\}^n \mapsto [0,1]$ iff for every $S \in \mathcal{F}$ it holds that $p(\omega \text{ covers } S) = \theta_S$.

Finally, we say that a set of binary variables X is *downard closed* if for all strict subsets $S \subset X$ we have θ_S, e.g., if we consider the triple of items $s = \{I_1, I_2, I_3\}$ then s is downward closed if we know the empirical singleton frequencies $\theta_{I_1}, \theta_{I_2}, \theta_{I_3}$ and empirical pair frequencies $\theta_{\{I_1,I_2\}}, \theta_{\{I_2,I_3\}}, \theta_{\{I_1,I_3\}}$.

We consider specifically the case where all itemsets of size 3 (triples) are downward closed and the triples are the itemsets which we wish to estimate the frequency of.

4.2.2 Estimation by extrapolation and independence assumption

We briefly describe the two estimators used for comparison. Let $X = \{I_1, I_2, I_3\} \subset I$, $|X| = 3$, be the triple of interest from sampled data set $D \subset \mathcal{D}$. The independence model assumes the occurrences of random variables to not be correlated, thus we have:

$$\theta^i_X = \theta_{I_1}\theta_{I_2}\theta_{I_3}$$

For the extrapolation estimator, let $occ(X)$ be the number of occurrences of X in D. To estimate the frequency θ_X in $\mathcal{D}$ we have:

$$\theta^e_X = occ(X)/|D|$$

Following from Chernoff bounds on independent variables θ^e_X is known to be a good estimate when θ_X is significant in D.

4.2.3 Maximum entropy of itemsets

We are interested in estimating the frequency of a specific set of items $G \subseteq I$. We will estimate such a frequency using the maximum entropy distribution, which intuitively can be thought of as the most uniform distribution given some observed frequencies. We will compute the needed entries of the maximum entropy distribution to be used to compare to the empirical distribution.

For a family of itemsets $\mathcal{F}$ and a variable set G let the projected family $\mathcal{F}_G$ be defined as

$$\mathcal{F}_G = \{X \in \mathcal{F} | X \subset G, X \neq \emptyset\}.$$

Then letting $\mathcal{P}$ be the set of all possible probability distributions that satisfy $\mathcal{F}_G$, the entropy of a distribution p is given as

$$H(p) = -\sum_{\omega \in \{0,1\}^{|G|}} p(\omega) \log p(\omega) \tag{4.3}$$

The maximum entropy distribution $p*$ can be found by maximizing $H(p)$ over all $p \in \mathcal{P}$

$$p^* = \arg\max_{p \in \mathcal{P}} H(p) \tag{4.4}$$

We note that $|\mathcal{F}_G| = \mathcal{O}\left(2^{|G|}\right)$ and if $\mathcal{F}_G = \emptyset$ then p^* is the uniform distribution. The set $\mathcal{P}$ contains the empirical distribution (Equation (4.2)) and hence is non-empty by construction. We shall denote by *maximum entropy estimate* θ_X^m of an itemset X the frequency of the itemset according to the maximum entropy distribution.

4.2.3.1 Classifying and computing the maxent estimate

For completeness we will state the approach used to compute the maximum entropy estimate of itemset frequency. We note that a similar proof of how to find the maxent estimate appears in [90], however we give explicit constants in Equation (4.14) whereas their proof shows existence of the constants.

High-level description. The overview of the proof is that for any $z > 1$ variables, when $\{\theta_X \mid X \subset G\}$ is given for each variable then the subspace of the joint probability space of z random boolean variables is 1-dimensional. It follows from this that the frequency estimate according to the maximum entropy distribution p^* is located on a line segment. The estimate thus be computed by doing a simple binary search on this line segment and the main point of doing this is that we avoid computing the entire maximum entropy distribution.

Given z random boolean variables and the marginal probabilities $\{\theta_X \mid X \subset G\}$ then the joint probability space is given by 2^z linear equations, where the rank of the corresponding matrix is $2^z - 1$. Let the z boolean random variables $Y_1, \ldots, Y_z$ have probabilities $\mathbf{Pr}[Y_1], \ldots, \mathbf{Pr}[Y_z]$. For $x \subseteq [z]$, $y = [z] \setminus x$ then denote by $f_x^=$ the probability $\mathbf{Pr}\left[\forall_{i \in x} Y_i = 1 \wedge \forall_{j \in y} Y_j = 0\right]$. and by θ_x the probability $\mathbf{Pr}\left[\forall_{i \in x} Y_i = 1\right]$.

Lemma 4.2. *For random boolean variables $Y_1, \ldots, Y_z$ with feasible marginal probabilities $\{\theta_X \mid X \subset [z]\}$ then the space of feasible distributions is determined entirely by $f_{[z]}^=$. More generally, for $S \subseteq [z]$ then $f_S^=$ follows from Equation* (4.5)*.*

$$f_S^= = \sum_{S' \supseteq S} (-1)^{|S' \setminus S|} \theta_{S'} \tag{4.5}$$

Proof. We prove this by induction on $|S|$. For $|S| = i$ we have

$$f_S^= = \sum_{S' \supseteq S} (-1)^{|S' \setminus S|} \theta_{S'}$$

For the inductive step we assume Equation (4.5) to hold for $|S| > i$. Then for $|S| = i$, Equation (4.6) holds as the sum is over supersets of S. By applying Equation (4.5) to the second term of the right hand side of Equation (4.6) we get Equation (4.7).

$$f_S^= = \theta_S - \sum_{S' \supset S} f_{S'}^= \tag{4.6}$$

$$f_S^= = \theta_S - \sum_{S' \supset S} \sum_{S'' \supseteq S'} (-1)^{|S'' \setminus S'|} \theta_{S''} \tag{4.7}$$

The double sum of Equation (4.7) can be split into two as shown in Equation (4.8)

$$f_S^= = \theta_S - \left(\sum_{S' \supseteq S} \sum_{S'' \supseteq S'} (-1)^{|S'' \setminus S'|} \theta_{S''} - \sum_{S' = S} \sum_{S'' \supseteq S'} (-1)^{|S'' \setminus S'|} \theta_{S''} \right) \tag{4.8}$$

The first double sum can be split into two parts

$$\sum_{S' \supset S} \sum_{S'' \supset S'} (-1)^{|S'' \setminus S'|} \theta_{S''} + \sum_{S' = S} \sum_{S'' = S'} (-1)^{|S'' \setminus S'|} \theta_{S''}$$

the first for which we will use Fact 4.3 below.

Fact 4.3. For a set space X and function $g : X \to \mathbb{R}$, for a double sum of sign-alternating supersets of any $x \in X$ we have

$$\sum_{x' \supset x} \sum_{x'' \supset x'} (-1)^{|x'' \setminus x'|} g(x) = 0$$

due to summands canceling out.

Then by application of Fact 4.3 to Equation (4.8) we get Equation (4.9), where the first double sum consists only of the element θ_S, hence we now arrive at the induction basis in Equation (4.10).

$$f_S^{=} = \theta_S - \left(\sum_{S'=S} \sum_{S''=S'} (-1)^{|S'' \setminus S'|} \theta_{S''} - \sum_{S'=S} \sum_{S'' \supseteq S'} (-1)^{|S'' \setminus S'|} \theta_{S''} \right) \tag{4.9}$$

$$f_S^{=} = \theta_S - \left(\theta_S - \sum_{S'=S} \sum_{S'' \supseteq S'} (-1)^{|S'' \setminus S'|} \theta_{S''} \right)$$

$$f_S^{=} = \sum_{S'' \supseteq S} (-1)^{|S'' \setminus S|} \theta_{S''} \tag{4.10}$$

■

Intuitively, Lemma 4.2 states that the space of probability distributions that satisfy the marginal probabilities can be traversed by varying $f_{[z]}^{=}$. We note that Lemma 4.2 was shown earlier (using a slightly different argument) by Calders & Goethals [92, eq. (1)] and that we include the proof for sake of completeness.

We shall now argue that on this line through 2^z dimensional space, there is a unique point, i.e. a unique feasible distribution p, that maximizes the entropy $H(p)$ and thus computing this point allows us to query the maximum entropy distribution p^* for a z-sized set of variables. Given a feasible distribution x consisting of entries $x_S \geq 0$ for every $S \subseteq [z]$, then by Lemma 4.2 the feasible distribution space $\mathcal{P}_f$ can be traversed by Equation (4.11).

$$\mathcal{P}_f = x + t \cdot v, t \in (l, r) \tag{4.11}$$

where v is a 2^z-sized $\{-1, +1\}$-vector with entries $v_S = (-1)^{z-|S|}$ for every $S \subseteq [z]$ and the range (l, r) is the range for which all coordinates in the vector $x + t \cdot v$ are non-negative. Consider the partitioning of subsets $S \subseteq [z]$ into $S_{\text{even}} = \{S \mid (z - |S|) \text{ is even}\}$ and $S_{\text{odd}} = \{S \mid (z - |S|) \text{ is odd}\}$. Then the t-value borders are given below.

$$l = \max\{x_S \mid S \in S_{\text{even}}\}$$
$$r = \min\{x_S \mid S \in S_{\text{odd}}\}$$

We note that feasible solution x exists by construction since we have observed a feasible distribution and that vector v corresponds to the null space of the 2^z-row matrix denoting the linear equalities which the joint distribution adheres to.

Location of the unique point corresponding to querying the max entropy distribution p^* is shown in Lemma 4.4 below.

Lemma 4.4. *For a feasible distribution space $\mathcal{P}_f = x + t \cdot v, t \in (l, r)$ there exists a point $p_{max} \in \mathcal{P}_f$ s.t. $\forall p \in \mathcal{P}_f, p \neq p_{max} \implies H(p_{max}) \geq H(p)$. In particular, Equation (4.13) below holds.*

$$\frac{\mathrm{d}}{\mathrm{d}t} H(x + t \cdot v) = \frac{\mathrm{d}}{\mathrm{d}t} \left(\sum_{S \subseteq [z]} (x_S + (-1)^{z-|S|} t) \log \left(\frac{1}{x_S + (-1)^{z-|S|} t} \right) \right) \tag{4.12}$$

$$= \sum_{S \subseteq [z]} (-1)^{z-|S|} \log \left(\frac{1}{x_S + (-1)^{z-|S|} t} \right) \tag{4.13}$$

Proof. We wish to show the existence of the scalar t_{max} that optimizes the entropy, i.e. $p_{max} = x + t_{max} \cdot v$. Equation (4.13) follows form standard derivative rules, from which we arrive at

$$\frac{\mathrm{d}}{\mathrm{d}t} H(x + t \cdot v) = \sum_{S \subseteq [z]} (-1)^{z-|S|} \left(\log \left(\frac{1}{x_S + (-1)^{z-|S|} t} \right) + \frac{\left(x_S + (-1)^{z-|S|} t\right)^2}{\left(x_S + (-1)^{z-|S|} t\right)^2} \right)$$

where the rightmost term will cancel out due to there being an equal number of odd- and even-sized subsets of $[z]$ for any integer z. ■

Corollary 4.5. *The value $t_f \in (l, r)$ that maximizes $H(x + t \cdot v)$ while $x + t \cdot v$ is a feasible solution can be found as $t_f = \text{median}\{l, r, t_{max}\}$.*

Proof. From Lemma 4.4 we have that t_{max} is the t-value that maximizes the entropy function $H(x + t \cdot v)$, which is at its unique maximum when Equation (4.14) holds.

$$\sum_{S \in S_{\text{odd}}} \log \left(\frac{1}{x_S - t} \right) = \sum_{S \in S_{\text{even}}} \log \left(\frac{1}{x_S + t} \right) \tag{4.14}$$

Let $t_f = \text{median}\{l, r, t_{max}\}$. For the line segment spanned by end points l and r, we have 3 cases: $t_{max} < l$ then $t_f = l$, $l \leq t_{max} \leq r$ then $t_f = t_{max}$ and $t_{max} > r$ then $t_f = r$. The median of the values l, r, t_{max} distinguishes these cases. ■

The t for which Equation (4.14) holds is t_{max}, that is, the equation holds strictly under maximum entropy. Since the solution space of (l, r) is always 1-dimensional if all frequencies of the $2^z - 1$ subsets are given, then we compute the value t_{max} by binary search in the space. We note that this search is an approximation, but the binary search converges to values arbitrarily close to t_{max} quickly in practice, e.g., 30 iterations of binary search was used to produce the results in this chapter. For an itemset X, the t-value we hold after 30 such iterations in the search space is then the maximum entropy estimate θ_X^m.

We note that by using a constant number of iterations and assuming a constant number of itemsets, then for each binary search the computation takes time linear in the number of itemsets for which we wish to compute the maximum entropy estimate. Up to a constant factor this is equivalent to the extrapolation and independence estimators.

4.2.3.2 Maxent on noisy inputs

Recall that a view on data is that all data comes from some smaller sample D sampled independently from a larger data set $\mathcal{D}$. As we wish to use the data from D to reason about triples from $\mathcal{D}$ we will show that the error introduced by sampling does not hurt the maxent estimate too much, in particular we will show that a maxent estimate based on D will not be far from a maxent estimate based on $\mathcal{D}$. We will show that the maximum entropy estimate on a triple computed on input with relative error $0 \leq \varepsilon < 0.5$ is only a factor $\mathcal{O}(\varepsilon)$ from the maximum entropy estimate computed using the true distribution from $\mathcal{D}$ as input.

For a distribution d over 3 variables X, Y, W we have $d = (\mathbf{Pr}(XYW), \mathbf{Pr}(XY \wedge \overline{W}), \ldots, \mathbf{Pr}(\overline{XYW}))$, i.e., $|d| = 2^3$. Let $|d_i|$ be the number of non-negated literals in d_i. The entries of d can (as in Section 4.2.3.1) be partitioned into two sets, $o = o_1, \ldots, o_4$ and $e = e_1, \ldots, e_4$ where $d_i \in o$ if $3 - |d_i|$ is odd and $d_i \in e$ if it is even.

Let two polynomials $E_{\varepsilon_1}(t)$ and $O_{\varepsilon_1}(t)$ with error $0 \leq |\varepsilon_1| < 1$ be defined

$$E_{\varepsilon_1}(t) = (e_1 + \varepsilon_1 + t)(e_2 + \varepsilon_1 + t)(e_3 + \varepsilon_1 + t)t \tag{4.15}$$

$$O_{\varepsilon_1}(t) = (o_1 + \varepsilon_1 - t)(o_2 + \varepsilon_1 - t)(o_3 + \varepsilon_1 - t)(o_4 + \varepsilon_1 - t) \tag{4.16}$$

where $t = \mathbf{Pr}(XYW)$ is the triple frequency. When we have

$$E_0(t) = O_0(t) \tag{4.17}$$

then t is the triple frequency under maximum entropy. We seek to bound the error on the output caused by the input error ε_1. Letting t_0 be the solution to $E_0(t) = Q_0(t)$ and t_{ε_1} be the solution to $E_{\varepsilon_1}(t) = O_{\varepsilon_1}(t)$ we wish to bound the error on t_{ε_1} in terms t_0. We will show the following lemma.

Lemma 4.6. *Let two polynomials with additive error t on the terms and where $0 < e_i, o_i, \leq 1$ be defined as below.*

$$\tilde{E}(t) = (e_1 + t)(e_2 + t)(e_3 + t) \tag{4.18}$$

$$\tilde{O}(t) = (o_1 - t)(o_2 - t)(o_3 - t)(o_4 - t) \tag{4.19}$$

For all $t \in [0; \varepsilon \min o_1, \dots, o_4, e_1, \dots, e_3]$, *where* $0 < \varepsilon \leq 1$, *the following bounds on Equations* (4.15) *and* (4.16) *hold*

$$\tilde{E}(0)t \leq E_0(t) \leq (1+\varepsilon)^3 \tilde{E}(0)t \tag{4.20}$$

$$(1-\varepsilon)^4 O_0(0) \leq \tilde{O}(t) \leq O_0(0) \tag{4.21}$$

Proof. The first inequality of eq. (4.20) follows trivially from $t > 0$ and from $\tilde{E}$ being monotonically increasing in t. For the second inequality let $e_{\min} = \min e_1, \dots, e_3$ and $t \leq \varepsilon e_{\min}$, $0 < \varepsilon \leq 1$. It follows that

$$\begin{aligned} E_0(t) &\leq (e_1 + \varepsilon e_{\min})(e_2 + \varepsilon e_{\min})(e_3 + \varepsilon e_{\min})\varepsilon e_{\min} \\ &\leq (1+\varepsilon)^3 e_1 e_2 e_3 t \\ &= (1+\varepsilon)^3 \tilde{E}(0)t \end{aligned}$$

The first inequality of eq. (4.21) follows analogously; let $o_{\min} = \min o_1, \dots, o_4$ and $t \leq \varepsilon o_{\min}$, $0 < \varepsilon \leq 1$. We then have

$$\begin{aligned} \tilde{O}(t) &\geq (o_1 - \varepsilon o_{\min})(o_2 - \varepsilon o_{\min})(o_3 - \varepsilon o_{\min})(o_4 - \varepsilon o_{\min}) \\ &\geq (1-\varepsilon)^4 O_0(0) \end{aligned}$$

The second inequality again follows from $t > 0$ and $\tilde{O}$ being monotonically decreasing in t. ■

It follows that a simple approximate formula for maximum entropy estimates on triples exist.

Lemma 4.7. *For a triple with distribution over entries* $\mathcal{D} = e_1, \dots, e_3, o_1, \dots, o_4$ *let*

$$\tilde{t} = \frac{o_1 o_2 o_3 o_4}{e_1 e_2 e_3}.$$

The triple frequency under maximum entropy $t < \varepsilon \min \mathcal{D}$ *for* $0 < \varepsilon \leq 0.5$ *can be bounded in terms of* $\tilde{t}$

$$t \in \left[(1 - 6.5\varepsilon)\tilde{t}, \tilde{t}\right] \tag{4.22}$$

Proof. Recall that t takes value under maximum entropy when $O_0(t)/E_0(t) = 1$. By Lemma 4.6 we have a lower bound on $O_0(t)/E_0(t)$

$$\begin{aligned}(1-\varepsilon)^4 O_0(0)/(1+\varepsilon)^3\tilde{E}(0) &\leq O(t)_0/E_0(t)\\ (1-\varepsilon)^4 o_1o_2o_3o_4/(1+\varepsilon)^3 e_1e_2e_3 &\leq O_0(t)/E_0(t)\\ \frac{(1-\varepsilon)^4}{(1+\varepsilon)^3}\tilde{t} &\leq O_0(t)/E_0(t)\end{aligned}$$

By expansion and using $0 < \varepsilon \leq 0.5$ we get the needed lower bound of eq. (4.22).

$$\begin{aligned}1 - c*\varepsilon &\geq \frac{(1-\varepsilon)^4}{(1+\varepsilon)^3}\\ c &\geq -\varepsilon^3 + 5\varepsilon^2 - 3\varepsilon + 7\\ c &\geq 6.5\end{aligned}$$

Equivalently we have the needed upper bound

$$\begin{aligned}O_0(t)/E_0(t) &\leq O_0(0)/\tilde{E}(0)\\ O_0(t)/E_(t) &\leq o_1o_2o_3o_4/e_1e_2e_3\\ O_0(t)/E_(t) &\leq \tilde{t}\end{aligned}$$

■

We will now assume relative error on the distribution entries e_i, o_i. The following lemma holds analogously to Lemma 4.7.

Lemma 4.8. *Let distribution $d = (e_1, \dots e_3, o_1, \dots, o_4)$. Let an approximation of distribution d be defined by $o'_i \in [(1-\varepsilon)o_i, (1+\varepsilon)o_i]$ and $e'_i \in [(1-\varepsilon)e_i, (1+\varepsilon)e_i]$ for each $o_i, e_i \in d$ and letting*

$$\begin{aligned}\tilde{t}_r &= \frac{o'_1o'_2o'_3o'_4}{e'_1e'_2e'_3},\\ \tilde{t} &= \frac{o_1o_2o_3o_4}{e_1e_2e_3},\end{aligned}$$

it holds that

$$\tilde{t}_r \in \big[(1-6.5\varepsilon')\tilde{t}, \tilde{t}\big] \tag{4.23}$$

Proof. (Theorem 4.1) The theorem follows directly from Lemmas 4.7 and 4.8. ■

4.3 Experimental Results

Our experiments were conducted on 5 real datasets shown in Table 4.1. For each dataset we prune the singletons with support below a specified threshold. We perform this pruning in order to construct datasets where intuitively the independence model has a chance to do well as it relies solely on the singletons, but also to keep the number of items down to a practical level, as the time complexity with n items is $\mathcal{O}(n^3)$ per experiment. AOL Queries [1] is a uniform sample of the infamous AOL search terms dataset. Docwords is transactions of words occurring together in documents. From MovieLens [2] we create a dataset where a transaction is the set of movies which a particular user rated 5/5 stars. Retail [3] is shopping baskets from an anonymous Belgian supermarket.

Overview of experiments. We start by showing how the independence and maxent estimators perform on the full datasets, i.e., when there is significant support for the triple whose frequency is to be estimated as well as its subsets. The maxent estimator shows better concentration and less variance even for the low-support triples. We then perform experiments for all datasets where 1% of the transactions are sampled and we wish to estimate the frequency of triples in the whole data set. For every triple X with $30 \leq occ(X) \leq 100$ we use the three estimators θ^m_X, θ^i_X and θ^e_X. As X occurs at most 100 times in the full dataset, it occurs in expectation at most once in the sample.

We also study precision and recall for the problem of approximating the set of frequent triples based on the estimators. Again we consider two cases: 1) Estimates are based on the full dataset, where the set to approximate consists of the 10% highest frequencies among all triples, and 2) as in the first case, but with estimates based on a 1% sample. In the latter case, if the threshold for being in top 10% is Δ, then we include in the estimate all triples that are estimated to have at least 0.9Δ occurrences. The number 0.9 was experimentally found to yield a good precision/recall tradeoff for the maximum entropy estimates.

Finally, using again a 1% sample of the data, we compute the average ratio between the error made by our maximum entropy estimate and estimates made by independence and extrapolation, respectively.

In summary our experiments show:

1. In most cases, the maximum entropy estimator provides the best estimate for low-support triples.

[1]`http://www.gregsadetsky.com/aol-data/`
[2]`http://www.grouplens.org/node/73`
[3]`http://fimi.ua.ac.be/data/`

Name	Occ. Threshold	#Items	#Transactions
AOL Queries	500	211	144038
Docwords	3000	142	49078
Movielens	3000	85	67312
Retail	500	85	88162

TABLE 4.1: Datasets used. All datasets are from real data and have previously been used for data mining purposes.

	Independence		Maxent	
Dataset	Precision	Recall	Precision	Recall
AOL Queries	0.0	0.0	0.54	1.0
Docwords	0.93	0.43	0.92	0.93
MovieLens	1.00	0.003	0.80	0.96
Retail	1.00	0.43	0.99	0.97

TABLE 4.2: Precision-recall for full datasets. Relevant triple threshold Δ is such that relevant triples are among the 10% most frequent.

	Ind.		Maxent		Extrapol.	
Dataset	Prec	Rec	Prec	Rec	Prec	Rec
AOL Queries	1.0	0.001	0.23	0.89	0.31	0.67
Docwords	0.88	0.44	0.56	0.72	0.53	0.58
MovieLens	1.0	0.009	0.51	0.90	0.49	0.85
Retail	0.93	0.40	0.51	0.78	0.49	0.73

TABLE 4.3: Precision-recall for sampled datasets. Relevant triple threshold Δ is such that relevant triples are among the 10% most frequent. Triples are reported when $occ \geq 0.9\Delta$ as this was observed to maximize precision/recall for θ^e_X.

Dataset	Independence	Extrapolation
AOL Queries	7.91	7.58
Docwords	6.31	14.32
MovieLens	11.55	7.22
Retail	3.22	4.42

TABLE 4.4: For n estimates and estimators est_1 and est_2 the table shows the normalized absolute error ratio: $\frac{1}{n}\sum_X (|est_1(X) - occ(X)|/|est_2(X) - occ(X)|)$, where est_2 is maximum entropy and est_1 is independence and extrapolation respectively for the two columns.

2. Sampling with higher probability increases concentration greatly for θ^m_X even when the sample is still of insufficient size for θ^e_X to be useful.

3. In almost all cases studied, the precision and recall are strictly better for θ^e_X (see tables 4.2 and 4.3).

4. When predicting the occurrences of triples in the full dataset using a 1% sample, we show θ^m_X improves the absolute error by a factor 3 – 14 compared to θ^i_X and θ^e_X (see Table 4.4).

4.3.1 Maxent vs. independence for full datasets

For all datasets we ran the estimators on every triple with occ > 30 on the full (unsampled) dataset. The figures show the concentration of the estimates by for each triple plotting its observed value (Y-axis) and estimated value (X-axis). The plots are shown in Figures 4.3 and 4.4.

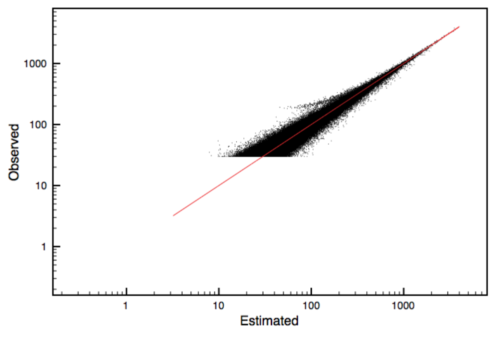
For the Docwords dataset both the maxent estimate (Figure 4.3a) and the independence estimate (Figure 4.3b) are concentrated around the $X = Y$, which denote the line of optimal estimations while for MovieLens we observe similar concentration for maxent (Figure 4.3c) while the independence estimator underestimates slightly (Figure 4.3d) and Retail behaves equivalently in this setting. For AOL Queries (Figure 4.4) using independence estimates we observe similar great under estimation due to high positive correlation, while the maxent estimator overestimates slightly but is far more concentrated.

Our precision and recall computations (Table 4.2) is based on relevance threshold Δ being such that relevant triples are among the top 10% most frequent and we report a triple if the estimate is at least Δ. Note that the high precision for the independence estimate is due to high underestimation - the estimators report too few triples as relevant, as the recall shows. An extreme case of this is the AOL dataset that reports zero triples. Maxent has similar precision but much higher recall for all datasets.

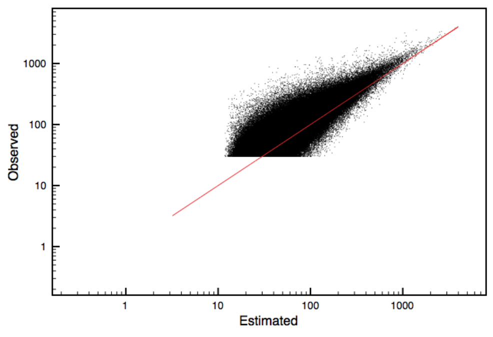
4.3.2 Low-support itemsets

On the same datasets we now examine triples X where $30 \leq occ(X) \leq 100$ and we sample independently at random every transaction with probability 1/100. We wish to estimate the θ_X in the full dataset, but since we have $occ(X) \leq 100$ then following from independent sampling the expected number of occurrences of X in our sample is ≤ 1. We will perform estimates using the extrapolation estimate θ_X^e, the independence estimate θ_X^i and the maxent estimate θ_X^m. We restrict ourselves to triples where all pairs occur in the sample.

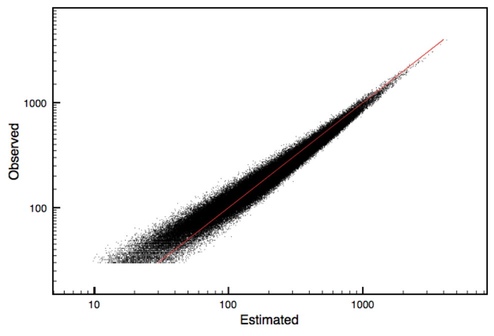
The extrapolation estimator θ_X^e performs similarly on all datasets. While θ_X^e is an unbiased estimator of θ_X, the variance is large as a consequence of the sampling - by the mode of independent Bernoulli trials we have that the most likely outcome of a triple is $\lceil \mu \rceil = 1$, with the outcome 0 and 2 slightly less probable. On the AOL, Retail and MovieLens datasets our experiments conclude similarly: θ_X^e doesn't give meaningful estimates, e.g., zero for the unsampled triples and overestimates on the sampled triples, while θ_X^i underestimates greatly and θ_X^m is fairly concentrated. See



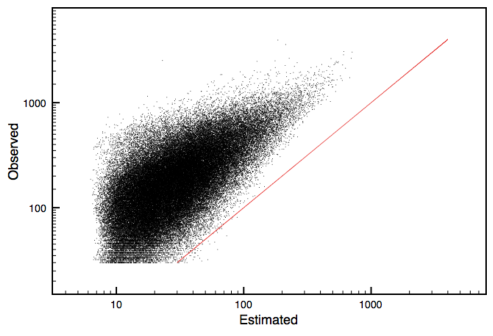
(A) Max. entropy estimates for all triples of occ > 30 in Docwords.



(B) Independence estimates for all triples of occ > 30 in Docwords.

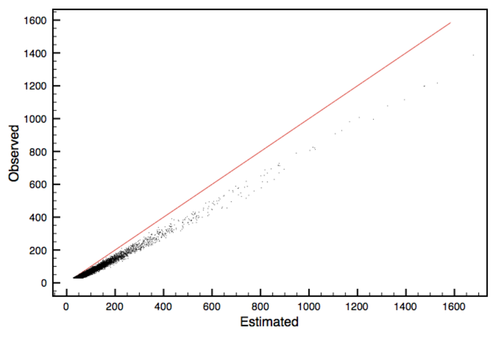


(C) Max. entropy estimates for all triples of occ > 30 in Movielens.

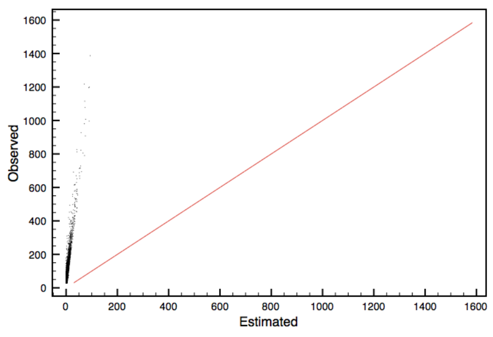


(D) Independence estimates for all triples of occ > 30 in Movielens.

FIGURE 4.3: Concentration plots for Docwords and Movielens datasets. Each point is a triple, the Y-value of a point is the empirical number of occurrences (occ) of the triple while the X-value is the estimated number of occurrences, using either independence or maxent estimators. The red line is X=Y. We observe better concentration on the maxent estimates, in particular when the statistical significance is high.



(A) Maxent estimates for all triples of occ > 30 in AOL Queries.



(B) Independence estimates for all triples of occ > 30 in AOL Queries.

FIGURE 4.4: Concentration plots for AOL Queries. Each point is a triple, the Y-value of a point is the empirical number of occurrences (occ) of the triple while the X-value is the estimated number of occurrences, using either independence or maxent estimators. The red line is X=Y. We observe better concentration for Maxent for all occurrences.

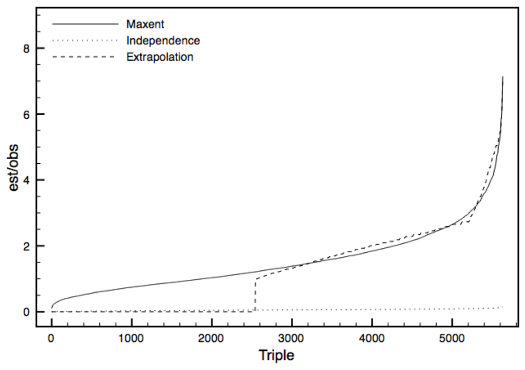


FIGURE 4.5: est/obs distribution plots for AOL Queries sampled at 1/100. We observe that independence underestimates greatly, while maxent has better concentration than extrapolation, in particular for the low occurrence triples.

Figure 4.5 for example of all three estimators on AOL. For Docwords in this setting, we get better concentration for θ^i_X than θ^m_X - this can be explained by pair frequencies being more vulnerable to noise introduced by sampling than single frequencies and θ^i_X being well concentrated for Docwords (recall Figure 4.3b). However, we observe that if we raise the sampling probability to 1/20 from 1/100 then θ^m_X has better concentration. In summary, there is a sampling rate where θ^e_X is very poorly concentrated where θ^m_X outperforms θ^i_X on all datasets.

Precision and recall values (Table 4.3) were computed by setting the relevance threshold Δ to be s.t. if $occ(X) \geq \Delta$ then triple X is among the 10% most frequent triples in the entire dataset with $occ(X) \geq 30$. Triples are reported if the occurrence estimate, which is based on the 1/100 independent sample of all transactions, is at least 0.9Δ as this value was observed experimentally to yield a good tradeoff. AOL and MovieLens using θ^i_X has full precision due to reporting very few triples. Note that θ^e_X has an advantage in terms of precision/recall due to it mostly performing overestimates, i.e., if a triple occurs in the sample then it will likely be reported – even so we see θ^m_X being better than θ^e_X except for AOL where only recall is better.

In the same setting, i.e., triples where $occ(X) \geq 30$ and using an independent 1% sample we compute the normalized ratio of the absolute error between estimators θ^i_X, θ^e_X and θ^m_X. That is, letting $est_m(X), est_i(X)$ denote estimates of triple X by maxent and independence respectively, the normalized ratio is $\frac{1}{n}\sum_X \frac{|est_i(X)-occ(X)|}{|est_m(X)-occ(X)|}$. This experiments show (see Table 4.4) that for all datasets in this setting we would improve, on average, our absolute error by a factor of 3 – 14 by switching from independence or extrapolation to maximum entropy.

Chapter 5

I/O-efficient Similarity Join in High Dimensions

5.1 Introduction

The ability to handle noisy or imprecise data is becoming increasingly important in computing. In database settings this kind of capability is often achieved using similarity join primitives that replace equality predicates with a condition on similarity. To make this more precise consider a space $\mathcal{U}$ and a distance function $d : \mathcal{U} \times \mathcal{U} \to \mathbf{R}$. The *similarity join* of sets $R, S \subseteq \mathcal{U}$ is the following: Given a radius r, compute the set

$$R \underset{\leq r}{\bowtie} S = \{(x, y) \in R \times S \mid d(x, y) \leq r\} \ .$$

This problem occurs in numerous applications, such as web deduplication [93, 94], document clustering [95], data cleaning [96, 97], click fraud detection [98], and many others [99, 100]. As such applications arise in large-scale datasets, the problem of scaling up similarity join for different metric distances is getting more important and more challenging.

Many known similarity join techniques (e.g. prefix filtering [96, 97], positional filtering [94], inverted index-based filtering [101]) are based on *filtering* techniques that often, but not always, succeed in reducing computational costs. If we let $N = |R| + |S|$ these techniques generally require $\Omega(N^2)$ comparisons for worst-case data. Another approach is *locality-sensitive hashing* (LSH) where candidate output pairs are generated using collisions of carefully chosen hash functions. LSH is able to break the N^2 barrier in cases where for some constant $c > 1$ the number of pairs in $R \bowtie_{\leq cr} S$ is not too large. In other words, there should not be too many pairs that have distance within a factor c

of the threshold, the reason being that such pairs are likely to become candidates, yet considering them does not contribute to the output.

In this chapter we consider I/O efficient similarity join methods based on LSH. That is, we are interested in minimizing the number of I/O operations where a block of B points from $\mathcal{U}$ is transferred between external memory and an internal memory with capacity for M points from $\mathcal{U}$. Our main result is the first I/O-efficient (in fact cache-oblivious) algorithm for similarity join that has provably sub-quadratic dependency on the data size and at the same time inverse polynomial dependency on M. In essence, where previous methods have an overhead factor of either N/M or $(N/B)^\rho$ we obtain an overhead of $(N/M)^\rho$, where $\rho \in (0; 1)$ is a parameter of the LSH employed, strictly improving both.

In the context of I/O-efficient algorithms it is natural to not require the *listing* of all near pairs (i.e., pairs with distance not greater than r), but rather we simply require that the algorithm *enumerates* all such near pairs. More precisely, the algorithm calls for each near pair (x, y) a function $\texttt{emit}(x, y)$. This is a natural assumption in external memory since it reduces the I/O complexity, and it is desired in many applications where join results are intermediate results pipelined to a subsequent computation, and are not required to be stored on external memory. Our upper bound can be easily adapted to list all instances by increasing the I/O complexity of an unavoidable additive terms of $\Theta\left(|R \bowtie_{\leq r} S|/B\right)$ I/Os. Whereas most methods in the literature focus on a single (or a few) distance measures, our methods work for an arbitrary space and distance measure that allows locality-sensitive hashing, e.g., Hamming, Manhattan (ℓ_1), Euclidean (ℓ_2), Jaccard, and angular metric distances.

5.2 Related work

The problem of performing similarity joins in high dimensions has attracted significant attention from the database and data mining community. We review some of the results most closely related to our work.

Index-based similarity join. A popular approach is to make use of indexing techniques to build a data structure for one relation, and perform queries using the points of the other relation. The indexes typically perform some kind of *filtering* to reduce the number of points that a given query point is compared to.

Solutions based on filtering techniques include PrefixFilter [97], PartEnum [96], All-Pairs [101], PP-Join [94], as well as I/O-efficient methods based on LSH [102, 103].

Satuluri and Parthasarathy [104] proposed the Bayesian LSH variant for performing candidate pruning, but provided no theoretical guarantees on the performance of the algorithm.

Indexing can be space consuming, in particular for LSH, but in the context of similarity join this is not a big concern since we have many queries, and thus can afford to construct each hash table "on the fly". On the other hand, it is clear that index-based similarity join techniques will not be able to take significant advantage of internal memory when $N \gg M$. Indeed, the query complexity stated in [102] is $\mathcal{O}\left((N/B)^{\rho}\right)$ I/Os. Thus the I/O complexity of using indexing for similarity join will be high.

Sorting-based. The indexing technique of [102] can be adapted to compute similarity joins more efficiently by making use of the fact that many points are being looked up in the hash tables. This means that all lookups can be done in a batched fashion using sorting. This results in a dependency on N that is $\mathcal{O}\left((N/B)^{1+\rho}\right)$ I/Os, where $\rho \in (0;1)$ is a parameter of the LSH family. In addition, there will be a cost that depends on the distribution of distances.

Generic joins. When N is close to M the I/O-complexity can be improved by using general join operators optimized for this case. We will make use of the following result on cache-oblivious nested loop joins:

Theorem 5.1. *(He and Luo [105]) For an arbitrary join condition the join of relations R and S can be computed by a cache-oblivious algorithm using*

$$\mathcal{O}\left((|R|+|S|)/B + (|R||S|)/(MB)\right) \textit{ I/Os.}$$

This number of I/Os suffices to generate the result in memory, but may not suffice to write it to disk.

Distributed computation. Recently, Bahmani et al. [106] proposed the Layered LSH variant, which leverages the Entropy LSH scheme [107] to improve the network communication cost for similarity join in a distributed setting. This approach attempts to collocate near points on the same machine, and distribute far points on different machines. Though Layered LSH achieves improvements on the communication cost over the simple distributed LSH, the I/O cost is unclear since the number of repetitions required to guarantee a correct result has not been analyzed. Our recursive algorithm will also be using repeated LSH, so may be seen as a development of the ideas in [106] with provable guarantees and working for any distance measure with a suitable LSH.

Many techniques have recently been suggested to perform similarity join using the *MapReduce* framework [100, 108–110]. However, the MapReduce-based algorithms often suffer from the problem of load imbalance among reducers due to heuristic data partitioning schemes on skewed data distributions. Moreover, there is no analysis to guarantee the I/O and communication cost of these solutions.

5.3 Preliminaries

5.3.1 Locality-sensitive hashing

Definition 5.2. Fix a distance function $d : \mathcal{U} \times \mathcal{U} \to \mathbf{R}$. For positive reals r, c, p_1, p_2, and $p_1 > p_2, c > 1$, a family of functions $\mathcal{H}$ is (r, cr, p_1, p_2)-*sensitive* if for uniformly chosen $h \in \mathcal{H}$ and all $x, y \in \mathcal{U}$:

- If $d(x, y) \leq r$ then $\Pr[h(x) = h(y)] \geq p_1$;
- If $d(x, y) > cr$ then $\Pr[h(x) = h(y)] \leq p_2$.

We say that $\mathcal{H}$ is *monotone* if $\Pr[h(x) = h(y)]$ is a non-increasing function of $d(x, y)$. We say that $\mathcal{H}$ uses space s if a function from $\mathcal{H}$ can be stored and evaluated using space s.

In what follows we will talk about *near* pairs at distance at most r (those that should be reported), *c-near* pairs at distance between r and cr, and *far* pairs at distance greater than cr. One will often concatenate hash function values to increase the gap of collision probability between near pairs and far pairs. The gap between near and far collision probability is polynomial, with an exponent of $\rho = \log p_1 / \log p_2$.

For ℓ_1 there is a simple construction of an (r, cr, p^ρ, p)-sensitive family with $\rho = 1/c$, which is optimal [111–113]. For ℓ_2 the strongest constructions are (r, cr, p^ρ, p)-sensitive family with $\rho = 1/c^2 + o(1)$ [114]. Internal memory LSH solutions that do not consider I/O-efficiency make use of (r, cr, p_1, p_2)-sensitive functions with $p_2 = 1/N$. This means that in expectation each $x \in R$ has collided with at most one point $y \in S$ with $d(x, y) > cr$, so $\mathcal{O}(N)$ comparisons suffice in expectation to investigate all such "false positives" of a given hash function.

It is worth noting that the standard LSHs for metric distances, including Hamming, ℓ_1, ℓ_2, Jaccard and angular distances, are monotone. Most common LSHs are space-efficient, and use space comparable to that required to store a point. One exception is the

LSH of [114] which requires space $N^{o(1)}$. We did not explicitly require the hash values themselves to be particularly small. However, using universal hashing we can always map to small bit strings while introducing no new collisions with high probability. Thus we may assume that B hash values fit in one memory block.

5.3.2 Computational model

This chapter will study algorithms for similarity join in the external memory model as defined in Section 1.3.2. For simplicity we will here measure block and memory size in units of points from $\mathcal{U}$, such that a block can contains B points. We let $\text{sort}(N) = \mathcal{O}\left((N/B)\log_{M/B}(N/B)\right)$ be shorthand for the I/O complexity [115] of sorting N points.

Our simplest algorithm (Section 5.5.1) is cache-aware: it uses the parameter M explicitly to achieve its I/O complexity. But the main algorithm of Theorem 5.3 is cache-oblivious as it does not explicitly use any model parameters. As a technical note, since we are not concerned with log-factors we do *not* need the so-called *tall cache assumption.*

5.4 Our results

In this chapter we show:

Theorem 5.3. *Consider $R, S \subseteq \mathcal{U}$, let $N = |R| + |S|$, assume $M < N$, $M \geq 20\log N + 3B$ and that there exists a monotone (r, cr, p_1, p_2)-sensitive family of functions with respect to distance measure d, using space B and with $p_2 < p_1 < 1/2$. Then there exists a cache-oblivious randomized algorithm computing $R \bowtie_{\leq r} S$ (wrt. d) with probability $1 - \mathcal{O}\left(\frac{1}{N}\right)$ using*

$$\tilde{\mathcal{O}}\left(p_1^{-1}\left(\frac{N}{M}\right)^{\rho}\left(\frac{N}{B} + \frac{|R \underset{\leq r}{\bowtie} S|}{MB}\right) + \frac{|R \underset{\leq cr}{\bowtie} S|}{MB}\right) \text{ I/Os,}$$

where $\rho = \log p_1 / \log p_2$ and the $\tilde{\mathcal{O}}(\cdot)$-notation hides a polylog(N) factor.

A primary technical hurdle is that we cannot use any kind of strong concentration bounds on the number of points having a particular value, since hash values of an LSH family may be correlated *by definition.* Our algorithm allows the same output pair to be generated more than once. If such duplicates are to be eliminated an additional sorting step is required. In Section 5.5.3 we discuss how to limit the number of times a pair is emitted.

Discussion We now argue (informally) that it seems reasonable to conjecture that the bound in Theorem 5.3 is close to the best possible for the class of "signature based" algorithms that work by generating a set of LSH values (from a black-box, monotone family) and checking all pairs that collide. We can split the I/O complexity of Theorem 5.3 in two parts:

$$T_1 = p_1^{-1}(N/M)^{\rho}\big(N/B + |R \underset{\leq r}{\bowtie} S|/(MB)\big)$$
$$T_2 = |R \underset{\leq cr}{\bowtie} S|/(MB)$$

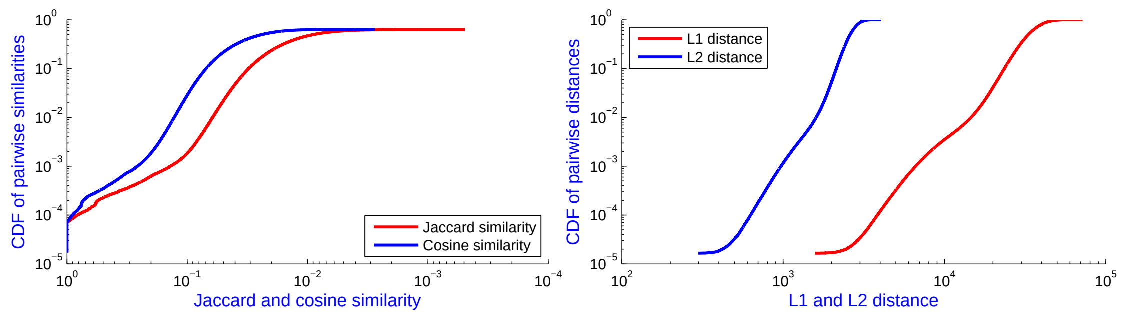
To see why T_1 I/Os may be needed first notice that to ensure N/B I/Os per hash function (including the necessary comparisons), matching the cost of writing hash values to disk to find collisions, we need a collision probability for far pairs that ensures at most NM far collisions in expectation. This is because during N/B I/Os we can bring at most NM distinct pairs into memory.

Now consider the case where there are $\Omega\left(N^2\right)$ pairs at distance cr. Then the collision probability for each pair must be $\mathcal{O}\left(M/N\right)$. In turn, this means that the collision probability for pairs at distance r must be at most $\mathcal{O}\left((M/N)^{\rho}\right)$. But since we have a black-box LSH the only way of reducing the collision probability is by composing several hash values, which means that we may "overshoot" the desired collision probability at distance r by a factor of close to p_1. So $\Omega\left(p_1^{-1}(N/M)^{\rho}\right)$ repetitions (different hash values) are needed before we expect a pair at distance r to collide at least once.

Then, we might need to examine, for each of the $\Omega\left((N/M)^{\rho}\right)$ hash functions, a constant fraction the pairs in $R \bowtie_{\leq r} S$ whose collision probability is constant. For example, this can happen if R and S include two clusters of close points. One could speculate that some pairs could be marked as "finished" during computation such that we do not have to compare their hash values again. However, it seems hard to make this idea work for an arbitrary distance measure where there may be very little structure to the output set.

To argue that the term T_2 is needed consider the case where all pairs in $R \bowtie_{\leq cr} S$ have distance $r + \varepsilon$ for a value ε small enough to make the collision probability of at distance $r + \varepsilon$ indistinguishable the collision probability of at distance r. Then the every pair in $R \bowtie_{\leq cr} S$ must be brought into internal memory to ensure a correct result, which requires T_2 I/Os.

Note that when $M = N$ we have $T_1 = \mathcal{O}\left(N/B\right)$ as we would expect, since just reading the input is optimal. At the other extreme, when $B = M = 1$ our bound matches the time complexity of internal memory techniques.



(a) Enron email dataset (b) MNIST dataset

FIGURE 5.1: The cumulative distributions of pairwise distances on samples of 10,000 points from Enron Email and MNIST datasets.

Finally, we carried out experiments to demonstrate that the first term T_1 often dominates the second term T_2 in real datasets. In particular, we depict the cumulative distribution function (cdf) in log-log scale of all pairwise distances (i.e., ℓ_1, ℓ_2) and all pairwise similarities (i.e., Jaccard and cosine) on two commonly used datasets: Enron Email[1] and MNIST[2], as shown on Figure 5.1.

Figure 5.1.a demonstrates an almost *monomial* relationship with some small exponent m between the number of pairwise similarities greater than the similarity threshold s and such s, where $s > 0.5$. This setting $s > 0.5$ is commonly used in many applications for both Jaccard and cosine similarities [94, 96, 101]. Similarly, Figure 5.1.b also shows an *monomial* relationship between the distance threshold r and the number of pairwise distances smaller than r. In turn, this means that the number of c-near pairs $|R \bowtie_{\leq cr} S|$ is not much greater than $c^m|R \bowtie_{\leq r} S|$. In other words, the second term T_2 is often much smaller than the first term T_1.

5.5 Our algorithms

In this section we describe our I/O efficient algorithms. As a warm-up, we start in Section 5.5.1 with a simple cache-aware algorithm, which uses an LSH family where the value of the collision probability is set to be a function of the internal memory size. Then, in Section 5.5.2, we propose our main result, that is the recursive and cache-oblivious algorithm. The algorithm uses the LSH with a black-box approach and does not make any assumption on the value of collision probability. Finally, in Section 5.5.3, we show how to reduce the expected number of times each near pair is emitted.

[1]https://archive.ics.uci.edu/ml/datasets/Bag+of+Words
[2]http://yann.lecun.com/exdb/mnist/

5.5.1 Warm-up: Cache-aware algorithm

To outline the main ideas we first outline a simple cache-aware algorithm (explicitly using the parameter M) that achieves the error probability and I/O bound of Theorem 5.3.

The algorithm repeatedly hashes all points into buckets using LSH. By composing several hash functions from $\mathcal{H}$ we can obtain collision probabilities p'_1 and p'_2 of the LSH such that:

1. For a point x the number of far points it collides with is in expectation at most M.
2. Near points collide with probability roughly $p'_1 \geq p_1(M/N)^\rho$.

We iterate through each bucket and look at all the pairs $(x, y) \in R \times S$ in the bucket. If a bucket has less than $M/2$ points from either R or S the cost of comparing all pairs in the bucket is bounded by the cost of reading the points. If a point has collided with a near point we emit that pair, if a point has collided with a c-near point we ignore it, but we count the number of times a point collides with far points. If a point reaches 4 times its expected number of collisions with far points we throw that point away. Using $1/p'_1$ LSH functions we have at least a constant probability of a near pair colliding in one of them, and we have probability above 3/4 that a point stays within 4 times its expected number of far collisions. We note that one of the primary technical difficulties is that since we distribute points using an LSH we do not have independence and hence all tail bounds must be done by Markov-like arguments. After constant success probability has been reached we boost the success rate by a standard repetition argument to be at least $1 - 1/N$ using $\mathcal{O}(\log N)$ repetitions.

We note that we have strong spatiality: In one block transfer of size B we can compute BM distance pairs since each point in the block can compared to each point residing in memory. The I/O bound is achieved the following way: For each of the $1/p'_1$ LSH functions we spend sort(N) on rehashing and $\mathcal{O}(N/B)$ on far collisions since there are N points in total, and in expectation each point collides with at most M far points so $\mathcal{O}(NM/(BM)) = \mathcal{O}(N/B)$ I/Os. Each near pair collides at most once for each of the $1/p'_1$ hash functions, and the c-near pairs collide in expectation a constant number of times due to monotonicity of the LSH family. See proof details below.

Proof. (of cache-aware warmup-algorithm) First assume $0 < \rho \leq 1$ and let $p_1 = (M/4N)^\rho$, $p_2 = M/4N$, $L = 5/p_1$ (assumed wlog to be integer). Further let $h_1, \ldots, h_L$ be hash functions drawn independently from the family of (r, cr, p_1, p_2)-sensitive hash functions.

For each h_i, $i \in \{1, \ldots, L\}$ we will hash all the points into buckets: For a fixed hash function h_i let $R_j \subseteq R$ and $S_j \subseteq S$ be the points that hash to bucket j. For each hash function, the hashing step happens by computing all N hash values and then sorting them all by their hash values using $\mathcal{O}\left(\text{sort}(N)\right) I/Os$.

For each hash function algorithm proceeds by examining each such R_j, S_j pair. Assume without loss of generality that $|R_j| \leq |S_j|$. Assume a bucket is an ordered array or list. The two cases to handle are then:

1. If $|R_j| \leq M$ then scan over S_j, computing all distances of pairs in $R_j \times S_j$.
2. Otherwise we have $|R_j| > M$ and hence $|R_j \times S_j| > M^2$. Then the algorithm proceeds by iteratively reading into memory size M subsets $R'_j \subset R_j$ and for each of such $|R_j|/M$ subsets scan over S_j and compute all distance pairs in $R'_j \times S_j$.

Since we wish to bound the I/Os spent in terms of the size of the c-near pairs $R \bowtie_{\leq cr} S$, we are going to keep count of the number of bad collisions. Specifically, we are with each point $x \in R \cup S$ going to associate with it a counter, C_x, that denotes the number of collisions with far away points, i.e. it is the size of the multiset of points y where $d(x, y) > cr$ and the pair collide. The counter C_x will be associated with the point x and can be thought of as adding another dimension to the point, however hash functions and comparisons do not take this dimension into account. We will allow ourselves for each $x \in R \cup S$ to compute $2LM$ distances $d(x, y)$ that are above cr and if this threshold is exceeded we will remove the point from the set. When processing a bucket pair R_j, S_j and a point $x \in S_j$ reaches the threshold then after the current scan over S_j, replace the bucket with $S_j \setminus x$ using $N/B + \mathcal{O}(1)$ I/Os. Since the scan uses $\mathcal{O}(N/B)$ itself this replacement doesn't change the complexity. Since $|R_j \leq |S_j|$ nothing needs to be removed when the threshold is reached for a point in R_j as it wastes no I/Os. In both cases (1) and (2) we have the following crucial spatiality property: When the algorithm scans S_j then for each block transfer of size B we compute distances between pairs in that block and our current size M subset of R_j hence MB distances are computed using one I/O.

I/Os spent on near pairs. Since we repeat for $L = 5/p_1$ hash functions and at most each pair from $R \bowtie_{\leq r} S$ collides in all of them, the I/Os spent on the near pairs is at most $\mathcal{O}\big((1/p_1)|R \bowtie_{\leq r} S|/(BM)\big)$.

I/Os spent on c-near pairs. A pair from $R \bowtie_{\leq cr} S$ appears in a bucket with probability at least p_2 due to monotonicity of our LSH family. Since we repeat for $L = 5/p_1$ hash functions for $p_1 > p_2$ we use in expectation $\mathcal{O}\big(|R \bowtie_{\leq cr} S|/(MB)\big)$ I/Os.

I/Os spent on far pairs. For each point $x \in R \cup S$ we throw x away as described above if it exceeds $2LM$ collisions with points that are more than distance cr away. The number of far away collisions examined is thus $2LM + M$ as due to case (2) we do not stop until before the full scan of S_j is complete. As we still get BM distance pairs computed for one block transfer of size B the total number of I/Os spent on far away collisions is $\mathcal{O}\left((2NLM + M)/MB\right) = \mathcal{O}\left(2NL/B\right)$.

Correctness. Consider a point $x \in R_j \cup S_j$ and let for a fixed hash function h_i let

$$F^i_{x,j} = \{y \mid y \in S_j \wedge d(x,y) \geq cr\}$$

be the set of far collisions with point x. By choice of p_2 we have $\mathbb{E}\left[|F_{x,j}|\right] = M/4$ and hence by Markov we have:

$$\Pr\left[|F^i_{x,j}| \geq 2M\right] \leq 1/8.$$

Then looking at the number of far collisions over all L hash functions we trivially have:

$$\Pr\left[\sum_{i \in \{1,\ldots,L\}} |F^i_{x,j}| \geq 2LM\right] \leq 1/8.$$

Consider a point $x \in R_v \cup S_v$ and let for a fixed hash function h_i let $F^i_{x,v} = \{y \mid y \in S_v \wedge d(x,y) \geq cr\}$ be the set of far collisions with point x. By choice of p'_2 we have $\mathbb{E}|F_{x,v}| \leq M$ and hence by Markov we have: $\mathbf{Pr}|F^i_{x,v}| \geq 4M \leq 1/4$. Then looking at the number of far collisions over all L hash functions we trivially have:

$$\mathbf{Pr} \sum_{i \in \{1,\ldots,L\}} |F^i_{x,v}| \geq 4LM \leq 1/4.$$

Consider then a pair $x \in R$, $y \in S$. By an union bound over the probability of the points being removed, the pair is removed with probability at most $1/2$. If for a pair $x \in R$, $y \in S$ it holds that $d(x,y) \leq r$, then by our choice of hash function they collide with probability at least p'_1. If we repeat for L hash functions we have that the probability of the pair not colliding in any of the L hash functions is at most $(1 - p'_1)^L \leq 1/e$. Therefore, we have that the probability that during the repetitions a near pair is not removed and collides is at least $1 - (1/2 + 1/(2e)) = 1/2 - 1/(2e) \geq 0.31$. When this event happens the pair is examined and if it is near it is reported.

I/Os spent in total. Initially we use $N/B + \mathcal{O}(1)$ I/Os to read the input. For each hash function we use $\mathcal{O}(\text{sort}(N))$ I/Os on hashing and sorting the points by their hash value and on far points we use at most $\mathcal{O}(N/B)$ I/Os. The I/Os spent on c-near pairs in total over all L hash functions is $\mathcal{O}\big(|R \bowtie_{\leq cr} S|/MB\big)$. Since we use $L = 5/p_1 = (4N/M)^\rho$

```
1 If $\psi = \Psi$ then compute $R \bowtie_{\leq r} S$ using the algorithm of Theorem 5.1 and return;
2 If $|R| > |S|$ then swap (the references to) the sets such that $|R| \leq |S|$;
3 Pick a random sample $S'$ of $20\Delta$ points from $S$ (or all points if $|S| < 20\Delta$);
4 Permute $R$ such that the set $R'$ of points from $R$ that have distance $> cr$ to at most $10\Delta$
  points in $S'$ is in the first positions;
5 Compute $R' \bowtie_{\leq r} S$ using the algorithm of Theorem 5.1;
6 Repeat $L = 1/p_1$ times:
7     Choose $h \in \mathcal{H}$ uniformly at random;
8     Use $h$ to partition (in-place) $R \backslash R'$ and $S$ in buckets $R_v$, $S_v$ of points with hash value $v$;
9     For each $v$ where $R_v$ and $S_v$ are nonempty, recursively call SimJoin($R_v, S_v, \psi + 1$);
```

Algorithm SimJoin(R, S, k)**:** R, S are the input sets, and ψ is the recursion depth.

hash functions and repeat everything $2 \log N$ times the stated I/O complexity follows. ∎

5.5.2 Cache-oblivious algorithm

We are now ready to describe our cache-oblivious I/O efficient algorithm for similarity join, named SimJoin. The algorithm receives in input two sets R and S of join attributes, and a parameter ψ denoting the depth in the recursion tree (initially, $\psi = 0$). Without loss of generality, we assume that S denotes the largest set (i.e., $|R| \leq |S|$) and let $N = |R| + |S|$. The input sets are stored in arrays and, since the order is immaterial, the algorithm is allowed to rearrange their points such that all manipulation of R and S is done in-place.

We use a monotone (r, cr, p_1, p_2)-sensitive family of functions $\mathcal{H}$.[3] As discussed in the introduction, monotonicity is a property of most LSH families. For the sake of simplicity, we initially assume that $1/p_1$ and $1/p_2$ are integers, and further assume without loss of generality that N is a power of two.

Pseudocode for a description of SimJoincan be seen at the top of the page. The parameters $\Delta = \log N$ and $\Psi = \lceil \log_{1/p_2} N \rceil$ in the pseudocode are global values that are kept invariant in the recursive levels (i.e., N is the initial input size). The recursion ends when there are no more points in an input sets, or when a subproblem reaches recursion level Ψ. We note that the cache-oblivious nested loop join algorithm of Theorem 5.1 is used for computing the similarity join in subproblems at level Ψ, as well as the similarity join in Step 5. If $1/p_1$ is not integer the last iteration is performed with probability $1/p_1 - \lfloor 1/p_1 \rfloor$, such that $L \in \{\lfloor 1/p_1 \rfloor, \lceil 1/p_1 \rceil\}$ and $\mathbb{E}[L] = 1/p_1$.

[3]The monotonicity requirement can be relaxed to the following: $\Pr[h(x) = h(y)] \geq \Pr[h(x') = h(y')]$ for every two pairs (x, y) and (x', y') where $d(x, y) \leq r$ and $d(x', y') > r$. A monotone LSH family clearly satisfies this assumption.

A single call to SimJoin(R, S) outputs a given pair with probability $o(1)$. To boost the probability to $1 - \mathcal{O}(1/N)$ that all pairs in the join result are output, SimJoin(R, S) needs to be called $\mathcal{O}\left(\log^{3/2} N\right)$ times.

5.5.2.1 Analysis of I/O complexity

We will bound the *expected* number of I/Os of the algorithm rather than the worst case. This can be converted to a fixed time bound by a standard technique of restarting the computation when the expected number of I/Os is exceeded by a factor 2. To succeed with probability $1-1/N$ it suffices to do $\mathcal{O}(\log N)$ restarts to complete within twice the expected time bound, and the logarithmic factor is absorbed in the $\tilde{\mathcal{O}}$-notation. If the computation does not succeed within this bound we fail to produce an output, slightly increasing the error probability. In this section, we let R and S denote the initial input sets, and let $\tilde{R}$ and $\tilde{S}$ denote the subsets given in input to a particular subproblem (note that $\tilde{R}$ can be a subset of R but also of S; similarity for $\tilde{S}$).

Our first lemma says that two properties of the choice of random sample in Step 3 are almost certain.

Lemma 5.4. *Consider a run of Steps 3 and 4 in any subproblem* SimJoin$(\tilde{R}, \tilde{S})$. *Then with probability at least* $1 - \mathcal{O}(1/N)$ *over the choice of sample* $\tilde{S}'$ *we have:*

$$|\tilde{R}' \underset{\leq cr}{\bowtie} \tilde{S}| > \frac{|\tilde{R}'||\tilde{S}|}{5} \ , \tag{5.1}$$

$$|(\tilde{R}\backslash\tilde{R}') \underset{> cr}{\bowtie} \tilde{S}| > \frac{4|\tilde{R}\backslash\tilde{R}'||\tilde{S}|}{5} \ . \tag{5.2}$$

Proof. If $|\tilde{R}| \leq 20\Delta$, the claims are clearly true with probability 1. Assume now that $|\tilde{R}| > 20\Delta$. Both claims follow from Chernoff bounds. Let $x \in \tilde{R}$ be a point which is c-near to at most one fifth of the points in $\tilde{S}$. A point x enters $\tilde{R}'$ if there are at least 10Δ c-near points in $\tilde{S}'$ and this happens with probability at most $1/N^{4.5}$ since the entries in $\tilde{S}'$ are randomly and independently chosen from $\tilde{S}$ and there is probability $1/5$ that an entry in $\tilde{S}$ is near to x.

Each point of $R \cup S$ appears in at most in $2L^{\Psi} \leq 2N^2$ subproblems and there are at most N points in $R \cup S$. Thus by an union bound we get that, in any subproblem SimJoin$(\tilde{R}, \tilde{S})$, with probability $1 - 2N^3N^{-4.5} \leq 1 - 2N^{-1}$ every point in $\tilde{R}'$ has at least $|\tilde{S}|/5$ c-near points in $\tilde{S}$, and thus the bound in Equation 5.1 follows. Similarly, we have that all points in $\tilde{R}\backslash\tilde{R}'$ are far from at least 4/5 points of $\tilde{S}$ with probability at least $1 - 2N^{-1}$, getting Equation 5.2. ■

In the remainder of the analysis, we assume that Equations 5.1 and 5.2 in Lemma 5.4 always hold, and thus our claims apply with probability at least $1 - \mathcal{O}(1/N)$.

To analyze the number of I/Os for subproblems of size more than M we bound the cost in terms of different types of *collisions*, i.e., pairs in $R \times S$ that end up in the same subproblem of the recursion. We say that (x, y) *is in* a particular subproblem $\textsc{SimJoin}(\tilde{R}, \tilde{S})$ if $(x, y) \in (\tilde{R} \times \tilde{S}) \cup (\tilde{S} \times \tilde{R})$. Observe that a pair (x, y) is in a subproblem if and only if x and y have colliding hash values on every step of the call path from the initial invocation of $\textsc{SimJoin}$.

Definition 5.5. Given $Q \subseteq R \times S$ let $C_i(Q)$ be the number of times a pair in Q is in a call to $\textsc{SimJoin}$ at the ith level of recursion. We also let $C_{i,k}(Q)$, with $0 \leq k \leq \log N$ denote the number of times a pair in Q is in a call to $\textsc{SimJoin}$ at the ith level of recursion where the smallest input set has size in $[2^k, 2^{k+1})$ if $0 \leq k < \log M$, and in $[M, +\infty)$ if $k = \log M$. The count is over all pairs and with multiplicity, so if (x, y) is in several subproblems at the ith level, all these are counted.

Our next lemma expresses the I/O complexity of $\textsc{SimJoin}$ in terms of $C_i(R \bowtie_{\leq cr} S)$ and $C_i(R \bowtie_{>cr} S)$, for any $0 \leq i < \Psi$. We will later upper bound the expected size of these quantities in Lemma 5.7 and then get the claim of Theorem 5.3.

Lemma 5.6. *Let $\ell = \lceil \log_{1/p_2}(N/M) \rceil$ and $M \geq 20 \log N + 3B$. With probability at least $1 - \mathcal{O}(1/N)$, the I/O complexity of* $\textsc{SimJoin}(R, S)$ *is*

$$\tilde{\mathcal{O}}\left(\frac{NL^\ell}{B} + \sum_{i=0}^{\ell} \frac{C_i\left(R \underset{\leq cr}{\bowtie} S\right)}{MB} + \sum_{i=\ell}^{\Psi-1} \sum_{k=0}^{\log M} \frac{C_{i,k}\left(R \underset{>cr}{\bowtie} S\right) L}{B 2^k}\right).$$

Proof. To ease the analysis we assume that no more than 1/3 of internal memory is used to store blocks containing elements of R and S, respectively. Since the cache-oblivious model assumes an optimal cache replacement policy this can not decrease the I/O complexity. Also, internal memory space used for other things than data (input and output buffers, the recursion stack of size at most Ψ) is less than $M/3$ by our assumption that $M \geq 20\Delta + 3B = \Omega(\log N)$. As a consequence, we have that the number of I/Os for solving a subproblem $\textsc{SimJoin}(\tilde{R}, \tilde{S})$ where $|\tilde{R}| \leq M/3$ and $|\tilde{S}| \leq M/3$ is $\mathcal{O}\left((|\tilde{R}| + |\tilde{S}|)/B\right)$, including all recursive calls. This is because there is space $M/3$ dedicated to both input sets and only I/Os for reading the input are required. By charging the cost of such subproblems to the writing of the inputs in the parent problem, we can focus on subproblems where the largest set (i.e., $\tilde{S}$) has size more than $M/3$.

We notice that the cost of Steps 3 and 4 is dominated by other costs by our assumption that the set $\tilde{S}'$ fits in internal memory, which implies that it suffices to scan data once to implement these steps. This cost is clearly negligible with respect to the remaining steps and thus we ignore them.

We first provide an upper bound on the I/O complexity required by all subproblems at a recursive level above ℓ. Let $\textsc{SimJoin}(\tilde{R},\tilde{S})$ be a recursive call at the ith recursive level, for $0 \leq i \leq \ell$. The I/O cost of the nested loop join in Step 5 in $\textsc{SimJoin}(\tilde{R},\tilde{S})$ is $\mathcal{O}\left(|\tilde{S}|/B + |\tilde{R}'||\tilde{S}|/(MB)\right)$ by Theorem 5.1. We can ignore the $\mathcal{O}\left(|\tilde{S}|/B\right)$ term since it is asymptotically negligible with respect to the cost of each iteration of Step 6, which is upper bounded later. By Equation 5.1 we have that $\tilde{R}' \bowtie_{\leq cr} \tilde{S}$ contains more than $|\tilde{R}'||\tilde{S}|/5$ collisions, and thus the cost of Step 5 in $\textsc{SimJoin}(\tilde{R},\tilde{S})$ can be bounded by $\mathcal{O}\left(|\tilde{R}' \bowtie_{\leq cr} \tilde{S}|/(MB)\right)$. This means that we can bound the total I/O cost of all executions of Step 5 at level i of the recursion with $\mathcal{O}\left(C_i\left(R \bowtie_{\leq cr} S\right)/(MB)\right)$ since each near pair (x,y) appears in $C_i((x,y))$ subproblems at level i. The second major part of the I/O complexity is the cost of preparing recursive calls in $\textsc{SimJoin}(\tilde{R},\tilde{S})$ (i.e., Steps 7-8). In fact, in each iteration of Step 6, the I/O cost is $\tilde{\mathcal{O}}\left((|\tilde{R}| + |\tilde{S}|)/B\right)$, which includes the cost of hashing and of sorting to form buckets. Since each point of $\tilde{R}$ and $\tilde{S}$ is replicated in L subproblem in Step 6, we have that each point of the initial sets R and S is replicated L^{i+1} times at level i. Since the average cost per entry is $\tilde{\mathcal{O}}\left(1/B\right)$, we have that the total cost for preparing recursive calls at level i is $\tilde{\mathcal{O}}\left(NL^{i+1}/B\right)$. By summing the above terms, we have that the total I/O complexity of all subproblems in the ith recursive level is upper bounded by:

$$\tilde{\mathcal{O}}\left(\frac{C_i\left(R \underset{\leq cr}{\bowtie} S\right)}{MB} + \frac{NL^{i+1}}{B}\right). \tag{5.3}$$

We now focus our analysis to bound the I/O complexity required by all subproblems at a recursive level below ℓ. Let again $\textsc{SimJoin}(\tilde{R},\tilde{S})$ be a recursive call at the ith recursive level, for $\ell \leq i \leq \Psi$. We observe that (part of) the cost of a subproblem at level $i+1 > \ell$ can be upper bounded by a suitable function of collisions among far points in $\textsc{SimJoin}(\tilde{R},\tilde{S})$. More specifically, consider an iteration of Step 6 in a subproblem at level i. Then, the cost for preparing the recursive calls and for performing Step 5 in each subproblem (at level $i+1$) generated during the iteration, can be upper bounded as

$$\tilde{\mathcal{O}}\left((|\tilde{R}\backslash\tilde{R}'| + |\tilde{S}|)/B + |(\tilde{R}\backslash\tilde{R}') \underset{\leq cr}{\bowtie} \tilde{S}|/(BM)\right),$$

since each near pair in $(\tilde{R}\backslash\tilde{R}') \bowtie_{\leq cr} \tilde{S}$ is found in Step 5 in at most one subproblem at level $i+1$ generated during the iteration. Since $|(\tilde{R}\backslash\tilde{R}') \bowtie_{\leq cr} \tilde{S}| \leq |\tilde{R}\backslash\tilde{R}'||\tilde{S}|$, we easily get that the above bound can be rewritten as

$$\tilde{\mathcal{O}}\left(|\tilde{R}\backslash\tilde{R}'||\tilde{S}|/(B\min\{M,|\tilde{R}\backslash\tilde{R}'|\})\right).$$

We observe that this bound holds even when $i=\Psi-1$: in this case the cost includes all I/Os required for solving the subproblems at level Ψ called in the iteration and which are solved using the nested loop in Theorem 5.1 (see Step 1). By Lemma 5.4, we have that the above quantity can be upper bounded with the number of far collisions between $\tilde{R}$ and $\tilde{S}$, getting $\tilde{\mathcal{O}}\left((|\tilde{R}\backslash\tilde{R}' \bowtie_{>cr} \tilde{S}|)/(B\min\{M,|\tilde{R}\backslash\tilde{R}'|\})\right)$. Recall that $C_{i,k}\left(Q\right)$ denotes the number of times a pair in Q is in a call to SimJoin at the ith level of recursion where the smallest input set has size in $[2^k, 2^{k+1})$ if $0 \leq k < \log M$, and in $[M,+\infty)$ if $k=\log M$. Then, the total cost for preparing the recursive calls in Step 7-8 in all subproblems at level i and for performing Step 5 in all subproblems at level $(i+1)$ is:[4]

$$\tilde{\mathcal{O}}\left(\sum_{k=0}^{\log M}\frac{C_{i,k}\left(R\bowtie_{>cr} S\right)L}{B2^k}\right). \tag{5.4}$$

The L factor in the above bound follows since far collisions at level i are used for amortizing the cost of Step 5 for each one of the L iterations of Step 6.

To get the total I/O complexity of the algorithm we sum the I/O complexity required by each recursive level. We bound the cost of each level as follows: for a level $i<\ell$ we use the bound in Equation 5.3; for a level $i>\ell$ we use the bound in Equation 5.4; for level $i=\ell$, we use the bound given in Equation 5.4 to which we add the first term in Equation 5.3 since the cost of Step 5 at level ℓ is not included in Equation 5.4 (note that the addition of Equations 5.3 and 5.4 gives a weak upper bound for level ℓ). The lemma follows. ■

Next, we analyze the expected sizes of the terms in Lemma 5.6. Clearly each pair from $R\times S$ is in the top level call, so the number of collisions is $|R||S|<N^2$. But in lower levels we show that the expected number of times that a pair collides either decreases or increases geometrically, depending on whether the collision probability is smaller or larger than p_1 (or equivalently, depending on whether the distance is greater or smaller than the radius r):

Lemma 5.7. *With probability* $1-\mathcal{O}\left(1/N\right)$ *and for each* $0\leq i\leq\Psi$, *we have*

1. $\mathbb{E}\left[C_i\left(R \underset{>cr}{\bowtie} S\right)\right] \leq |R \underset{>cr}{\bowtie} S|\left(p_2/p_1\right)^i$

[4]We note that the true input size of a subproblem is $|\tilde{R}|$ and not $|\tilde{R}\backslash\tilde{R}'|$. However, the expected value of $C_{i,k}\left(R\bowtie_{>cr} S\right)$ is computed assuming the worst case where there are no close pairs an thus $\tilde{R}'=\emptyset$.

2. $\mathbb{E}\left[C_i\left(R \underset{>r,\leq cr}{\bowtie} S\right)\right] \leq |R \underset{>r,\leq cr}{\bowtie} S|$

3. $\mathbb{E}\left[C_i\left(R \underset{\leq r}{\bowtie} S\right)\right] \leq |R \underset{\leq r}{\bowtie} S|\, L^i$

4. $\mathbb{E}\left[C_{i,k}\left(R \underset{>cr}{\bowtie} S\right)\right] \leq N 2^{k+1} \left(p_2/p_1\right)^i$, *for any* $0 \leq k < \log M$

Proof. Let $x \in R$ and $y \in S$. We are interested in upper bounding the number of collisions of the pair at the ith recursive level. We envision the problem as *branching process* (more specifically a GaltonWatson process, see e.g. [116]) where the expected number of children (i.e., recursive calls that preserve a particular collision) is $\Pr\left[h(x) = h(y)\right]/p_1$ for random $h \in \mathcal{H}$. It is a standard fact from this theory that the expected population size at generation i (i.e., number of times (x, y) is in a problem at recursive level i) is $\left(\Pr\left[h(x) = h(y)\right]/p_1\right)^i$ [116, Theorem 5.1]. If $d(x, y) > cr$, we have that $\Pr\left[h(x) = h(y)\right] \leq p_2$ and each far pair appears $(p_2/p_1)^i$ times in expectation at level i, from which follows Equation 1. Moreover, since the probability of collisions is monotonic in the distance, we have that $\Pr\left[h(x) = h(y)\right] \leq 1$ if $r < d(x, y) \leq cr$, and $\Pr\left[h(x) = h(y)\right] \leq 1/p_1$, from which follow Equations 2 and 3.

In order to get the last bound we observe that each entry of R and S is replicated $L^i = p_1^{-i}$ times at level i. Thus, we have that $N2^{k+1}L^i$ is the total maximum number of far collisions in subproblems at level i where the smallest input set has size in $[2^k, 2^{k+1})$. Each one of these collisions survives up to level i with probability p_2^i, and thus the expected number of these collisions is $N2^{k+1}(p_1/p_2)^i$. ■

We are now ready to prove the I/O complexity of SimJoin as claimed in Theorem 5.3. By the linearity of expectation and Lemma 5.6, we get that the expected I/O complexity of SimJoin is

$$\tilde{\mathcal{O}}\left(\frac{NL^\ell}{B} + \sum_{i=0}^{\ell} \frac{\mathbb{E}\left[C_i\left(R \underset{\leq cr}{\bowtie} S\right)\right]}{MB} + \sum_{i=\ell}^{\Psi-1} \sum_{k=0}^{\log M} \frac{\mathbb{E}\left[C_{i,k}\left(R \underset{>cr}{\bowtie} S\right)\right] L}{B2^k}\right),$$

where $\ell = \lceil \log_{1/p_2}(N/M) \rceil$. Then, by noticing that $C_{i,\log M}\left(R \bowtie_{>cr} S\right) \leq C_i\left(R \bowtie_{>cr} S\right)$, $|R \bowtie_{>cr} S| \leq N^2$ and $C_i\left(R \bowtie_{\leq cr} S\right) = C_i\left(R \bowtie_{\leq r} S\right) + C_i\left(R \bowtie_{>r, \leq cr} S\right)$, and by plugging in the bounds on the expected number of collisions given in Lemma 5.7, we get the claimed result.

5.5.2.2 Analysis of correctness

Next, we would like to argue that a pair (x, y) with $d(x, y) \leq r$ is output with good probability. Let $X_i = C_i((x, y))$ be the number of subproblems at level i containing (x, y). Recall that $\mathbb{E}[X_i] = (\Pr[h(x) = h(y)]/p_1)^i$. If $\Pr[h(x) = h(y)]/p_1 > 1$ then in fact there is positive constant probability that (x, y) survives indefinitely, i.e., does not go extinct [116]. Since at every branch of the recursion we eventually compare points that collide under all hash functions on the path from the root call, this implies that (x, y) is reported with positive constant probability.

In the *critical case* where $\Pr[h(x) = h(y)]/p_1 = 1$ we need to consider the variance of X_i, which by [116, Theorem 5.1] is equal to $i\sigma^2$, where σ^2 is the variance of the number of children (hash collisions in recursive calls). If $1/p_1$ is integer the number of children in our branching process follows a binomial distribution with mean 1. This implies that $\sigma^2 < 1$. Also in the case where $1/p_1$ is not integer it is easy to see that the variance is bounded by 2. That is, we have $\mathrm{Var}(X_i) \leq 2i$, which by Chebychev's inequality means that for some integer $j^* = 2\sqrt{i} + \mathcal{O}(1)$:

$$\sum_{j=j^*}^{\infty} \Pr[X_i \geq j] \leq \sum_{j=j^*}^{\infty} \mathrm{Var}(X_i)/j^2 \leq 1/2 \ .$$

Since $\mathbb{E}[X_i] = \sum_{j=1}^{\infty} \Pr[X_i \geq j] = 1$ this implies that $\sum_{j=1}^{j^*-1} \Pr[X_i \geq j] > 1/2$, and since $\Pr[X_i \geq j]$ is non-increasing with j this implies that $\Pr[X_i \geq 1] \geq 1/(2j^*) = \Omega\left(1/\sqrt{i}\right)$. Since the recursion depth is $\mathcal{O}(\log N)$ this implies that the probability that a close pair is found is $\Omega\left(1/\sqrt{\log N}\right)$. Thus, by repeating $\mathcal{O}\left(\log^{3/2} N\right)$ times we can make the error probability $\mathcal{O}\left(1/N^3\right)$ for a particular pair and $\mathcal{O}(1/N)$ for the entire output by applying the union bound.

5.5.3 Removing duplicates

The definition of LSH requires that the probability $p(x, y) = \Pr[h(x) = h(y)]$ of two near points x and y of being hashed on the same value is at least p_1. If $p(x, y) \gg p_1$, our SimJoin algorithm can emit (x, y) many times. As an example suppose that the algorithm ends in one recursive call: then, the pair (x, y) is expected to be in the same bucket for $p(x, y)L$ iterations of Step 6 and thus it is emitted $p(x, y)L \gg 1$ times in expectation. Moreover, if the pair is not emitted in the first recursive level, the expected number of emitted pairs increases as $(p(x, y)L)^i$ since the pair (x, y) is contained in $(p(x, y)L)^i$ subproblems at the ith recursive level. A simple solution requires to store all emitted near pairs on the external memory, and then using a cache-oblivious sorting algorithm [4] for removing repetitions. However, this approach requires $\tilde{\mathcal{O}}\left(\kappa\frac{|R\bowtie_{\leq r}S|}{B}\right)$ I/Os, where κ is the expected average replication of each emitted pair, which can dominate the complexity of SimJoin.

In this section we show how the number of emitted replica can be easily reduced, without storing near pairs on the external memory. The approach requires that the probability $\Pr\left[h(x)=h(y)\right]$ can be explicitly computed in $\mathcal{O}(1)$ time and no I/Os for each pair (x,y) at distance $\leq r$. This is the case of LSHs for many metrics, including Hamming [111], ℓ_1 and ℓ_2 [117], Jaccard [95], and angular [118] distances. The approach is the following: for each near pair (x,y) that is found at the ith recursive level, with $i\geq 0$, the pair is emitted with probability $1/(p(x,y)L)^i$ and is ignored otherwise.

Theorem 5.8. *The above approach guarantees that each near pair is emitted with constant probability.*

Proof. Consider a near pair (x,y). Let G_i and H_i be random variables denoting respectively the number of subproblems at level i containing the pair (x,y), and the number of subproblems at level i where (x,y) is not found by the cache-oblivious nested loop join algorithm in Theorem 5.1. Let also K_i be a random variable denoting the actual number of times the pair (x,y) is emitted at level i. We have: (1) $\mathbb{E}\left[K_i|G_i,H_i\right]=(G_i-H_i)/(p(x,y)L)^i$ since a near pair is emitted with probability $1/(p(x,y)L)^i$ only in those subproblems where the pair is found by the join algorithm; (2) $\mathbb{E}\left[G_i\right]=(p(x,y)L)^i$ since a near pair is in the same bucket with probability $p(x,y)^i$ (it follows from the previous analysis based on standard branching); (3) $G_0=1$ since each pair exists at the beginning of the algorithm; (4) $H_\Psi=0$ since each pair surviving up to the last recursive level is found by the nested loop join algorithm.

We are interested in upper bounding $\mathbb{E}\left[\sum_{i=0}^{\Psi}K_i\right]$. We prove by induction that

$$\mathbb{E}\left[\sum_{i=0}^{l}K_i\right]=1-\frac{\mathbb{E}\left[H_l\right]}{(p(x,y)L)^l},$$

for any $0\leq l\leq\Psi$. For $l=0$ (i.e., the first call to SimJoin) the equality is verified since

$$\mathbb{E}\left[K_0\right]=\mathbb{E}\left[\mathbb{E}\left[K_0|G_0,H_0\right]\right]=\mathbb{E}\left[G_0-H_0\right]=1-\mathbb{E}\left[H_0\right],$$

since $\mathbb{E}\left[G_0\right]=G_0=0$. Consider now a generic level $l>0$. Since a pair propagates in a lower recursive level with probability $p(x,y)$, we have $\mathbb{E}\left[G_l\right]=\mathbb{E}\left[\mathbb{E}\left[G_l|H_{l-1}\right]\right]=p(x,y)L\mathbb{E}\left[H_{l-1}\right]$. Thus

$$\mathbb{E}\left[K_l\right]=\mathbb{E}\left[\mathbb{E}\left[K_l|G_l,H_l\right]\right]=\mathbb{E}\left[\frac{G_l-H_l}{(p(x,y)L)^l}\right]=\frac{\mathbb{E}\left[H_{\ell-1}\right]}{(p(x,y)L)^{l-1}}-\frac{\mathbb{E}\left[H_\ell\right]}{(p(x,y)L)^l}$$

By exploiting the inductive hypothesis, we get

$$\mathbb{E}\left[\sum_{i=0}^{l} K_i\right] = \mathbb{E}\left[K_l\right] + \mathbb{E}\left[\sum_{i=0}^{l-1} K_i\right] = 1 - \frac{\mathbb{E}\left[H_l\right]}{(p(x,y)L)^l}.$$

Since $H_\Psi = 0$, we have $\mathbb{E}\left[\sum_{i=0}^{\Psi} K_i\right] = 1$ and the claim follows. ■

We observe that the proposed approach is equivalent to use an LSH where $p(x, y) = p_1$ for each near pair. Thus, the probability of emitting a pair is at least $\Omega\left(1/\sqrt{\Psi}\right)$ as shown in the second part of Section 5.5.2.2 and $\mathcal{O}\left(\log^{3/2} N\right)$ repetitions of SimJoin suffices to find all pairs with high probability (however, the expected number of replica of a given near pair becomes $\mathcal{O}\left(\log^{3/2} N\right)$, even with the proposed approach).

Chapter 6

Load Balancing with Limited Independence

6.1 Introduction

This chapter will deal with a technical problem, motivated by the repeated use of hash functions in the algorithms of this thesis (and algorithms and data structures in general). The unifying metric of strength of the hash functions is the *independence* of the function (see Definition 1.9). Hence a question of theoretical interest is, for each algorithmic application, *how much independence is required?* A typical assumption when performing algorithmic analysis is to just assume full independence, that is, that for input size n then the hash function is n-wise independent. In fact, we did this previously in this thesis in Section 2.5 but then relaxed the assumption afterwards in Section 2.6.1. Besides the interest from a theoretic perspective, the question of how much independence is required is in fact interesting from a practical perspective: Hash functions with lower independence are as a rule of thumb faster in practice than those with higher independence, hence if it is proven that the algorithmic application needs only l_1-wise independence to work, then it can provide a speedup for the implementation to specifically pick a fast construction that provides the required l_1-independence.

We will in this chapter consider one of the most fundamental applications of random hashing: Throwing n balls into $2n$ bins using an l-wise independent hash function and analyzing the size of the largest bin. This can be seen as a load balancing as the balls can represent "tasks" and the bins represent processing units. Our main result is a family of l-wise independent hash functions, which when used in this setting implies $\Omega(n^{1/l})$ load with constant probability. This result was unknown for the entire range of l except

$l = 2$ due to Alon et al. [15], who partially posed the result of this chapter as an open problem.

As an example of the usefulness of such load balancing bounds, consider the fundamental data structure; the dictionary. Widely used algorithms books such as Cormen et a [119] teaches as the standard method to implement a dictionary as an array with *chaining* used to handle collisions between keys. Chaining here simply means that for each key, corresponding to an entry in the array, we have a linked list and when a new key-value pair is inserted then simply insert it on the end of the linked list. Clearly then, searching for a particular key-value pair takes worst-case time proportional to the size of the largest chain. Hence, if one is interested in worst-case lookup time guarantees then the expected maximal bucket size formed by the keys in the dictionary is of great importance.

6.2 Related work

We will briefly review related work on the topic of bounding the independence used as well as mention some of the popular hash function constructions.

As briefly mentioned earlier, our result is almost the generalization of Alon et al., STOC'97, specifically [15, Theorem 2]. They show that for a (perfect square) field $\mathbb{F}$ then the class $\mathcal{H}$ of all linear transformations between $\mathbb{F}^2$ and $\mathbb{F}$ has the property that when a hash function is picked uniformly at random from $h \in \mathcal{H}$ then there exists an input set of size n such that largest bucket has size at least $\sqrt{n}$.

On the upper bound side, remember that a family $\mathcal{H}_u$ of hash functions that map from $\mathcal{U}$ to $[n]$ is *universal* [120] if for a h picked uniformly from $\mathcal{H}_u$ it holds

$$\forall x \neq y \in \mathcal{U} : \mathbf{Pr}(h(x) = h(y)) \leq 1/n.$$

Universal hash functions are known to have expected largest bucket size at most $\sqrt{n} + 1/2$, hence essentially tight compared to the bound $\sqrt{n}$ lower bound of Alon et al. On the other end of the spectrum, full independence is known to give expected largest bucket size $\Theta(\log n/\text{loglog}n)$ due to a standard application of Chernoff bounds. This bound was proven to hold for $\Theta(\log n/\text{loglog}n)$-wise independence as well [52]. In Section 6.4 we state an upper bound that coincides with our new lower bound, however we consider it to be folk lore as it uses standard methods.

The line of research that considers the amount of independence required is deep. As examples, Pagh et al. [121] showed that linear probing works with 5-wise independence. For the case of ε-min-wise hashing ("almost" min-wise-hashing as used e.g. in Chapter 2

Indyk showed that $\mathcal{O}(\log \frac{1}{\varepsilon})$-wise independence is sufficient. For both of the above problems Thorup and Patrascu [122] showed optimality: They show existence of explicit families of hash functions that for linear probing is 4-wise independent and leads to $\Omega(\log n)$ probes and for ε-min-wise hashing is $\Omega(\log \frac{1}{\varepsilon})$-wise independent that implies 2ε-min-wise hashing. Additionally, they show that the popular multiply-shift hashing scheme by Dietzfelbinger et al. [19] is not sufficient for linear probing and ε-min-wise hashing.

Since the question of how much independence is needed from a practical perspective often could be rephrased "how fast of a hash function can I use and maintain algorithmic guarantees?" we will briefly recap some used hash functions. Functions with lower independence are typically faster in practice than functions with higher. The formalization of this is due to Siegel's lower bound [123] where he shows that in the cell probe model, to achieve l-wise independence and number of probes $t < l$ then you need space $l(n/l)^{1/t}$. Since space usage scales with the independence l then for high l the effects of the memory hierarchy will mean that even if the time is held constant the practical time will scale with l as cache effects etc. impact the running time.

The most used hashing scheme in practice is, as mentioned, the 2-wise independent multiply-shift by Dietzfelbinger et al. [19], which can be twice as fast [124] compared to even the simplest linear transformation $x \mapsto (ax+b) \mod p$. For 3-wise independence we have due to (analysis by) Thorup and Patrascu the simple tabulation scheme [20], which can be altered to give 5-universality [125]. For general l-wise independent hash functions the standard solution is degree $l-1$ polynomials, however especially for low l these are known to run slowly, e.g. for $l = 5$ then polynomial hashing is 5 times slower than the tabulation based solution of [125]. Alternatively for high independence the double tabulation scheme by Thorup[126], which builds on Siegels result [123], can potentially be practical. On smaller universes Thorup gives explicit and practical parameters for 100-wise independence. For generating l-wise independent variables then Christiani and Paghs constant time generator [127] performs well - their method is at an order of magnitude faster than evaluating a polynomial using fast fourier transform. We note that even though constant time generators as the above exists, the practical evaluation will actually scale with the independence, as the memory usage of the methods depend on the independence and so the effects of the underlying memory hierarchy comes to effect.

Finally, we would like to note that the paradigm of independence has its limitations in the sense that even though one can prove that l-wise independence in itself doesn't imply certain algorithmic guarantees, it is not ruled out there exists l-wise hash functions that do. That is, lower bound proofs typically construct artificial families to provide counter examples, which in practice would not come into play. As an example, consider that linear probing needs 5-wise independence to work as mentioned above but it has been proven to work with simple tabulation hashing, which only has 3-wise independence.

6.3 Our results

We consider the fundamental case of throwing n balls into $2n$ bins. The main result is a surprisingly simple l-wise independent hash function (family) which when used to throw the balls imply that with constant probability the largest bin has $\Omega(n^{1/l})$ balls. We show the theorem below.

Theorem 6.1. *Consider the setting where n balls are distributed among $2n$ bins using a random hash function h. For all constants $l \in \mathbb{N}$ there exists a family $\mathcal{H}$ of l-wise independent hash functions such that the max load is $\Omega\left(n^{1/l}\right)$ with probability $\Omega(1)$ when h is chosen uniformly at random from $\mathcal{H}$.*

An implication of Theorem 6.1 is that we now have the full understanding of the parameter space for this problem, as it was well known that independence $l = \mathcal{O}(\log n/\text{loglog}n)$ implied $\Theta(\log n/\text{loglog}n)$ balls in the largest bin. We summarize with corollary below.

Corollary 6.2. *Consider the setting where n balls are distributed among $2n$ bins using a random hash function h. There exists two families that are l-wise independent for $1 \leq l \leq n^{1/l}$ and $l > n^{1/l}$ respectively, such that given an l and drawing a h uniformly at random from the corresponding family we get mad load L as:*

(a) if $l \leq n^{1/l}$ then $L = \Omega\left(n^{1/l}\right)$ with probability $\Omega(1)$.

(b) if $l > n^{1/l}$ then $L = \Omega\left(\log n/\log\log n\right)$ with probability $\Omega(1)$.

We note that the result of Theorem 6.1 is not quite the generalization of the lower bound of Alon et al: They show $\Omega(n^{1/2})$ load for any linear transformation while our result gives a concrete worst-case l-wise scheme to achieve load $\Omega(n^{1/l})$. However, as is evident from the proof in the next section, our scheme is not that artificial: In fact it is "almost" simply standard polynomial hashing, which gives hope that the true generalization of Alon et al. can be shown.

6.4 Upper bound

We will briefly show the upper bound that matches our lower bound presented in the next section. We are unaware of literature that includes the upper bound, but note that it follows from a standard argument and is included for the sake of completeness.

Lemma 6.3. *Consider the setting where n balls are distributed among n buckets using a random hash function h. For $m = \Omega\left(\frac{\log n}{\log\log n}\right)$ and any $l \in \mathbb{N}$ such that $l < n^{1/l}$ then if h is l-independent the largest bucket size is $\mathcal{O}(m)$ with probability at least $1 - \frac{n}{m^l}$.*

Proof. Define $x^{\underline{l}}$ as $x(x-1)\dots(x-(l-1))$ for real x and positive integer l. Consider random variables $B_1, \dots, B_n$, where B_i denotes the number of balls that are distributed to bin i. By definition, the largest bucket size is $\max_i B_i$. Since $(\max_i B_i)^{\underline{l}} \le \sum_i (B_i)^{\underline{l}}$ for any threshold t we see that

$$\mathbf{Pr}(\max_i B_i \ge t) = \mathbf{Pr}\left((\max_i B_i)^{\underline{l}} \ge t^{\underline{l}}\right) \le \mathbf{Pr}\left(\sum_i (B_i)^{\underline{l}} \ge t^{\underline{l}}\right).$$

Since $\sum_i (B_i)^{\underline{l}}$ is exactly the number of ordered l-tuples being assigned to the same bucket we see that $\mathbb{E}\left(\sum_i (B_i)^{\underline{l}}\right) = n^{\underline{l}} \cdot \frac{1}{n^{l-1}}$, because there are exactly $n^{\underline{l}}$ ordered l-tuples. Hence we can apply Markov's inequality

$$\mathbf{Pr}\left(\sum_i (B_i)^{\underline{l}} \ge t^{\underline{l}}\right) \le \frac{\mathbb{E}\left(\sum_i (B_i)^{\underline{l}}\right)}{t^{\underline{l}}} = \frac{n^{\underline{l}}}{n^l} \cdot \frac{n}{t^{\underline{l}}} \le \frac{n}{t^{\underline{l}}}.$$

Since $l < n^{1/l}$ implies $l = \mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ we see that $l + m = \Theta(m)$. Letting $t = l + m$ we get the desired upper bound $\frac{n}{m^l}$ on the probability that $\max_i B_i \ge m + l$ since $(m+l)^{\underline{l}} > m^l$. ■

6.5 Lower bound

We explore the standard case of throwing n balls into $2n$ bins. The max load of a distribution is then size of the bin with the largest number of balls in it. Particularly, we wish to construct a family of hash functions that are l-wise independent (Definition 1.9) and where the maximum load is $\Omega\left(n^{1/l}\right)$.

At a high level, our hashing scheme is to divide the keys into buckets and in each bucket polynomial hashing is used if the bucket is "not full". The key point is then to see that for polynomial hashing, the probability that a particular polynomial hashes some set of keys to the same value can be bounded by the probability of all coefficients of the polynomial being zero. Using this along with the bound of the number of buckets that are "not full" provides the result.

Proof. (of Theorem 6.1) Our hash function h will be composed of two hash functions h_1 and h_2 where h_1 will be used to hash elements into "super buckets" and h_2 will be used inside the super buckets to hash into the $2n$ bins. Note that the bits that make up the hash value of a key can then simply be seen as the concatenation of the bits of the hash values from h_1 and h_2.

Let $p \in \left(n^{1/l}, 2n^{1/l}\right)$ be a prime. By Bertrand's Postulate [128] such a p exists. Assume wlog that p divides n, as if this is not the case we would simply have another super bucket of size at most p where would always use independent hashing and the result would not change.

We will hash an element from $[n]$ independently and uniformly at random into one of the n/p super buckets, i.e. we have

$$h_1 : [n] \mapsto [n/p].$$

We will in each super bucket distribute the balls to the bins based on how many are distributed to the super buckets. If the load in a super bucket is $d \le p$ then we will distribute the d balls by polynomial hashing. We define the distribution scheme to the p bins in the super bucket to be

$$h_2(x) = \Big(a_{l-1}x^{l-1} + a_{l-2}x^{l-2} + \ldots + a_0\Big) \mod p.$$

In a super bucket with load $d \le p$ we distribute the balls as $h_2(0), h_2(1), \ldots, h_2(d-1)$. If the load in a super bucket is $d > p$ then we distribute the balls uniformly at random to the p bins. Inside a super bucket the names $0, \ldots, d-1$ are given to keys according to the ordering of the keys. That is, the keys are given names according to $\text{rank}_{B_{h_1}}(x)$, which maps key x to the zero-indexed rank inside bucket B_{h_1} according to the total ordering on the keys.

We now examine the maximum bin size due to the distribution scheme as above. We have n balls and $2n$ bins and so for any super bucket $c \in [n/p]$ we have in expectation a number of balls

$$\mathbb{E}\left(|\{x \in [n] \mid h_1(x) = c\}|\right) = \frac{n/2}{n/p} = p/2.$$

Since the distribution to super buckets is independent random processes (since h_1 distributes independently and uniformly) we have by standard Chernoff bounds that

$$\mathbf{Pr}\left(|\{x \in [n] \mid h_1(x) = c\}| \in [p/5, 3p/5]\right) = 1 - e^{-\Omega(p)}. \tag{6.1}$$

By a union bound over the error probability of Equation (6.1) then with probability $1 - o(1)$ all super buckets contain between $p/5$ and $3p/5$ balls. Consider the case where load of a super bucket is $d \leq p$. Then the event that every ball maps to the same of the p possible bins happens when all random coefficients of h_2 are drawn to be 0, i.e. the probability of that event is at least

$$\mathbf{Pr}\left(a_{l-1} = a_{l-2} = \ldots = a_1 = 0\right) = \frac{1}{p^{l-1}} \tag{6.2}$$

We have n/p super buckets each with probability $\frac{1}{p^{l-1}}$ of having large load. We wish to bound the probability of the event that no bucket has large load. This corresponds to n/p boolean variables that are 1 with probability $\frac{1}{p^{l-1}}$ and 0 otherwise and we wish to bound the probability of the sum of those variables being 0. The expected number of 1 variables is

$$\frac{n}{p}\frac{1}{p^{l-1}} = \frac{n}{p^l} = \Omega(1),$$

where we use that we picked $p = \Theta(n^{1/l})$. It follows directly from Chernoff bounds that the probability of there being no large buckets is $1 - \Omega(1)$.

In a super bucket where all balls map to the same bin, we will by Equation (6.1) have at least $p/5$ balls, hence the max load is $\Theta(p) = \Theta\left(n^{1/l}\right)$ which completes the proof for (a) of the theorem.

Part (b) of the theorem follows trivially: In the case of $d \leq p$ balls in a bin we use polynomial hashing by a degree $l-1$ polynomial, which is known to be l-wise independent over the p bins of the super bucket (see e.g. [129]). If the load of a super bucket is $d > p$ then we distribute the balls fully independently to the bins. This completes the proof. ■

Since it is well known that using $\mathcal{O}(\log n/\text{loglog}n)$-wise uniform hash function to distribute the balls will imply $\Omega\left(\log n/\log\log n\right)$ maximum load, Corollary 6.2 provides the full understanding of the maximum bin size.

Proof. (of Corollary 6.2) Part (a) follows directly from Theorem 6.1. Part (b) follows since $l > n^{1/l}$ implies $l > \log n/\log\log n$ and so we apply the $\Omega\left(\log n/\log\log n\right)$ bound from [52]. ■

Chapter 7

Summary and open problems

Here we will make concluding remarks on all the presented results as well as discuss possible future directions of the work.

In general, the thesis deals with the computation of dot products and set intersection sizes of various kinds. The motivation is that methods to compute set intersection sizes and dot products are highly applicable in several areas and due to the "data flood" the demand for algorithms with lower time and space complexity is present. The thesis presents results that beat state of the art methods for summarizing set intersection sizes, sparse matrix multiplication in both the I/O and the word-RAM model, similarity join in the I/O model as well as providing the full understanding of how large the biggest bin is when hashing n balls to $\mathcal{O}(n)$ bins using a hash function with limited independence.

In Chapter 2 we presented a new method to "summarize" set intersection: For each set in the input, we compute a randomized subset - a summary - of each set, such that by querying only those summaries we can answer approximately intersection size queries on the original sets. Our new method relies on one-permutation min-wise hashing, that is, computing one random permutation and storing the k first elements whereas the most closely related previous method computed k permutations and stored the first element of each. For m sets in the input our method yields a factor $\mathcal{O}(m/\log m)$ improvement on the space used to achieve an (ε, δ)-estimate of the intersection size, as compared to the k permutation method. However, for constant number of input sets our new lower bound of $\Omega(\varepsilon^{-2}(n/t))$ bits per set implies that the old k-permutation method was in fact optimal and that our new method is optimal up to log-factors for non-constant m. In terms of future work directly related to the results mentioned we have the two obvious ones, namely improving the lower bound to match the k-permutation upper bound entirely (currently missing is a factor $\log(1/\delta)$) and also improve the upper bound to be tight, if possible.

Open Problem 7.1. Show a lower bound on summarizing set intersection that holds already for two sets of size n and that states the required space is $\Omega(\varepsilon^{-2}(n/t)\log(1/\delta))$ bits to achieve an (ε, δ)-estimate on the set intersection size t.

Open Problem 7.2. Does there exist an upper bound on the space usage for summarizing set intersection that improves the dependence on the number of sets m, e.g. the dependency is $\log m$ in our estimator.

If the answer to Open Problem 7.2 is that the space usage must depend on m, it would likewise be interesting to see a lower bound that incorporates the number of sets m into the space usage. A possible proof direction could be via multi-party communication complexity, see e.g. [130], which incorporates several players contrary two only two, Alice and Bob, in the two-party one-way model used for the lower bound in this thesis.

In Chapter 3 we focused on the problem of computing the product of two matrices A and C where the number of nonzero entries in A and C was N and the number of nonzero entries of the output matrix AC was Z. In the I/O model we improved the current start of the art by a factor $M^{3/8}$ to achieve I/O complexity $\tilde{\mathcal{O}}\left(N\sqrt{Z}/(B\sqrt{M})\right)$. Contrary to most previous work our algorithm works with arbitrary cancellations of terms. This bound was further proved to be tight by extending the technique of Hong and Kung [5], in fact the lower bound is more general as it incorporates a factor $1/P$ for P processing units. An interesting future direction would be to remove the Monte Carlo component from the upper bound.

Open Problem 7.3. For matrices A and C with N nonzero entries in total and Z nonzero entries in AC, does there exist an algorithm that uses $\tilde{\mathcal{O}}\left(N\sqrt{Z}/(B\sqrt{M})\right)$ I/Os and always computes the nonzero entries of AC?

It should be clear that Open Problem 7.3 requires ideas and techniques different from those used in this thesis: We crucially use Monte Carlo methods to partition the matrix and even more crucially to handle cancellations of terms. Since the number of nonzero elementary products is bounded only in terms of the number of nonzeros in the input it seems that creating an algorithm to not waste I/Os computing elementary products that later cancel out requires significantly new techniques. Another interesting direction would be to parallelize the algorithm.

Open Problem 7.4. For matrices A and C with N nonzero entries in total, Z nonzero entries in AC and P being the number of processing units, does there exist an algorithm that uses $\tilde{\mathcal{O}}\left(N\sqrt{Z}/(PB\sqrt{M})\right)$ I/Os in the parallel external memory model?

It seems that the main hurdle to parallelize the method presented in this thesis is the partitioning step, at least it would seem that it would be difficult to parallelize for larger P than the number of partitions. The size estimation sub-procedure is based on distinct items estimation in a streaming setting and hence should parallelize easily. A parallel version algorithm may even be interesting in practice (in particular if the Monte Carlo component could be removed), as the number of hidden log-factors in the stated complexity is not that large. Any algorithm that solves Open Problem 7.4 will not work for arbitrary P and it would be interesting what amount of parallelization can be achieved.

In the word-RAM model we considered the setting where the input matrices were $U \times U$ in dimensions and dense while the product had Z output entries. Further, we here assume that we work over a field such that Strassen-like ("fast matrix multiplication") methods are applicable. In this setting we presented an algorithm that computes the matrix product using $\tilde{\mathcal{O}}\left(U^2(Z/U)^{\omega-2} + Z + N\right)$ word-RAM operations, which beats current state of the art methods when $Z = \omega(U)$ (see Table 3.1). The algorithm was based on partitioning the input into smaller dense subproblems that could then be solved efficiently using the fast square matrix multiplication black box. Besides the problem of removing the Monte Carlo component of the algorithm, an intriguing direction for this setting is to be able to exploit sparseness in the input as well as the output. To the knowledge of the author, there doesn't exist a nontrivial upper bound that depends on density N instead of dimension U in this setting.

Open Problem 7.5. For matrices A and C over a field, with N nonzero entries in total and Z nonzero entries in AC, does there exist an algorithm that uses a number of operations proportional to N, e.g. $\tilde{\mathcal{O}}\left(N(Z/\sqrt{N})^{\omega-2} + Z + N\right)$?

The main hurdle for Open Problem 7.5 is that in order to exploit the fast matrix multiplication black box, one needs to invoke it on dense input matrices or dense sub-parts of input matrices, since the black box uses $\mathcal{O}(U^\omega)$ for two $U \times U$ matrices. And so the problem is to come up with a partitioning scheme that can somehow group the nonzero entries of the input together in a way that preserves the output. The complexity proposed in Open Problem 7.5 seems to be the "correct" one to aim for: For $N = U^2$ it corresponds to our complexity.

In Chapter 4 we examined the setting where we are interested in computing the probabilities of three random events occurring at the same time, where we knew the single probabilities as well as all the pair-probabilities. The problem was well-motivated by applications in recommender systems, where this could correspond to predicting a purchase. In particular we considered the setting where the probability of all three events occurring was so low that the event was not observed in the sample of the dataset that was available to us. Our proposed solution to this problem is to use the maximum entropy estimate. The theoretical contribution was an explicit formula that gives an approximation to the maximum entropy estimate under certain assumptions. Through experiments we observed that the maximum entropy estimate of the triple probabilities was superior to the estimates coming from assuming independence and extrapolation, which often times gave estimates that were not useful. The standing open problem, as also noted by Meo [89], is an explicit formula for the precise maximum entropy estimate.

Open Problem 7.6. Does there exist a closed form of the maximum entropy estimate?

Even such a formula for triple probabilities would be interesting, as this would enable faster computation of the maximum entropy estimate, which judging by our experiments should see more use than it currently does.

In Chapter 5 we examined the problem of computing the similarity join of two relations in an external memory setting. Our new cache-aware algorithm of Section 5.5.1 and cache-oblivious algorithm of Theorem 5.3 improve upon current state of the art by around a factor of around M/B I/Os unless the number of c-near pairs is huge (more than NM). We believe this is the first cache-oblivious algorithm for similarity join, and more importantly the first subquadratic algorithm whose I/O performance improves significantly when the size of internal memory grows. It would be interesting to investigate if our approach is also practical, which it seems it might be after some parameter tuning. Our I/O bound is probably not easy to improve significantly, but interesting open problems are to remove the error probability of the algorithm and to improve the implicit dependence on dimension in B and M: We assumed for simplicity that the unit of M and B is number of points, but in general we may get tighter bounds by taking into account the gap between the space required to store a point and the space for e.g. hash values. The result in this thesis is made with general spaces in mind and so an interesting direction would be to make a more specific algorithm for a fixed space.

Open Problem 7.7. For a specific high-dimensional metric space, does there exist a similarity join algorithm that significantly improves upon the I/O complexity of the algorithm presented here?

Finally in Chapter 6 we examined the fundamental technical problem of throwing n balls into $2n$ bins using a l-wise independent hash function. The quantity of interest was then the relationship between the size of the largest bin (the load) and the amount of independence used. Here it was known that full independence gives $\Theta(\log n/\text{loglog}n)$ load and $l = 2$ implies load $> \sqrt{n}$. We provided a family of hash functions where when one was picked at random and used to distribute the balls into the bins then with constant probability the load would be $\Omega(n^{1/l})$. Since this new bound completes our understanding of this setting entirely the most interesting open problems related to this work is to examine the independence needed for popular algorithms to work. As our family is “artificial” in the sense that it is a worst-case family of functions constructed specifically to imply high load, the natural related question is to replace the family with degree $l-1$ polynomial hashing and see if the bound still applies.

Open Problem 7.8. Distributing n balls to $\mathcal{O}(n)$ bins using a function picked uniformly at random from the family of degree $l-1$ polynomials, is the load $\Omega(n^{1/l})$ with constant probability?

We will note that this open problem is mainly of algebraic interest, at least for small l, since polynomial hashing has been shown experimentally to run quite slow (see Section 6.2 for discussion). However, as noted by Alon et al. [15], it is an interesting problem to examine the strength of polynomials as they examined linear maps.

Appendix A

Communication Complexity Lower Bound Proof Details

This appendix contains the proof details for the communication complexity lower bound of Section 2.4 which implies Theorem 2.2.

A.1 The GapAnd Problem

For bit vectors $\mathbf{z}, \mathbf{z}'$ of the same length, let $\mathrm{AND}(\mathbf{z}, \mathbf{z}')$ be the vector $\mathbf{z}''$ in which $z''_i = z_i \wedge z'_i$. For a vector $\mathbf{z}$, let $\mathrm{wt}(\mathbf{z})$ denote its Hamming weight, i.e., the number of its coordinates equal to 1.

Definition A.1. In the GapAnd problem, Alice and Bob have $\mathbf{z}, \mathbf{z}' \in \{0,1\}^r$, respectively. We define GapAnd to be:

$$\begin{aligned} &1 \text{ if } wt(AND(\mathbf{z}, \mathbf{z}')) \geq \frac{r}{4} + C\varepsilon r \\ &0 \text{ if } wt(AND(\mathbf{z}, \mathbf{z}')) \leq \frac{r}{4} - C\varepsilon r. \end{aligned}$$

This is a promise problem, and if neither case occurs, the output can be arbitrary.

A.2 The Index Problem

Consider the following Index problem.

Definition A.2. In the Index problem, Alice has an input $\mathbf{Y} \in \{0,1\}^d$ and Bob has an input $K \in [d]$, where $\mathbf{Y}$ and K are independent and uniformly distributed over their respective domains. We define Index to be:

$$1 \text{ if } Y_K = 1$$
$$0 \text{ if } Y_K = 0$$

Suppose $\mathbf{W}$ is the public coin and κ is the set of randomized one-way δ-error protocols Π. Let γ denote this distribution on the inputs. Say a 1-way protocol Π for Index with private randomness R and public randomness $\mathbf{W}$ is δ-error if

$$\mathbf{Pr}_{(\mathbf{Y},K)\sim\gamma,R,\mathbf{W}}[\Pi(\mathbf{Y},K,R,\mathbf{W}) = \mathbf{Y}_K] \geq 1-\delta.$$

Let $M(\mathbf{Y})$ be the message function associated with the 1-way protocol Π (which is a randomized function of R and $\mathbf{W}$). Let

$$CIC_{\delta}^{\gamma,1-way}(\mathsf{Index}) = \min_{\Pi\in\kappa} I(M(\mathbf{Y});\mathbf{Y} \mid \mathbf{W}).$$

Fact A.3. For $\delta \leq \frac{1}{2} - \Omega(1)$, $CIC_{\delta}^{\gamma,1-way}(\mathsf{Index}) = \Omega(d)$.

Proof. We note that this fact is folklore, but existing references, e.g., Theorem 5.5 of [131] only explicitly state the bound for deterministic protocols, whereas we want such a bound for protocols with both private randomness and public randomness $\mathbf{W}$. We provide the simple proof here.

Let Π be a δ-error protocol with (randomized) message function M and $Y = (Y_1,\ldots,Y_d)$. By the chain rule,

$$I(M(\mathbf{Y});\mathbf{Y} \mid \mathbf{W}) = \sum_{i=1}^{d} I(M(\mathbf{Y});Y_i \mid Y_1,\ldots,Y_{i-1},\mathbf{W}).$$

By independence and the fact that conditioning cannot increase entropy,

$$\sum_{i=1}^{d} I(M(\mathbf{Y});Y_i \mid Y_1,\ldots,Y_{i-1},\mathbf{W}) \geq \sum_{i=1}^{d} I(M(\mathbf{Y});Y_i \mid \mathbf{W}).$$

If Π is δ-error for $\delta = 1/2 - \Omega(1)$, then by Markov's inequality, for an $\Omega(1)$ fraction of i, $\Pi(\mathbf{Y}, i) = Y_i$ with probability $1/2 + \Omega(1)$. Call such an i *good.* Then

$$\begin{aligned}\sum_{i=1}^{d} I(M(\mathbf{Y});Y_i, \mid \mathbf{W}) &\geq \Omega(d)\cdot \min_{\text{good } i} I(M(\mathbf{Y});Y_i, \mid \mathbf{W})\\ &= \Omega(d)\cdot \min_{\text{good } i}(1 - H(Y_i \mid M(\mathbf{Y}),\mathbf{W})).\end{aligned}$$

By Fano's inequality (Fact 2.7) and using that i is good, we have $H(Y_i \mid M(\mathbf{Y}), \mathbf{W}) = 1 - \Omega(1)$. This completes the proof. ■

A.3 Proof of Theorem 2.2

Proof. It suffices to prove the theorem for a sufficiently small constant probability of error δ, since

$$R_{1/3}^{1-way}(\mathsf{GapAndIndex}) = \Theta(R_\delta^{1-way}(\mathsf{GapAndIndex}).$$

Let Π be a 1-way randomized (both public and private) δ-error protocol for $\mathsf{GapAndIndex}$. For ease of presentation, we let $M = M(\mathbf{X})$ when the input $\mathbf{X}$ is clear from context. Note that M also implicitly depends on Alice's private coins as well as a public coin $\mathbf{W}$. We need to show that $I(M; \mathbf{X} \mid \mathbf{W})$ is $\Omega(rd)$, for $r = \Theta(\varepsilon^{-2})$.

We start with the following claim, which does not directly look at the information Π conveys about its inputs, but rather the information Π conveys about certain bits in its input.

Claim A.4. $I(M; X_{i^1}^1, \dots, X_{i^r}^r \mid \mathbf{W}) = \Omega(r)$.

Proof. We will need the following fact, which follows from work by Braverman et al. [132].

Fact A.5. ([132]) Let ρ be the uniform distribution on bits $c^1, \dots, c^r$ and $d^1, \dots, d^r$. Let $\mathbf{C} = (C^1, \dots, C^r)$ and $\mathbf{D} = (D^1, \dots, D^r)$ for vectors $\mathbf{C}$ and $\mathbf{D}$ drawn from ρ.

There is a sufficiently small constant δ for which for any private randomness protocol Π which errs with probability at most δ on GapAnd, over inputs $\mathbf{C}$ and $\mathbf{D}$ drawn from ρ and the private randomness of Π and the public randomness $\mathbf{W}$, satisfies

$$I(\Pi(\mathbf{C}, \mathbf{D}); \mathbf{C}, \mathbf{D} \mid \mathbf{W}) = \Omega(r).$$

Proof. The work of Braverman et al. [132] establishes this for the problem of deciding if $\sum_{i=1}^r (C^i \oplus D^i) \geq r/2 + \sqrt{r}$ or $\sum_{i=1}^r (C^i \oplus D^i) \leq r/2 - \sqrt{r}$, which corresponds to the Hamming distance $\Delta(\mathbf{C}, \mathbf{D})$ of vectors drawn from ρ.

If $\mathrm{wt}(\mathbf{C})$ denotes the Hamming weight of $\mathbf{C}$, then we have

$$wt(\mathbf{C}) + wt(\mathbf{D}) - 2 \cdot \mathsf{And}(\mathbf{C}, \mathbf{D}) = \Delta(\mathbf{C}, \mathbf{D}),$$

where $\mathsf{And}(\mathbf{C}, \mathbf{D})$ is the number of coordinates i for which $C^i = D^i = 1$. Therefore, if Alice and Bob exchange $\mathrm{wt}(\mathbf{C})$ and $\mathrm{wt}(\mathbf{D})$ using $2 \log r$ bits, then together with a protocol Π for GapAnd, they can solve this Hamming distance problem. It follows that

$$I(\Pi(\mathbf{C},\mathbf{D}),\mathrm{wt}(\mathbf{C}),\mathrm{wt}(\mathbf{D});\mathbf{C},\mathbf{D}\mid\mathbf{W})=\Omega(r),$$

and so by the chain rule for mutual information one has $I(\Pi(\mathbf{C},\mathbf{D});\mathbf{C},\mathbf{D}\mid\mathbf{W})=\Omega(r)-I(\mathrm{wt}(\mathbf{C}),\mathrm{wt}(\mathbf{D});\mathbf{C},\mathbf{D}\mid\mathbf{W},\Pi(\mathbf{C},\mathbf{D}))=\Omega(r)-H(\mathrm{wt}(\mathbf{C}),\mathrm{wt}(\mathbf{D}))=\Omega(r)-2\log r=\Omega(r)$. ■

First, observe that if $\mathbf{I}$ denotes a uniformly random value of $\mathbf{i}$, then

$$\begin{aligned}&I(M;X^1_{I^1},\ldots,X^r_{I^r}\mid\mathbf{I},\mathbf{W})\\&=H(M\mid\mathbf{I},\mathbf{W})-H(M\mid X^1_{I^1},\ldots,X^r_{I^r},\mathbf{I},\mathbf{W})\\&=H(M)-H(M\mid X^1_{I^1},\ldots,X^r_{I^r})\\&=I(M;X^1_{I^1},\ldots,X^r_{I^r}\mid\mathbf{W}),\end{aligned}$$

where we use that M and $X^1_{I^1},\ldots,X^r_{I^r}$ are jointly independent of $\mathbf{I}$, conditioned on $\mathbf{W}$.

Hence, using also the independence of $\mathbf{I}$ and $\mathbf{W}$,

$$\begin{aligned}&I(M;X^1_{I^1},\ldots,X^r_{I^r}\mid\mathbf{W})\\&=I(M;X^1_{I^1},\ldots,X^r_{I^r}\mid\mathbf{I},\mathbf{W})\\&=\sum_i I(M;X^1_{i^1},\ldots,X^r_{i^r}\mid\mathbf{I}=\mathbf{i},\mathbf{W})\cdot\mathbf{Pr}[\mathbf{I}=\mathbf{i}]\\&\geq\frac{1}{2}\min_{\mathbf{i}}I(M;X^1_{i^1},\ldots,X^r_{i^r}\mid\mathbf{I}=\mathbf{i},\mathbf{W}).\end{aligned}$$

We claim that for each $\mathbf{i}$, $I(M;X^1_{i^1},\ldots,X^r_{i^r}\mid\mathbf{I}=\mathbf{i},\mathbf{W})=\Omega(r)$. To see this, define a 1-way protocol $\Pi_{\mathbf{i}}$ for GapAnd as follows. Alice and Bob are given inputs $\mathbf{C}$ and $\mathbf{D}$ to GapAnd, respectively, distributed according to ρ. For each $j\in[r]$, Alice sets $X^j_{i^j}=C^j$, while Bob sets $B^j=D^j$. Alice then chooses an independent uniform random bit for X^j_k for each j and $k\neq i^j$. The players then run the protocol $\Pi(\mathbf{X},\mathbf{i},\mathbf{B})$, and outputs whatever Π outputs.

By construction, $\Pi_{\mathbf{i}}(\mathbf{C},\mathbf{D})=\mathsf{GapAnd}(\mathbf{X},\mathbf{i},\mathbf{B})$, and so the correctness probability of $\Pi_{\mathbf{i}}$ is at least $1-\delta$.

Moreover, if $M_{\mathbf{i}}$ denotes the message function of Alice in $\Pi_{\mathbf{i}}$, then by construction we have that for a sufficiently small constant δ,

$$\begin{aligned}&I(M;X^1_{i^1},\ldots,X^r_{i^r}\mid\mathbf{I}=\mathbf{i},\mathbf{W})\\&=I(M_{\mathbf{i}}(X^1_{i^1},\ldots,X^r_{i^r});X^1_{i^1},\ldots,X^r_{i^r}\mid\mathbf{W})=\Omega(r)\end{aligned}$$

using Fact A.5. ■

By Claim A.4 and the chain rule, for $\Omega(1)$ fraction of $j \in [r]$ we have $I(M; X^j_{I^j} \mid X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}, \mathbf{W}) = \Omega(1)$. Call such an index j *informative.* For each informative j, a value x of the vector $(X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}})$ is *informative* if $I(M; X^j_{I^j} \mid (X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x, \mathbf{W}) = \Omega(1)$. Since

$$I(M; X^j_{I^j} \mid X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}, \mathbf{W}) = \Omega(1),$$

it follows that an $\Omega(1)$ fraction of x are informative for an informative j.

We now lower bound $I(M(\mathbf{X}); \mathbf{X} \mid \mathbf{W})$. Let $\mathbf{X}^{<j} = (X^1, \ldots, X^{j-1})$. Applying the chain rule, as well as the definition of informative and the bounds on informative j and x above,

$$\begin{aligned}
&I(M(\mathbf{X}); \mathbf{X} \mid \mathbf{W}) \\
&= \sum_{j=1}^{r} I(M(\mathbf{X}); X^j \mid \mathbf{X}^{<j}, \mathbf{W}) \\
&\geq \sum_{j=1}^{r} I(M(\mathbf{X}); X^j \mid X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}, \mathbf{W}) \\
&= \sum_{j=1}^{r} \sum_{x} I(M(\mathbf{X}); X^j \mid (X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x, \mathbf{W}) \\
&\qquad \cdot \mathbf{Pr}[(X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x] \\
&\geq \sum_{\text{inform.} j,x} I(M(\mathbf{X}); X^j \mid (X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x, \mathbf{W}) \\
&\qquad \cdot \mathbf{Pr}[(X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x] \\
&\geq \Omega(r) \cdot \min_{\text{inform.} j,x} I(M(\mathbf{X}); X^j \mid (X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x, \mathbf{W}),
\end{aligned}$$

where the first inequality follows from the fact that X^j is independent of $\mathbf{X}^{<j}$, together with the fact that conditioning cannot increase entropy.

We now lower bound

$$\min_{\text{informative } j,x} I(M(\mathbf{X}); X^j \mid (X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x, \mathbf{W}).$$

To do so, we build a 1-way protocol $\Pi_{j,x}$ with j and x hardwired, for solving the Index problem with a uniform distribution γ on its inputs. Suppose Alice is given the random input $Y \in \{0,1\}^d$, and Bob is given the random input $K \in [d]$, where Y and K are uniformly distributed over $\{0,1\}^d$ and $[d]$, respectively. Alice and Bob create inputs for protocol Π as follows. Namely, Alice sets $X^j = Y$, and uses the hardwiring of x to set

$(X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x$. Further, Alice uses her private randomness to fill in the remaining coordinates of $X^1, \ldots, X^{j-1}$, as well as to choose $X^{j+1}, \ldots, X^r$ (all coordinates of such vectors are independent of Bob's inputs and uniformly distributed, so Alice can choose such inputs without any communication). Further, Bob sets $I^j = K$, and chooses $I^{j'}$ for $j' \neq j$ uniformly and independently in $[d]$. Bob also chooses his input $\mathbf{B}$ to be independent of all other inputs and uniformly distributed.

Given this setting of inputs, in $\Pi_{j,x}$ Alice and Bob then run protocol Π on these inputs, resulting in a message function $M'(\mathbf{Y}) = M(\mathbf{X})$. Since j and x are informative, it follows that $I(M(\mathbf{X}); X^j_{I^j} | (X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x, \mathbf{W}) = \Omega(1)$, which implies that $I(M'(\mathbf{Y}); Y_K \mid \mathbf{W}) = \Omega(1)$, or equivalently,

$$H(Y_K | M'(\mathbf{Y}), \mathbf{W}) = 1 - \Omega(1).$$

It follows from Fact 2.6 that Bob, given $M'(\mathbf{Y})$ and $\mathbf{W}$, can predict Y_k with probability $1/2 + \Omega(1)$, and solve Index on the uniform distribution γ. By Fact A.3, it follows that $I(M'(\mathbf{Y}); \mathbf{Y} \mid \mathbf{W}) = \Omega(d)$. Notice, though, that by construction of $\Pi_{j,x}$ that $I(M'(\mathbf{Y}); \mathbf{Y} \mid \mathbf{W}) = I(M(\mathbf{X}); X^j \mid (X^1_{I^1}, \ldots, X^{j-1}_{I^{j-1}}) = x, \mathbf{W})$.

We conclude that $I(M(\mathbf{X}); \mathbf{X} \mid \mathbf{W}) = \Omega(dr)$, which completes the proof. ■

Bibliography

[1] Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, September 1988. ISSN 0001-0782. doi: 10.1145/48529.48535. URL `http://doi.acm.org/10.1145/48529.48535`.

[2] Eyal Kushilevitz. Communication complexity. *Advances in Computers*, 44:331–360, 1997.

[3] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213. ACM, 1979.

[4] Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of FOCS'99*, pages 285–297, 1999.

[5] Hong Jia-Wei and H. T. Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 326–333, New York, NY, USA, 1981. ACM. doi: 10.1145/800076.802486. URL `http://doi.acm.org/10.1145/800076.802486`.

[6] Rasmus Pagh, Morten Stöckel, and David P. Woodruff. Is min-wise hashing optimal for summarizing set intersection? In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2014.

[7] Rasmus Pagh and Morten Stöckel. The input/output complexity of sparse matrix multiplication. In *Algorithms - ESA*, 2014.

[8] Riko Jacob and Morten Stöckel. Fast output-sensitive matrix multiplication. In *Algorithms - ESA*, 2015.

[9] Rasmus Pagh and Morten Stöckel. Association rule mining using maximum entropy. *CoRR*, abs/1501.02143, 2015.

[10] Rasmus Pagh, Ninh Pham, Francesco Silvestri, and Morten Stöckel. I/O-efficient similarity join. In *Algorithms - ESA*, 2015.

[11] Mathias Bæk Tejs Knudsen and Morten Stöckel. Quicksort, largest bucket, and min-wise hashing with limited independence. In *Algorithms - ESA*, 2015.

[12] Erik D. Demaine. Cache-oblivious algorithms and data structures. In *Lecture Notes from the EEF Summer School on Massive Data Sets*. BRICS, University of Aarhus, Denmark, June 27–July 1 2002.

[13] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969. ISSN 0029-599X. doi: 10.1007/BF02165411. URL `http://dx.doi.org/10.1007/BF02165411`.

[14] Rasmus Resen Amossen and Rasmus Pagh. Faster join-projects and sparse matrix multiplications. In *Proceedings of the 12th International Conference on Database Theory*, ICDT '09, pages 121–126, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-423-2. doi: 10.1145/1514894.1514909. URL `http://doi.acm.org/10.1145/1514894.1514909`.

[15] Noga Alon, Martin Dietzfelbinger, Peter Bro Miltersen, Erez Petrank, and Gábor Tardos. Is linear hashing good? In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 465–474, New York, NY, USA, 1997. ACM. ISBN 0-89791-888-6. doi: 10.1145/258533.258639. URL `http://doi.acm.org/10.1145/258533.258639`.

[16] Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009. ISBN 0521884276, 9780521884273.

[17] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1995. ISBN 0-521-47465-5, 9780521474658.

[18] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, March 2008. ISSN 0097-5397. doi: 10.1137/060658400. URL `http://dx.doi.org/10.1137/060658400`.

[19] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19 – 51, 1997. ISSN 0196-6774. doi: http://dx.doi.org/10.1006/jagm.1997.0873. URL `http://www.sciencedirect.com/science/article/pii/S0196677497908737`.

[20] Mihai Ptraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM (JACM)*, 59(3):14, 2012.

[21] M. Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 90–99, Oct 2013. doi: 10.1109/FOCS.2013.18.

[22] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, March 2004. ISSN 0097-5397. doi: 10.1137/S0097539701386216. URL `http://dx.doi.org/10.1137/S0097539701386216`.

[23] Florin Rusu and Alin Dobra. Sketches for size of join estimation. *ACM Trans. Database Syst.*, 33(3), 2008.

[24] Rishi Rakesh Sinha and Marianne Winslett. Multi-resolution bitmap indexes for scientific data. *ACM Trans. Database Syst.*, 32(3):16, 2007.

[25] Zhewei Wei and Ke Yi. Beyond simple aggregates: indexing for summary queries. In Maurizio Lenzerini and Thomas Schwentick, editors, *PODS*, pages 117–128. ACM, 2011. ISBN 978-1-4503-0660-7.

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

[27] Ping Li and Arnd Christian König. Theory and applications of b-bit minwise hashing. *Commun. ACM*, 54(8):101–109, August 2011. ISSN 0001-0782. doi: 10.1145/1978542.1978566. URL `http://doi.acm.org/10.1145/1978542.1978566`.

[28] Hagai Cohen and Ely Porat. Fast set intersection and two-patterns matching. In *Proceedings of the 9th Latin American Conference on Theoretical Informatics*, LATIN'10, pages 234–242, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-12199-3, 978-3-642-12199-9. doi: 10.1007/978-3-642-12200-2_22. URL `http://dx.doi.org/10.1007/978-3-642-12200-2_22`.

[29] Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

[30] Piotr Indyk and David P. Woodruff. Tight lower bounds for the distinct elements problem. In *Proceedings of Foundations of Computer Science (FOCS)*, pages 283–288, 2003.

[31] David P. Woodruff. Optimal space lower bounds for all frequency moments. In *SODA*, pages 167–175, 2004.

[32] Amit Chakrabarti and Oded Regev. An optimal lower bound on the communication complexity of gap-hamming-distance. *SIAM J. Comput.*, 41(5):1299–1317, 2012.

[33] T. S. Jayram. Information complexity: a tutorial. In *PODS*, pages 159–168, 2010.

[34] David P. Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *STOC*, pages 941–960, 2012.

[35] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:327–336, 1998.

[36] Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '10, pages 41–52, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0033-9. doi: 10.1145/1807085.1807094. URL `http://doi.acm.org/10.1145/1807085.1807094`.

[37] Philippe Flajolet, Eric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm.

[38] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, 1997. URL `http://dl.acm.org/citation.cfm?id=283554.283370`.

[39] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD*, KDD '09, pages 219–228, 2009. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557049. URL `http://doi.acm.org/10.1145/1557019.1557049`.

[40] Ping Li and Christian König. b-bit minwise hashing. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 671–680, 2010. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772759. URL `http://doi.acm.org/10.1145/1772690.1772759`.

[41] Ping Li, Arnd Christian König, and Wenhao Gui. *b*-bit minwise hashing for estimating three-way similarities. In *Proceedings of Annual Conference on Neural Information Processing Systems (NIPS)*, pages 1387–1395, 2010.

[42] Edith Cohen and Haim Kaplan. Leveraging discarded samples for tighter estimation of multiple-set aggregates. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 251–262, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-511-6. doi: 10.1145/1555349.1555379. URL `http://doi.acm.org/10.1145/1555349.1555379`.

[43] Edith Cohen, Haim Kaplan, and Subhabrata Sen. Coordinated weighted sampling for estimating aggregates over multiple weight assignments. *Proc. VLDB Endow.*, 2 (1):646–657, August 2009. ISSN 2150-8097. URL `http://dl.acm.org/citation.cfm?id=1687627.1687701`.

[44] Edith Cohen and Haim Kaplan. What you can do with coordinated samples. In *APPROX-RANDOM*, volume 8096 of *Lecture Notes in Computer Science*, pages 452–467. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40327-9. doi: 10.1007/978-3-642-40328-6_32. URL `http://dx.doi.org/10.1007/978-3-642-40328-6_32`.

[45] Ping Li, Art Owen, and Cun-Hui Zhang. One permutation hashing for efficient search and learning. *CoRR*, abs/1208.1259, 2012.

[46] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[47] Devdatt Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *RANDOM STRUCTURES & ALGORITHMS*, 13:99–124, 1996.

[48] Peter Neumann. Über den Median der Binomial- und Poissonverteilung. *Wissenschaftliche Zeitschrift der Humboldt-Universität zu Berlin. Reihe Mathematik/-Naturwissenschaften*, 16:62–64, 1967. ISSN 0863-0631.

[49] Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal of Computing*, 31(2):353–363, 2001.

[50] Philip Bille, Anna Pagh, and Rasmus Pagh. Fast evaluation of union-intersection expressions. In *Proceedings of the 18th International Symposium on Algorithms And Computation (ISAAC '07)*, pages 739–750. 2007.

[51] Mikkel Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. *STOC*, 2013.

[52] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-hoeffding bounds for applications with limited independence. *SIAM J. Discret. Math.*, 8 (2):223–250, May 1995. ISSN 0895-4801. doi: 10.1137/S089548019223872X. URL `http://dx.doi.org/10.1137/S089548019223872X`.

[53] Stijn van Dongen. *Graph Clustering by Flow Simulation.* PhD thesis, University of Utrecht, 2000.

[54] M. O. Rabin and V. V. Vazirani. Maximum matchings in general graphs through randomization. *J. Algorithms*, 10(4):557–567, December 1989. ISSN 0196-6774.

doi: 10.1016/0196-6774(89)90005-9. URL http://dx.doi.org/10.1016/0196-6774(89)90005-9.

[55] Ketan Mulmuley, UmeshV. Vazirani, and VijayV. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987. ISSN 0209-9683. doi: 10.1007/BF02579206. URL http://dx.doi.org/10.1007/BF02579206.

[56] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995. ISSN 0004-5411. doi: 10.1145/210332.210337. URL http://doi.acm.org/10.1145/210332.210337.

[57] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. ISSN 0178-4617. doi: 10.1007/BF02523189. URL http://dx.doi.org/10.1007/BF02523189.

[58] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 887–898, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1245-5. doi: 10.1145/2213977.2214056. URL http://doi.acm.org/10.1145/2213977.2214056.

[59] Alok Aggarwal and Jeffrey S. Vitter. The Input/Output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. ISSN 0001-0782.

[60] Michael A. Bender, Gerth Stølting Brodal, Rolf Fagerberg, Riko Jacob, and Elias Vicari. Optimal sparse matrix dense vector multiplication in the I/O-model. *Theory of Computing Systems*, 47(4):934–962, 2010. ISSN 1432-4350. doi: 10.1007/s00224-010-9285-4. URL http://dx.doi.org/10.1007/s00224-010-9285-4.

[61] Gero Greiner and Riko Jacob. The I/O complexity of sparse matrix dense matrix multiplication. In *LATIN 2010: Theoretical Informatics*, pages 143–156. Springer, 2010.

[62] Rasmus Pagh. Compressed matrix multiplication. *ACM Trans. Comput. Theory*, 5(3):9:1–9:17, August 2013. ISSN 1942-3454. doi: 10.1145/2493252.2493254. URL http://doi.acm.org/10.1145/2493252.2493254.

[63] R. Williams and H. Yu. *Finding orthogonal vectors in discrete structures*, chapter 135, pages 1867–1877. doi: 10.1137/1.9781611973402.135. URL http://epubs.siam.org/doi/abs/10.1137/1.9781611973402.135.

[64] Edith Cohen. Structure prediction and computation of sparse matrix products. *Journal of Combinatorial Optimization*, 2(4):307–332, 1998. ISSN 1382-6905. doi: 10.1023/A:1009716300509. URL `http://dx.doi.org/10.1023/A%3A1009716300509`.

[65] E. Cohen. Estimating the size of the transitive closure in linear time. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 190–200, Washington, DC, USA, 1994. IEEE Computer Society. ISBN 0-8186-6580-7. doi: 10.1109/SFCS.1994.365694. URL `http://dx.doi.org/10.1109/SFCS.1994.365694`.

[66] Rasmus Resen Amossen, Andrea Campagna, and Rasmus Pagh. Better size estimation for sparse matrix products. APPROX/RANDOM'10, pages 406–419, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15368-2, 978-3-642-15368-6. URL `http://dl.acm.org/citation.cfm?id=1886521.1886554`.

[67] Philippe Flajolet and G. Nigel Martin. Probabilistic Counting Algorithms for Data Base Applications. *Journal of Computer and System Sciences.*

[68] Andrew McGregor. *Algorithms for Signals, book draft.* 2013. URL `http://people.cs.umass.edu/~mcgregor/book/signals.pdf`.

[69] Robert S. Boyer and J Strother Moore. MJRTY - A fast majority vote algorithm. Technical Report AI81-32, The University of Texas at Austin, Department of Computer Sciences, February 1 1981. URL `ftp://ftp.cs.utexas.edu/pub/AI-Lab/tech-reports/UT-AI-TR-81-32.pdf`.

[70] Dror Irony, Sivan Toledo, and Alexander Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 64(9):1017–1026, 2004.

[71] Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. Space-round tradeoffs for mapreduce computations. In *Proceedings of the 26th ACM international conference on Supercomputing*, pages 235–244. ACM, 2012.

[72] Rasmus Pagh and Francesco Silvestri. The Input/Output complexity of triangle enumeration. In *Proceedings of PODS'14*, pages 224–233, 2014.

[73] Gero Greiner. *Sparse Matrix Computations and their I/O Complexity.* Dissertation, Technische Universitat Munchen, Munchen, 2012.

[74] Andrew James Stothers. On the complexity of matrix multiplication. 2010.

[75] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251 – 280, 1990. ISSN 0747-7171. doi: http://dx.doi.org/10.1016/S0747-7171(08)80013-2. URL `http://www.sciencedirect.com/science/article/pii/S0747717108800132`. Computational algebraic complexity editorial.

[76] François Le Gall. Powers of tensors and fast matrix multiplication. *arXiv preprint arXiv:1401.7714*, 2014.

[77] M.A. Iwen and C.V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Information Processing Letters*, 109(10):468 – 471, 2009. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/j.ipl.2009.01.010. URL `http://www.sciencedirect.com/science/article/pii/S0020019009000131`.

[78] Andrzej Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. In Amos Fiat and Peter Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 408–419. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04127-3. doi: 10.1007/978-3-642-04128-0_37. URL `http://dx.doi.org/10.1007/978-3-642-04128-0_37`.

[79] Timoty M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *SODA 2015*. 2015.

[80] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, July 2005. ISSN 1549-6325. doi: 10.1145/1077464.1077466. URL `http://doi.acm.org/10.1145/1077464.1077466`.

[81] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[82] E. T. Jaynes. Information Theory and Statistical Mechanics. *Physical Review*, 106:620–630, May 1957. doi: 10.1103/PhysRev.106.620.

[83] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(4): 380 –393, apr 1997. ISSN 0162-8828. doi: 10.1109/34.588021.

[84] Nikolaj Tatti. Computational complexity of queries based on itemsets. *Inf. Process. Lett.*, 98(5):183–187, June 2006. ISSN 0020-0190. doi: 10.1016/j.ipl.2006.02.003. URL `http://dx.doi.org/10.1016/j.ipl.2006.02.003`.

[85] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. ISBN 1558609016.

[86] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993. ISSN 0163-5808. doi: 10.1145/170036.170072. URL `http://doi.acm.org/10.1145/170036.170072`.

[87] Nikolaj Tatti. Maximum entropy based significance of itemsets. *Data Mining, IEEE International Conference on*, 0:312–321, 2007. ISSN 1550-4786. doi: http://doi.ieeecomputersociety.org/10.1109/ICDM.2007.43.

[88] Michael Mampaey, Nikolaj Tatti, and Jilles Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 573–581, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7. doi: 10.1145/2020408.2020499. URL `http://doi.acm.org/10.1145/2020408.2020499`.

[89] Rosa Meo. Theory of dependence values. *ACM Trans. Database Syst.*, 25(3): 380–406, September 2000. ISSN 0362-5915. doi: 10.1145/363951.363956. URL `http://doi.acm.org/10.1145/363951.363956`.

[90] R. Meo. Maximum independence and mutual information. *Information Theory, IEEE Transactions on*, 48(1):318 –324, jan 2002. ISSN 0018-9448. doi: 10.1109/18.971763.

[91] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *Information Theory, IEEE Transactions on*, 14(3):462–467, 1968. ISSN 0018-9448. doi: 10.1109/TIT.1968.1054142.

[92] Toon Calders and Bart Goethals. Non-derivable itemset mining. *Data Min. Knowl. Discov.*, 14(1):171–206, February 2007. ISSN 1384-5810. doi: 10.1007/s10618-006-0054-6. URL `http://dx.doi.org/10.1007/s10618-006-0054-6`.

[93] Monika Rauch Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of SIGIR'06*, pages 284–291, 2006.

[94] Chuan Xiao, Wei Wang, Xuemin Lin, and Jeffrey Xu Yu. Efficient similarity joins for near duplicate detection. In *Proceedings of WWW'08*, pages 131–140, 2008.

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

[96] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *Proceedings of VLDB'06*, pages 918–929, 2006.

[97] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In *Proceedings of ICDE'06*, page 5, 2006.

[98] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Detectives: detecting coalition hit inflation attacks in advertising networks streams. In *Proceedings of WWW'07*, pages 241–250, 2007.

[99] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. *IEEE Trans. Knowl. Data Eng.*, 13(1):64–78, 2001.

[100] Ahmed Metwally and Christos Faloutsos. V-smart-join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. *PVLDB*, 5(8): 704–715, 2012.

[101] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. Scaling up all pairs similarity search. In *Proceedings of WWW'07*, pages 131–140, 2007.

[102] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of VLDB'99*, pages 518–529, 1999.

[103] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of SIGMOD'09*, pages 563–576, 2009.

[104] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *PVLDB*, 5(5):430–441, 2012.

[105] Bingsheng He and Qiong Luo. Cache-oblivious nested-loop joins. In *Proceedings of CIKM'06*, pages 718–727, 2006.

[106] Bahman Bahmani, Ashish Goel, and Rajendra Shinde. Efficient distributed locality sensitive hashing. In *Proceedings of CIKM'12*, pages 2174–2178, 2012.

[107] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *Proceedings of SODA'06*, pages 1186–1195, 2006.

[108] Akash Das Sarma, Yeye He, and Surajit Chaudhuri. Clusterjoin: A similarity joins framework using map-reduce. *PVLDB*, 7(12):1059–1070, 2014.

[109] Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of SIGMOD'10*, pages 495–506, 2010.

[110] Ye Wang, Ahmed Metwally, and Srinivasan Parthasarathy. Scalable all-pairs similarity search in metric spaces. In *Proceedings of KDD'13*, pages 829–837, 2013.

[111] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of STOC'98*, pages 604–613, 1998.

[112] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. In *Proceedings of SOCG'06*, pages 154–157, 2006.

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

[114] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of FOCS'06*, pages 459–468, 2006.

[115] Jeffrey Scott Vitter. *Algorithms and Data Structures for External Memory.* Now Publishers Inc., 2008.

[116] Theodore E Harris. *The theory of branching processes.* Courier Dover Publications, 2002.

[117] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of SOCG'04*, pages 253–262, 2004.

[118] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of STOC'02*, pages 380–388, 2002.

[119] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms.* McGraw-Hill Higher Education, 2nd edition, 2001. ISBN 0070131511.

[120] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979. ISSN 0022-0000. doi: http://dx.doi.org/10.1016/0022-0000(79)90044-8. URL `http://www.sciencedirect.com/science/article/pii/0022000079900448`.

[121] Anna Pagh, Rasmus Pagh, and Milan Ruzic. Linear probing with constant independence. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 318–327, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-631-8. doi: 10.1145/1250790.1250839. URL `http://doi.acm.org/10.1145/1250790.1250839`.

[122] Mihai Ptracu and Mikkel Thorup. On the k-independence required by linear probing and minwise independence. In Samson Abramsky, Cyril Gavoille, Claude

Kirchner, Friedhelm Meyer auf der Heide, and PaulG. Spirakis, editors, *Automata, Languages and Programming*, volume 6198 of *Lecture Notes in Computer Science*, pages 715–726. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-14164-5. doi: 10.1007/978-3-642-14165-2_60. URL `http://dx.doi.org/10.1007/978-3-642-14165-2_60`.

[123] A. Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, 2004.

[124] Mikkel Thorup. Even strongly universal hashing is pretty fast. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 496–497, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics. ISBN 0-89871-453-2. URL `http://dl.acm.org/citation.cfm?id=338219.338597`.

[125] Mikkel Thorup and Yin Zhang. Tabulation based 5-universal hashing and linear probing. In *ALENEX'10*, pages 62–76, 2010.

[126] M. Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. FOCS'13*, pages 90–99, 2013.

[127] Tobias Christiani and Rasmus Pagh. Generating k-independent variables in constant time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 196–205, 2014. doi: 10.1109/FOCS.2014.29. URL `http://dx.doi.org/10.1109/FOCS.2014.29`.

[128] S. Ramanujan. A proof of bertrand's postulate. *J. Indian Math. Soc*, 11, 1919.

[129] A. Joffe. On a set of almost deterministic k-independent random variables. *Ann. Probab.*, 2(1):161–162, 02 1974. doi: 10.1214/aop/1176996762. URL `http://dx.doi.org/10.1214/aop/1176996762`.

[130] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 209–213, New York, NY, USA, 1979. ACM. doi: 10.1145/800135.804414. URL `http://doi.acm.org/10.1145/800135.804414`.

[131] Ziv Bar-Yossef. *The complexity of massive data set computations.* PhD thesis, University of California at Berkeley, 2002.

[132] Mark Braverman, Ankit Garg, Denis Pankratov, and Omri Weinstein. Information lower bounds via self-reducibility. In *CSR*, pages 183–194, 2013.