

UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

Approximation Algorithms for Similarity Measures and Tensor Contractions

Permalink

<https://escholarship.org/uc/item/3wc3x7m0>

ISBN

9798247946878

Author

Heddes, Mike

Publication Date

2026-06-03

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, available at <https://creativecommons.org/licenses/by/4.0/>

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA,
IRVINE

Approximation Algorithms for Similarity Measures and Tensor Contractions

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Mike Heddes

Dissertation Committee:
Distinguished Professor Alex Nicolau, Chair
Professor Tony Givargis
Assistant Professor Ioannis Panageas

Chapter 2 © 2023 Igor Nunes, Mike Heddes, Pere Vergés, Danny Abraham, Alex
Veidenbaum, Alex Nicolau, Tony Givargis
Chapter 3 © 2024 Mike Heddes, Igor Nunes, Tony Givargis, Alex Nicolau
Chapter 4 © 2026 Mike Heddes, Igor Nunes, Tony Givargis, Alex Nicolau
All other materials © 2026 Mike Heddes

DEDICATION

To my fiancée, Melanie, and my family for their support.

Genius is 1% inspiration, 99% perspiration.

— Thomas Edison

TABLE OF CONTENTS

	Page
LIST OF FIGURES	vi
LIST OF TABLES	vii
LIST OF ALGORITHMS	viii
ACKNOWLEDGMENTS	ix
VITA	x
ABSTRACT OF THE DISSERTATION	xiii
1 Introduction	1
1.1 A brief history of sketching algorithms	3
1.2 Concentration inequalities	5
1.2.1 Markov’s inequality	5
1.2.2 Chebyshev’s inequality	6
1.2.3 Exponential Markov’s inequality	7
1.2.4 Chernoff bound	7
1.2.5 Hoeffding’s inequality	9
1.2.6 Median trick	11
1.3 Universal hashing	11
1.3.1 k -independent hashing	12
1.4 Examples of sketching algorithms	13
1.4.1 Unique count	13
1.4.2 Inner product	15
2 Set similarity estimation	17
2.1 Introduction	17
2.2 Set similarity measures	21
2.2.1 Jaccard index	21
2.2.2 Adamic-Adar index	22
2.2.3 Resource allocation index	22
2.3 Set similarity estimators	23

2.3.1	MinHash	23
2.3.2	SimHash	24
2.3.3	Related work	25
2.4	DotHash	27
2.4.1	Computing the intersection size	28
2.4.2	Generalization to orthogonal vectors	28
2.4.3	Exploiting <i>quasi</i> -orthogonality	30
2.4.4	Estimating the Adamic-Adar index	32
2.4.5	General family of supported metrics	34
2.5	Application: link prediction	34
2.5.1	Datasets	35
2.5.2	Results	36
2.6	Application: document deduplication	38
2.6.1	Datasets	39
2.6.2	Results	40
2.7	Conclusion	41
3	Join size estimation	42
3.1	Introduction	42
3.2	Background	46
3.2.1	Streaming data	46
3.2.2	Linear sketching	48
3.2.3	AMS sketch	50
3.2.4	Count sketch	51
3.2.5	Extensions to the count sketch	53
3.2.6	Multi-join with AMS sketches	55
3.2.7	Other related work	57
3.3	UCI sketch	59
3.3.1	Key insight for preserving information	59
3.3.2	General formulation by example	60
3.3.3	Analysis	64
3.3.4	Integration with query optimizers	67
3.4	Experiments	67
3.4.1	Databases and queries	68
3.4.2	Estimation accuracy	70
3.4.3	Execution times	73
3.4.4	Comparison with learning-based methods	75
3.4.5	PostgreSQL query execution time	76
3.5	Conclusion	77
4	Approximate Tensor Network Contraction	79
4.1	Introduction	79
4.1.1	Technical overview	83
4.2	Preliminaries	85
4.2.1	Count sketch	87

4.2.2	Tensor sketch	88
4.2.3	Recursive sketch	89
4.3	Related work	90
4.4	Methods	91
4.4.1	Without loss of generality assumptions	92
4.4.2	Approximating any tensor network contraction	93
4.4.3	Exponential lower bound	96
4.4.4	Improved approximation for acyclic tensor networks	98
4.4.5	Main result	101
4.5	Applications of tensor network contraction	103
4.5.1	Join size estimation	104
4.5.2	Triangle counting	105
Bibliography		107
Appendix A Proofs		121
Appendix B Supplementary algorithms		138
Appendix C Supplementary empirical results		141

LIST OF FIGURES

	Page
1.1 Data volume worldwide from 2010 to 2024	2
1.2 Illustration of the steps in a sketching algorithm	3
2.1 Set intersection size calculation using one-hot encoding	29
2.2 Set intersection size calculation using quasi-orthogonal encoding	31
2.3 Intersection estimate bounds for DotHash	32
2.4 Link prediction accuracy results	37
2.5 Normalized average link prediction execution time	38
3.1 Streaming query-processing scheme	43
3.2 Comparison of the AMS and count sketches performing an update	52
3.3 Example join graph and corresponding SQL query	55
3.4 Comparison of the Hadamard product and circular convolution	60
3.5 Total number of entries across all columns of both the STATS and IMDB databases, grouped by the best fit Zipf parameter of each column	69
3.6 Absolute relative error comparison of methods at varying sketch sizes	71
3.7 Kernel density estimates of the power law exponents from the absolute relative error plots	73
3.8 Execution times for the three stages (initialization, sketching/training, and inference) of the methods at varying sketch/model sizes	74
3.9 Cumulative distribution function of the q-error for the STATS-CEB and JOB- light sub-queries	76
4.1 Tensor diagram of an example tensor network	81
4.2 Tensor diagram of a simple acyclic and cyclic tensor network	93
4.3 Tensor diagram of an example acyclic tensor network	101
4.4 Example SQL query and corresponding tensor network diagram	104

LIST OF TABLES

	Page
2.1 Bit string representation of sets A and B	28
2.2 Accuracy and computation time (s) results for document duplicate detection	41
3.1 Notation for join-size estimation	47
3.2 Comparison of time complexity by stage	66
3.3 Database size statistics	68
3.4 Percentage of relations among all queries by their number of joins, and the percentage of queries by their relation with the maximum number of joins .	70
3.5 Throughput in tuples processed per second	75
3.6 PostgreSQL plan execution time and percentage improvement using different cardinality estimators	77
4.1 Complexity comparison of approximate full tensor network contraction . . .	84

LIST OF ALGORITHMS

	Page
1 UCI sketch join-size estimation procedure	61
2 UCI sketch creation procedure	63
3 UCI sketch inference procedure	65
4 Approximating general full tensor network contractions	97
5 Approximating acyclic full tensor network contraction	102

ACKNOWLEDGMENTS

I would like to express deep gratitude to my committee chair and advisor, Dr. Alex Nicolau, for the opportunity to pursue my doctoral studies at UC Irvine. Over the course of this journey, I have grown immensely both as a scientist and as an individual. I am especially grateful to both Dr. Nicolau and my co-advisor, Dr. Tony Givargis, for their support and for providing me the intellectual freedom to explore my own research interests.

I would also like to thank Dr. Ioannis Panageas for serving on my dissertation committee and for his insightful feedback and guidance throughout my research.

I am grateful for the invaluable professional experiences I have had during my doctoral studies. I want to thank Dr. Narayan Srinivasa for the opportunity to intern at Intel Labs. I also want to thank Dr. Vahab Mirrokni and Dr. Thomas Fu for the opportunity to intern at Google Research. These experiences were among the highlights of my doctoral journey, and let me realize multiple of my professional ambitions.

I am also grateful for the productive collaboration I had with Igor Nunes. We conducted much of our doctoral research together and we share many of our academic successes. I want to thank him for all the fruitful intellectual discussions and for being a good friend.

Finally, I want to recognize those whose support prepared me for my doctoral studies. I want to thank Dr. Daria Meijers for her guidance and support during my undergraduate studies at the Amsterdam University of Applied Sciences. I also want to thank Dr. Dario Izzo and Dr. Marcus Märtens for the opportunity to intern with the Advanced Concepts Team at the European Space Agency (ESA). It was during my time at ESA that I was first exposed to a research environment, and it motivated me to pursue an advanced degree in Computer Science. For this, I am incredibly grateful.

The text of Chapter 2 of this dissertation is an adaptation of the material as it appears in “DotHash: Estimating Set Similarity Metrics for Link Prediction and Document Deduplication” in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2023), used with permission from ACM (CC BY 4.0). The coauthors listed in this publication are Igor Nunes, Pere Vergés, Danny Abraham, Alex Veidenbaum, Alex Nicolau, and Tony Givargis.

The text of Chapter 3 of this dissertation is an adaptation of the material as it appears in “Convolution and Cross-Correlation of Count Sketches Enables Fast Cardinality Estimation of Multi-Join Queries” in *Proceedings of the ACM on Management of Data* (2024), used with permission from ACM (CC BY-NC 4.0). The coauthors listed in this publication are Igor Nunes, Tony Givargis, and Alex Nicolau.

VITA

Mike Heddes

EDUCATION

Doctor of Philosophy in Computer Science

University of California, Irvine

2020–2026

Irvine, California

Bachelor of Science in Engineering

Amsterdam University of Applied Sciences

2015–2019

Amsterdam, Netherlands

RESEARCH EXPERIENCE

Graduate Student Researcher

University of California, Irvine

2021–2026

Irvine, California

Student Researcher

Google Research

2024–2025

New York, New York

Artificial Intelligence Research Intern

Intel Labs

2022

Santa Clara, California

Advanced Concepts Team Intern

European Space Agency

2019

Noordwijk, Netherlands

TEACHING EXPERIENCE

Teaching Assistant

University of California, Irvine

2020–2025

Irvine, California

REFEREED JOURNAL PUBLICATIONS

P. Vergés, **M. Heddes**, I. Nunes, D. Kleyko, T. Givargis, and A. Nicolau, “Classification using Hyperdimensional Computing: A Review with Comparative Analysis.” *Artificial Intelligence Review* 58, 173, 2025.

M. Heddes, I. Nunes, T. Givargis, A. Nicolau, and A. Veidenbaum, “Hyperdimensional Computing: A Framework for Stochastic Computation and Symbolic AI.” *Journal of Big Data*, 2024.

M. Heddes, I. Nunes, P. Vergés, D. Kleyko, D. Abraham, T. Givargis, A. Nicolau and A. Veidenbaum, “Torchhd: An Open Source Python Library to Support Research on Hyperdimensional Computing and Vector Symbolic Architectures.” *Journal of Machine Learning Research (JMLR)*, 2023.

N. Watkinson, F. Zaitsev, A. Shivam, M. Demirev, **M. Heddes**, T. Givargis, A. Nicolau, and A. Veidenbaum, “EdgeAvatar: An Edge Computing System for Building Virtual Beings.” *Electronics*, 2021.

REFEREED CONFERENCE PUBLICATIONS

M. Heddes, A. Javanmard, K. Axiotis, G. Fu, M. Bateni, and V. Mirrokni, “DeepCrossAttention: Supercharging Transformer Residual Connections.” In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

M. Heddes, N. Srinivasa, T. Givargis, and A. Nicolau, “Always-Sparse Training by Growing Connections with Guided Stochastic Exploration.” In *Proceedings of the 2025 International Joint Conference on Neural Networks (IJCNN)*, 2025.

M. Heddes, I. Nunes, T. Givargis, and A. Nicolau, “Convolution and Cross-Correlation of Count Sketches Enables Fast Cardinality Estimation of Multi-Join Queries.” In *Proceedings of the ACM on Management of Data (SIGMOD)*, 2024.

P. Vergés, I. Nunes, **M. Heddes**, T. Givargis, and A. Nicolau, “Molecular Classification using Hyperdimensional Graph Classification.” In *Proceedings of the 2024 International Joint Conference on Neural Networks (IJCNN)*, 2024.

P. Vergés, I. Nunes, **M. Heddes**, T. Givargis, and A. Nicolau, “Accelerating Permute and N-Gram Operations for Hyperdimensional Learning in Embedded Systems.” In *Proceedings of the 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2023.

I. Nunes, **M. Heddes**, P. Vergés, D. Abraham, A. Veidenbaum, A. Nicolau, and T. Givargis, “DotHash: Estimating Set Similarity Metrics for Link Prediction and Document Deduplication.” In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2023.

I. Nunes, **M. Heddes**, T. Givargis, and A. Nicolau, “An Extension to Basis-Hypervectors for Learning from Circular Data in Hyperdimensional Computing.” In *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

M. Heddes, I. Nunes, T. Givargis, A. Nicolau, and A. Veidenbaum, “Hyperdimensional Hashing: A Robust and Efficient Dynamic Hash Table.” In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 2022.

I. Nunes, **M. Heddes**, T. Givargis, A. Nicolau, and A. Veidenbaum, “GraphHD: Efficient Graph Classification using Hyperdimensional Computing.” In *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe (DATE)*, 2022. **Best Paper Nominee**.

SOFTWARE

Torchhd <https://github.com/hyperdimensional-computing/torchhd>
An open source Python library to support research on hyperdimensional computing and vector symbolic architectures.

ABSTRACT OF THE DISSERTATION

Approximation Algorithms for Similarity Measures and Tensor Contractions

By

Mike Heddes

Doctor of Philosophy in Computer Science

University of California, Irvine, 2026

Distinguished Professor Alex Nicolau, Chair

As modern systems grapple with massive amounts of data, traditional algorithms requiring polynomial or even linear time and space have become prohibitively expensive for some critical applications. Examples include document deduplication, link prediction, and data analytics. This has motivated research into approximation algorithms characterized by a sub-linear time or space complexity. This dissertation covers approximation algorithms, particularly those leveraging sketching and dimensionality reduction, that trade a marginal loss in precision for substantial gains in efficiency. These algorithms have formal bounds on their error and success probability, and often achieve an optimal tradeoff between accuracy and cost. Collectively, these insights provide a robust framework for the design of algorithms in the big data era.

Chapter 2 covers approximation algorithms for set similarity metrics. Set similarity metrics are a core aspect of several data mining tasks. To remove duplicate results in a Web search, for example, a common approach looks at the *Jaccard index* between all pairs of pages. In social network analysis, a much-celebrated metric is the *Adamic-Adar index*, widely used to compare node neighborhood sets in the important problem of predicting links. However, with the increasing amount of data to be processed, calculating the exact similarity between all pairs can be intractable. The challenge of working at this scale has motivated research into

efficient estimators for set similarity metrics. The two most popular estimators, *MinHash* and *SimHash*, are indeed used in applications such as document deduplication and recommender systems where large volumes of data need to be processed. Chapter 2 describes a novel method, called DotHash, which is an unbiased estimator for the weighted intersection size of two sets. DotHash can be used to estimate the Jaccard index and is the first method that can also estimate the Adamic-Adar index and a family of related metrics. This family of metrics is formally defined, theoretical bounds on the probability of estimate errors are provided, and its empirical performance is analyzed. The empirical results indicate that DotHash is more accurate than the other estimators in link prediction and detecting duplicate documents with the same complexity and similar comparison time.

Chapter 3 covers approximation algorithms in the streaming data setting. With the increasing rate of data generated by critical systems, estimating functions on streaming data has become essential. This demand has driven numerous advancements in algorithms designed to efficiently query and analyze one or more data streams while operating under memory constraints. The primary challenge arises from the rapid influx of new items, requiring algorithms that enable efficient incremental processing of streams in order to keep up. A prominent algorithm in this domain is the *AMS sketch*. Originally developed to estimate the second frequency moment of a data stream, it can also estimate the join size of the equi-join between two relations. Since then, two important advancements are the *count sketch*, a method which significantly improves upon the sketch update time, and secondly, an extension of the AMS sketch to accommodate multi-join queries. However, combining the strengths of these methods to maintain sketches for multi-join queries while ensuring fast update times is a non-trivial task, and has remained an open problem for decades. Chapter 3 describes the UCI sketch, a novel sketching method that successfully addresses this problem and has fast updates, even for sketches capable of accurately estimating the join size of complex multi-join queries. The estimator is unbiased and has the same error guarantees as the AMS-based method. Empirical results confirm the significant improvement in update time

complexity, resulting in orders of magnitude faster estimates, with equal or better estimation accuracy.

Chapter 4 generalizes and improves upon the results from Chapter 3. Specifically, Chapter 4 shows that join size estimation is equivalent to approximating a tensor network contraction, which is a fundamental mathematical operation that generalizes the dot product and matrix multiplication. It finds applications in numerous domains besides database systems, such as graph theory, machine learning, probability theory, and quantum mechanics. Tensor network contractions are computationally expensive, in general requiring exponential time and space. The prior sketching methods for tensor network contraction, however, only support acyclic tensor networks. Chapter 4 describes the first method capable of approximating arbitrary tensor network contractions, including those of cyclic tensor networks. Additionally, the prior sketching methods require a computational complexity that grows exponentially with the number of contractions. Chapter 4 describes a second method, for acyclic tensor networks, whose space and time complexity depends only polynomially on the number of contractions.

Chapter 1

Introduction

The volume of data created and consumed is growing at an ever increasing rate (see Figure 1.1). As a consequence, it becomes increasingly common that traditional algorithms, with a polynomial or even a linear time and space complexity, are prohibitively expensive. For example, imagine that we want to track the number of unique users who visited our website. There is a simple algorithm that allows us to track this count using a hash set of the n visitors seen so far. This simple algorithm has a time and space complexity of $O(n)$, that is, linear in the number of visitors, and thus seems perfectly scalable. However, it becomes costly when one considers that Google has around 83 billion monthly visitors (Statista, 2024). Moreover, we might want to track the unique visitors not just for a single webpage but for all pages of a website, further increasing the storage cost.

To mitigate the high space requirement, one might set out to create an algorithm that computes the unique count using $o(n)$ space, that is, sublinear with respect to the number of visitors. However, the $\Omega(n)$ space lower bound for the problem, discussed in Section 1.4, implies that this would be a futile attempt. One way to get around this lower bound is by relaxing the problem. If we settle for an approximation of the unique count, then Flajolet

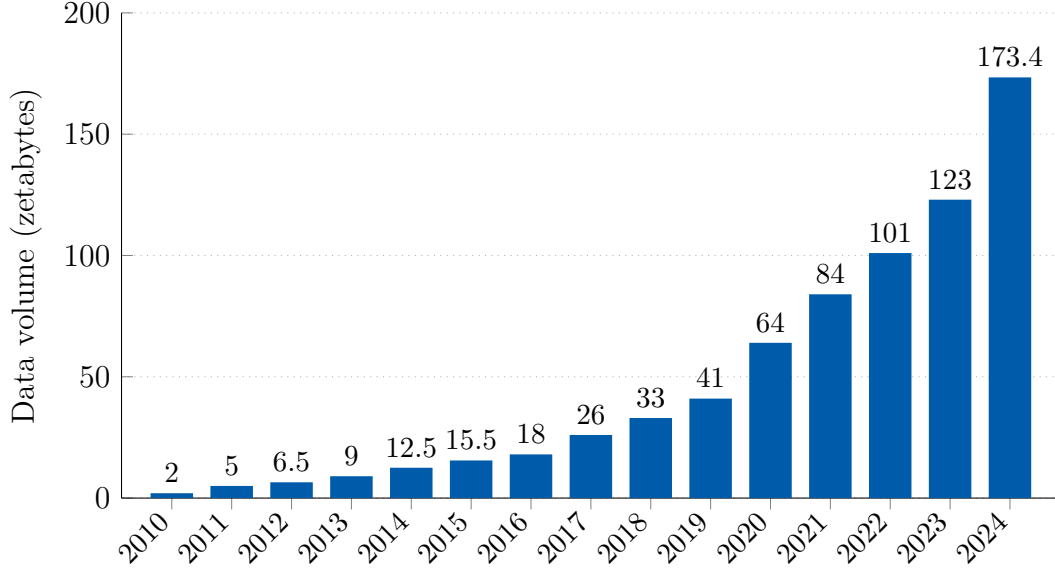


Figure 1.1. Data volume worldwide from 2010 to 2024 (Statista, 2025)

and Martin (1985) showed that we can improve upon the $\Omega(n)$ lower bound and achieve a space complexity of $O(1/\epsilon^2)$, where ϵ is the level of error. Interestingly, the space complexity of their approximation algorithm is independent of the unique count and is constant for a given level of acceptable error. As a result, their method can scale to any data volume.

Similar improvements in space or time complexity have been obtained for many other problems where approximate solutions are permissible. Notable examples include quantiles (Greenwald and Khanna, 2001; Shrivastava et al., 2004; Karnin et al., 2016), inner products (Alon et al., 1999; Cormode and Garofalakis, 2005), heavy hitters (Misra and Gries, 1982; Cormode and Muthukrishnan, 2005; Metwally et al., 2006), as well as many problems in numerical linear algebra (Woodruff, 2014). The approximation algorithms for these problems are called *sketching algorithms* because they construct a *sketch*, which is a compressed summary of a dataset that preserves essential geometric or statistical properties, to approximate answers to complex queries, often in sublinear space or time (see Figure 1.2).

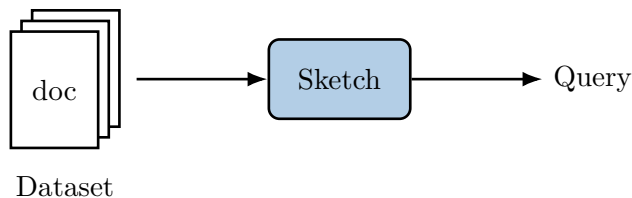


Figure 1.2. Illustration of the steps in a sketching algorithm

1.1 A brief history of sketching algorithms

This section provides a brief overview of the development of sketching algorithms. The description of the initial developments is based on the paper by Cormode (2023).

Perhaps the most basic and earliest used sketch is a uniform random sample of the data. The reservoir sampling algorithm made this sketch practical for potentially unbounded streams by creating the random sample in a single pass over the stream. The algorithm was already known in the 70s due to Waterman¹. Besides random sampling, the first sketching algorithm is perhaps the Bloom filter (Bloom, 1970) for approximate set membership queries. Given a set, the sketch can tell whether an item is for sure not in the set, otherwise the item might be in the set. Thus, the algorithm only suffers from false positives. The Bloom filter is commonly used as an efficient check to potentially avoid an expensive operation and has found many applications (Broder and Mitzenmacher, 2004). Unlike the other sketching algorithms, the Bloom filter does not provide an asymptotic improvement in complexity over the exact membership test, it does, however, significantly reduce the constant of proportionality.

Among the first sketching algorithms that provided an improvement in asymptotic complexity over the exact algorithm is the Morris counter (Morris, 1978). The algorithm allows counting to within ϵ relative error of n with probability $1 - \delta$ using $O(\epsilon^{-2} \log(1/\delta) \log \log(n/(\epsilon\delta)))$ bits. For constant ϵ and δ , this is an improvement compared to the $O(\log n)$ bits for the exact count. Flajolet and Martin (1985) subsequently introduced the first idealized sub-linear approximation algorithm for the unique count. The algorithm is idealized because

¹See the discussion in: <https://markkm.com/blog/reservoir-sampling/>

it assumes a truly random hash function, which requires $\Omega(n)$ bits for a universe of size n (Wegman and Carter, 1979). The non-idealized KMV algorithm by Bar-Yossef et al. (2002) uses 2-wise independent hash functions instead, which require only $O(\log n)$ bits, as will be discussed in Section 1.3.1. Kane et al. (2010) later obtained an optimal space, non-idealized algorithm for the problem. In practice, the HyperLogLog algorithm (Flajolet et al., 2007) is more commonly used due to its superior empirical performance despite assuming a truly random hash function and having a worse asymptotic complexity.

The seminal work by Alon et al. (1996) formalized the data stream computation model and introduced the AMS sketch, which provided the first sublinear algorithm for estimating the second frequency moment and the inner product. The second frequency moment F_2 is the sum of squared frequencies, that is, $F_2 = \sum_x f_t(x)^2$, where $f_t(x)$ is the frequency (or count) of x in the stream at time t . Similarly, $F_0 = \sum_x f_t(x)^0$, which is the number of unique items in the stream. Alon et al. (1996) also provided space lower bounds for computing frequency moments in a streaming setting, further motivating the need for approximation algorithms.

In parallel, the Johnson-Lindenstrauss (JL) lemma (1984) proved that any set of n points in a high-dimensional Euclidean space can be embedded in a space of dimension $O(\epsilon^{-2} \log n)$ while preserving pairwise distances up to a $1 \pm \epsilon$ factor. Since any method that satisfies the JL lemma can also be used to approximate the second frequency moment, the two areas converged with the works of Alon et al. (1999) and Achlioptas (2003).

Constructions of the JL lemma rely heavily on random projections, that is, given vectors $x_1, x_2, \dots, x_n \in \mathbb{R}^d$ the methods embed them as $\Pi x_1, \Pi x_2, \dots, \Pi x_n$ using a random matrix $\Pi \in \mathbb{R}^{m \times d}$ with $m = O(\epsilon^{-2} \log n)$. Many variations of the random projection matrix Π , also known as the JL matrix, have been proposed. The classical result uses independent standard normal random variables for each element of Π . Alon et al. (1999) and Achlioptas (2003) showed that a matrix of random signs also satisfies the JL lemma. Since then, two directions have been explored to speed up the embedding process: using a sparse JL matrix (Achlioptas,

2003; Matoušek, 2008; Kane and Nelson, 2014) and using a structured JL matrix (Ailon and Chazelle, 2006). The JL lemma and JL matrices have found many applications, including in numerical linear algebra, where it is applied to problems such as least squares regression and low rank approximation (Woodruff, 2014). In 2017, Larsen and Nelson proved that the JL lemma is optimal.

1.2 Concentration inequalities

The correctness of randomized approximation algorithms hinges on bounding the probability that the randomized estimate deviates significantly from the true value. This is achieved via concentration inequalities, which will be discussed in this section.

We will first introduce the expectation $\mathbb{E} X$ of a random variable X , which is an important property of a random variable and will appear in multiple concentration inequalities. The expected value is given by the sum over all possible realizations of X weighted by their probability.

Definition 1.1 (Expected value). *Let X be a discrete random variable taking values in some countable subset $S \subset \mathbb{R}$, then*

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

1.2.1 Markov's inequality

Markov's inequality is the most fundamental bound since many concentration inequalities are based on Markov's inequality. The inequality states that the probability that a nonnegative random variable X is at least t is at most the expectation of X divided by t .

Lemma 1.1 (Markov's inequality). *If X is a nonnegative random variable and $t > 0$, then*

$$\Pr(X \geq t) \leq \frac{\mathbb{E} X}{t}.$$

Proof. Using the Iverson bracket notation, we have $[X \geq t] = 1$ if the event $X \geq t$ occurs, and $[X \geq t] = 0$ if $X < t$. Then, given $t > 0$, $t[X \geq t] \leq X$ which is clear from the two possible outcomes of $X \geq t$. If $X < t$, then $[X \geq t] = 0$, and so $t[X \geq t] = 0 \leq X$. Otherwise, we have $X \geq t$, for which $[X \geq t] = 1$ and so $t[X \geq t] = t \leq X$. Taking the expectation of both sides yields $\mathbb{E} t[X \geq t] \leq \mathbb{E} X$. Now, using the linearity of expectations and $\mathbb{E}[E] = \Pr(E)$ for any event E , the left side of this inequality is the same as $t \Pr(X \geq t)$. Thus, we have $t \Pr(X \geq t) \leq \mathbb{E}(X)$ and since $t > 0$, we can divide both sides by t . $\square$

1.2.2 Chebyshev's inequality

Chebyshev's inequality applies Markov's inequality to the variance of the random variable to obtain a stronger concentration.

Lemma 1.2 (Chebyshev's inequality). *If X is a random variable and $t > 0$, then*

$$\Pr(|X - \mathbb{E} X| \geq t) \leq \frac{\text{Var}(X)}{t^2}.$$

Proof. Since $t > 0$, $|X - \mathbb{E} X| \geq t$ implies $(X - \mathbb{E} X)^2 \geq t^2$, where $(X - \mathbb{E} X)^2$ is a nonnegative random variable and $t^2 \geq 0$. We can thus apply Markov's inequality. Specifically,

$$\Pr(|X - \mathbb{E} X| \geq t) = \Pr((X - \mathbb{E} X)^2 \geq t^2) \leq \frac{\mathbb{E}[(X - \mathbb{E} X)^2]}{t^2} = \frac{\text{Var}(X)}{t^2},$$

where the last equality uses the definition of the variance. $\square$

1.2.3 Exponential Markov's inequality

The exponential Markov's inequality applies Markov's inequality to the moment generating function $M_X(\lambda) = \mathbb{E} \exp(\lambda X)$ of a random variable X to obtain a stronger concentration than both Markov's and Chebyshev's inequalities.

Lemma 1.3 (Exponential Markov's inequality). *If X is a random variable, then*

$$\Pr(X \geq t) \leq \mathbb{E}[\exp(\lambda X)] \exp(-\lambda t) \quad (\lambda > 0),$$

$$\Pr(X \leq t) \leq \mathbb{E}[\exp(\lambda X)] \exp(-\lambda t) \quad (\lambda < 0).$$

Proof. First, for the right tail bound $\lambda > 0$, thus $\Pr(X \geq t) = \Pr(\exp(\lambda X) \geq \exp(\lambda t))$. For the left tail bound $\lambda < 0$, thus $\Pr(X \leq t) = \Pr(\exp(\lambda X) \geq \exp(\lambda t))$. The inequalities then follow from Markov's inequality since $\exp(\lambda X)$ and $\exp(\lambda t)$ are nonnegative. $\square$

1.2.4 Chernoff bound

The exponential Markov's inequality is commonly used to obtain tight concentration bounds. For instance, the Chernoff bound applies the exponential Markov's inequality to the mean of independent and identically distributed (i.i.d.) Bernoulli random variables.

Lemma 1.4 (Chernoff bound). *Let $X_1, \dots, X_n$ be independent Bernoulli(p) random variables for some $p \in (0, 1)$, and $X = \frac{1}{n} \sum_{i=1}^n X_i$. Then for any $t \geq p$,*

$$\Pr(X \geq t) \leq \exp(-np) \left(\frac{ep}{t} \right)^{nt}.$$

Proof. Substituting the definition of X and rewriting the probability gives

$$\Pr(X \geq t) = \Pr\left(\frac{1}{n} \sum_{i=1}^n X_i \geq t\right) = \Pr\left(\sum_{i=1}^n X_i \geq tn\right).$$

Applying the exponential Markov's inequality for all $\lambda > 0$,

$$\Pr\left(\sum_{i=1}^n X_i \geq tn\right) \leq \mathbb{E}\left[\exp\left(\lambda \sum_{i=1}^n X_i\right)\right] \exp(-\lambda tn).$$

Using the independence property of X_i and $1 + x \leq \exp(x)$ for all $x \geq 0$,

$$\mathbb{E}\left[\exp\left(\lambda \sum_{i=1}^n X_i\right)\right] = \prod_{i=1}^n \mathbb{E}[\exp(\lambda X_i)] = \prod_{i=1}^n (1 + p(e^\lambda - 1)) \leq \exp(np(e^\lambda - 1)).$$

This gives $\Pr(X \geq t) \leq \exp(np(e^\lambda - 1) - \lambda tn)$. Minimize in λ to get $\lambda = \ln(t/p)$ when $t \geq p$.

Thus, $\Pr(X \geq t) \leq \exp(nt - np - \ln(t/p)tn) = \exp(-np)\left(\frac{ep}{t}\right)^{nt}$. $\square$

For small deviations from the mean, the Chernoff bound results in an exponential decaying tail bound, which matches the tail bound of a normal distribution.

Corollary 1.1. *Let $X_1, \dots, X_n$ be independent Bernoulli(p) random variables for some $p \in (0, 1)$ and $X = \frac{1}{n} \sum_{i=1}^n X_i$. Then for any $\epsilon \in (0, 1)$,*

$$\Pr(X > (1 + \epsilon)p) < \exp(-\epsilon^2 pn/6).$$

Proof. Setting $t = (1 + \epsilon)p$ in the Chernoff bound gives

$$\begin{aligned}
\exp(-np) \left(\frac{ep}{t}\right)^{nt} &= \exp(-np) \left(\frac{e}{1 + \epsilon}\right)^{(1+\epsilon)pn} \\
&= \exp(-np) \exp((1 + \epsilon)pn) \left(\frac{1}{1 + \epsilon}\right)^{(1+\epsilon)pn} \\
&= \exp(\epsilon pn) \left(\frac{1}{1 + \epsilon}\right)^{(1+\epsilon)pn} \\
&= \exp(\epsilon pn - (1 + \epsilon)pn \ln(1 + \epsilon)) \\
&= \exp(pn(\epsilon - (1 + \epsilon) \ln(1 + \epsilon))) \\
&\leq \exp(-\epsilon^2 pn/6).
\end{aligned}$$

Using Taylor's theorem, one can show that $\epsilon - (1 + \epsilon) \ln(1 + \epsilon) \leq -\epsilon^2/6$, when $0 < \epsilon < 1$. $\square$

1.2.5 Hoeffding's inequality

Hoeffding's inequality bounds the sum of independent bounded random variables with an exponential decay on the tail of the distribution, matching that of a normal distribution.

Lemma 1.5 (Hoeffding's inequality). *Let $X_1, \dots, X_n$ be independent random variables such that $a_i \leq X_i \leq b_i$ almost surely and $X = \sum_{i=1}^n X_i$. Then, for all $t > 0$,*

$$\Pr(|X - \mathbb{E} X| \geq t) \leq 2 \exp\left(-\frac{2t^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

To prove this, we first proof Hoeffding's lemma.

Lemma 1.6 (Hoeffding's lemma). *Let X be a random variable such that $a \leq X \leq b$ almost surely. Then, for all $\lambda \in \mathbb{R}$,*

$$\mathbb{E}[\exp(\lambda(X - \mathbb{E} X))] \leq \exp\left(\frac{\lambda^2(b - a)^2}{8}\right).$$

Proof. Without loss of generality, assume $\mathbb{E} X = 0$, so that $a \leq 0 \leq b$. Since $e^{\lambda x}$ is a convex function of x , we have that for all $x \in [a, b]$,

$$\mathbb{E}[e^{\lambda X}] \leq \frac{b - \mathbb{E}[X]}{b - a} e^{\lambda a} + \frac{\mathbb{E}[X] - a}{b - a} e^{\lambda b} = \frac{b}{b - a} e^{\lambda a} + \frac{-a}{b - a} e^{\lambda b} = e^{L(\lambda(b-a))},$$

where $L(h) = \frac{ha}{b-a} + \ln(1 + \frac{a-e^h a}{b-a})$. By computing derivatives, we find $L(0) = L'(0) = 0$ and $L''(h) = -\frac{abe^h}{(b-ae^h)^2}$. From the AM-GM inequality, $(b - ae^h)^2 \leq -4abe^h$ and thus $L''(h) \leq \frac{1}{4}$ for all h . Using Taylor's theorem, there is thus some $\theta \in [0, 1]$ such that $L(h) = L(0) + hL'(0) + \frac{1}{2}h^2L''(h\theta) \leq \frac{1}{8}h^2$. Thus, $\mathbb{E}[e^{\lambda X}] \leq e^{\frac{1}{8}\lambda^2(b-a)^2}$. $\square$

Proof of Hoeffding's inequality. For $\lambda, t > 0$, the exponential Markov's inequality gives

$$\begin{aligned} \Pr(X - \mathbb{E} X \geq t) &\leq \exp(-\lambda t) \mathbb{E}[\exp(\lambda(X - \mathbb{E} X))] \\ &= \exp(-\lambda t) \prod_{i=1}^n \mathbb{E}[\exp(\lambda(X_i - \mathbb{E} X_i))]. \end{aligned}$$

Using Hoeffding's lemma, $\mathbb{E}[\exp(\lambda(X_i - \mathbb{E} X_i))] \leq \exp\left(\frac{\lambda^2(b_i - a_i)^2}{8}\right)$, thus

$$\begin{aligned} \Pr(X - \mathbb{E} X \geq t) &\leq \exp(-\lambda t) \prod_{i=1}^n \exp\left(\frac{\lambda^2(b_i - a_i)^2}{8}\right) \\ &= \exp\left(-\lambda t + \frac{1}{8}\lambda^2 \sum_{i=1}^n (b_i - a_i)^2\right). \end{aligned}$$

Minimizing in λ gives $\lambda = \frac{4t}{\sum_{i=1}^n (b_i - a_i)^2}$. Substituting this value for λ provides the desired bound. Repeating a similar argument for $\Pr(X - \mathbb{E} X \leq -t)$ and using the union bound of both events concludes the proof. $\square$

1.2.6 Median trick

In the analysis of approximation algorithms, a common technique is to first prove that an algorithm produces good estimates with some constant probability of success $p > 1/2$, and then select the median of $O(\log(1/\delta))$ such independent estimates to amplify the probability of success to $1 - \delta$. This technique is commonly referred to as the *median trick*.

Lemma 1.7 (Median trick). *Given independent random variables $X_1, \dots, X_n$ such that $\Pr(t - \epsilon \leq X_i \leq t + \epsilon) \geq p > 1/2$ for some fixed t and ϵ . Then,*

$$\Pr(t - \epsilon \leq \text{median}(X_1, \dots, X_n) \leq t + \epsilon) \geq 1 - \exp(-2n(p - 1/2)^2).$$

Proof. Let Z_i be the indicator variable for the event that $X_i \notin [t \pm \epsilon]$. Then, the event $\text{median}(X_1, \dots, X_n) \in [t \pm \epsilon]$ does not occur only if at least half of $Z_i = 1$, that is, $\frac{1}{n} \sum_{i=1}^n Z_i \geq 1/2$. Using Hoeffding's inequality with $t = n/2 - \sum_{i=1}^n \mathbb{E} Z_i$, this event occurs with probability at most $\exp(-2n(p - 1/2)^2)$. $\square$

1.3 Universal hashing

Many sketching algorithms use universal hash functions in their design. At a high level, a universal hash function is a function sampled at random from a family of functions that map values from the domain uniformly to the codomain in expectation. For example, with $[n] = \{1, 2, \dots, n\}$ the hash function $h : [n] \rightarrow [m]$ maps the integers in $[n]$ uniformly to the integers in $[m]$ such that $\Pr(h(x) = y) = 1/m$ for $x \in [n]$ and $y \in [m]$, where the probability is over the random hash function.

Section 1.1 mentioned the idealized approximation algorithm for the unique count by Flajolet and Martin (1985), which assumes a fully independent hash function. However, storing a

fully independent hash function, which outputs an independent random number for each value of the domain, is expensive because it requires $\Omega(n)$ bits of space, where n is the size of the domain (Wegman and Carter, 1979). This becomes prohibitive for large domains. For example, a hash function that maps 32-bit integers to independent random bits requires 2^{32} bits of space.

To combat this issue, we have to give up the full independence between the random numbers and settle for k -wise independence instead. This means that the outputs of any k distinct inputs are independent of each other.

1.3.1 k -independent hashing

Wegman and Carter (1979) introduced the idea of k -wise independent hash functions.

Definition 1.2 (k -wise independence). *A family of hash functions $H = \{h : [n] \rightarrow [m]\}$ is said to be k -wise independent if for any k distinct items $x_1, x_2, \dots, x_k$ the hashed values $h(x_1), h(x_2), \dots, h(x_k)$ are independent and uniformly distributed in $[m]$.*

Interestingly, k -wise independent hash functions can be stored using $O(k \log n)$ independent random bits. The most well-known construction of such hash functions are polynomials of the form

$$h_a(x) = \sum_{i=1}^k a_i x^i \bmod p,$$

where $x \in [p]$ and the coefficients $a_1, \dots, a_k$ are uniformly sampled from $[p]$ for any prime p . From this hash function, one can then construct approximately k -wise independent hash functions over the domain $[d]$ for $d < p$ by taking the hash function $h_a(x) \bmod d$.

1.4 Examples of sketching algorithms

This section provides a brief description of two approximation algorithms for the fundamental problems of unique count and inner product. While neither of the two methods are the current state of the art, they provide a simple example of the construction of sketching algorithms that builds the intuition required for the advanced methods presented later.

1.4.1 Unique count

The unique count problem, also known as the count distinct, distinct elements, or F_0 problem, is formally defined as follows.

Definition 1.3 (Unique count). *Given a stream of integers $x_1, x_2, \dots, x_t \in [n]$. Let $f_t(x)$ be the frequency of x at time t . Then, the unique count of the stream at time t is $F_0 = |\{x_1, x_2, \dots, x_t\}|$, that is, the number of unique integers seen in the stream.*

Solving this problem exactly with a deterministic algorithm requires at least $\Omega(n)$ bits of space. The following lemma shows this formally through an adversarial argument using the pigeonhole principle.

Lemma 1.8. *Any deterministic algorithm for the unique count of a stream with items from a universe of size n , requires $\Omega(n)$ bits of space.*

Proof. Recall that there are 2^n possible distinct subsets of items that could arrive in the stream. Suppose the algorithm uses b bits of memory. This means that it can be in at most 2^b distinct states. If $b < n$ bits, then $2^b < 2^n$ and by the pigeonhole principle, there must exist at least two distinct subsets A and B of the universe that cause the algorithm to end up in the exact same state after processing them. Because the algorithm is in an identical state for both A and B , it must behave exactly the same way for any future items added to the stream. If an adversary then streams all the elements of A , and the original stream was

A , the new set is $A \cup A$, and the unique count is $|A|$. However, if the original stream was B , the new set is $B \cup A$. Since A and B are distinct subsets, the unique count is $|B \cup A|$, which is strictly greater than $|A|$. Because the algorithm cannot distinguish whether the first part of the stream was A or B , it will inevitably output the wrong count for one of those scenarios. Therefore, any exact deterministic algorithm requires at least n bits of space. $\square$

Alon et al. (1996) showed that this lower bound holds even for randomized exact algorithms that fail with some small probability δ and for deterministic approximate algorithms that output the unique count with some small error ϵ . This shows that both randomization and approximation are necessary to enable an improvement in asymptotic complexity.

The following algorithm for the unique count is called the k -minimum values (KMV) algorithm by Bar-Yossef et al. (2002). The intuition for approximating the unique count is based on the properties of uniform hashing. If we apply a hash function to map the items in the stream uniformly over some universe of size M , the number of unique hash evaluations expected in any subset of the universe of size S will be tS/M , with t the unique count.

Based on this intuition, the KMV algorithm first samples a random hash function $h : [n] \rightarrow [M]$ for $M = n^3$ from a 2-wise independent hash family. It then tracks the $k = O(1/\epsilon^2)$ items in the stream with the smallest hash evaluations. When k or less unique hash evaluations have been observed at the time of a query, the algorithm outputs the number of distinct hash evaluations seen. Otherwise, if X is the k th smallest hash evaluation, then the algorithm provides an estimate as kM/X .

Bar-Yossef et al. (2002) showed that the above estimator uses $O(1/\epsilon^2 \log m)$ bits of space to be within a $1 \pm \epsilon$ factor of the unique count with some constant probability of success $p > 1/2$. To boost the probability of success to $1 - \delta$, the authors use the median trick by selecting the median of $O(\log(1/\delta))$ independent estimates. The final algorithm then uses $O(\epsilon^{-2} \log(1/\delta) \log(m))$ bits of space and has an update time complexity of $O(\log(1/\epsilon) \log(m))$.

by using a balanced binary search tree to store the minimum k values.

1.4.2 Inner product

The remainder of this dissertation will focus on approximations of the inner product, or dot product, and generalizations thereof.

Definition 1.4 (Inner product). *Given two streams of integers $x_1, x_2, \dots, x_t \in [n]$ and $y_1, y_2, \dots, y_t \in [n]$. Let $f_t(x)$ be the frequency of x in the first stream at time t and similarly $g_t(x)$ the frequency of x in the second stream at time t . Then, the inner product of the streams at time t is $\langle f_t, g_t \rangle = \sum_{i=1}^n f_t(i)g_t(i)$.*

Any randomized exact algorithm for the inner product of two n -dimensional vectors requires $\Omega(n)$ space due to the lower bound for the *Index* problem (Jayram et al., 2008).

The following randomized approximate algorithm is called the AMS sketch by Alon et al. (1999). The AMS sketch applies a random linear transformation $\Pi \in \{-1, 1\}^{m \times n}$ to both of the vectors to embed them in a lower dimensional space. The matrix Π has random sign entries determined by 4-wise independent hash functions. Specifically, the algorithm starts by sampling m i.i.d. 4-wise independent hash functions $h_i : [n] \rightarrow \{-1, 1\}$ for $i \in [m]$. The matrix Π is then specified as $\Pi_{i,j} = h_i(j)$. The inner product of vectors $x, y \in \mathbb{R}^n$ is approximated using Πx and Πy as $(\Pi x)^\top (\Pi y) / m$.

To understand why this works, let us first look at a single element of Πx and Πy . The i th element uses the hash function h_i , thus $(\Pi x)_i = \sum_{j=1}^n h_i(j)x_j$ and similarly $(\Pi y)_i = \sum_{j=1}^n h_i(j)y_j$. If we look at the expected value of their product, we see

$$\mathbb{E}[(\Pi x)_i(\Pi y)_i] = \sum_{j,k=1}^n \mathbb{E}[h_i(j)h_i(k)]x_jy_k = \sum_{j,k=1}^n [j=k]x_jy_k = \sum_{j=1}^n x_jy_j = x^\top y.$$

Thus, the product of a single element is an unbiased estimator of the inner product. Since we

sampled an i.i.d. hash function of each element of the resulting sketches, the product of each element is an unbiased estimator of the inner product. Taking the mean of m independent estimates then improves the concentration of the random variable by a factor of $1/m$.

Alon et al. (1999) showed that setting $m = O(1/\epsilon^2)$ results in estimates that are within $(1 \pm \epsilon)\|x\|_2\|y\|_2$ of the inner product with some constant probability of success $p > 1/2$. The authors then apply the standard median trick to boost the probability of success to $1 - \delta$ by selecting the median of $O(\log(1/\delta))$ independent estimates. The final algorithm then uses $O(\epsilon^{-2} \log(1/\delta) \log(n))$ bits of space to store $O(\epsilon^{-2} \log(1/\delta))$ hash functions and has an update time complexity of $O(\epsilon^{-2} \log(1/\delta))$.

Chapter 2

Set similarity estimation

Many current challenges—and opportunities—in computer science stem from the sheer scale of the data to be processed (Fan et al., 2014; Hariri et al., 2019). Among these challenges, one of the most outstanding is comparing collections of objects, or simply *sets*¹. This demand arises, for example, in important problems such as comparing text documents or social media profiles. The challenge is often not the size of each set, but the number of pairwise comparisons over a large dataset of sets. This has motivated research on *estimating* existing set similarity measures, which is the main subject of this chapter.

2.1 Introduction

The search for methods to compare sets of elements is long-standing: more than a century ago, Gilbert (1884) and Jaccard (1912) independently proposed a measure that is still widely used, known as the *Jaccard index*. The metric is defined as the ratio between the sizes of the intersection and the union of two sets. With the explosion of available data, brought about mainly by the advent of the Web, the Jaccard index has become prevalent as an essential

¹We shall refer to *sets*, indistinctly, as collections with or without repeated elements.

tool in data mining and machine learning. Important applications include information retrieval (Ni wattanakul et al., 2013; Sefid et al., 2019), natural language processing (Yih and Meek, 2007; Strube and Ponzetto, 2006), and image processing (Liu et al., 2020; Padilla et al., 2020), among several others (Choi et al., 2010).

Another academic field in which set similarity has become crucial is *network science*. This field studies network representations of physical, biological and social phenomena and is used to understand complex systems in various disciplines (Barabási, 2013). Such networks are modeled using graphs, a mathematical abstraction tool where sets are ubiquitous. Marked by its growing relevance, this area also gave rise to one of the most famous set similarity metrics: the *Adamic-Adar index* (Adamic and Adar, 2003). The index was proposed as an alternative to the Jaccard index for the problem of predicting links, such as friendship or co-authorship, in social networks.

Link prediction is a widely studied problem, with applications in several Web-related tasks, including hyperlink-prediction (Zhu et al., 2002), recommender systems in e-commerce (Li and Chen, 2009), entity resolution (Malin et al., 2005), and friend recommendation in social networks (Daud et al., 2020; Freschi, 2009), among others (Hasan and Zaki, 2011). In this problem, each node is characterized by its set of adjacent nodes, or *neighbors*. The intuition is that nodes of similar neighborhoods tend to become neighbors. The Adamic-Adar index is used to compare these sets of neighbors, but unlike the Jaccard index, it assigns different weights to neighbors (see Section 2.2.2). The Adamic-Adar index is known to be superior to the Jaccard index for modeling the phenomenon of link emergence in networks in various real-world applications (Liben-Nowell and Kleinberg, 2007; Martínez et al., 2016). The success of the Adamic-Adar index also motivated the emergence of several metrics with different ways of weighting neighbors (Dong et al., 2011; Liu et al., 2017; Cannistraci et al., 2013), which will be discussed in Section 2.2.3.

A second prime example of an application marked by demanding an enormous number of

set comparisons is the removal of (near-)duplicate pages in Web search. Eliminating such pages saves network bandwidth, reduces storage costs and improves the quality of search results (Manku et al., 2007). In this domain, each document is commonly treated as a set of word sequences. To find duplicates in a corpus of 10 million pages, a relatively small scale for Web applications (Manku et al., 2007), it would already be necessary to compute the set similarity metric about 50 trillion times. It was precisely in the face of this challenge that the problem of *estimating* set similarity metrics has become highly relevant and has triggered numerous scientific endeavors.

MinHash (Broder, 1997) and *SimHash* (Charikar, 2002), the two best-known estimators, were initially developed for the above-mentioned problem and used respectively in the AltaVista and Google Web search engines. Other current applications taking advantage of set similarity estimation include genomic and metagenomic analysis (Ondov et al., 2016; Koslicki and Zabeti, 2019), graph comparison (Teixeira et al., 2012), collaborative filtering (Das et al., 2007), natural language dataset preparation (Lee et al., 2021), and duplicate detection of other types of data such as images (Chum et al., 2008). SimHash is also used in *locality sensitive hashing* (LSH), a technique applied to detect if the similarity of sets exceeds a given threshold, used in problems such as dimensionality reduction, nearest neighbor search, entity resolution and fingerprint matching (Leskovec et al., 2020). An important remark is that, despite being related, LSH and the problem of estimating metrics directly are distinct and both individually relevant, as we will discuss in Section 2.3. The wide range of relevant applications illustrates the importance and potential of set similarity estimators to push the boundaries of problems where they are applied.

Despite the importance of the estimators mentioned above, previous works reveal limitations of these techniques. As is common with estimators, their accuracy is a function of the input as well as the value to be estimated. In the original MinHash paper, Broder (1997) indicates that the estimator’s accuracy is at its worst when the Jaccard index is around 0.5.

Koslicki and Zabeti (2019) show that the probability of the estimator deviating from the true value increases exponentially with the difference in size between the sets to be compared. Nonetheless, Shrivastava and Li (2014) provides theoretical arguments for the superiority of MinHash over the other baseline, SimHash. More importantly, these techniques are not able to estimate indices like the Adamic-Adar index. Sections 2.5 and 2.6 will show empirically that this limitation makes them less appropriate for the link prediction and near-duplicate detection problems.

Commendable effort has been devoted to reshaping these techniques to overcome these limitations for specific applications, as will be discussed in Section 2.3.3. Nevertheless, given its relevance, it is also important to pursue alternative techniques that explore new perspectives on the problem. With this in mind, this chapter describes DotHash—a novel set similarity estimator. Like its predecessors, DotHash is based on creating a fixed-size representation of sets (often called a *sketch*, *signature* or *fingerprint* in the literature) that can be compared to estimate the similarity between the sets. Creating these compressed representations introduces preprocessing time, but dramatically mitigates the time for each comparison. The central idea of DotHash is to exploit valuable features of high-dimensional random vector spaces, in particular their *tendency towards orthogonality* (Kanerva, 1988), to create fixed-dimension sketches while retaining as much information from the original space as possible. Theorem 2.2 shows that the cardinality of the intersection of two sets can be estimated, without bias, by a simple *dot product* between their DotHash sketches. Since the dot product is such a fundamental operation, DotHash can take advantage of recent progress in modern hardware for enhanced performance and energy efficiency (Yamanaka et al., 2008; Moughan, 2021).

In addition to the theoretical contribution of a new baseline framework to the problem of comparing sets in its general formulation, DotHash also has prompt practical relevance. Experiments with several popular datasets for the problems of link prediction and near-

duplicate detection show that the accuracy obtained by DotHash is higher than that obtained with SimHash and MinHash. These results exploit the fact that DotHash is able to estimate a more general family of metrics, which are better adapted to applications, as is the case of the Adamic-Adar index for link prediction. It is worth noting that the time complexity for each comparison is linear in the size of the DotHash sketches regardless of the metric, since it consists of computing the dot product between them. As previously mentioned, these pair-wise comparisons between sketches dictate the overall time complexity.

2.2 Set similarity measures

This section introduces relevant set similarity metrics, starting with the Jaccard and Adamic-Adar indices, for which theoretical and empirical results are shown. An overview of other metrics that DotHash is able to estimate directly is then provided. It is worth noting that many alternative set similarity metrics have been proposed, specifically tailored to better suit specific scenarios by adapting the basic metrics. For an extensive overview of set similarity metrics, the works by Martínez et al. (2016) and Lü and Zhou (2011) are recommended.

2.2.1 Jaccard index

Consider the sets A and B , and let $A \cap B$ denote their intersection and $A \cup B$ their union. The Jaccard index (Jaccard, 1912) between A and B is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|},$$

where $|A|$ denotes the cardinality of the set A . This is one of the oldest and most established ways of comparing sets. Over time, numerous adaptations of this simple metric have emerged, specializing it for particularly interesting applications. One of these adaptations, which is widely used especially in network science (Barabási, 2013), is described next.

2.2.2 Adamic-Adar index

The Adamic-Adar index was created for the problem of link prediction in social graphs (Adamic and Adar, 2003). Let $G = (V, E)$ be a graph, composed of a set of nodes V and a set of edges E . Each $e = (u, v) \in E$ represents an edge (or link) between nodes $u, v \in V$. With $\Gamma(v) \subseteq V$ we denote the subset of nodes that are adjacent to v , i.e., the *neighbors* of v . The cardinality of a neighborhood $|\Gamma(v)|$ is referred to as the *degree* of v .

In the context of graphs, the Jaccard index can be used to compare pairs of nodes by looking at how many connections they have in common, normalized by the size of the union of their neighborhoods. The Adamic-Adar index seeks to improve this comparison based on the intuition that the more popular a node in the intersection is (i.e., the higher its degree), the less informative it is about the similarity of the nodes being compared. In the case of social networks, for example, a common connection with a celebrity says little about the chance of two people connecting with each other compared to a less popular mutual friend. To account for this, the Adamic-Adar index penalizes the number of connections that each shared connection has by taking the logarithm of its degree. Formally, the Adamic-Adar index between two nodes is defined as

$$A(u, v) = \sum_{x \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(x)|}.$$

2.2.3 Resource allocation index

Given the importance of link prediction, several other metrics have emerged for the purpose of comparing neighborhoods. Similar to the Adamic-Adar index, these metrics take the local properties of the nodes in the intersection into account. Many of these metrics can also be directly estimated using DotHash. One such example is the resource allocation index, used to evaluate the resource transmission between two unconnected nodes through their neighbors

which is defined as

$$\text{RA}(u, v) = \sum_{x \in \Gamma(u) \cap \Gamma(v)} \frac{1}{|\Gamma(x)|}.$$

Section 2.4.5 provides a formal description of the family of set similarity metrics that DotHash can directly estimate.

2.3 Set similarity estimators

This section describes the two most popular estimators, MinHash and SimHash, which will then be used as baselines for the evaluation of DotHash. It is important to highlight that DotHash is compared with these two methods because they constitute the state of the art for the problem in its most general formulation and are used in current applications, as previously mentioned.

2.3.1 MinHash

MinHash (Broder, 1997) is a probabilistic method for estimating the Jaccard index. The technique is based on a simple intuition: if we *uniformly sample* one element x from the set $A \cup B$, then

$$\Pr(x \in A \cap B) = \frac{|A \cap B|}{|A \cup B|} = J(A, B),$$

which makes the result of this experiment an estimator for the Jaccard index. However, an important problem remains: *how to uniformly sample from $A \cup B$?* Explicitly computing the union is at least as expensive as computing the intersection, that is, it would be as expensive as calculating the Jaccard index exactly. The main merit of MinHash is an efficient way of circumventing this problem.

Let $h : A \cup B \rightarrow \mathbb{N}$ denote a min-wise independent hash function, i.e., for any subset of the domain, the output of any of its elements is equally likely to be the minimum (see Broder et al. (2000) for a detailed discussion), then

$$\Pr(\min_{a \in A} h(a) = \min_{b \in B} h(b)) = \frac{|A \cap B|}{|A \cup B|} = J(A, B).$$

Given the above, the problem of uniformly sampling an element of the union and checking if it belongs to the intersection can be emulated as follows: hash the set elements and check if the smallest value obtained in both sets is the same. Although the result of this random variable is an unbiased estimator of the Jaccard index, its variance is high when the Jaccard is around 0.5. The idea of MinHash is therefore to do k such experiments with independent hash functions and return the sample mean to reduce the variance.

2.3.2 SimHash

SimHash (Charikar, 2002), sometimes indistinctly called *angular LSH*, is another popular estimator of set similarity. The sketches of sets are fixed-length binary vectors and are generated as follows: (1) all elements of the superset S are mapped to independent d -dimensional standard normal random vectors; (2) for each set $X \subseteq S$ a d -dimensional vector is created by adding the vectors of its elements; (3) the *SimHash sketch* of the set is a bit string obtained by transforming each positive entry to one and the non-positive entries to zero. The similarity between pairs of sets is then measured by the Hamming distance between these sketches.

Despite being a general estimator for set similarity metrics, SimHash owes its popularity largely to a specific use. Manku et al. (2007) showed an efficient way to solve the following problem: *in a collection of SimHash sketches, quickly find all sketches that differ at most k bits from a given sketch, where k is a small integer*. This particular formulation is very

useful in the context of duplicate text detection and its efficient solution led to SimHash being used by the Google Crawler.

The problem described above is an instance of the problem known as *locality sensitive hashing* (LSH) which, in general, tries to detect pairs of objects whose similarity exceeds a threshold by maximizing the collision probability of the hashes of these objects (Indyk and Motwani, 1998; Indyk et al., 1997). Note that LSH is used to group similar objects and the output is binary, i.e. *two objects are either similar or not* (Leskovec et al., 2020). Therefore, the problem of estimating the metrics directly, as discussed in this chapter, is different from LSH. Directly estimating the similarity has other possible outcomes, such as ordering pairs by similarity, which is crucial in some applications like query optimization (Cormode et al., 2011) and link prediction, as will be discussed in Section 2.5.

Although SimHash is more popular in the context of LSH, for completeness, SimHash is included as a baseline for the general problem. However, the other baseline, MinHash, is much more common in the literature for the problem of estimating the actual value of metrics as will be discussed in the next section.

2.3.3 Related work

The primary focus of this chapter is on estimating set similarity metrics in its broadest sense. However, it is important to acknowledge other advancements in the field that are not directly aligned with the content of this chapter. These advancements, discussed below, primarily involve enhancements tailored to specific contexts and applications. It is important to note that DotHash is not intended to directly compete with these notable developments in each individual application, but rather aims to serve as a new baseline for the general problem. However, DotHash also brings practical contributions to the state of the art. This is achieved by supporting a wider range of metrics, which is formally defined in Section 2.4.5. As will be

shown in Sections 2.5 and 2.6, this allows for greater accuracy in important problems such as link prediction and document deduplication.

While MinHash remains the standard framework for estimating the actual value of set similarity metrics, several techniques have been proposed to enhance its accuracy and efficiency in specific application contexts. For instance, Chum and Matas (2012) propose an efficient method to compute MinHash sketches for image collections using inverted indexing. Another technique, introduced by Koslicki and Zabeti (2019), employs Bloom filters for fast membership queries and is known as *containment MinHash*. They demonstrate the superiority of this technique in metagenomic analysis by more accurately estimating the Jaccard index when dealing with sets that significantly differ in size.

Several other works have focused on a variant of the problem that involves estimating the *weighted* Jaccard index (Wu et al., 2020a). For datasets with sparse data, Ertl (2018) and Christiani (2020) have explored the concept of *consistent weighted sampling* (CWS) (Ioffe, 2010) with their respective *BagMinHash* and *DartMinHash* techniques. Conversely, when dealing with dense data, methods based on *rejection sampling* have been demonstrated to be more efficient (Shrivastava, 2016; Li and Li, 2021).

Another important recent endeavor has been to develop LSH techniques based on deep learning, known as “learn to hash” methods. These include *Deep Hashing* (Zhu et al., 2016) and various others (Dutt et al., 2019; Hasan et al., 2020; Kipf et al., 2018). In general, these techniques do not estimate any particular metric directly, but seek to create sketches that allow for approximate nearest neighbor search. Another key distinction lies in the methodology employed by these approaches. They rely on constructing trained models through annotated data, where the concept of similarity is derived from the mapping of training examples to a specific target. Consequently, this similarity may not necessarily extend to other datasets, making it potentially non-generalizable. In contrast, the methods discussed in this chapter estimate the similarity between two sets solely based on the sets

themselves.

Although “learn to hash” methods have demonstrated promising accuracy in situations where supervised learning is viable, their broader adoption has also been hindered by other challenges. These limitations encompass high costs associated with training and inference, the inherent unpredictability due to unknown bounds in estimation error, and their high sensitivity to data distribution, often concealed by the reliance on purely empirical assessments (Kim et al., 2022; Wang et al., 2021). As a result, traditional methods such as MinHash and SimHash continue to be utilized in important applications such as the ones mentioned earlier.

2.4 DotHash

This section starts by describing a simple method to compute the cardinality of the intersection between two sets. This provides the basis from which the intuition for DotHash is given. The intuition builds on a generalization of the simple method and a subtle feature of high-dimensional vector spaces, leading to an estimator for the intersection cardinality. Notably, a number of set similarity metrics are direct functions of the intersection cardinality, combined with other easily obtained quantities such as the size of the sets (Martínez et al., 2016). The fact that DotHash estimates the intersection cardinality directly makes it naturally extendable to all these metrics. One of the few exceptions is a family of metrics that assign different weights to the intersection items, such as the Adamic-Adar index. The section concludes by demonstrating how DotHash can be adapted to estimate this larger family of metrics as well, being the first estimator to enable this.

2.4.1 Computing the intersection size

A common way of representing sets is by using bit strings (Rosen, 2012; Fletcher et al., 2018). In this representation, an arbitrary order $[x_1, x_2, \dots, x_d]$ of the elements of the superset S is established. Then, each set $X \subseteq S$ is represented by an $|S|$ -dimensional binary vector whose i th bit is one if $x_i \in X$, and zero otherwise. Table 2.1 illustrates this representation for sets $A = \{a, b, c, d\}$ and $B = \{b, c, d, e\}$. This representation is especially common for graphs, where it is called an *adjacency matrix*, and each set consists of the neighborhood of a node.

Table 2.1. Bit string representation of sets A and B

	a	b	c	d	e
A	1	1	1	1	0
B	0	1	1	1	1

It is easy to observe that the size of the intersection between A and B is given by the number of columns where both elements are one. This provides a straightforward way to get the size of the intersection of sets: calculate the *dot product* between their bit strings (Fletcher et al., 2018; Brusco et al., 2021).

2.4.2 Generalization to orthogonal vectors

An alternative way of visualizing the set bit strings, important for the generalization, is: consider that each *element* $x_i \in S$ is encoded by an $|S|$ -dimensional vector of value one in position i , and zero elsewhere. This representation is usually called *one-hot encoding*. From this, we can define the bit string of a set as *the sum of its one-hot encoded elements*, as illustrated in Figure 2.1.

Theorem 2.1 shows that the dot product results in the intersection of sets not only when they are the sum of one-hot encoded elements, but more generally when the elements are encoded using any orthonormal basis of $\mathbb{R}^{|S|}$, of which one-hot is a particular case—the *standard*

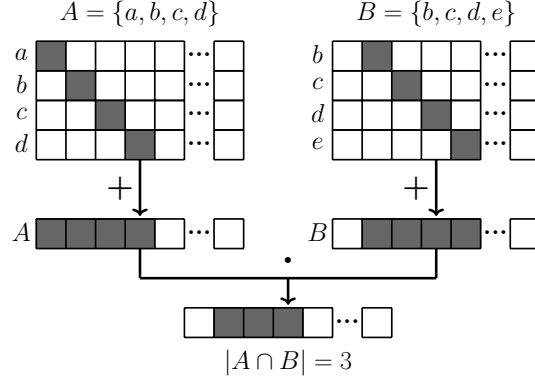


Figure 2.1. Set intersection size calculation using one-hot encoding

basis. Although this, arguably trivial, generalization alone may not seem advantageous at this point, the next section describes how it is fundamental in the transition from exact to approximate.

Theorem 2.1. *Consider arbitrary sets $A, B \subseteq S$. Let $\phi : S \rightarrow \mathbb{R}^{|S|}$ be any injective map of their elements to vectors to one of the orthonormal bases of $\mathbb{R}^{|S|}$, and let*

$$\mathbf{a} = \sum_{a \in A} \phi(a) \quad \text{and} \quad \mathbf{b} = \sum_{b \in B} \phi(b).$$

Then, $\mathbf{a} \cdot \mathbf{b} = |A \cap B|$.

While the above yields a way of representing sets so that intersection sizes can be computed by simple dot products, notice that each set is represented using $|S|$ bits, resulting in a time and space complexity of $O(|S|)$. Although this can be useful in certain scenarios, clearly this method becomes prohibitively expensive for very large supersets S , for example, in the large scale applications described in the previous sections. Another problem is that in many real applications, such as those related to social networks, the sets change over time, so there is no way to establish the superset size a priori.

In some cases the time and space complexity can be improved to $O(|A| + |B|)$ by restricting ϕ to the standard basis encoding, represented using a sparse vector format so that only the

non-zero elements are stored. With this modification, the dot product can be computed by iterating over both vectors at once, similar to merging lists. However, because of the overhead of sparse vector representations, this is mainly useful when $|A| + |B| \ll |S|$.

There are still limitations to the sparse vector improvement, especially when even $|A| + |B|$ is very large. This happens both in social networks where nodes can have more than a million connections, and in document deduplication where large documents can hold many word sequences. The next section describes a method to improve the time complexity to a constant value by giving up the exact intersection size in favor of an estimate.

2.4.3 Exploiting *quasi*-orthogonality

The method described above seems unsuitable for large scale applications as it requires a number of dimensions equal to the size of the superset. This constraint is imposed by the fact that the smallest real vector space with $|S|$ orthogonal vectors is $\mathbb{R}^{|S|}$. Intuitively, orthogonality is needed so that $\phi(x)^\top \psi(A) = 1$ if, and only if, $x \in A$, and zero otherwise. This ensures that $\psi(A)^\top \psi(B) = |A \cap B|$ (for a detailed discussion, see the proof of Theorem 2.1 in the Appendix A.1.1). From an information theory perspective, this guarantees a lossless representation of the sets by the sum of the encoded elements, since by the above operation we can verify exactly which elements make up the set.

The DotHash estimator relies on a very interesting property of high-dimensional vector spaces: uniformly sampled vectors are nearly orthogonal, or *quasi*-orthogonal, to each other with high probability (Kanerva, 2009). This valuable feature has been explored in several other domains, especially to model human cognition and memory (Kanerva, 1988; Gayler, 2003). It is based on this insight that DotHash turns the above method into an estimator for the size of the intersection of sets.

Instead of using a *precisely* orthonormal basis of vectors of $\mathbb{R}^{|S|}$ to encode the elements of

S , DotHash uses unit vectors sampled from $\mathbb{R}^d$, with $d < |S|$. The set *sketches* (fixed-length representations) are then built in the same way, by adding the encodings of their elements, as depicted in Figure 2.2. Intuitively from the above, each encoded element would be quasi-orthogonal to all others, allowing to approximate the dot product relations mentioned above. Theorem 2.2 formalizes this idea, showing that the dot product between the DotHash sketches of sets is an unbiased estimator for their intersection cardinality.

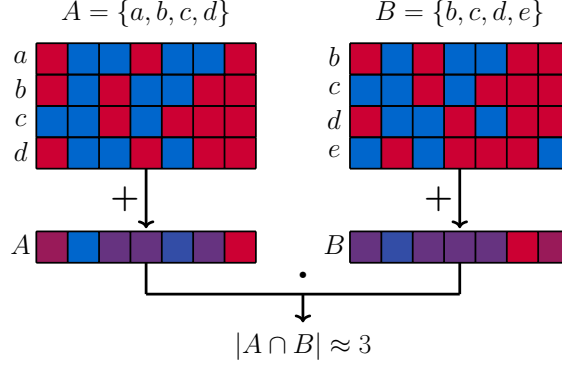


Figure 2.2. Set intersection size calculation using quasi-orthogonal encoding

Theorem 2.2.

Consider arbitrary sets $A, B \subseteq S$ and any constant $d \in \mathbb{N}^+$. Let $\psi : S \rightarrow \mathbb{R}^d$ be a uniform random mapping of elements in S to unit vectors which are the vertices of a d -dimensional hypercube, and let

$$\mathbf{a} = \sum_{a \in A} \psi(a) \quad \text{and} \quad \mathbf{b} = \sum_{b \in B} \psi(b).$$

Then, $\mathbb{E}[\mathbf{a} \cdot \mathbf{b}] = |A \cap B|$, and $\text{Var}(\mathbf{a} \cdot \mathbf{b}) = \frac{1}{d} (|A||B| + |A \cap B|^2 - 2|A \cap B|)$.

Using the variance provided in Theorem 2.2 and Chebyshev's inequality, we can bound the probability of error by

$$\Pr(|X - \mu| \geq \epsilon\mu) \leq \frac{\text{Var}(\mathbf{a} \cdot \mathbf{b})}{(\epsilon|A \cap B|)^2}.$$

If we use the observation that each dimension can be interpreted as an independent sample, we can use the Central Limit Theorem (CLT) to approximate the probability of error as

$$\lim_{d \rightarrow \infty} \Pr(|X - \mu| \geq \epsilon\mu) = 2 \left(1 - \Phi \left(\frac{\epsilon|A \cap B|}{\sqrt{\text{Var}(\mathbf{a} \cdot \mathbf{b})}} \right) \right),$$

where Φ denotes the standard normal cumulative distribution function. Figure 2.3 provides the CLT estimate (solid line) and the empirical probability (dashed line).

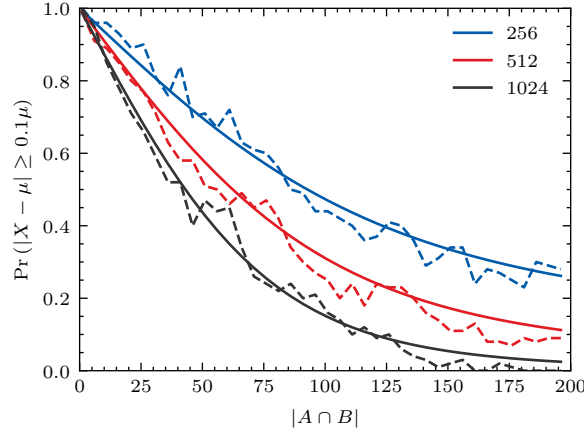


Figure 2.3. Intersection estimate bounds for DotHash with $|A| = |B| = 200$ and $d \in \{256, 512, 1024\}$. The dashed lines indicate empirical results and the solid lines indicate the Central Limit Theorem (CLT) estimates.

We can rewrite the CLT (or Chebyshev's inequality) to get the required number of dimensions d to obtain an error greater or equal to $\epsilon|A \cap B|$ with a given probability p :

$$d \approx \text{Var}(\mathbf{a} \cdot \mathbf{b}) \left(\frac{\Phi^{-1}\left(1 - \frac{p}{2}\right)}{\epsilon|A \cap B|} \right)^2,$$

where Φ^{-1} denotes the standard normal percent point function.

2.4.4 Estimating the Adamic-Adar index

Now that we have established a method for estimating the size of the intersection, this section describes how to adapt DotHash to estimate the Adamic-Adar index. The idea starts from

the fact that $\mathbf{a}$ and $\mathbf{b}$ are sums of the vectors that encode the elements of their respective sets. Given this and the distributive property of the dot product over addition, we have

$$\mathbf{a} \cdot \mathbf{b} = \sum_{a \in A} \sum_{b \in B} \psi(a) \cdot \psi(b).$$

Observe that $\mathbb{E}[\psi(a) \cdot \psi(b)] = 1$ if $a = b$, and zero otherwise (see the proof of Theorem 2.2), thus

$$\mathbb{E}[\mathbf{a} \cdot \mathbf{b}] = \sum_{a \in A} \sum_{b \in B} \mathbb{E}[\psi(a) \cdot \psi(b)] = \sum_{a \in A} \sum_{b \in B} [a = b] = \sum_{x \in A \cap B} 1,$$

which uses the Iverson bracket notation, where $[S]$ is one if the statement S is true and zero otherwise. The right-hand side of this equation is similar to the Adamic-Adar index in that both sum values over the intersection items. The key missing part is that the value to be summed must be a function of the size of the neighborhoods, not a constant.

In the above case, the summation parameter is one because every element is encoded to a unit vector. However, the construction of DotHash allows the summation parameter to be adjusted by modifying the magnitude of the vectors used to represent each element. To obtain an estimator for the Adamic-Adar index, we want each element, in this case each *node*, to be encoded in such a way that

$$\psi(v) \cdot \psi(v) = \frac{1}{\log |\Gamma(v)|}.$$

Theorem 2.3 shows how to adapt the vector magnitudes to obtain the Adamic-Adar index.

Theorem 2.3.

Consider an arbitrary graph $G = (V, E)$ with nodes V and edges E . Take any constant $d \in \mathbb{N}^+$ and let $\Gamma(v) \subseteq V$ denote the neighbors of node $v \in V$. Let $\psi : V \rightarrow \mathbb{R}^d$ be a uniform

random mapping of nodes in V to unit vectors which are the vertices of a d -dimensional hypercube, and let

$$\mathbf{u} = \sum_{x \in \Gamma(u)} \psi(x) \sqrt{\frac{1}{\log |\Gamma(x)|}} \quad \text{and} \quad \mathbf{v} = \sum_{y \in \Gamma(v)} \psi(y) \sqrt{\frac{1}{\log |\Gamma(y)|}}.$$

Then, $\mathbb{E}[\mathbf{u} \cdot \mathbf{v}] = A(u, v)$.

2.4.5 General family of supported metrics

This section extends the results presented in the previous section to encompass a general formulation of all set similarity metrics that can be directly estimated using DotHash. This family includes all metrics of the form $\sum_{x \in A \cap B} f(x)$, where $f : S \rightarrow \mathbb{R}$ is any function on intersection elements. Besides the item x , the function f can use any global parameters such as the cardinalities of A , B or S . The DotHash sketches for this general set similarity metric are given by

$$\mathbf{a} = \sum_{a \in A} \psi(a) \sqrt{f(a)} \quad \text{and} \quad \mathbf{b} = \sum_{b \in B} \psi(b) \sqrt{f(b)},$$

and the estimate is obtained by $\mathbf{a} \cdot \mathbf{b}$. Observe that the intersection size, the Adamic-Adar index, and the resource allocation index all fit into this general framework: the intersection size corresponds to $f(x) = 1$, the Adamic-Adar index to $f(x) = \frac{1}{\log |\Gamma(x)|}$, and the resource allocation index to $f(x) = \frac{1}{|\Gamma(x)|}$. This group of metrics directly supported by DotHash includes the majority of metrics listed by Martínez et al. (2016) and Lü and Zhou (2011).

2.5 Application: link prediction

The link prediction problem consists of inferring which links will appear in the future, called the *inference time interval*, given a snapshot of a graph (Liben-Nowell and Kleinberg, 2007).

In practice, the task is seen as a ranking problem for pairs of nodes, i.e., the approaches compare pairs of nodes and predict that the most similar pairs are those that are likely to connect in the future.

This section presents experiments comparing DotHash to the baselines discussed in Section 2.3 for the link prediction problem. While the baseline estimators, MinHash and SimHash, are limited to estimating the Jaccard index, DotHash offers the ability to estimate more complex metrics, known to be more effective for certain applications, including link prediction.

The quality of the methods is evaluated based on how well they rank pairs that effectively form in the inference time interval, against random pairs that do not connect. These edges are respectively called the positive and negative samples. The most popular metric, Hits@ K , counts the ratio of positive edges that are ranked K or above the negative edges (Hu et al., 2020). The Jaccard, Adamic-Adar, common neighbors, and resource allocation indices are all used in the literature for establishing this ranking.

The experiments were conducted as follows. The methods were implemented using PyTorch (Paszke et al., 2019) and the Torchhd library (Heddes et al., 2023). The experiments were executed on a machine with 20 Intel Xeon Silver 4114 CPUs, 93 GB of RAM and 4 Nvidia TITAN Xp GPUs. However, the experiments only used a single CPU or GPU. Each experiment was repeated 5 times on the CPU and 5 times on the GPU. The code is available at: <https://github.com/mikeheddes/dot hash>.

2.5.1 Datasets

The following graph datasets were used to evaluate DotHash and the baselines for the link prediction problem:

- Drugs (Guney, 2017): This dataset represents the interaction between drugs, where the joint effect of using both drugs is significantly different from their effects separately.
- Wikipedia (Rozemberczki et al., 2021): This dataset represents a webpage network where each node represents a web page and the edges represent hyperlinks between them.
- Facebook (Rozemberczki et al., 2021): This dataset represents a network of verified Facebook pages where nodes correspond to Facebook pages and edges are the mutual likes between pages.
- Proteins (Szklarczyk et al., 2019): This dataset represents a protein network where nodes represent proteins from different species and edges show biological meaningfulness between the proteins associations.
- Epinions (Richardson et al., 2003): This dataset represents the who-trusts-whom social network of the general consumer review site `epinions.com`, where each node represents a user, and each edge is a directed trust relation.
- Slashdot (Leskovec et al., 2008): This dataset represents the Slashdot social network as of February 2009, where each node is a user and each edge is a directed friend/foe link.

2.5.2 Results

The results, presented in Figure 2.4, provide evidence to substantiate the claim that estimating more appropriate metrics makes DotHash a better estimator for the link prediction problem. By employing sufficient dimensions and selecting suitable metrics, DotHash consistently outperforms the baselines across all datasets and approaches the exact indices, shown in dashed lines. Each solid line represents the mean of the values observed in the five repeti-

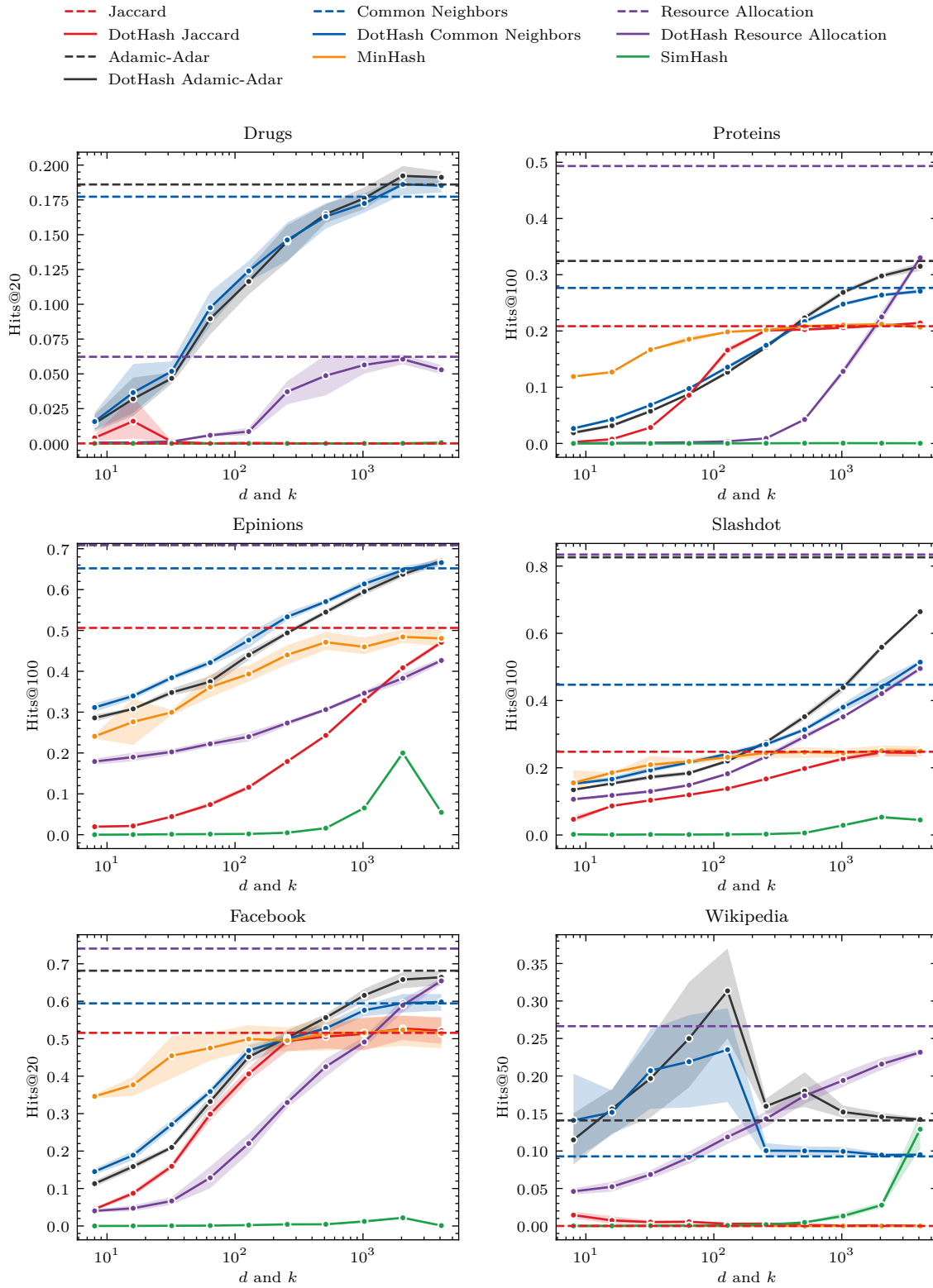


Figure 2.4. Link prediction accuracy results while varying the number of dimensions d and hashes k .

tions, and the corresponding colored shades show the 95% confidence interval. Importantly, as explained in the previous section, the adoption of DotHash does not impose a significant additional computational burden compared to the baseline methods. Figure 2.5 shows the normalized execution time of each method on the same datasets. For more details on execution times, please refer to Appendix C.

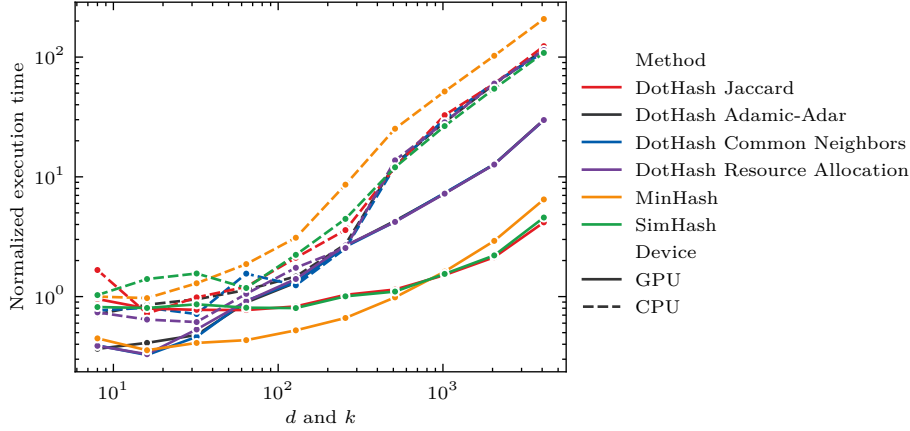


Figure 2.5. Normalized average link prediction execution time of different methods, relative to MinHash with $k = 8$ running on CPU. The average is calculated over all link prediction datasets.

2.6 Application: document deduplication

The detection of duplicate documents was the first major application to motivate the development of set similarity estimators. Both MinHash and SimHash were developed and became popular for their use in deduplicating web pages in Google and Alta Vista search engines. The experiments in this section seek to show how DotHash compares to these methods in this important problem.

Once again, the broader capability of DotHash enables the estimation of richer metrics than the Jaccard coefficient between documents. While MinHash and SimHash give equal weight to shingles (sequences of words) when comparing documents, DotHash can assign different weights to reflect how important each shingle is to each document in the corpus. The

intuition is that the more important a term is to identify each text, the more its presence in different texts indicates their similarity.

One of the most popular ways of evaluating how important a term is to a document is by the *inverse document frequency*, or IDF (Jones, 1972). The measure is widely used in the information retrieval literature for text mining (Robertson, 2004; Henzinger et al., 2003), and is defined as

$$\text{idf}(x) = \log \frac{|D|}{|\{d \in D : x \in d\}|},$$

where $|D|$ is the number of documents in the entire corpus and $|\{d \in D : x \in d\}|$ the number of documents that contain the term x . Given this, documents A and B can be compared not only by the number, but also by the importance of common shingles as

$$\text{sim}_{\text{idf}}(A, B) = \sum_{x \in A \cap B} \text{idf}(x),$$

which is in the family of functions that DotHash can estimate.

The experiments were conducted in a manner similar to the link prediction experiments. That is, the methods were implemented using PyTorch (Paszke et al., 2019) and the Torchhd library (Heddes et al., 2023). The experiments were executed on a machine with 20 Intel Xeon Silver 4114 CPUs, 93 GB of RAM and 4 Nvidia TITAN Xp GPUs. However, the experiments only used a single CPU or GPU. Each experiment was repeated 5 times on the CPU and 5 times on the GPU. The code is available at: <https://github.com/mikeheddes/dot hash>.

2.6.1 Datasets

The following datasets were used to evaluate DotHash and the baselines for the document deduplication problem:

- CORE Deduplication Dataset 2020 (Gyawali et al., 2020): This dataset consists of more than 1.3M scholar documents labeled as duplicates or non-duplicates.
- Harvard Dataverse Duplicate Detection Restaurant dataset (Ali, 2022). This dataset consists of a collection of 864 restaurant records containing 112 duplicates.
- Harvard Dataverse Duplicate Detection cddb dataset (Ali, 2022): This dataset contains a set of 9,763 records with 299 duplicated entries, with each row representing information about a particular audio recording.

2.6.2 Results

Table 2.2 shows the comparative results between the three estimators for the near-duplicate detection problem for the three different datasets. For each estimator, the near-duplicity detection accuracy is presented in terms of Hits@25 and execution time in seconds. The number of hash values and dimensions for MinHash, SimHash, and DotHash were set to 128, 500, and 10,000, respectively. These values were chosen to ensure comparable accuracy and enable the observation of differences in computational efficiency between the algorithms.

The numerical results presented indicate that DotHash is able to surpass the accuracy of MinHash in all datasets, even with a number of dimensions in which its execution time is between 0.5 and $3\times$ faster. The same is observed in the comparison with SimHash, which obtains the lowest accuracy in all cases. These empirical results reinforce the findings observed in the link prediction experiments, highlighting the advantage of DotHash. By efficiently estimating richer metrics through a single dot product computation between set sketches, DotHash consistently delivers superior cost-benefit compared to other estimators.

Table 2.2. Accuracy and computation time (s) results for document duplicate detection

Dataset	Metric	MinHash	SimHash	DotHash
CORE 20'	Hits@25	0.6246	0.3991	0.6286
	Time	0.0269	0.0284	0.0088
Rest	Hits@25	0.9598	0.7745	0.9819
	Time	0.0014	0.0011	0.0006
cddb	Hits@25	0.9058	0.6496	0.9085
	Time	0.0094	0.0084	0.0066

2.7 Conclusion

DotHash is a new baseline method for estimating the similarity between sets. The method takes advantage of the tendency to orthogonality of sets of random high-dimensional vectors to create fixed-size representations for sets. A simple dot product of these sketches serves as an unbiased estimator for the size of the intersection of sets. DotHash allows estimating a larger set of metrics than existing estimators. Experiments show that this makes it more appropriate for link prediction and duplicate detection tasks. Adding the theoretical and practical contribution, DotHash is a new framework for a problem of increasing relevance in data mining and related areas.

Chapter 3

Join size estimation

This chapter presents a novel approach to the problem of estimating the join size of multi-join queries in database systems, also known as cardinality estimation. In this setting, the join size is the number of tuples that result from a join. In general, the execution time of a database query is highly sensitive to the join execution order (Leis et al., 2015). Since determining the optimal join order is known to be NP-hard (Ibaraki and Kameda, 1984), query optimizers often rely on intermediate join size estimates¹ as their primary input to assess the cost of a particular join order. The chapter describes the UCI sketch², a solution to this problem within the general streaming data framework depicted in Figure 3.1 and elaborated upon below.

3.1 Introduction

The analysis of streaming data has amassed considerable attention, driven by the increasing demand for real-time data processing, and the remarkable advancements in algorithms that

¹The intermediate query result can itself be expressed as a query (a subquery of the query to optimize).

²The name “UCI sketch” was coined by Mayer et al. (2026).

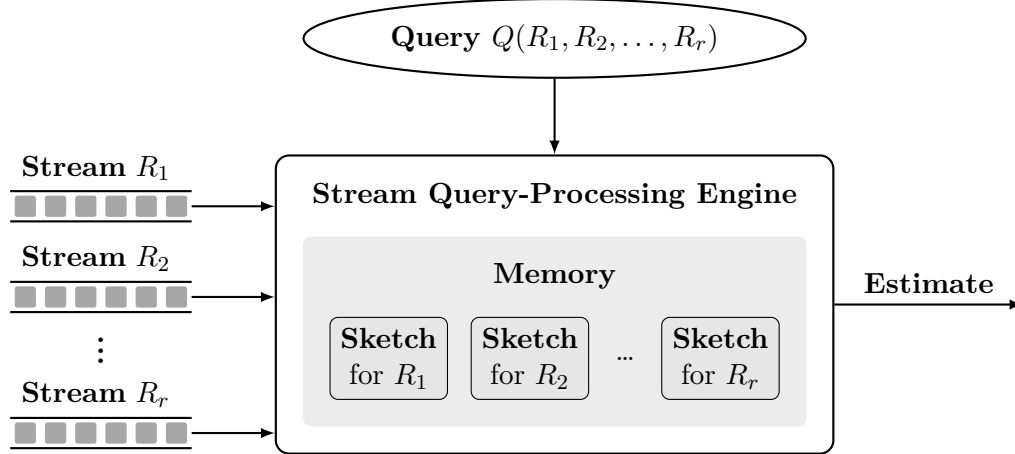


Figure 3.1. Streaming query-processing scheme

enable efficiently querying and analyzing data streams under memory constraints. Streaming data refers to data that is received sequentially and is often too large to be stored in its entirety, hence requiring algorithms that can process the data on-the-fly (Babu and Widom, 2001; Gilbert et al., 2001). Efficiently providing answers to queries over streaming data is vital in numerous application environments, including recommendation systems (Huang et al., 2015; Chen et al., 2013, 2020), smart cities (Giatrakos et al., 2017; Biem et al., 2010), network traffic monitoring (Cormode et al., 2003; Dobra et al., 2002), natural language processing (Goyal et al., 2012), and analysis of market data in financial systems (Cao et al., 2014; Ross et al., 2011; Stonebraker et al., 2005). The focus on the streaming data setting in this chapter stems from its generality. Streaming algorithms are not only effective in streaming settings but also seamlessly extend their applicability to non-streaming scenarios.

The rise of data-intensive applications has created a need for data structures that can handle massive volumes of data efficiently. This context motivated the emergence of *synopsis data structures* (Gibbons and Matias, 1999), a family of data structures designed to represent large quantities of data with sublinear space complexity, which is imperative in the given context. Examples include random samples, histograms, wavelets and sketches (Cormode et al., 2011), all of which are actively being researched as a means of analyzing and querying streaming data (Cormode, 2022; Li and Li, 2018; Cormode et al., 2011; Aggarwal and Yu,

2007). These algorithms operate by generating a compressed representation of the original data, which can then be utilized to estimate a specific property or a set thereof. For example, the popular Bloom Filter (Bloom, 1970) is widely used for membership testing, while the Count-Min sketch (Cormode and Muthukrishnan, 2005) is commonly used for frequency estimation. Both of these methods are examples of sketches. Besides their supported queries, various factors differentiate sketching methods, including sketch size, sketch initialization time, update time, and inference time (Cormode et al., 2011) (see Section 3.2 for details). These characteristics serve as catalysts for diverse research avenues and are crucial to consider when utilizing or developing a sketching method that is tailored to a specific use case.

Aside from basic statistical properties such as count, sum, and mean, much useful information from a data stream is derived from its frequency distribution, or histogram. This becomes particularly relevant when we need to compare or estimate functions across multiple data sets, such as the number of shared items. Frequency-based sketches are a class of sketching methods specifically designed for estimating functions of the frequency vector. Among these, the *AMS* (Alon-Matias-Szegedy) sketch (Alon et al., 1996), also known as *Tug-of-War* sketch, stands out as a prime example, renowned for its established reputation of being both simple and remarkably effective in a wide array of applications. The AMS sketch was initially introduced to estimate the second frequency moment of a data stream, but it was later demonstrated to also estimate the cardinality of any equi-join between two relations (Alon et al., 1999).

Interestingly, it turns out that many important functions on the frequency vector can be expressed as the cardinality of an equi-join. This equivalence is an important driver behind the development of sketches, often seen as an approximate query processing (AQP) technique (Li and Li, 2018). One particularly relevant use case is estimating the join cardinality, which is crucial for query optimizers to efficiently assess the cost of candidate physical join plans. The challenge of determining an appropriate join order is a highly researched problem in the

field of databases (Chaudhuri, 1998; Lan et al., 2021), and the methods employed typically rely on cardinality estimates as the primary input (Leis et al., 2015).

Two significant breakthroughs emerged a few years after the introduction of the AMS sketch. First, Charikar et al. (2002) proposed the *count sketch*, which divides estimates into “buckets” instead of computing the mean of multiple independent and identically distributed (i.i.d.) estimates. This approach makes the sketch more accurate for skewed data and dramatically speeds up its updates (Rusu and Dobra, 2008; Thorup and Zhang, 2004). Second, Dobra et al. (2002) proposed a generalization of the AMS sketch that enables the cardinality estimation of multi-join queries, thus considerably expanding the algorithm’s applicability.

Although both methods have gained popularity for their respective advantages, the existing literature has highlighted the task of integrating all these benefits into a unified approach as a challenging and unresolved problem (Cormode et al., 2011; Izenov et al., 2021). The specific challenge lies in effectively handling multi-join queries with fast updates. This challenge becomes even more significant when considering the prevalence of such multi-joins, as they constitute the majority of queries (see Section 3.4.1). The difficulty of combining the count sketch with the AMS-based multi-join query estimation method arises from the use of binning, as will be discussed in Section 3.3, yet binning is essential for achieving the benefits of the count sketch.

This chapter describes the UCI sketch, a new sketching method that combines insights from both Charikar et al. (2002) and Dobra et al. (2002). The method relies on the intuitive observation that the operation used to merge single-item AMS sketches to form sketches of tuples, the Hadamard product, is incongruous with the sparse nature of the count sketch. In essence, when two count sketches undergo the Hadamard product, the resulting sketch will likely lose information due to the sparsity of the count sketches.

The core innovation of the new approach lies in employing circular convolution instead of the

Hadamard product for counting tuples in a data stream. Unlike the Hadamard product, this operation ensures the preservation of information from the operands in the resulting count sketch. This is complemented by incorporating circular cross-correlation in the estimation procedure. The UCI sketch not only exhibits superior estimation accuracy when applied to real data and queries, but also operates within the same memory constraints. Moreover, the time complexity of the sketch update process is significantly improved, enabling estimates to be computed orders of magnitude faster. Theorem 3.4 proves that the estimator is unbiased and offers error guarantees equivalent to Dobra et al. (2002). Importantly, the UCI sketch does not require prior knowledge of the data distribution. The empirical findings presented in Section 3.4 support the practical applicability of the method, underscoring its significant advancement in addressing the aforementioned open problem.

3.2 Background

This section provides the necessary background and introduces the key methodologies and notation used in this chapter. For an overview of the notation see Table 3.1.

3.2.1 Streaming data

The focus of this chapter is on *streaming data* analysis, a prominent application area for synopsis data structures (Gibbons and Matias, 1999), which involves real-time processing of data that arrives at a high frequency. Streaming data naturally arises in many big data applications, including network traffic monitoring (Cormode et al., 2003; Dobra et al., 2002), recommendation systems (Huang et al., 2015; Chen et al., 2013, 2020), natural language processing (Goyal et al., 2012), smart cities (Giatrakos et al., 2017; Biem et al., 2010), and analysis of market data in financial systems (Cao et al., 2014; Ross et al., 2011; Stonebraker et al., 2005). Algorithms for streaming data are designed to handle data that can only be

Table 3.1. Notation for join-size estimation

Symbol	Definition
$[n] = \{1, 2, \dots, n\}$	Domain of items
(i, Δ)	Tuple of item and frequency change
$f, g \in \mathbb{R}^n$	Frequency vectors
$\Pi \in \mathbb{R}^{m \times n}$	Random matrix
$c \in \mathbb{R}^m$	Vector of counters, i.e., the sketch
$s_j : [n] \rightarrow \{-1, +1\}$	Random sign function
$h_j : [n] \rightarrow [m]$	Random bin function
$R_1, R_2, \dots, R_r$	Database relations
$Q(R_1, R_2, \dots, R_r)$	Query over relations
$u, v \in [w]$	Joined attribute names (vertices)
$\{u, v\} \in E$	Join from all joins (edges)
$u \in \Omega(R_k)$	Joined attribute u of relation R_k
$v \in \Gamma(u)$	Joined attribute v with u
$\Psi(u) \in [w - r + 1]$	Join graph component of attribute u
$I_k = [n] \times \dots \times [n]$	Domain of relation R_k
$\mathcal{F}_k(i)$	Frequency of tuple i in relation R_k
$X, \mathbb{E}[X]$	Estimate and expected value
ϵ, δ	Error bound and confidence
$y, \hat{y}$	True and predicted cardinality

observed once, in arbitrary order, as it continuously arrives (Dobra et al., 2002). Consequently, these algorithms must be highly efficient in processing each input, while utilizing limited memory resources, to keep up with the rapid influx of new data.

Describing the method within the streaming data setting demonstrates its applicability to a broad range of scenarios. This is because streaming algorithms are also applicable when multiple data accesses or a specific access order are allowed. Inversely, algorithms that require multiple data accesses or a specific access order, like many learning-based methods, clearly do not apply in a streaming data setting. Even when a streaming algorithm is not strictly necessary, optimizing for fewer data accesses remains advantageous because it minimizes potentially costly I/O operations.

The problem is formulated as follows, based on Cormode and Muthukrishnan (2005): consider a vector $f(t) \in \mathbb{R}^n$, which is assumed too large to be stored explicitly and is there-

fore presented implicitly in an incremental fashion. Starting as a zero vector, $f(t)$ is updated by a stream of pairs (i_t, Δ_t) which increments the i_t -th element by Δ_t , meaning that $f_{i_t}(t) = f_{i_t}(t-1) + \Delta_t$, while the other dimensions remain unchanged. The *items* i_t are members of the *domain*³ $[n] = \{1, 2, \dots, n\}$; $\Delta_t \in \mathbb{R}$ are the *changes in frequency* and $f(t)$ is called the *frequency vector*. At any time t , a *query* may request the computation of a function on $f(t)$. Specific streaming settings are further classified by their type of updates, as follows:

- **cash-register:** $\Delta_t > 0$ on every update;
- **strict turnstile:** for some updates Δ_t can be negative, but $f_i(t) \geq 0$ for all i and t ;
- **general turnstile:** both updates and entries of the vector $f(t)$ can assume negative values at any time $t > 0$.

The algorithms discussed in this chapter utilize synopsis data structures to efficiently handle data streams, eliminating the need to explicitly store and compute over $f(t)$ to answer a set of supported queries. The following section will introduce a group of sketching techniques known as *linear sketches*. This family of methods, which includes the novel approach described in this chapter, is designed to support the most general streaming setting, i.e., the general turnstile.

3.2.2 Linear sketching

Sketching techniques are a popular set of methods for dealing with streaming data and approximate query processing (Cormode et al., 2011). Both the baselines and the method described in this chapter are *linear sketches*, meaning that the summaries they generate can be represented as a *linear transformation* of the input. In contrast, the Bloom filter (Bloom, 1970) serves as a classic example of a *non-linear* sketch.

³Without loss of generality, we can assume $i \in [n]$ (Dobra et al., 2002; Ganguly et al., 2004).

Formally, for a given vector $x \in \mathbb{R}^n$, we define a linear sketch as a vector obtained by Πx , where $\Pi \in \mathbb{R}^{m \times n}$ is some random matrix, and $m \ll n$. The linearity of the transformation offers notable advantages (Cormode et al., 2011): it allows for processing items in any order and combining different sketches through addition. This enables efficient handling of data and supports map-reduce style processing of large data streams. In every sketching method, the random matrix is thoughtfully designed to enable the estimation of one or multiple functions over x , utilizing only its “summary” captured by Πx , thereby eliminating the necessity for accessing x itself. When sketching techniques are used in a streaming scenario they are often referred to as *frequency-based sketches*, where the input vector x is the frequency vector $f(t)$ defined in Section 3.2.1. Hereafter, the time argument will be omitted from the frequency vector for brevity.

At this point, one naturally wonders: how can we transform the vector f of size n , which is already considered too large, using a matrix Π that is even larger with size mn ? Streaming algorithms cleverly represent the matrix Π succinctly using hash functions, enabling them to generate just the column of Π that is needed to add a given item. Further details on this process will be provided later. Furthermore, it is crucial to ensure that the sketch is efficiently updated as f changes, i.e, as new items stream in. This can be achieved by making Π sparse, as we will explore shortly. The effectiveness and versatility of a sketching method primarily relies on the following key properties (Cormode et al., 2011):

- **Sketch size:** The total number of counters and random seeds required by the sketch, determined by the parameter m .
- **Initialization time:** The time it takes to initialize the sketches. Typically, this involves simply setting a block of memory to zeros, and sampling the random seeds for the hash functions.
- **Update time:** In streaming settings, algorithms must keep pace with the high influx

of items. The update time determines the highest item throughput rate that can be sustained.

- **Inference time:** The time it takes to compute an estimate from the generated sketches.
- **Accuracy:** It is crucial to understand the accuracy of an estimator for a given memory budget (limiting the sketch size) and throughput requirement (constraining the update time).
- **Supported queries:** Each sketch is designed to enable estimation of a specific set of functions on the input vector. Typically, a query-specific procedure needs to be performed on the sketch to approximate the value of a particular function.

The following section will describe some important sketching methods from the existing literature that have particularly space- and time-efficient ways of representing and computing Πf .

3.2.3 AMS sketch

The *AMS* sketch, also referred to as the *Tug-of-War* or *AGMS* sketch, is a pioneering technique for frequency-based sketching that was first introduced by Alon, Matias, and Szegedy (AMS) (Alon et al., 1996). The method was originally proposed as a way to estimate the *second frequency moment* F_2 of a data stream, where $F_2 = \|f\|_2^2 = \sum_{i=0}^{n-1} f_i^2$ and f is the frequency vector of the stream as defined in Section 3.2.1. The AMS sketch is represented by a vector $\mathbf{c}$, containing $m = O(1/\epsilon^2)$ counters c_j , for $j \in [m]$, where $0 < \epsilon < 1$ is the relative error bound. The counters are i.i.d. random variables obtained by $c_j = \sum_{i=0}^{n-1} f_i s_j(i)$, where each $s_j : [n] \rightarrow \{-1, +1\}$ is drawn from a family of 4-wise independent hash functions (see Definition 1.2). These hash functions are used to compute the random projection Πf with-

out representing Π explicitly, since $\Pi_{j,i} = s_j(i)$. To establish a confidence level δ , one can take the median of $O(\log 1/\delta)$ independent estimates. The overall method then requires only $O((1/\epsilon^2) \log 1/\delta)$ counters. Taking the median of i.i.d. estimates, sometimes called the “median trick”, is universal among sketching methods because it is an effective way to rapidly improve the confidence level of an estimate by Hoeffding’s inequality (see Section 1.2.6). This concept will therefore be revisited in the subsequent discussion of other methods.

In their original work, Alon et al. (1996) showed that $\frac{1}{m}\langle \Pi f, \Pi f \rangle$ is an unbiased estimator of F_2 . In fact, it can be demonstrated more generally that for any two vectors f and g , the normalized inner product of their AMS sketches $\frac{1}{m}\langle \Pi f, \Pi g \rangle$ is an unbiased estimator for their inner product $\langle f, g \rangle$ (Alon et al., 1999). Notably, when f and g correspond to the frequency vectors of a given attribute of two database relations, this estimated value corresponds to the equi-join size of these relations over that attribute. Theorem 3.1 formally states the expectation, and bounds the variance of the AMS sketch.

Theorem 3.1 (AMS sketch). *For any vectors $f, g \in \mathbb{R}^n$ and a random matrix $\Pi \in \mathbb{R}^{m \times n}$ constructed by 4-wise independent hash functions $s_j: [n] \rightarrow \{-1, +1\}$ for $j \in [m]$ and $\Pi_{j,i} = s_j(i)$,*

$$\mathbb{E} \left[\frac{\langle \Pi f, \Pi g \rangle}{m} \right] = \langle f, g \rangle, \quad \text{and} \quad \text{Var} \left(\frac{\langle \Pi f, \Pi g \rangle}{m} \right) \leq \frac{2}{m} \|f\|_2^2 \|g\|_2^2.$$

The proof is given by Alon et al. (1999, Lemma 4.4).

3.2.4 Count sketch

In order to ensure fast sketch updates, i.e., sublinear with respect to its size m , it is desirable for sketching methods to use a *sparse* linear transformation Π when processing input vectors. However, note that the AMS sketch requires changes in all m counters for each

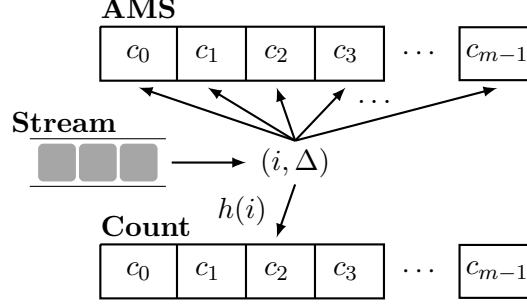


Figure 3.2. Comparison of the AMS and count sketches performing an update

update. Consequently, the accuracy of the estimator is constrained by the need to maintain a throughput that corresponds to the rate of incoming items as well as the available memory budget.

The *count sketch* is another linear sketching method that emerged after AMS and overcomes this limitation by allowing the update time to be independent of the sketch size. It achieves this by employing a technique called the “hashing trick,” (Weinberger et al., 2009; Cormode et al., 2011) which ensures that only one counter per estimate is modified during each update. As a result, the count sketch improves the update time complexity from $O((1/\epsilon^2) \log 1/\delta)$ to just $O(\log 1/\delta)$, while maintaining not only the same error bounds, but also the same space and inference time complexities as the AMS sketch. The AMS sketch and count sketch update procedures are compared in Figure 3.2. Although the count sketch was originally introduced for the *heavy hitters* problem (Charikar et al., 2002), the same hashing trick can be applied to speed up the AMS sketch, which is commonly known as the *Fast-AMS sketch* (Cormode and Garofalakis, 2005).

The value of each counter in the count sketch is given by $c_j = \sum_{i=0:h(i)=j}^{n-1} f_i s(i)$, where $h : [n] \rightarrow [m]$ is a random bin function drawn from a family of 2-wise independent hash functions, and $s : [n] \rightarrow \{-1, +1\}$ is again a random sign function drawn from a family of 4-wise independent hash functions. The corresponding random matrix Π is specified by $\Pi_{j,i} = s(i)[h(i) = j]$. Notice that Π has only one non-zero value per column, making it highly sparse. Also, the count sketch requires only one sign and bin hash function per estimate,

regardless of the required precision, resulting in a further reduction in memory usage. An estimate is obtained by taking the inner product between sketches. Theorem 3.2 formally states the expectation and bounds the variance of the count sketch.

Theorem 3.2 (Count sketch). *For any vectors $f, g \in \mathbb{R}^n$ and a random matrix $\Pi \in \mathbb{R}^{m \times n}$ constructed by 4-wise independent hash function $s: [n] \rightarrow \{-1, +1\}$ and 2-wise independent hash function $h: [n] \rightarrow [m]$ with $\Pi_{j,i} = s(i)[h(i) = j]$,*

$$\mathbb{E}[\langle \Pi f, \Pi g \rangle] = \langle f, g \rangle, \quad \text{and} \quad \text{Var}(\langle \Pi f, \Pi g \rangle) \leq \frac{2}{m} \|f\|_2^2 \|g\|_2^2.$$

The proof is given by Weinberger et al. (2009, Appendix 22) and Alon et al. (1999, Lemma 4.4).

The count sketch has also been shown to outperform the AMS sketch in estimation precision for skewed data distributions (Rusu and Dobra, 2008). This is because the count sketch is able to separate out the few high frequency components with high probability. This is an important trait, as it has been widely acknowledged in the literature that the majority of real-world data distributions exhibit a skewed nature (Dobra et al., 2002; Leis et al., 2015; Manerikar and Palpanas, 2009; Yang et al., 2017; Roy et al., 2016). This topic will be discussed further in Section 3.4.1.

3.2.5 Extensions to the count sketch

The count sketch was originally introduced for single-dimensional data which is represented by the frequency vector f . More recently, however, several extensions to the count sketch have been proposed which enable its usage for higher-dimensional data represented by a frequency *tensor* $\mathcal{F}$ instead (Liu et al., 2022). The most notable extensions are the *tensor sketch* (Pham and Pagh, 2013) and the *higher-order count (HOC) sketch* (Shi and Anand-

kumar, 2019). Both methods set out to reduce the computational complexity of machine learning applications. The tensor sketch was used to approximate the polynomial kernel, but finds its origin in estimating matrix multiplication (Pagh, 2013). The HOC sketch was introduced to compress the training data or neural network parameters, in order to speed up training and inference processes.

For each incoming item of the stream, both methods start by encoding all axes separately using independent instances of the count sketch. They differ in the way these individually sketched axes are combined: the tensor sketch employs circular convolution (see Definition 3.1), generating a sketch vector, whereas the HOC sketch utilizes the tensor product to produce a sketch tensor of the same order but with reduced dimensions compared to $\mathcal{F}$. The use of the tensor product ensures that mode information about the sketched data is preserved, at the cost of an exponential increase in sketch size with the order of the tensor. The circular convolution, in contrast, preserves the dimensionality of the sketch vector for any tensor order, i.e., it maps tensors to vectors.

In the context of databases, COMPASS (Izenov et al., 2021) uses HOC sketches to estimate the cardinality of multi-join queries. They additionally propose a method to approximate HOC sketches by merging count sketches. While they show promising results for query optimization, their estimation method lacks theoretical error guarantees. Moreover, Section 3.4.2 will show that, in practice, the UCI sketch achieves significantly higher estimation accuracy. The tensor sketch (see Definition 3.2) is the most related to the UCI sketch, however, the UCI sketch and the application are novel and solve an important open problem in the streaming and databases community, as will be detailed in Section 3.3.

Definition 3.1 (Circular convolution). *The circular convolution $x * y$ of any two vectors $x, y \in \mathbb{C}^m$ is a vector with elements given by $(x * y)_j = \sum_{i=1}^m x_i y_{(j-i \bmod m)+1}$ for all $j \in [m]$.*

Definition 3.2 (Tensor sketch). *Consider any order d tensor $\mathcal{F} \in \mathbb{R}^{n^d}$ and random*

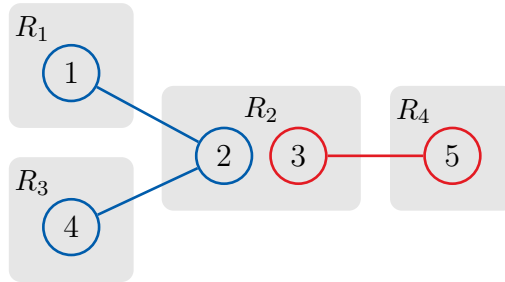
matrix $\Pi \in \mathbb{R}^{m \times n^d}$ constructed by 2-wise independent hash functions $h_k: [n] \rightarrow [m]$ and 4-wise independent hash functions $s_k: [n] \rightarrow \{-1, +1\}$ for each mode $k \in [d]$. Let $\Pi_{j,i} = S(i)[H(i) = j]$ with the decomposable hash functions

$$H(i) = \left(\sum_{k=1}^d h_k(i_k) - 1 \right) \bmod m + 1, \quad \text{and} \quad S(i) = \prod_{k=1}^d s_k(i_k).$$

Then, the tensor sketch is given by $\Pi\mathcal{F}$.

3.2.6 Multi-join with AMS sketches

Another significant advancement in linear sketching techniques emerged around the same period as the count sketch. Dobra et al. (2002) proposed a generalization of the AMS sketch that enables the estimation of complex multi-join aggregate queries, such as count and sum queries. These estimates are useful for big data analytics and query optimization (Leis et al., 2015). The proposed method addresses the scenario where a query $Q(R_1, R_2, \dots, R_r)$ involves multiple relations R_k for $k \in [r]$. An example is illustrated in Figure 3.3, which provides an intuitive visualization of this type of complex query as a disconnected, undirected graph. In this abstraction, each vertex corresponds to an attribute, edges represent the joins between them, and attributes are grouped to form relations.



```
SELECT COUNT(*) FROM R1, R2, R3, R4
WHERE R1.1 = R2.2 AND R3.4 = R2.2 AND R4.5 = R2.3
```

Figure 3.3. Example join graph and corresponding SQL query. Additional attributes in each relation, not involved in the join, are omitted for clarity.

The technique generates sketches for each relation by iterating over all the *tuples* i in each relation once. This iterative traversal of the tuples aligns with the streaming data scenario described in Section 3.2.1, enabling the sketches to be created on-the-fly as the relations are updated. The sketch c_k for relation R_k is given by

$$c_{k,j} = \sum_{i \in I_k} \mathcal{F}_k(i) \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{j,\{u,v\}}(i_u), \quad (3.1)$$

where i represents a tuple that belongs to the domain I_k of relation R_k ; I_k is the cross product of item domains $[n] \times \dots \times [n]$ for each joined attribute of relation R_k ; $\mathcal{F}_k(i)$ gives the frequency of tuple i in relation R_k ; i_u denotes the value in tuple i for attribute u ; u is an attribute from the set of joined attributes of R_k in the query, denoted as $\Omega(R_k)$ (for example, $\Omega(R_2) = \{2, 3\}$ in Figure 4.4); v is an attribute from the set of attributes joined with u , denoted as $\Gamma(u)$ (for example, $\Gamma(2) = \{1, 4\}$ in Figure 4.4). We represent a join between two attributes with $\{u, v\}$. Both $u, v \in [w]$ are from the set of all joined attributes. The notation assumes that all attributes are globally unique, which can easily be achieved in practice, for instance, by concatenating the relation and attribute names. Moreover, following Dobra et al. (2002), it is assumed that joins are non-cyclic, a self-join is thus represented as a join with a fictitious copy of the relation. It is worth noting that the copy does not need to be physically created, this is done solely to simplify the notation. Note also that the functions Γ and Ω are defined for a specific query Q . This dependence is omitted in the notation for brevity, as it is evident from their definitions.

Once the sketches are created, a query estimate is derived by performing the element-wise multiplication of the sketches, often referred to as the *Hadamard product* of sketches. This is followed by calculating the mean over the counters. Formally, this can be expressed as

$$X = \frac{1}{m} \sum_{j=1}^m \prod_{k=1}^r c_{k,j},$$

where X is an unbiased estimate of the cardinality of query Q . The expectation and variance of X are formally stated in Theorem 3.3.

Theorem 3.3. *Given an acyclic query of relations R_k for $k \in [r]$, let Equation 3.1 provide the sketches for each relation and $X = \frac{1}{m} \sum_{j=0}^{m-1} \prod_{k=0}^{r-1} c_{k,j}$ the cardinality estimate of the query, then*

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i \in I_1 \times \dots \times I_r} \mathcal{F}_1(i) \cdots \mathcal{F}_r(i) \prod_{\{u,v\} \in E} [i_u = i_v] \\ \text{Var}(X) &\leq \frac{1}{m} 3^{r-1} \prod_{k=1}^r \|\mathcal{F}_k\|_2^2\end{aligned}$$

In Lemmas 3.1 and 3.2 of Dobra et al. (2002), similar results are presented, albeit with a slightly looser bound on the variance. The proof for Theorem 3.3 is provided in Appendix A.2 for the sake of completeness.

From the perspective of the count sketch extensions discussed in Section 3.2.5, this method can be interpreted as a similar generalization but for the AMS sketch. It can thus be seen as one of the first methods to generalize sketching for tensor data, although this aspect was not explicitly mentioned in the original work.

3.2.7 Other related work

This section discusses other recent work in cardinality estimation. The Pessimistic Estimator (Cai et al., 2019) is an interesting sketching technique which provides an upper bound of the cardinality. They show that it improves upon the cardinality estimator within PostgreSQL. However, the practical use of the method is limited due to its lengthy estimation time (Han et al., 2021), at times exceeding the query execution time, as also mentioned by the authors.

Since the inception of sketching, a number of techniques have been proposed that complement the aforementioned sketches. Among these techniques, the Augmented Sketch (Roy et al., 2016) and the JoinSketch (Wang et al., 2023) aim to improve the accuracy of sketches for skewed data by separating the high- from the low-frequency items in the sketch. The counters of the high-frequency items are explicitly represented in an additional data structure, thereby preventing them from causing high estimation error due to hash collisions. Another notable technique is the Pyramid sketch (Yang et al., 2017), which employs a specialized data structure that dynamically adjusts the number of allocated bits for each counter, preventing overflows in the case of high-frequency items. It is important to note that these techniques are proposed as complementary tools, compatible with a variety of sketching methods, including the one proposed in this chapter.

There has been a parallel effort aimed at harnessing the power of machine learning for cardinality estimation. Among the various approaches, the most promising ones are data-driven methods that build query-independent models to estimate the joint probability of tuples (Han et al., 2021). Notable examples of such techniques include DeepDB (Hilprecht et al., 2020), BayesCard (Wu et al., 2020b), NeuroCard (Yang et al., 2020), and FLAT (Zhu et al., 2021). While machine learning-based cardinality estimation techniques have been receiving increasing attention, they still face important limitations: these methods are presently viable only in scenarios where supervised training is feasible, their accuracy has not consistently lived up to expectations (Müller, 2022), and they prove impractical in situations with frequent data updates, such as streaming settings, due to their high cost of model updates (Han et al., 2021). Section 3.4.4 demonstrates that the UCI sketch, described in the next section, not only avoids these performance limitations but also achieves significantly higher accuracy compared to the machine learning techniques.

3.3 UCI sketch

This section describes the UCI sketch, which solves the longstanding challenge of integrating the key advantages of the count sketch, such as its efficient update mechanism and superior accuracy when handling skewed data, into a method that effectively estimates the cardinality of multi-join queries. Devising such a method is recognized as a challenging task, as underscored not only by the author of the Count-Min sketch (Cormode et al., 2011) but also by other work in the field (Izenov et al., 2021). This acknowledgment highlights the importance of the contribution described in this chapter. The importance of solving this problem is further underscored when considering the prevalence of multi-join queries. In fact, an analysis of two sets of widely recognized benchmarking queries, as discussed in Section 3.4.1, reveals that multi-joins constitute approximately 97% of all their queries (Leis et al., 2015; Han et al., 2021).

3.3.1 Key insight for preserving information

The main challenge of combining the count sketch with the method proposed by Dobra et al. (2002) lies in the inability to effectively merge count sketches using the Hadamard product. Dobra et al. (2002) create the sketch for a tuple as the Hadamard product of the AMS sketches for each value in the tuple. As discussed earlier, the advantages of the count sketch stem from its sparsity. However, as illustrated in the left part of Figure 3.4, it is precisely this characteristic that results in a loss of information, with high probability, when combining sketches with the Hadamard product. Essentially, due to the sparsity, the non-zero entry in each sketch is highly likely to appear at a different position, causing the result of the element-wise multiplication to yield a zero vector, devoid of information.

To address this issue, the core concept behind the described method, as depicted in the right part of Figure 3.4, involves the utilization of circular convolution paired with circular

$$\begin{array}{lcl}
\Pi_{,a}^{(1)} = & \boxed{\begin{array}{ccccc} 0 & 0 & -1 & 0 & 0 \end{array}} & \boxed{\begin{array}{ccccc} 0 & 0 & -1 & 0 & 0 \end{array}} \\
& \circ & * \\
\Pi_{,b}^{(2)} = & \boxed{\begin{array}{ccccc} 0 & 0 & 0 & 1 & 0 \end{array}} & \boxed{\begin{array}{ccccc} 0 & 0 & 0 & 1 & 0 \end{array}} \\
& = & = \\
& \boxed{\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \end{array}} & \boxed{\begin{array}{ccccc} -1 & 0 & 0 & 0 & 0 \end{array}}
\end{array}$$

Figure 3.4. Comparison of the Hadamard product (left) and circular convolution (right) on two single-item count sketches. The resulting sketch represents the 2-tuple (a, b) .

cross-correlation (see Definitions 3.1 and 3.3) during inference. With circular convolution, the resulting sketch has the product of the non-zero entries in the bin that corresponds to the sum of the non-zero indices, modulo m . Unlike the Hadamard product, this operation guarantees that the information is preserved. Subsequent sections will delve deeper into this intuition and formalize it. Crucially, the circular convolution of single-item count sketches can be computed in $O(1)$ time, with respect to the sketch size m . This means that the sketch of a stream can be updated in constant time for each arriving tuple, in contrast to the $O(m)$ time required by the AMS sketch with the Hadamard product. As will be shown empirically in Section 3.4.3, this translates to sketch updates that are orders of magnitude faster.

Definition 3.3 (Circular cross-correlation). *The circular cross-correlation $x \star y$ of any two vectors $x, y \in \mathbb{C}^m$ is a vector with elements given by $(x \star y)_j = \sum_{i=1}^m \bar{x}_i y_{(j+i-2 \bmod m)+1}$ for all $j \in [m]$, where $\bar{x}_i$ denotes the complex conjugate of x_i .*

3.3.2 General formulation by example

To facilitate a better understanding of the UCI sketch, the following illustrative example query serves as an initial demonstration. This walkthrough of the estimation process provides insights into the method’s inner workings. Subsequently, the method is formally presented via its pseudocode. Mathematical proofs demonstrate that the method serves as an unbiased cardinality estimator for the previously defined family of multi-join queries, accompanied by established bounds on the estimation error. A general overview of the estimation procedure

is available in Algorithm 1. The following description utilizes the example query illustrated in Figure 3.3.

Algorithm 1 UCI sketch join-size estimation procedure

input: Relations R_k for $k \in [r]$, query $Q(R_1, R_2, \dots, R_r)$, and sketch size m .

output: Estimate X

- 1: $s \leftarrow \text{SampleSignHashes}(Q, m)$
 - 2: $h \leftarrow \text{SampleBinHashes}(Q, m)$
 - 3: **for each** relation R_k **do**
 - 4: $c_k \leftarrow \text{CreateSketch}(R_k, s, h, Q, m)$
 - 5: **end for**
 - 6: $X \leftarrow \text{GetQueryEstimate}(c_1, c_2, \dots, c_r, Q, m)$
-

Initialization The method starts by initializing the necessary hash functions and counters. An independent random *sign function* $s_{\{u,v\}}: [n] \rightarrow \{-1, +1\}$ is sampled from a family of 4-wise independent hash functions for every join $\{u, v\} \in E$, represented by an edge in Figure 4.4. Moreover, each graph component is assigned an independent random *bin function* $h_{\Psi(u)}: [n] \rightarrow [m]$, drawn from a family of 2-wise independent hash functions. Here, $\Psi(u)$ represents the graph component to which attribute u belongs. A graph component comprises a set of attributes connected by joins, forming a subgraph that is not part of any larger connected subgraph. In the example, there are two graph components: $\{1, 2, 4\}$ and $\{3, 5\}$, identified by the edge colors in Figure 4.4. A bin function is shared within a graph component because all the attributes that form a graph component must, by definition, be joined on equal values. Note that the equal values are mapped to the same bin by using the same bin function. Lastly, for each relation, a zero vector of m counters is initialized.

Sketching Once the counters and hash functions are initialized, the tuples from each relation stream are processed. When a tuple streams in, it is mapped to a single sign and bin, derived from the signs and bins of the joined attributes. To determine the sign of a tuple, all the signs of the joined attributes are multiplied together. For instance, tuple i from relation R_2 is hashed as follows: $s_{\{2,4\}}(i_2)s_{\{2,1\}}(i_2)s_{\{3,5\}}(i_3)$, where i_u is the value for

attribute u , and $\{u, v\}$ denotes the join between attributes u and v . This is because R_2 has two joined attributes, and one of those is joined twice. Formally, the sign of a tuple i from relation R_k is given by:

$$S_k(i) = \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{\{u,v\}}(i_u) \quad (3.2)$$

To determine the bin of the tuple, the bin indices of all the joined attributes are summed, followed by taking the modulo m . Continuing our example, we have: $(h_{\Psi(2)}(i_2) + h_{\Psi(3)}(i_3) - 2) \bmod m + 1$ because R_2 has two joined attributes. The bin index of a tuple i from relation R_k is formally given by:

$$H_k(i) = \left(\sum_{u \in \Omega(R_k)} h_{\Psi(u)}(i_u) - 1 \right) \bmod m + 1 \quad (3.3)$$

Subsequently, the sign of the tuple multiplied by the change of frequency is added to the counter at the bin index of the tuple.

The sketching process for a tuple is equivalent to circular convolution between the sketches for each joined attribute value in the tuple. Since the individual sketches have only one non-zero value, the result of the circular convolution also has one non-zero value, which can be computed in constant time with respect to the sketch size, as explained earlier. The pseudocode for the general sketch creation procedure is provided in Algorithm 2. The sketch c_k for relation R_k is formally stated as follows:

$$c_{k,j} = \sum_{i \in I_k : H_k(i)=j} \mathcal{F}_k(i) S_k(i) \quad (3.4)$$

where $\mathcal{F}_k(i)$ denotes the frequency of tuple i in relation R_k . The tuple i is from the domain $I_k = [n] \times \dots \times [n]$ of relation R_k .

Algorithm 2 UCI sketch creation procedure

```

1: function CREATESKETCH( $R_k, s, h, Q, m$ )
2:    $c_k \leftarrow \mathbf{0}$  ▷ Size:  $m$ 
3:   for each tuple  $(i, \Delta)$  in  $R_k$  do ▷ The stream of tuples
4:      $x = 1$  ▷ For accumulation of signs
5:      $j = 0$  ▷ For accumulation of bins
6:     for each attribute  $u$  in  $\Omega(R_k)$  do
7:        $j \leftarrow j + h_{\Psi(u)}(i_u)$ 
8:       for each attribute  $v$  in  $\Gamma(u)$  do
9:          $x \leftarrow s_{\{u,v\}}(i_u)x$ 
10:      end for
11:    end for
12:     $j \leftarrow j \bmod m$ 
13:     $c_{k,j} \leftarrow c_{k,j} + x\Delta$ 
14:  end for
15:  return sketch  $c_k$ 
16: end function

```

Inference Upon creating the sketches for each relation, one can proceed to estimate the query's cardinality by combining sketches using either the Hadamard product or circular cross-correlation. The computation consists of summations over the sketch size for each graph component. The sketch for each relation is indexed by the sums of the graph components that have an attribute in that relation. Sketches of relations with multiple joined attributes will, therefore, also have multiple indices, and these are summed to obtain the final index. The sketch values inside the sums are all multiplied. An estimate of the example query is then obtained as follows: $\sum_{j_1=1}^m \sum_{j_2=1}^m c_{1,j_1} c_{3,j_1} c_{2,(j_1+j_2-2) \bmod m+1} c_{4,j_2}$ because there are two graph components, and R_2 is part of both. This can be factorized as the Hadamard product between sketches c_1 and c_3 whose result is circular cross-correlated with c_2 , followed by another Hadamard product with c_4 , and lastly a summation over the elements. A cardinality estimate X is formally obtained as follows:

$$X = \sum_{j \in J} \prod_{k=1}^r c_{k,G_k(j)}, \quad \text{with} \quad G_k(j) = \left(\sum_{u \in \Omega(R_k)} j_{\Psi(u)} - 1 \right) \bmod m + 1 \quad (3.5)$$

where J is the cross product of bin domains $[m] \times \cdots \times [m]$ for each graph component.

Computing the estimates naively using Equation 3.5 has an exponential time complexity with the number of graph components. However, this can be improved significantly by factorizing the problem. One can then rely on the fact that circular cross-correlation can be computed efficiently in $O(m \log m)$ time using the fast Fourier transform (FFT). To perform this efficient estimation process methodically, one starts by selecting any joined attribute, say attribute 4 in the example. The process now aggregates all the sketches towards attribute 4 to obtain the estimate. This is implemented as a depth first traversal of the join graph with attribute 4 as the root node of a rooted tree and attributes 0 and 3 as the leaves. The general procedure for combining sketches to efficiently compute an estimate of the query is provided in Algorithm 3. The pseudocode ensures that the recursion only moves away from any selected root attribute $o \in [w]$ by keeping track of the visited nodes V . The functions (I)FFT denote the (inverse) fast Fourier transform. This procedure reduces the inference time complexity to $O(rm \log m)$, that is, nearly linear with respect to the sketch size.

3.3.3 Analysis

Now that we have discussed the estimation procedure, we present a theoretical analysis of the proposed method. We show that it is an unbiased estimator for the cardinality of multi-join queries in Theorem 3.4, and provide guarantees on the estimation error. Lastly, we provide the time complexity for each estimation stage.

Theorem 3.4. *Given an acyclic query of relations R_k for $k \in [r]$, let Equation 3.4 provide the sketch for each relation and Equation 3.5 the cardinality estimate X of the query, then*

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i \in I_1 \times \dots \times I_r} \mathcal{F}_1(i) \cdots \mathcal{F}_r(i) \prod_{\{u,v\} \in E} [i_u = i_v] \\ \text{Var}(X) &\leq \frac{1}{m} 3^{r-1} \prod_{k=1}^r \|\mathcal{F}_k\|_2^2.\end{aligned}$$

Algorithm 3 UCI sketch inference procedure

```
1: function GETQUERYESTIMATE( $c_1, c_2, \dots, c_r, Q, m$ )
2:    $o \leftarrow \text{AnyJoinedAttribute}(Q)$  ▷ The root attribute
3:    $V \leftarrow \{\}$  ▷ Global set of visited attributes
4:    $X \leftarrow \text{SumElements}(\text{CombineSketches}(o, V, m))$ 
5:   return estimate  $X$ 
6: end function
7: function COMBINESKETCHES( $u, V, m$ )
8:    $R_k \leftarrow \text{RelationOf}(u)$ 
9:    $x \leftarrow c_k$  ▷ Sketch of relation  $R_k$ 
10:   $\text{add}(V, u)$  ▷ Adds attribute  $u$  to the visited set
11:  ▷ Recurse through the other attributes in the relation.
12:  for each attribute  $u'$  in  $\Omega(R_k) \setminus \{u\}$  do
13:     $\text{add}(V, u')$ 
14:     $a \leftarrow \mathbf{1}$  ▷ Size:  $m$ 
15:    ▷ By the definition of  $\Omega$  there is at least one iteration.
16:    for each attribute  $v$  in  $\Gamma(u')$  do
17:       $a \leftarrow \text{CombineSketches}(v, V, m) \circ a$ 
18:    end for
19:    ▷ Efficient circular cross-correlation
20:     $x \leftarrow \text{IFFT}(\overline{\text{FFT}(a)} \circ \text{FFT}(x))$ 
21:  end for
22:  ▷ Recurse over the attributes joined with the current.
23:  for each attribute  $v$  in  $\Gamma(u) \setminus V$  do
24:     $x \leftarrow \text{CombineSketches}(v, V, m) \circ x$ 
25:  end for
26:  return intermediate sketch  $x$ 
27: end function
```

We present the proof in Appendix A.2.

Using Chebyshev’s inequality and the upper bound on the variance from Theorem 3.4, the absolute estimation error can be bounded by $\epsilon > 0$ with $m \geq 3^r \epsilon^{-2} \prod_{k=1}^r \|\mathcal{F}_k\|_2^2$. Furthermore, to guarantee the error with probability at most $1 - \delta$, one selects the median of $l = O(\log 1/\delta)$ i.i.d. estimates by the Chernoff bound (Alon et al., 1996). The exponential term 3^r indicates that accurately estimating queries involving many relations quickly becomes infeasible. It remains an open problem whether this exponential dependence can be improved. However, as will be shown in the following section, for moderate sized queries, involving up to 6 relations, the estimation accuracy constitutes a significant improvement over the baselines.

Table 3.2. Comparison of time complexity by stage

Method	Initialization	Update	Inference
AMS	$O(rlm)$	$O(rlm)$	$O(rlm)$
COMPASS (partition)	$O(lm^r)$	$O(rl)$	$O(lm^r)$
COMPASS (merge)	$O(rlm)$	$O(rl)$	$O(rlm^r)$
UCI sketch	$O(rlm)$	$O(rl)$	$O(rlm \log m)$

Table 3.2 presents the time complexity of each estimation stage, comparing the UCI sketch with the AMS-based technique by Dobra et al. (2002) and the two variations of COMPASS (partition and merge) (Izenov et al., 2021). The symbols r , l , and m denote the number of relations, medians, and the sketch size, respectively. The update time of the UCI sketch for each incoming tuple is remarkably efficient, with a time complexity of only $O(r \log 1/\delta)$. The efficient update time complexity, independent of the estimation error ϵ , enables the sketching of high-throughput streams even when requiring a high level of accuracy. While COMPASS also has fast updates, its exponential dependence on r during inference limits its practical use even for moderately sized queries. The UCI sketch achieves fast updates yet introduces only an additional $\log m$ term during the inference stage, compared to the AMS baseline. This slight increase in inference time is negligible when considering the substantial improvement in update time. For instance, the experiments in Section 3.4.2 go up to $m = 10^6$ with $l = 5$,

which means that the UCI sketch achieves roughly 10^6 times faster sketch updates, while having only $\log_2(10^6) \approx 20$ times slower inference. As a result, the UCI sketch effectively minimizes the overall estimation time in various crucial scenarios. These claims are further supported by the empirical results detailed in Section 3.4.

3.3.4 Integration with query optimizers

The quintessential application of the UCI sketch is the cardinality estimator within query optimizers. Query optimizers use a plan enumeration algorithm to find a good join order. The cost of a join order depends upon the sizes of the intermediate results. The cardinality estimator’s role is to provide the estimates for these intermediate sizes (Lan et al., 2021). Each intermediate cardinality can be expressed as a sub-query which the UCI sketch can estimate. The sketches for all the evaluated sub-queries can be created in a single pass over the data. Typically, each sub-query requires its own sketches; however, in cases where an attribute is joined multiple times, the sketches can be reused for each join involving that attribute. For example, to decide the join order of joins $\{1, 2\}$ and $\{2, 4\}$ in Figure 4.4, the sketch for attribute 2 can be reused for attributes 1 and 4. Section 3.4.5 demonstrates the improvement in query execution time after integrating the UCI sketch cardinality estimator into the query optimizer of PostgreSQL.

3.4 Experiments

This section presents an empirical analysis to evaluate the effectiveness of the UCI sketch and compares it to various baseline approaches. These baselines include the AMS-based method proposed by Dobra et al. (2002) and the two variations of COMPASS (Izenov et al., 2021), namely partition and merge. The primary objective is to assess the accuracy of the estimator against the baselines for a specified memory budget. Furthermore, the initialization,

Table 3.3. Database size statistics

Database	Relations	Tuples	Storage size
IMDB	21	74.2M	3.88 GB
STATS	8	1.03M	39.6 MB

sketching, and inference times of the UCI sketch are compared with those of the baselines. Secondly, the estimation error and execution time of the UCI sketch are compared with the four data-driven machine learning techniques discussed in Section 3.2.7: DeepDB (Hilprecht et al., 2020), BayesCard (Wu et al., 2020b), NeuroCard (Yang et al., 2020), and FLAT (Zhu et al., 2021). Finally, the impact of the cardinality estimator is evaluated on the query execution time of PostgreSQL.

The two sketching baselines and the presented method were implemented using the PyTorch tensor library (Paszke et al., 2019). The hash functions are implemented using efficient random polynomials over a Mersenne prime field (Ahle et al., 2020b). All experiments were conducted on an internal cluster of Intel Xeon Gold 6148 CPUs. Each experiment utilized a single CPU and 24 GB of memory. The source code for the experiments, extended results, and cardinality estimates are available online⁴.

3.4.1 Databases and queries

The limitations of synthetic databases in accurately reflecting real-world performance have been widely acknowledged in the literature (Leis et al., 2015; Han et al., 2021). To address this concern, the experiments were conducted using two established benchmarking databases containing real data: the IMDB database (Leis et al., 2015), which encompasses information on movies, actors, and their associated production companies, and the STATS database (Han et al., 2021), which comprises user-contributed content from the Stats Stack Exchange network. To provide insights into the characteristics of the databases, Table 3.3

⁴Source code: <https://github.com/mikeheddes/fast-multi-join-sketch>

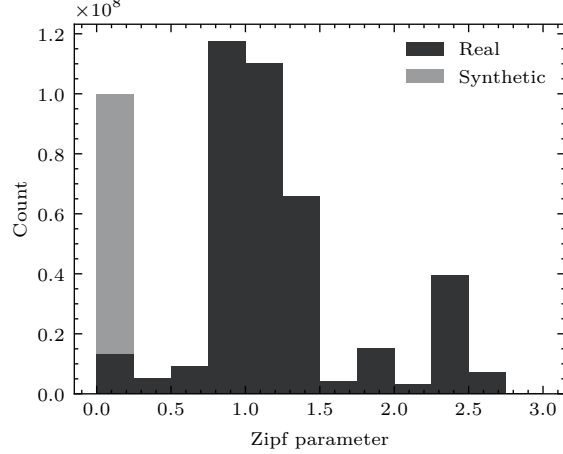


Figure 3.5. Total number of entries across all columns of both the STATS and IMDB databases, grouped by the best fit Zipf parameter of each column. Synthetic entries refer to the id and md5sum columns, which are unique by design, the real entries include all other columns.

presents statistics detailing their sizes. Additionally, Figure 3.5 showcases the distribution skewness of the database entries, highlighting the significant skewness often observed in real data (Dobra et al., 2002; Leis et al., 2015; Manerikar and Palpanas, 2009; Yang et al., 2017; Roy et al., 2016). The experiments cover all the 146 STATS-CEB and 70 JOB-light queries, in addition to the 3299 associated sub-queries from the cardinality estimation benchmark (Han et al., 2021). These queries collectively represent a diverse range of real-world workloads.

The key feature of the UCI sketch is its ability to efficiently estimate multi-join queries. As previously mentioned, the motivation for this capability is rooted in the prevalence of such queries in real-world scenarios. To further substantiate this motivation, an analysis of all the queries in both the cardinality estimation benchmark (Han et al., 2021) and the join order benchmark (Leis et al., 2015) was conducted. The results, displayed in Table 3.4, indicate that 44% of the relations in the queries are involved in multiple joins, with single joins being the most common at 57%. Notably, 97% of the queries contain at least one relation which participates in multiple joins. These statistics underscore the significance of supporting multi-join queries to effectively address the majority of real-world query scenarios.

Table 3.4. Percentage of relations among all queries by their number of joins, and the percentage of queries by their relation with the maximum number of joins

Joins	1	2	3	4	5+
Relations	57%	12%	9%	9%	12%
Queries	3%	24%	30%	20%	23%

Following Izenov et al. (2021), in the experiments the filter predicates of each query are processed during ingestion of the tuples from their respective relation streams. In many streaming algorithms, the query is assumed to be known in advance of the stream. Consequently, filtering the tuples at the time of ingestion offers advantages in terms of performance and accuracy. This approach eliminates the need to update the sketch for tuples that do not satisfy the filters. In addition, the estimation accuracy is significantly improved as some data is already filtered out from the sketches (Izenov et al., 2021). However, it is worth mentioning that sketching methods, including the UCI sketch, are also capable of handling filter predicates during inference. This can be achieved by treating the filters as joins with imaginary tables, a technique employed, for example, by Cormode et al. (2011) and Vengerov et al. (2015). Lastly, in the experiments, the median of $l = 5$ i.i.d. estimates is reported as the cardinality estimate for all sketching methods.

3.4.2 Estimation accuracy

We first compare the estimation accuracy of the different sketching methods in terms of the absolute relative error, defined as $|y - \hat{y}|$ divided by $\max(y, 1)$, where y is the true cardinality and $\hat{y}$ its estimate. This metric aligns with the formulation of the theoretical error bound outlined in Section 3.3.3. Figure 3.6 presents the median and the 95th percentile of the error at varying sketch sizes. The statistics are derived from 30 repetitions, each with distinct random initializations. The results are presented for a representative subset of the queries, necessitated by space constraints, but detailed results for all 216 queries can be found online⁴.

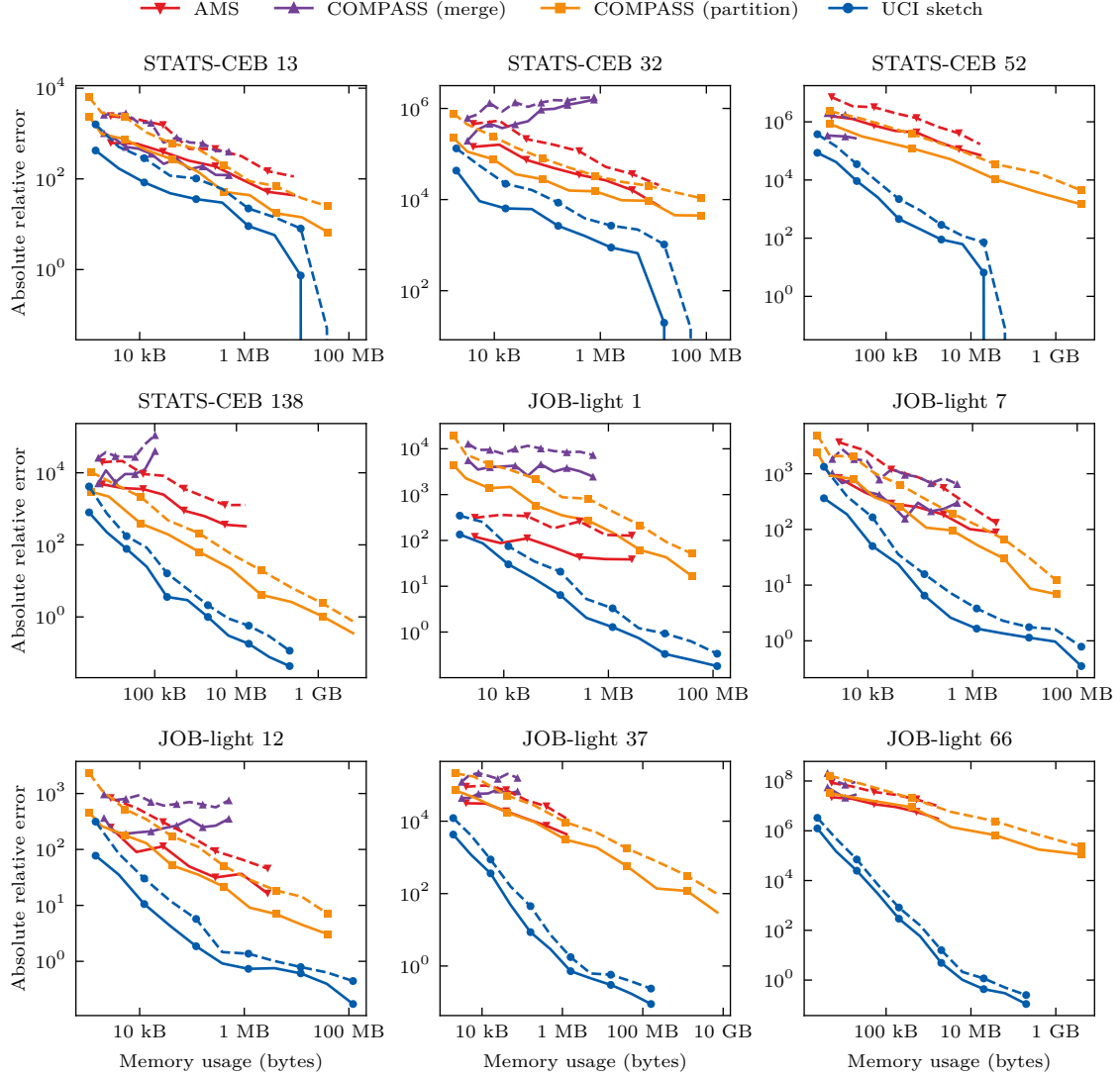


Figure 3.6. Absolute relative error of our method compared to the baselines at varying sketch sizes. Solid lines represent the median error and dashed lines denote the 95th percentile. The memory usage includes the counters of the sketches and the random seeds for the hash functions, but excludes the space needed for the intermediate inference calculations.

It is important to note that the experiments for AMS and COMPASS (merge) do not extend to the highest memory usage levels. This limitation arises due to the extensive time required to run the AMS-based experiments, which increases exponentially with each additional data point, rendering their execution quickly infeasible. Additionally, COMPASS (merge) encountered memory constraints during the inference stage, as its memory demand grows exponentially with the number of joins. Notice that the depicted memory usage excludes intermediate representations during inference, thus understating the actual memory required for COMPASS (merge).

Upon analyzing the results, we observe that the UCI sketch delivers comparable or lower error rates across all queries, often demonstrating orders of magnitude greater accuracy. The UCI sketch achieves zero error on many queries with large sketches. In realistic sketching applications, a margin of error is acceptable; therefore, sketches ranging from 1 to 10 MB could be employed for the STATS-CEB and JOB-light queries. For certain queries, like JOB-light 37 and 66, the UCI sketch not only achieves significantly lower error but also demonstrates a more rapid reduction in error with each increase in memory.

To assess the rate at which the UCI sketch improves the estimation error compared to the baselines, Figure 3.7 presents kernel density estimates of the slopes derived from the absolute relative error results for all 216 queries. These slopes are obtained by least-squares linear regression of the data for each method and query in log-log space. This means that the slope represents the exponent of a power law relationship, where the error at memory usage m is given by am^k , with k as the slope and a as the error at $m = 1$.

Remarkably, the UCI sketch exhibits a significantly faster reduction in error, highlighting its ability to achieve high accuracy with substantially less memory compared to the baselines. Considering that the real data in the experiments is skewed, as indicated in Figure 3.5, we speculate that the method successfully inherits and expands upon the advantages associated with the count sketch, particularly its effectiveness in handling skewed data. In the context

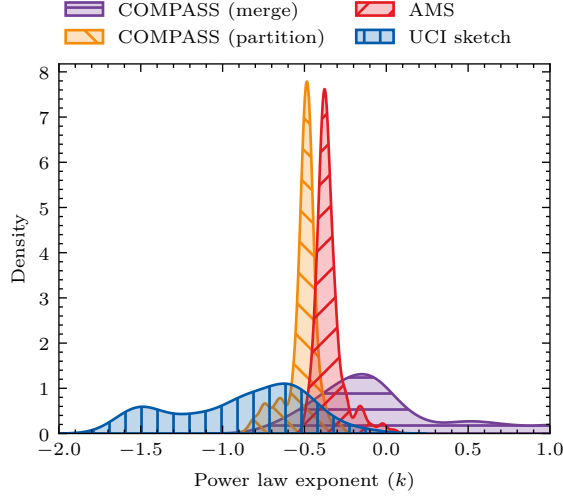


Figure 3.7. Kernel density estimates of the power law exponents from the absolute relative error plots for all 216 queries. A higher concentration on the left signifies a greater improvement in accuracy as the memory budget increases.

of multi-joins, the UCI sketch capitalizes on these benefits, demonstrating its ability to compute accurate cardinality estimates.

3.4.3 Execution times

The second set of experiments look into the execution time of the UCI sketch and how it compares to the baselines across varying sketch sizes. Figure 3.8 presents the execution times for each stage of the estimation process: initialization, sketching, and inference. The figure shows the best fit of a Gaussian process regression to the empirical results from all queries, totaling 232,323 experiments. It also contains data for the learning-based methods which will be discussed in the following section. The individual timing results for all queries are provided online⁴.

While the initialization time exhibits a similar trend for all methods, in sketching time there is a notable disparity between AMS and the other methods. This disparity directly reflects the difference in update time complexity. The methods also differ in initialization time complexity, but this is not visible in the timing results because they are plotted with

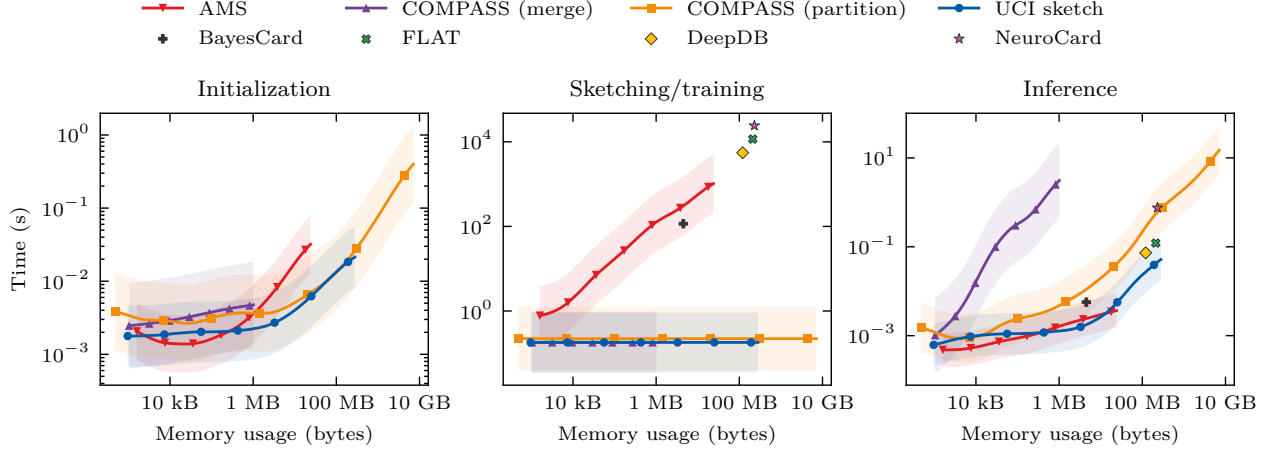


Figure 3.8. Execution times for the three stages (initialization, sketching/training, and inference) of the baselines and the UCI sketch at varying sketch/model sizes. The plots show the best fit of a Gaussian process regression of the data in log-log space, along with standard deviation. The memory usage includes the counters of the sketches and the random seeds for the hash functions. The data for BayesCard (Wu et al., 2020b), FLAT (Zhu et al., 2021), DeepDB (Hilprecht et al., 2020), and NeuroCard (Yang et al., 2020) is obtained from Han et al. (2021).

respect to their memory usage rather than the sketch size. That is, for a given sketch size COMPASS (partition) allocates more memory, but for a given memory budget, all methods have similar initialization time. Among the inference times, the UCI sketch demonstrates remarkable overall efficiency. Even for the most complex queries with the largest sketch size, the UCI sketch computes its estimate within ten seconds. In contrast, the AMS-based method requires hours to compute estimates, with the majority of that time spent on sketching, all while delivering higher error rates.

To further validate the fast update time of the UCI sketch, we assessed the maximum stream throughput for all baselines, as outlined in Table 3.5. The memory usage includes the counters of the sketches and the random seeds for the hash functions. The results were obtained by performing linear least-squares regression on the throughput measurements for each method on all queries. For smaller sketch sizes, the AMS-based method can achieve a throughput similar to the count sketch-based approaches. However, when working with larger sketch sizes required for high estimation accuracy, the AMS-based method becomes

limited to handling just a few hundred tuples per second. This significant limitation severely restricts the practical usability of AMS in streaming scenarios.

Table 3.5. Throughput in tuples processed per second

Memory usage	1 kB	10 kB	100 kB	1 MB	10 MB
AMS	5.2M	576k	63.5k	7.0k	774
COMPASS (partition)	5.9M	5.9M	5.9M	5.9M	5.9M
COMPASS (merge)	6.3M	6.2M	6.0M	5.8M	5.6M
UCI sketch	7.0M	6.8M	6.6M	6.5M	6.3M

3.4.4 Comparison with learning-based methods

The following set of experiments will compare the cardinality estimation performance of the UCI sketch with the four data-driven machine learning techniques discussed in Section 3.2.7. Specifically, we compare their execution time and estimation quality. To evaluate the quality of cardinality estimates, the q-error metric, defined as $\max(y/\hat{y}, \hat{y}/y)$ if $\hat{y} > 0$ and ∞ otherwise (Moerkotte et al., 2009), is used. Figure 3.9 presents the cumulative distribution function of the q-error for all 3299 sub-queries, showing the fraction of queries that were estimated within a certain q-error. The UCI sketch was configured with $m = 1,000,000$ bins, resulting in an average estimation time of 0.30 seconds and consuming an average of 137 MB of memory. To maintain consistency with the results of the learning methods, as obtained from the cardinality estimation benchmark (Han et al., 2021), the UCI sketch estimated the cardinality of each sub-query only once. The cardinality estimates for the sub-queries are provided together with the source code⁴ to facilitate further comparisons with the presented method and to ensure reproducibility.

The results presented in Figure 3.9 highlight the superior performance of the UCI sketch, which delivers error-free estimates for approximately 70% of the sub-queries. Furthermore, the method achieves q-error values of less than 2 for about 95% of the sub-queries. In contrast, even the best-performing learning-based method, BayesCard, achieves this level

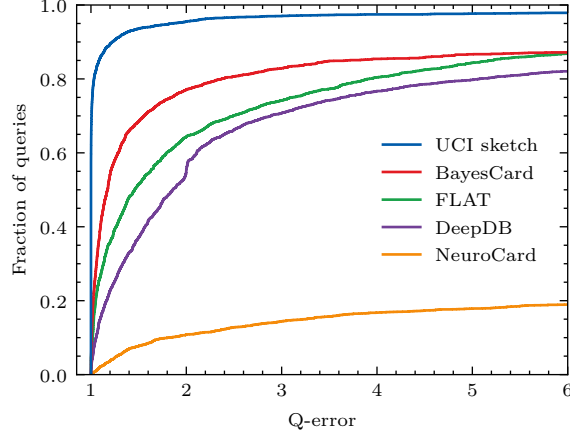


Figure 3.9. Cumulative distribution function of the q-error for the STATS-CEB and JOB-light sub-queries. The legend follows the ordering of the lines in the figure.

of accuracy for less than 80% of the sub-queries. These findings demonstrate the remarkable estimation accuracy of the UCI sketch. Moreover, the UCI sketch provides theoretical guarantees on the estimation error which are lacking for the learning-based methods.

The UCI sketch is also notably efficient when compared to the learning-based methods. As depicted in Figure 3.8, the training phase of the learning-based methods requires 3 to 5 orders of magnitude more time than creating the sketches. In addition, Table 3.5 shows that the UCI sketch can handle over 6 million updates per second. This is because the method simply adds another item to the sketch. BayesCard, on the other hand, takes 12 seconds to process an update (Han et al., 2021), and the other learning-based methods take several minutes. Consequently, these learning-based methods prove impractical for scenarios involving frequent updates.

3.4.5 PostgreSQL query execution time

This last set of experiments will evaluate the impact of the UCI sketch cardinality estimator on the query execution time of PostgreSQL. The experiments use the evaluation setup devised by Han et al. (2021), they modified PostgreSQL to enable the injection of cardinality estimates for all the sub-queries of the STATS-CEB and JOB-light queries. The UCI sketch

is compared against PostgreSQL’s own cardinality estimator as well as the aforementioned learning-based methods. Table 3.6 shows the total execution time for all the queries. The injected sub-query cardinality estimates are those reported in Section 3.4.4. In the results, PostgreSQL refers to the default PostgreSQL cardinality estimator, and True Cardinality denotes an oracle method with access to the actual intermediate sizes.

The UCI sketch achieved the lowest total execution time with an improvement of 43% compared to PostgreSQL. On STATS-CEB, the method improves over PostgreSQL by 48%, falling just short of FLAT, which showed an improvement of 55%. On JOB-light, the UCI sketch achieved equivalent execution time to the True Cardinality, while most other learning-based methods, with the exception of BayesCard, do not improve over PostgreSQL. These results underscore the practical advancement enabled by the UCI sketch due to its superior estimation accuracy, in addition to its exceptional efficiency.

Table 3.6. PostgreSQL plan execution time and percentage improvement using different cardinality estimators

Method	STATS-CEB		JOB-light		Total	
True Cardinality	1.78 h	56%	0.84 h	23%	2.62 h	49%
PostgreSQL	4.04 h	0%	1.08 h	0%	5.13 h	0%
BayesCard	2.42 h	40%	0.87 h	20%	3.29 h	36%
FLAT	1.82 h	55%	1.73 h	-60%	3.55 h	31%
DeepDB	2.24 h	45%	1.81 h	-67%	4.05 h	21%
NeuroCard	4.55 h	-13%	2.51 h	-132%	7.06 h	-38%
UCI sketch	2.09 h	48%	0.83 h	23%	2.92 h	43%

3.5 Conclusion

We have described a new sketching method, called the UCI sketch, that significantly enhances the cardinality estimation for multi-join queries. The presented approach provides fast update times, which remain constant irrespective of the required estimation accuracy. This crucial feature allows for efficient processing of high-throughput streams, all the while

delivering superior estimation accuracy compared to state-of-the-art baseline approaches. This is substantiated by the bound on the estimation error, as well as the empirical findings. The results underscore the practical suitability of the UCI sketch for applications such as query optimization and approximate query processing, surpassing the capabilities of previous methods. The UCI sketch successfully addresses the longstanding challenge of integrating the key advantages of the count sketch with the AMS-based method for multi-join queries.

Chapter 4

Approximate Tensor Network Contraction

This chapter will present two methods that generalize and improve upon the results presented in the previous chapter. Specifically, this chapter shows that the join size within a database query can equivalent be stated as a tensor network contraction. Tensor network contraction (TNC) is a fundamental mathematical operation with applications in various disciplines, including quantum physics, machine learning, database systems, graph theory, and probability theory. Analogously to how tensors generalize vectors and matrices to higher orders, TNC generalizes the dot product and matrix multiplication.

4.1 Introduction

TNC can be understood as a *tensor contraction* (see Definition 4.1) applied to the *tensor product* (see Definition 4.2) of multiple tensors. Tensor contraction is specified by a set of shared indices that form connections between the tensors, creating a *tensor network*.

The definition of TNC below uses the *Iverson bracket* notation $[S]$, which equals to *one* if the statement S is true and *zero* otherwise; the consecutive integer set $[k] \equiv \{1, \dots, k\}$ for any non-negative integer k ; and $\mathbf{i}_K$ to denote the *multi-index* $\mathbf{i} \equiv (i_1, \dots, i_q)$, which is restricted to the indices in $K \subseteq [q]$. More details on tensors and multi-index notation are provided in Section 4.2.

Definition 4.1 (Tensor contraction). *Given any order- q tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times \dots \times n_q}$ and contractions $E \subset [q] \times [q]$, such that for every $(u, v) \in E$, $u \neq v$ and $n_u = n_v$. Let $K = \{u, v : (u, v) \in E\}$ and $K' = [q] \setminus K$. The tensor contraction $\text{tc}(\mathcal{X}, E)$ is an order- $|K'|$ tensor specified by*

$$\text{tc}(\mathcal{X}, E)(\mathbf{i}_{K'}) = \sum_{\mathbf{i}_K = \mathbf{1}_K}^{\mathbf{n}_K} \mathcal{X}(\mathbf{i}) \prod_{(u,v) \in E} [i_u = i_v], \quad \forall \mathbf{1}_{K'} \leq \mathbf{i}_{K'} \leq \mathbf{n}_{K'}.$$

Definition 4.2 (Tensor product). *Given any two tensors $\mathcal{A} \in \mathbb{R}^{n_1 \times \dots \times n_p}$ and $\mathcal{B} \in \mathbb{R}^{m_1 \times \dots \times m_q}$ of orders p and q respectively, their tensor product $\mathcal{A} \times \mathcal{B}$ is specified by $(\mathcal{A} \times \mathcal{B})(\mathbf{i}, \mathbf{j}) = \mathcal{A}(\mathbf{i})\mathcal{B}(\mathbf{j})$ for all $\mathbf{1} \leq \mathbf{i} \leq \mathbf{n}$ and $\mathbf{1} \leq \mathbf{j} \leq \mathbf{m}$, where $\mathcal{A} \times \mathcal{B} \in \mathbb{R}^{n_1 \times \dots \times n_p \times m_1 \times \dots \times m_q}$.*

Definition 4.3 (Tensor network contraction (TNC)). *For any $p \in \mathbb{N}^+$, tensors $\mathcal{X}_1, \dots, \mathcal{X}_p$, and contractions E , the tensor network contraction is given by $\text{tc}(\mathcal{X}_1 \times \dots \times \mathcal{X}_p, E)$.*

The following example of a TNC is included to clarify the definitions above. The tensor network in Example 4.1 is shown in Figure 4.1 using *tensor diagram notation*, also known as *Penrose graphical notation*. In this notation, tensors are denoted by shapes with legs, where each leg represents a mode, and connected legs denote a contraction.

Example 4.1. *The TNC $\text{tc}(\mathcal{X}_1 \times \mathcal{X}_2 \times \mathcal{X}_3 \times \mathcal{X}_4, E) = \mathcal{Y}$ (shown in Figure 4.1), consists of the four tensors $\mathcal{X}_1 \in \mathbb{R}^{n_1}$, $\mathcal{X}_2 \in \mathbb{R}^{n_2 \times n_3}$, $\mathcal{X}_3 \in \mathbb{R}^{n_4 \times n_5 \times n_6}$, and $\mathcal{X}_4 \in \mathbb{R}^{n_7 \times n_8}$; and the three contractions $E = \{(1, 5), (3, 4), (6, 7)\}$. Then, from Definition 4.1, we have $K =$*

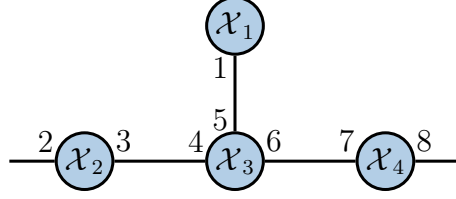


Figure 4.1. Tensor diagram of an example tensor network

$\{1, 3, 4, 5, 6, 7\}$, $K' = \{2, 8\}$, and after a reduction of the Iverson brackets, we get

$$\mathcal{Y}(i_2, i_8) = \sum_{i_1=1}^{n_1} \sum_{i_3=1}^{n_3} \sum_{i_6=1}^{n_6} \mathcal{X}_1(i_1) \mathcal{X}_2(i_2, i_3) \mathcal{X}_3(i_3, i_1, i_6) \mathcal{X}_4(i_6, i_8),$$

for all $(1, 1) \leq (i_2, i_8) \leq (n_2, n_8)$.

The general problem of TNC is known to be **NP**-hard (Bridgeman and Chubb, 2017; Arad and Landau, 2010). Markov and Shi (2008) show that exact algorithms have a running time exponential in the treewidth of the tensor network. This chapter presents a polynomial-complexity (ϵ, δ) -approximation for TNCs of acyclic tensor networks based on sketching. Additionally, it presents the first method capable of approximating arbitrary TNCs, including those of cyclic tensor networks. To our knowledge, this is the first work to directly address the general problem with sketching, and it establishes a new bound for important instances of the problem.

Sketching, or dimensionality reduction, has attracted significant attention for its ability to swiftly scale down problems while maintaining an approximation of the solution space. This technique has been applied to a number of fundamental problems in numerical linear algebra and machine learning, including least squares regression and low-rank approximation (Woodruff, 2014). Interest in these methods surged following the influential paper (Alon et al., 1996), which introduced the *AMS sketch*—a method for estimating the second frequency moment of a data stream. A notable improvement of the AMS sketch, called the *count sketch* (Charikar et al., 2002), significantly reduced the sketching time complexity.

Both sketches later gained popularity as inner product estimators (Alon et al., 1999; Cormode and Garofalakis, 2005).

In the realm of database theory, the estimation of inner products holds significant importance. It serves as a core aspect of query optimization, in particular, join size estimation. This importance has motivated the database community to further advance sketching methods. Among the notable results are two generalizations of inner product sketches to accommodate multi-join queries: first of the AMS sketch (Dobra et al., 2002), and later of the count sketch (Heddes et al., 2024). Building upon this, this chapter introduces a new perspective by showing that the problem of join size estimation, as addressed in these previous works, corresponds to the problem of approximating TNC.

Intuitively, expanding the estimation from the join size of single-joins to that of multi-joins, mirrors the broader generalization from approximating inner products to approximating TNCs. Given this perspective, the motivation for a new method arises from the observation that (1) the aforementioned join size estimators only support acyclic queries and (2) their accuracy deteriorates exponentially with an increase in the number of joins, which correspond to contractions. In fact, Section 4.4.3 shows an exponential lower bound for the technique proposed by (Heddes et al., 2024).

While the initial motivation stems from database theory, the reformulation of the underlying problem into one of TNC reveals the broader applicability of the presented techniques. Section 4.5 discusses the important role that the more general problem of approximating TNC plays beyond databases. For example, TNCs are used to simulate quantum computers (Bridgeman and Chubb, 2017); in probability theory, TNC corresponds to marginalization in graphical models (Robeva and Seigal, 2019); and counting the number of triangles in a graph can be reduced to a TNC.

4.1.1 Technical overview

This chapter presents two methods that are (ϵ, δ) -approximate tensor network contractions (ATNCs) (see Definition 4.4). The error bound of an ATNC is defined relative to the norm of the input tensor, rather than the output of the TNC. This is unavoidable as any error bound relative to the output of a TNC would imply a sublinear-space one-way protocol for the *Index* problem, contradicting its $\Omega(n)$ randomized communication lower bound (Jayram et al., 2008). A similar input-relative error bound is standard for approximate matrix multiplication (Woodruff, 2014), which is a special case of a TNC¹.

Definition 4.4 (Approximate tensor network contraction (ATNC)). *Given $\epsilon, \delta > 0$ and any positive integer p , a random function $\hat{\text{tc}}$ is an (ϵ, δ) -approximate tensor network contraction when, for any tensor network contraction $\text{tc}(\mathcal{X}, E)$ with $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_p$,*

$$\Pr(\|\hat{\text{tc}}(\mathcal{X}, E) - \text{tc}(\mathcal{X}, E)\|_{\text{F}} \leq \epsilon \|\mathcal{X}\|_{\text{F}}) \geq 1 - \delta.$$

The first method enables the approximation of *any* TNC, while prior sketching methods were only able to approximate *acyclic* TNCs (Deeds, 2025). A TNC is *cyclic* when its tensor network contains a cycle, and is *acyclic* otherwise. Key to this result is the *complement count sketch*, a circularly reversed count sketch, introduced in Section 4.4.2. Each contraction is assigned an independent count sketch and its complement so that the estimation process can use circular convolution of the sketches instead of circular cross-correlation. The use of circular cross-correlation by Heddes et al. (2024) prevents it from approximating cyclic TNCs. Section 4.4.3 then shows an exponential lower bound for the current best methods, motivating the need for a new approach.

The second method eliminates the exponential dependence on the number of contractions for acyclic TNCs. To accomplish this, the tensor network is interpreted as a tree structure

¹For matrices $A, B \in \mathbb{R}^{n \times n}$, the matrix multiplication $AB = \text{tc}(A \times B, \{(2, 3)\})$.

and acyclic TNC is formulated as a recursive function, consisting of a series of matrix multiplications with Kronecker products, in Section 4.4.4. The Kronecker products are then sketched using the recursive sketching technique from Ahle et al. (2020a). To ensure a space and time-efficient algorithm, the sketches are constructed incrementally from the leaves of the tree up to the root. These two methods yield the main result of this chapter, as stated in Theorem 4.1. The main result is stated for *full* TNCs, which are TNCs that result in a scalar value. Section 4.4.5 further demonstrates that the methods can also be applied to *partial* TNCs, which result in a tensor of nonzero order.

Theorem 4.1. *For every $p \in \mathbb{N}^+$, any order- q_k tensors $\mathcal{X}_k$ for $k \in [p]$, and every $\epsilon, \delta > 0$, there exists an (ϵ, δ) -ATNC of a full TNC that can be computed in time $O((pm \log m + qN) \log 1/\delta)$ using $O(mp \log 1/\delta)$ space, with (1) $m = \Omega(t/\epsilon^2)$ in the acyclic setting and (2) $m = \Omega(3^t/\epsilon^2)$ in general, where $N = \sum_{k=1}^p \text{nnz}(\mathcal{X}_k)$, $q = \sum_{k=1}^p q_k$, and $t = |E|$.*

Table 4.1 compares the time and space complexities of the two presented methods with the existing approaches. The complexity is parameterized with $N = \sum_{k=1}^p \text{nnz}(\mathcal{X}_k)$, $q = \sum_{k=1}^p q_k$, and $t = |E|$, which denote the number of non-zero components, modes, and contractions, respectively. Note that the existing sketching methods only support acyclic tensor networks. In this setting, the presented method brings an exponential improvement in the sketching dimension and, hence, in time and space complexity. Additionally, we present the first sketching method that can approximate general TNCs, including those with cyclic tensor networks.

Table 4.1. Complexity comparison of approximate full tensor network contraction. The AMS sketch-based method is due to Dobra et al. (2002) and the count sketch-based method is due to Heddes et al. (2024).

Method	Setting	Time	Space	Sketch size (m)
AMS	Acyclic	$O((pm + mqN) \log 1/\delta)$	$O(mp \log 1/\delta)$	$\Omega(3^t/\epsilon^2)$
Count	Acyclic	$O((pm \log m + qN) \log 1/\delta)$	$O(mp \log 1/\delta)$	$\Omega(3^t/\epsilon^2)$
<i>Ours</i>	Acyclic	$O((pm \log m + qN) \log 1/\delta)$	$O(mp \log 1/\delta)$	$\Omega(t/\epsilon^2)$
<i>Ours</i>	General	$O((pm \log m + qN) \log 1/\delta)$	$O(mp \log 1/\delta)$	$\Omega(3^t/\epsilon^2)$

4.2 Preliminaries

This section introduces the notation and concepts used throughout the chapter. We first revisit concepts from tensor algebra and their notation, based on Golub and Van Loan (2013).

An *order- q* tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \dots \times n_q}$ can be conceptualized as a q -dimensional array where the index of the k th *mode* ranges from 1 to n_k . A component $\mathcal{A}(i_1, \dots, i_q)$ of an order- q tensor is specified by q indices, with $1 \leq i_k \leq n_k$, for all k from 1 to q . With *multi-index* $\mathbf{i} = (i_1, \dots, i_q)$, a tensor component is equivalently indexed as $\mathcal{A}(\mathbf{i}) \equiv \mathcal{A}(i_1, \dots, i_q)$. The all-ones multi-index of length q is denoted by $\mathbf{1}_q$, or simply $\mathbf{1}$ when its length is clear from the context. If $\mathbf{i}$ and $\mathbf{j}$ are multi-indices of the same length, then $\mathbf{i} \leq \mathbf{j}$ means that $i_k \leq j_k$ for all k .

The squared *Frobenius norm* of $\mathcal{A}$ is given by

$$\|\mathcal{A}\|_{\text{F}}^2 = \sum_{\mathbf{i}=1}^{\mathbf{n}} \mathcal{A}(\mathbf{i})^2 \equiv \sum_{i_1=1}^{n_1} \dots \sum_{i_q=1}^{n_q} \mathcal{A}(i_1, \dots, i_q)^2,$$

the number of nonzero components of $\mathcal{A}$ is denoted by $\text{nnz}(\mathcal{A})$, and the *Hadamard product* is the component-wise multiplication between tensors.

Definition 4.5 (Hadamard product). *Given any two tensors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{n_1 \times \dots \times n_q}$, their Hadamard product $\mathcal{A} \circ \mathcal{B}$ is specified by $(\mathcal{A} \circ \mathcal{B})(\mathbf{i}) = \mathcal{A}(\mathbf{i})\mathcal{B}(\mathbf{i})$ for $\mathbf{1} \leq \mathbf{i} \leq \mathbf{n}$, where $\mathcal{A} \circ \mathcal{B} \in \mathbb{R}^{n_1 \times \dots \times n_q}$.*

Besides indexing components of a tensor, it is also possible to extract *subtensors*. A first and second-order extraction from a tensor are called a *fiber* and a *slice*, respectively. For example, if $\mathcal{A} \in \mathbb{R}^{2 \times 3 \times 4 \times 5}$ and $b = \mathcal{A}(1, :, 2, 4)$, then b is a fiber of $\mathcal{A}$ with $b(i) = \mathcal{A}(1, i, 2, 4)$. Moreover, if $B = \mathcal{A}(1, :, 2, :)$, then B is a slice of $\mathcal{A}$ with $B(i, j) = \mathcal{A}(1, i, 2, j)$. In this notation, a colon is used to indicate the extraction of a mode.

A tensor can be reshaped in various ways. The *vectorization* of a tensor $\mathcal{A} \in \mathbb{R}^{n_1 \times \dots \times n_q}$, denoted as $\text{vec}(\mathcal{A}) \in \mathbb{R}^{n_1 \dots n_q}$, is a column vector with the components of $\mathcal{A}$ in lexicographic order. The *mode- k flattening* of $\mathcal{A}$, given by $\text{mat}_k(\mathcal{A}) \in \mathbb{R}^{n_k \times n_1 \dots n_q / n_k}$, is a matrix whose columns correspond to the mode- k fibers of $\mathcal{A}$ arranged in lexicographic order. When k is omitted, as in $\text{mat}(\mathcal{A})$, it is assumed to be one.

We continue our discussion with operations specific to first and second-order tensors, better known as vectors and matrices. We start by defining the *Kronecker product* between two matrices.

Definition 4.6 (Kronecker product). *Given any two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$, their Kronecker product $A \otimes B$ is an (mp) -by- (nq) matrix with components given by $(A \otimes B)(\alpha, \beta) = A(i, j)B(k, l)$, where $\alpha = p(i - 1) + k$ and $\beta = q(j - 1) + l$.*

When the Kronecker product is applied to vectors, the vectors are assumed to be column matrices. This means that for vectors x and y , $x \otimes y = \text{vec}(x \times y)$. With $x^{\otimes p} \equiv x \otimes \dots \otimes x$ we denote the p -times Kronecker product of the vector x with itself. A sequence of Kronecker products for every member u of the set S is denoted by $\bigotimes_{u \in S} X_u$. For an empty set, we have that $\bigotimes_{u \in \emptyset} X_u = 1$. The *row-wise Kronecker product* applies the Kronecker product to the rows of two matrices.

Definition 4.7 (Row-wise Kronecker product). *Given any two matrices $A \in \mathbb{R}^{p \times n}$ and $B \in \mathbb{R}^{p \times m}$, their row-wise Kronecker product $A \bullet B$ is a p -by- (nm) matrix with components given by $(A \bullet B)(k, \alpha) = A(k, i)B(k, j)$, where $\alpha = m(i - 1) + j$.*

For vectors $x, y \in \mathbb{C}^n$, their inner product is denoted by $\langle x, y \rangle$, the component-wise complex conjugate of x is denoted by $\bar{x}$, and e_i is the i th standard basis vector. With $x * y$, we denote their *circular convolution*, and with $x \star y$ their *circular cross-correlation*, see definitions 3.1 and 3.3, respectively.

With I we denote the identity matrix, and the matrix $F_n \in \mathbb{C}^{n \times n}$, or simply F when n is clear from the context, denotes the *discrete Fourier transform* matrix. The products $F_n x$ and $F_n^{-1} x$ can be efficiently computed in time $O(n \log n)$ using the *fast Fourier transform*. The following lemma relates circular convolution and circular cross-correlation with the Hadamard product.

Lemma 4.1. *For any vectors $x, y \in \mathbb{C}^n$, $F(x * y) = Fx \circ Fy$, and $F(x \star y) = \overline{Fx} \circ Fy$.*

4.2.1 Count sketch

The *count sketch* (see Definition 4.9) was introduced to find frequent items in data streams (Charikar et al., 2002). It has since been used more broadly as a dimensionality reduction technique (Woodruff, 2014). Each column of the count sketch matrix has a single random sign in a random row. The sign is determined by a 4-wise independent hash function (see Definition 4.8) and the row is determined by a 2-wise independent hash function. The count sketch of a vector x can be efficiently computed in time $O(\text{nnz}(x))$ if x is represented as a list of its nonzero components.

Definition 4.8 (*k-wise independence*). *A family of hash functions $H = \{h : [n] \rightarrow [m]\}$ is said to be k -wise independent if for any k distinct items $x_1, \dots, x_k \in [n]$ the hashed values $h(x_1), \dots, h(x_k)$ are independent and uniformly distributed in $[m]$.*

Definition 4.9 (*Count sketch*). *The count sketch matrix is a random matrix $C \in \mathbb{R}^{m \times n}$, with each component specified as $C(j, i) = s(i)[h(i) = j]$, where $s : [n] \rightarrow \{-1, 1\}$ and $h : [n] \rightarrow [m]$ are drawn from families of 4 and 2-wise independent hash functions, respectively. Given any vector $x \in \mathbb{R}^n$, the count sketch of x is obtained by Cx .*

Lemma 4.2. *For any positive integers m, n , let $C \in \mathbb{R}^{m \times n}$ be a count sketch matrix. Then, for any vectors $x, y \in \mathbb{R}^n$,*

$$\mathbb{E}[(Cx)^\top(Cy)] = x^\top y \quad \text{and} \quad \text{Var}((Cx)^\top(Cy)) \leq \frac{2}{m} \|x\|_2^2 \|y\|_2^2.$$

The proof is given by Weinberger et al. (2009, Lemma 2).

4.2.2 Tensor sketch

The *tensor sketch* (see Definition 4.10) was a pioneer in introducing a version of the count sketch that can be used to efficiently sketch tensor products. Its insights came forth from work on compressed matrix multiplication (Pagh, 2013) applied to approximate the polynomial kernel $\langle x, y \rangle^q = \langle x^{\otimes q}, y^{\otimes q} \rangle$ for any vectors x and y , in order to accelerate kernel machines (Pham and Pagh, 2013).

Definition 4.10 (Tensor sketch). *The tensor sketch matrix $T \in \mathbb{R}^{m \times n^q}$ of order q is a random matrix defined as $T = F^{-1}((FC_1) \bullet (FC_2) \bullet \dots \bullet (FC_q))$, where $C_1, \dots, C_q \in \mathbb{R}^{m \times n}$ are independent count sketch matrices. Given any vector $x \in \mathbb{R}^{n^q}$, the tensor sketch of x is obtained by Tx .*

With tensor sketch matrix $T \in \mathbb{R}^{m \times n^q}$ and a vector $x \in \mathbb{R}^{n^q}$, the sketch Tx can be computed in time $O(q \text{nnz}(x))$. However, given a vector $y \in \mathbb{R}^n$, the sketch $T(y^{\otimes q})$ can be computed in time $O(qm \log m + q \text{nnz}(y))$ with the fast Fourier transform. This can be significantly more efficient than first computing $y^{\otimes q}$ and then applying the tensor sketch.

Lemma 4.3. *For any positive integers m, n, q , let $T \in \mathbb{R}^{m \times n^q}$ be a tensor sketch matrix of order q . Then, for any vectors $x, y \in \mathbb{R}^{n^q}$,*

$$\mathbb{E}[(Tx)^\top(Ty)] = x^\top y \quad \text{and} \quad \text{Var}((Tx)^\top(Ty)) \leq \frac{3^q}{m} \|x\|_2^2 \|y\|_2^2.$$

The proof is given by Avron et al. (2014, Lemma 2).

4.2.3 Recursive sketch

The *recursive sketch* (see Definition 4.11) improves on a limitation of the tensor sketch, whose variance has an exponential dependence on the order of the tensor. The main insight is that the exponential dependence can be avoided when the modes of a tensor are sketched recursively, following a binary tree. Multiple sketches are presented by Ahle et al. (2020a) which can be used to perform the recursion. We use a recursion of tensor sketches in our definition below. For a given sketch size, the recursive sketch only increases the time complexity of the tensor sketch by a constant factor, while offering an exponential improvement in approximation accuracy.

Definition 4.11 (Recursive sketch). *The recursive sketch matrix $R \in \mathbb{R}^{m \times n^q}$ of order q , with q a power of two², is a random matrix defined as $R = Q_2 Q_4 \cdots Q_{q/2} Q_q S$, where $S = C_1 \otimes \cdots \otimes C_q$ and $Q_l = T_{l,1} \otimes T_{l,2} \otimes \cdots \otimes T_{l,l/2}$, with independent count sketch matrices $C_1, \dots, C_q \in \mathbb{R}^{m \times n}$, and independent tensor sketch matrices $T_{l,1}, \dots, T_{l,l/2} \in \mathbb{R}^{m \times m^2}$ for $l \in \{2, 4, \dots, q/2, q\}$. Given any vector $x \in \mathbb{R}^{n^q}$, the recursive sketch of x is obtained by Rx .*

Lemma 4.4. *For any positive integers m, n, q , let $R \in \mathbb{R}^{m \times n^q}$ be a recursive sketch matrix of order q . Then, for any vectors $x, y \in \mathbb{R}^{n^q}$,*

$$\mathbb{E}[(Rx)^\top(Ry)] = x^\top y \quad \text{and} \quad \text{Var}((Rx)^\top(Ry)) \leq \left(\left(1 + \frac{8}{m}\right)^{2q} - 1 \right) \|x\|_2^2 \|y\|_2^2.$$

The proof follows from Ahle et al. (2020a, Lemmas 12 and 15).

²For notational simplicity, we allow q to be any positive integer. The actual order is implied to be the next power of two $\tilde{q}$, and the tensors in $\mathbb{R}^{n^q}$ are embedded into $\mathbb{R}^{n^{\tilde{q}}}$ through Kronecker products with e_1 , as described by Ahle et al. (2020a). Importantly, this preserves the variance bound and does not affect the asymptotic complexity.

4.3 Related work

This section reviews related works and compare them with the presented methods.

Sketch-based tensor network contraction. A generalization of the AMS sketch to enable the join size estimation of multi-join queries was proposed by Dobra et al. (2002). Although not explicitly mentioned by Dobra et al. (2002), the problem of join size estimation is equivalent to the problem of approximating a full TNC (see Section 4.5.1). Their approach involves sketching an order- q tensor using q independent AMS sketch matrices. Once all the tensors are sketched, the approximation is obtained by performing the Hadamard product of the sketches, followed by an averaging of their components. Since the AMS sketch matrix is dense, computing the m -dimensional sketch of an order- q tensor $\mathcal{X}$ requires $O(mq \text{nnz}(\mathcal{X}))$ time. The method presented in the previous chapter improves this by a factor m by using sparse count sketch matrices to sketch the tensors. Both methods only support acyclic tensor networks and have a space and time complexity that depends exponentially on the number of contractions (see Table 4.1 for a comparison with the methods presented in this chapter).

Marginalization. Robeva and Seigal (2019) established that marginalization in undirected graphical models with discrete variables is equivalent to TNC. Consequently, classical inference algorithms for graphical models can be viewed as contraction algorithms for tensor networks. This includes exact methods such as belief propagation on acyclic tensor networks and the junction tree algorithm for cyclic tensor networks (Wainwright et al., 2008). As shown by Wainwright et al. (2008) and Markov and Shi (2008), the running time of such exact algorithms is exponential in the treewidth of the tensor network. To mitigate this complexity, approximate methods such as loopy belief propagation (Wainwright et al., 2008) are often employed in practice, though they typically lack worst-case approximation guarantees.

Tensor decomposition. Tensor decomposition methods aim to factor a given tensor into a tensor network, with the specific network structure determined by the chosen decomposi-

tion (e.g., CP, Tucker, or tensor train). This differs from the TNC problem studied in this chapter, where the tensor network is given and the goal is to efficiently compute its contraction. A substantial body of work studies the use of randomized algorithms and sketching to accelerate tensor decompositions. For instance, Wang et al. (2015) proposes algorithms for CP decomposition that use sketches to speed up tensor power iterations. Malik and Becker (2018) and Ma and Solomonik (2021) presented methods for sketched alternating least squares for low-rank Tucker decomposition. More recently, Mahankali et al. (2024) presented algorithms for tensor train, Tucker, and CP decompositions, as well as decompositions into arbitrary tensor networks. While these works demonstrate the effectiveness of sketching for tensor decomposition, they do not address the problem studied in this chapter.

4.4 Methods

Two methods are presented that are (ϵ, δ) -approximate tensor network contractions. The methods are first described for full TNCs and it will later be shown that the methods are also approximate partial TNCs, which result in a tensor of nonzero order.

After establishing four simplifying assumptions (without loss of generality), we start this section by describing the first method, which can approximate any TNC. The need for a new sketching technique is then demonstrated by showing that the variance of the current best method requires an exponential dependence on the number of contractions. Motivated by this, a recursive expression for acyclic full TNC is presented that serves as a foundation for the introduction of the second method, which has a polynomial dependence on the number of contractions.

4.4.1 Without loss of generality assumptions

To simplify the explanation and notation going forward, the following four assumptions about TNC are made without loss of generality (WLOG):

WLOG 1. Every mode is part of exactly one contraction. Otherwise, *virtual copies* of the modes participating in multiple contractions can be created. Consider the following TNC of vectors a, b, c : $\text{tc}(a \times b \times c, \{(1, 2), (2, 3)\})$, where the single mode of b is part of two contractions. Defining $B(i, j) = b(i)[i = j]$, this can instead be written as $\text{tc}(a \times B \times c, \{(1, 2), (3, 4)\})$.

WLOG 2. Every contraction involves two tensors. When the modes being contracted belong to the same tensor, such as in the trace of a matrix, the exact contraction can trivially be computed in time proportional to the number of non-zero components of the tensor.

WLOG 3. At most a single contraction exists between any two tensors. Multiple contractions can be merged into one larger contraction by reshaping the tensors. The following example of matrices A and B : $\text{tc}(A \times B, \{(1, 3), (2, 4)\})$ can be expressed as $\text{tc}(\text{vec}(A) \times \text{vec}(B), \{(1, 2)\})$.

WLOG 4. Every mode has the same number of components. The methods discussed in this paper depend on the number of nonzero components of the tensors; therefore, zeros can be appended to smaller tensors to ensure that all modes have the same number of components.

It is easy to verify that these transformations do not increase the number of nonzero components or the Frobenius norm of the transformed tensors. Additionally, they do not introduce any extra tensors or contractions. These are sufficient conditions for the results to hold.

4.4.2 Approximating any tensor network contraction

The insights for the first method stem from an observation of the method by Heddes et al. (2024); namely, that the use of circular cross-correlation to combine sketches results in a pattern where each contraction has one conjugated mode when the TNC is acyclic. This pattern arises because circular cross-correlation conjugates its first argument when expressed as a Hadamard product (see Lemma 4.1), resulting in a layering of complex conjugates that cancel out when their number is even. The tensors in every other level of the contraction tree therefore have their modes conjugated, ensuring that one mode of every contraction is conjugated. However, this relies on the tensor network having a tree structure and therefore does not work for cyclic tensor networks.

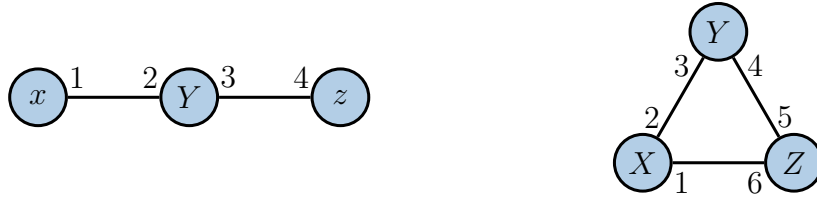


Figure 4.2. Tensor diagram of a simple acyclic (left) and cyclic (right) tensor network

The following two examples of simple acyclic and cyclic TNCs (see Figure 4.2) show that each contraction in the acyclic TNC (Example 4.2) has one conjugated mode (both are part of tensor Y), while contraction $(1, 6)$ of the cyclic TNC (Example 4.3) has no conjugated mode. We emphasize this difference because the proof for the general setting (presented in Appendix A.3.2) relies on each contraction having one conjugated mode.

Example 4.2. *With the method by Heddes et al. (2024), the acyclic TNC $\text{tc}(x \times Y \times z, \{(1, 2), (3, 4)\})$ (shown on the left of Figure 4.2) with $x, z \in \mathbb{R}^n$ and $Y \in \mathbb{R}^{n \times n}$ is approximated as follows. First, contractions $(1, 2)$ and $(3, 4)$ are assigned independent count sketch matrices C_1 and C_2 , respectively. The count sketches $C_1 x$ and $C_2 z$ and tensor sketch $F^{-1}(FC_1 \bullet FC_2) \text{vec}(Y)$ are then computed and combined using circular cross-correlation. Finally, the first component of the resulting vector is selected, yielding the following expres-*

sion:

$$\begin{aligned} & e_1^\top (C_1 x \star F^{-1}(FC_1 \bullet FC_2) \text{vec}(Y) \star C_2 z) \\ &= e_1^\top F^{-1}(FC_1 x \circ \overline{(FC_1 \bullet FC_2) \text{vec}(Y)} \circ FC_2 z). \end{aligned}$$

Example 4.3. *With the method by Heddes et al. (2024), the cyclic TNC $\text{tc}(X \times Y \times Z, \{(2, 3), (4, 5), (1, 6)\})$ (shown on the right of Figure 4.2) with $X, Y, Z \in \mathbb{R}^{n \times n}$ would be approximated as follows. First, contractions $(2, 3)$, $(4, 5)$, and $(1, 6)$ are assigned independent count sketch matrices C_1 , C_2 and C_3 , respectively. The tensor sketches $F^{-1}(FC_1 \bullet FC_2) \text{vec}(X)$, $F^{-1}(FC_2 \bullet FC_3) \text{vec}(Y)$, and $F^{-1}(FC_3 \bullet FC_1) \text{vec}(Z)$ are then computed and combined using circular cross-correlation. Finally, the first component of the resulting vector is selected, yielding the following expression:*

$$\begin{aligned} & e_1^\top (F^{-1}(FC_1 \bullet FC_2) \text{vec}(X) \star F^{-1}(FC_2 \bullet FC_3) \text{vec}(Y) \star F^{-1}(FC_3 \bullet FC_1) \text{vec}(Z)) \\ &= e_1^\top F^{-1}((FC_1 \bullet FC_2) \text{vec}(X) \circ \overline{(FC_2 \bullet FC_3) \text{vec}(Y)} \circ (FC_3 \bullet FC_1) \text{vec}(Z)). \end{aligned}$$

To describe the method for estimating cyclic TNCs, we first introduce the *complement count sketch*, which is key to the approach.

Definition 4.12 (Complement count sketch). *Given a count sketch matrix $C \in \mathbb{R}^{m \times n}$ with hash functions $s : [n] \rightarrow \{-1, 1\}$ and $h : [n] \rightarrow [m]$, the complement count sketch matrix C' of C is specified by $C'(j, i) = s(i)[-h(i) + 2 \equiv j \pmod{m}]$.*

Remark 4.1. *The complement of a count sketch matrix is itself a count sketch matrix, and the complement of a complement yields the original, that is, $(C')' = C$.*

The complement count sketch is the circularly reversed count sketch, which implies that $FC'x = \overline{FCx}$ for any $x \in \mathbb{R}^n$ as shown in Lemma 4.5 (with proof in Appendix A.3.1).

Instead of relying on circular cross-correlation to conjugate one mode of every contraction, the complement count sketch matrix can be used to explicitly control which mode of each contraction is conjugated.

Lemma 4.5. *For any positive integers m and n , let $C \in \mathbb{R}^{m \times n}$ be a count sketch matrix and C' its complement. Then, for any $x \in \mathbb{R}^n$, $FC'x = \overline{FCx}$.*

The method first samples independent count sketch matrices for one mode of each contraction and then assigns its complement count sketch matrix to the other mode. Specifically, for each $(u, v) \in E$, C_u is an independent count sketch matrix and $C_v := C'_u$. The approximation of any full TNC is then given by

$$\hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E) = e_1^\top F^{-1}(x_1 \circ \cdots \circ x_p), \quad x_k = (\bullet_{u \in V(k)} FC_u) \text{vec}(\mathcal{X}_k), \quad (4.1)$$

where $V(k)$ are the modes of $\mathcal{X}_k$ and $\bullet_{u \in S}$ denotes the row-wise Kronecker product for each member u of S . The following example shows that, with the first method, each contraction of the simple cyclic TNC used in Example 4.3 now has one conjugated mode. The proof of the expectation and variance bound of the presented estimator in Lemma 4.6 (with proof in Appendix A.3.2) relies on this property.

Example 4.4. *With the first method, the cyclic TNC $\text{tc}(X \times Y \times Z, \{(2, 3), (4, 5), (1, 6)\})$ (shown on the right of Figure 4.2) with $X, Y, Z \in \mathbb{R}^{n \times n}$ is approximated as follows. First, modes 2, 4, and 1 are assigned independent count sketch matrices C_2, C_4 , and C_1 , respectively. Then, modes 3, 5, and 6 are assigned the complement count sketch matrices C'_2, C'_4 , and C'_1 , respectively. The tensor sketches $F^{-1}(FC_1 \bullet FC_2) \text{vec}(X)$, $F^{-1}(FC'_2 \bullet FC_4) \text{vec}(Y)$, and $F^{-1}(FC'_4 \bullet FC'_1) \text{vec}(Z)$ are then computed and combined using circular convolution. Finally,*

the first component of the resulting vector is selected, yielding the following expression:

$$\begin{aligned} & e_1^\top (F^{-1}(FC_1 \bullet FC_2) \text{vec}(X) * F^{-1}(FC'_2 \bullet FC_4) \text{vec}(Y) * F^{-1}(FC'_4 \bullet FC'_1) \text{vec}(Z)) \\ &= e_1^\top F^{-1}((FC_1 \bullet FC_2) \text{vec}(X) \circ (\overline{FC_2} \bullet FC_4) \text{vec}(Y) \circ (\overline{FC_4} \bullet \overline{FC_1}) \text{vec}(Z)). \end{aligned}$$

We now show that the first method is an unbiased estimator for arbitrary TNCs, and bound its variance through a reduction to the tensor sketch in Lemma 4.6 (with proof in Appendix A.3.2). While the first method has the same variance bound and asymptotic complexity as the method by Heddes et al. (2024), to the best of our knowledge, the method is the first sketching method that enables the estimation of cyclic TNCs.

Lemma 4.6. *For every $m, p \in \mathbb{N}^+$ and any full TNC, let $\hat{\text{tc}}$ be defined as in Eq. (4.1) with sketch size m . Then, with $y = \hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E)$ and $t = |E|$,*

$$\mathbb{E}[y] = \text{tc}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E), \quad \text{and} \quad \text{Var}(y) \leq \frac{3^t}{m} \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2.$$

The pseudocode for the first method is provided in Algorithm 4. The estimator supports the most general streaming data setting, the general turnstile, in which a stream of updates can change any component by any amount. This is realized by storing the sketches x_k for all $k \in [p]$ (see Algorithm 4 line 8), with each update being applied in time $O(q_k)$.

4.4.3 Exponential lower bound

The best known bounds for approximate full TNC using sketching methods, including those introduced by Dobra et al. (2002), Heddes et al. (2024), and the first method described in Section 4.4.2, all have an exponential dependence on the number of contractions, as shown in Table 4.1. To motivate the need for a new approach, in Lemma 4.7 (with proof in

Algorithm 4 Approximating general full tensor network contractions

input: tensors $\mathcal{X}_k \in \mathbb{R}^{n \times \dots \times n}$ of degree q_k for $k \in [p]$, contractions E , sketch size m

output: approximate tensor network contraction $y \in \mathbb{R}$

```

1: for  $(u, v) \in E$  do
2:   Let  $C_u \in \mathbb{R}^{m \times n}$  be an independent count sketch matrix.
3:    $C_v \leftarrow C'_u$ 
4: end for
5:  $z \in \{1\}^m$ 
6: for  $k \in [p]$  do
7:    $T_k \leftarrow F^{-1}(\bullet_{u \in V(k)} F C_u) \triangleright$  Compose a tensor sketch matrix of count sketch matrices
8:    $x_k \leftarrow T_k \text{vec}(\mathcal{X}_k)$ 
9:    $z \leftarrow z \circ F x_k$ 
10: end for
11:  $y \leftarrow \frac{1}{m} \sum_{i=1}^m z_i$ 

```

Appendix A.3.3) we claim that the method by Heddes et al. (2024) requires an exponential dependence on the number of contractions. In particular, we show a lower bound on the variance for approximating a sequence of matrix multiplications based on (Ahle et al., 2020a, Lemma 43), which provides a variance lower bound for norm approximation using the tensor sketch.

Lemma 4.7. *For every positive integers n, m, q , where q is even, let $C_1, \dots, C_q \in \mathbb{R}^{m \times n}$ be independent count sketch matrices, and $T_i = F^{-1}((F C_{i-1}) \bullet (F C_i))$. Then, for the all-ones tensors $\mathcal{X}_1, \mathcal{X}_{q+1} \in \{1\}^n$ and $\mathcal{X}_i \in \{1\}^{n \times n}$ for $2 \leq i \leq q$,*

$$\text{Var}(e_1^\top (C_1 \mathcal{X}_1 \star T_2 \text{vec}(\mathcal{X}_2) \star \dots \star T_q \text{vec}(\mathcal{X}_q) \star C_q \mathcal{X}_{q+1})) \geq \left(\frac{3^q}{2m^2} - 1 \right) \prod_{i=1}^{q+1} \|\mathcal{X}_i\|_{\text{F}}^2.$$

This result, which extends to the first method, demonstrates that the exponential dependence of the prior best methods cannot be overcome merely through a tighter analysis. We thereby conclude that a new method is necessary for achieving an improved bound for the problem of ATNC. In the remainder of this section, we introduce the second approximation method and show how this new approach eliminates the exponential dependence on the number of contractions.

4.4.4 Improved approximation for acyclic tensor networks

For the second method, besides assuming a full TNC, we assume that the tensor network is acyclic. Acyclic tensor networks are common in several important settings. In particular, many relational database queries have an acyclic join graph, and estimating the join size of these queries (which corresponds to a TNC) is critical for efficient database systems (see Section 4.5.1). Although tractable algorithms exist for TNC with acyclic tensor networks, the intermediate tensors may still be prohibitively large, motivating the need for approximate TNC.

To facilitate the explanation of the proposed solution, we first introduce some additional notation and concepts. Specifically, we introduce the notion of a root tensor $\mathcal{X}_o$, with $o \in [p]$, which is arbitrarily selected from the contracted tensors to form a rooted tree structure from the tensor network. We then define $\Gamma(k)$ as the outgoing neighbors³ of tensor $\mathcal{X}_k$, that is, the set of tensors that share a contraction with $\mathcal{X}_k$ and farther from the root tensor. Note that all the leaves of the rooted tree are vectors. With this, we are able to write acyclic full TNC recursively as shown in Lemma 4.8 (with proof in Appendix A.3.4).

Lemma 4.8. *Given any $o \in [p]$, any acyclic full TNC can be expressed recursively as*

$$\text{tc}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E) = \text{vec}(\mathcal{X}_o)^\top g(o), \quad g(k) = \bigotimes_{l \in \Gamma(k)} \text{mat}(\mathcal{X}_l)g(l).$$

The second approach for approximate TNC builds upon the following insight: the recursive formulation of TNC consists of a sequence of matrix multiplications involving Kronecker products. Leveraging this observation, we sketch the Kronecker products using recursive

³For notational simplicity, the neighbor function Γ is assumed to yield neighbors in an order matching the modes of the corresponding tensor, and the first mode of a non-root tensor is assumed to contract with its parent in the tree. This arrangement can always be achieved by simply permuting the indices of the tensor.

sketch matrices. This leads to the following approximation of TNCs:

$$\hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E) = \text{vec}(\mathcal{X}_o)^\top R_o^\top R_o \hat{g}(o), \quad \hat{g}(k) = \bigotimes_{l \in \Gamma(k)} \text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l), \quad (4.2)$$

where each R_k maps a vectorized order- $(q_k - 1)$ tensor to a vector of size m , with the exception of R_o , which maps a vectorized order- q_o tensor to a vector of size m .

In Lemma 4.9, we show that the second method is an unbiased estimator of acyclic full TNC, and we bound its variance. The proof, provided in Appendix A.3.5, bounds the variance by accumulating $(1 + 8/m)^p$ terms from the recursive sketch variance bound in Lemma 4.4 while traversing the rooted tree. The expectation result follows from the expectation of the recursive sketch.

Lemma 4.9. *For every $m, p \in \mathbb{N}^+$ and any acyclic full TNC, let $\hat{\text{tc}}$ be defined as in Eq. (4.2) with sketch size m . Then, with $y = \hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E)$ and $t = |E|$,*

$$\mathbb{E}[y] = \text{tc}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E) \quad \text{and} \quad \text{Var}(y) \leq \left(\left(1 + \frac{8}{m} \right)^{2t} - 1 \right) \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2.$$

Computing an estimate efficiently is not straightforward, particularly since each $\text{mat}(\mathcal{X}_l) R_l^\top$ results in a (potentially large) n -by- m matrix. To address this, we decompose the recursive sketch matrix from the previous recursion step and integrate its initial dimensionality reduction directly into the following step of the recursion. Since $R_k = Q_k S_k$, where $S_k = \bigotimes_{l \in \Gamma(k)} C_{k,l}$ is a Kronecker product of independent count sketch matrices, we can use

the mixed-product identity $(A \otimes B)(x \otimes y) = (Ax) \otimes (By)$ to obtain the following factorization:

$$\begin{aligned} & \text{vec}(\mathcal{X}_o)^\top R_o^\top Q_o S_o \left(\bigotimes_{l \in \Gamma(o)} \text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l) \right) \\ &= \text{vec}(\mathcal{X}_o)^\top R_o^\top Q_o \left(\bigotimes_{l \in \Gamma(o)} C_{o,l} \text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l) \right), \end{aligned}$$

which can be applied to every recursion step. This yields $C_{k,l} \text{mat}(\mathcal{X}_l) R_l^\top$, which are manageable m -by- m matrices that can be efficiently computed in time $O(q_l \text{nnz}(\mathcal{X}_l))$. As a result, we have transformed a problem involving tensors of any order into a significantly more tractable problem that deals only with second-order tensors.

To eliminate the need for storing intermediate matrices altogether, we propose a progressive computation scheme that traverses the rooted tree of the tensor network from its leaves to the root. This approach efficiently computes the matrix-vector product during the creation of each subsequent sketch, resulting in compact sketch vectors at every recursion step.

Specifically, at the leaves we compute the count sketches of first-order tensors as $x_l = C_{k,l} \text{vec}(\mathcal{X}_l)$, which takes $O(\text{nnz}(\mathcal{X}_l))$ time. Using these leaf-level sketch vectors, we then proceed one level up the tree. At this level, we first compute $r_k = Q_k(\bigotimes_{l \in \Gamma(k)} x_l)$, which represents a recursive sketch of the leaf-level sketch vectors. We then compute $x_k = C_{l,k} \text{mat}(\mathcal{X}_k) R_k^\top r_k$, combining the computation of the sketch matrix $C_{l,k} \text{mat}(\mathcal{X}_k) R_k^\top$ and the matrix-vector product involving r_k . When carried out jointly, these operations require $O(q_k \text{nnz}(\mathcal{X}_k))$ time, improving upon the $O(q_k \text{nnz}(\mathcal{X}_k) + m^2)$ time needed if performed separately. The pseudocode for this procedure (called the `SKETCHEDMATRIXVECTORPRODUCT`) is provided in Appendix B, Algorithm 8, and the pseudocode for the second method is provided in Algorithm 5. Example 4.5 illustrates the second method applied to the acyclic tensor network shown in Figure 4.3.

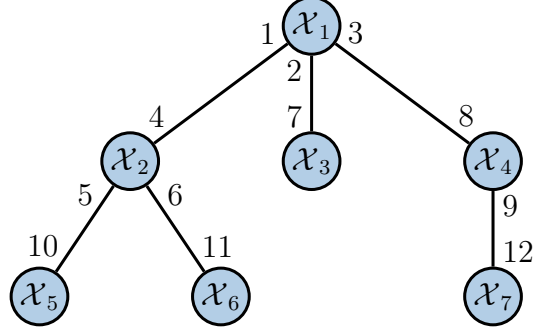


Figure 4.3. Tensor diagram of an example acyclic tensor network

Example 4.5. Using the second method, the TNC shown in Figure 4.3,

$$\text{tc}(\mathcal{X}_1 \times \mathcal{X}_2 \times \mathcal{X}_3 \times \mathcal{X}_4 \times \mathcal{X}_5 \times \mathcal{X}_6 \times \mathcal{X}_7, \{(1, 4), (2, 7), (3, 8), (5, 10), (6, 11), (9, 12)\}),$$

with $\mathcal{X}_1, \mathcal{X}_2 \in \mathbb{R}^{n \times n \times n}$, $\mathcal{X}_4 \in \mathbb{R}^{n \times n}$, and $\mathcal{X}_3, \mathcal{X}_5, \mathcal{X}_6, \mathcal{X}_7 \in \mathbb{R}^n$ is approximated as follows. First, let $\mathcal{X}_1$ be selected as the arbitrary root tensor, which allows us to interpret the tensor network as a rooted tree. Then, each tensor is assigned an independent recursive sketch matrix whose order matches the number of children the tensor has in the rooted tree. Thus, we have $R_1 \in \mathbb{R}^{m \times n^3}$, $R_2 \in \mathbb{R}^{m \times n^2}$, and $R_4 \in \mathbb{R}^{m \times n}$. Recall that each recursive sketch matrix can be decomposed as $R_k = Q_k(\bigotimes_{l \in \Gamma(k)} C_{k,l})$, where $C_{k,l}$ are independent count sketch matrices. Now, starting at the lowest level of the tree, we compute the count sketches $x_5 = C_{2,5} \text{vec}(\mathcal{X}_5)$, $x_6 = C_{2,6} \text{vec}(\mathcal{X}_6)$, $x_7 = C_{4,7} \text{vec}(\mathcal{X}_7)$. We then move one level up the tree, first computing the recursive sketches $r_2 = Q_2(x_5 \otimes x_6)$ and $r_4 = Q_4 x_7$, and then $x_2 = C_{1,2} \text{mat}(\mathcal{X}_2) R_2^\top r_2$, $x_3 = C_{1,3} \text{vec}(\mathcal{X}_3)$, and $x_4 = C_{1,4} \text{mat}(\mathcal{X}_4) R_4^\top r_4$ using Algorithm 8. Lastly, at the root of the tree, we compute $x_1 = R_1 \text{vec}(\mathcal{X}_1)$, $r_1 = Q_1(x_2 \otimes x_3 \otimes x_4)$, and $y = x_1^\top r_1$.

4.4.5 Main result

In this section, we combine the results from the previous sections to obtain the main result stated in Theorem 4.1, which shows that the first method is an (ϵ, δ) -ATNC for any full TNC and that the second method is an (ϵ, δ) -ATNC for acyclic full TNCs. The proof

Algorithm 5 Approximating acyclic full tensor network contraction

input: tensors $\mathcal{X}_k \in \mathbb{R}^{n \times \dots \times n}$ of degree q_k for $k \in [p]$, contractions E , sketch size m

output: approximate tensor network contraction $y \in \mathbb{R}$

```

1: Let  $o$  be any member of  $[p]$ .
2: Let  $R_o \in \mathbb{R}^{m \times n^{q_o}}$  be an independent recursive sketch.
3: for  $k \in [p] \setminus \{o\}$  do
4:   Let  $R_k = Q_k(\bigotimes_{l \in \Gamma(k)} C_{k,l}) \in \mathbb{R}^{m \times n^{q_k-1}}$  be an independent recursive sketch.
5: end for
6: function NEIGHBORSKETCH( $k$ )
7:   if  $|\Gamma(k)| = 0$  then return 1
8:   for  $l \in \Gamma(k)$  do
9:      $r_l \leftarrow \text{NEIGHBORSKETCH}(l)$ 
10:     $x_l \leftarrow \text{SKETCHEDMATRIXVECTORPRODUCT}(C_{k,l}, \mathcal{X}_l, R_l, r_l) \triangleright C_{k,l} \text{mat}(\mathcal{X}_l) R_l^\top r_l$ 
11:   end for
12:   return  $Q_k(\bigotimes_{l \in \Gamma(k)} x_l) \triangleright$  Apply remainder of the recursive sketch
13: end function
14:  $x_o \leftarrow R_o \text{vec}(\mathcal{X}_o)$ 
15:  $r_o \leftarrow \text{NEIGHBORSKETCH}(o)$ 
16:  $y \leftarrow x_o^\top r_o$ 

```

(presented in Appendix A.3.6) uses Chebyshev's inequality with the expected value and the variance upper bound from Lemmas 4.6 and 4.9 to obtain estimators with constant success probability. By selecting the median of $O(\log 1/\delta)$ independent estimates, each with a constant success probability, we can use the Chernoff bound to obtain an estimator that succeeds with probability $1 - \delta$. This approach is sometimes referred to as the *median trick*. It is commonly applied to boost an estimator with constant success probability to success with high probability.

Theorem 4.1. *For every $p \in \mathbb{N}^+$, any order- q_k tensors $\mathcal{X}_k$ for $k \in [p]$, and every $\epsilon, \delta > 0$, there exists an (ϵ, δ) -ATNC of a full TNC that can be computed in time $O((pm \log m + qN) \log 1/\delta)$ using $O(mp \log 1/\delta)$ space, with (1) $m = \Omega(t/\epsilon^2)$ in the acyclic setting and (2) $m = \Omega(3^t/\epsilon^2)$ in general, where $N = \sum_{k=1}^p \text{nnz}(\mathcal{X}_k)$, $q = \sum_{k=1}^p q_k$, and $t = |E|$.*

Lastly, in Corollary 4.1 (with proof in Appendix A.3.7), we show that the two methods can also be applied to partial TNC, that is, those that evaluate to tensors of nonzero order. The

time complexity of this generalization grows linearly with the number of components in the resulting tensor. Interestingly, the computational complexity of the methods only depends on the sizes of the input and output tensors.

Corollary 4.1. *For every $p, n \in \mathbb{N}^+$, any order- q_k tensors $\mathcal{X}_k \in \mathbb{R}^{n \times \dots \times n}$ for $k \in [p]$, and every $\epsilon, \delta > 0$, there exists an (ϵ, δ) -ATNC of a partial TNC which results in an order- $\bar{q}$ tensor that can be computed in time $O((n^{\bar{q}}pm \log m + qN) \log 1/\delta)$ using $O(n^{\bar{q}}mp \log 1/\delta)$ space, with **(1)** $m = \Omega(t/\epsilon^2)$ in the acyclic setting and **(2)** $m = \Omega(3^t/\epsilon^2)$ in general, where $N = \sum_{k=1}^p \text{nnz}(\mathcal{X}_k)$, $q = \sum_{k=1}^p q_k$, and $t = |E|$.*

We have now established that the first method is an ATNC in general and that the second method is an ATNC in the acyclic setting. In the remainder of this manuscript, we highlight applications of TNC for which approximate solutions are permissible.

4.5 Applications of tensor network contraction

TNC, by virtue of its generality, is important in many disciplines. From the mathematical modeling of physical phenomena to addressing both theoretical and practical issues in computer science. In quantum mechanics, tensors model many-body quantum wave functions, and approximate TNCs are used to simulate quantum computers (Bridgeman and Chubb, 2017). In machine learning, the contraction of efficient tensor representations seeks to mitigate the computational cost of training large models (Oseledets, 2011; Novikov et al., 2015). Meanwhile, in probability theory, the joint probability distribution of discrete random variables is naturally represented by a tensor, and marginalization in graphical models has been shown to be equivalent to TNC (Robeva and Seigal, 2019). In graph theory, the number of q -colorings of a graph can be reduced to TNC (Bridgeman and Chubb, 2017). In the remainder of this section, we highlight two applications of TNC: join size estimation in database systems and triangle counting in graphs.

4.5.1 Join size estimation

One particularly relevant application of approximate TNC lies within database systems, namely join size estimation. In this setting, the join size is the number of tuples that result from a join. In general, the execution time of a query is highly sensitive to the join execution order (Leis et al., 2015). Since determining the optimal join order is known to be **NP**-hard (Ibaraki and Kameda, 1984), query optimizers often rely on intermediate join size estimates⁴ as their primary input to assess the cost of a particular join order.



Figure 4.4. Example SQL query (left) and corresponding tensor network diagram (right). For all $k \in [4]$, every component $\mathcal{X}_k(\mathbf{i})$ denotes the frequency of $\mathbf{i}$ in relation R_k .

The join size of a query containing equi-joins can be expressed as a TNC, in which each relation is represented by a tensor and the joins specify the contractions (Alon et al., 1999; Dobra et al., 2002; Heddes et al., 2024). Join size estimation can therefore be formulated as an ATNC problem. Consequently, the presented results for ATNC yield improved bounds for join size estimation in the case of acyclic queries, and generalize previous results to cyclic queries as stated in Corollary 4.2. An example SQL query and its corresponding tensor network diagram are provided in Figure 4.4.

Corollary 4.2. *For every $p \in \mathbb{N}^+$, any query of equi-joins of relations R_k for $k \in [p]$, and every $\epsilon, \delta > 0$, there exists an (ϵ, δ) -approximation of the join size (as in Definition 4.4) that can be computed in time $O((pm \log m + qN) \log 1/\delta)$ using $O(mp \log 1/\delta)$ space, with (1) $m = \Omega(t/\epsilon^2)$ when the query is acyclic and (2) $m = \Omega(3^t/\epsilon^2)$ in general, where $N = \sum_{k=1}^p |R_k|$, $q = \sum_{k=1}^p q_k$, and $t = |E|$.*

⁴The intermediate query result can itself be expressed as a query (a subquery of the query to optimize).

Proof. Let $\mathcal{X}_k$ be the frequency tensor of relation R_k , such that for all $k \in [p]$, every component $\mathcal{X}_k(\mathbf{i})$ denotes the frequency of $\mathbf{i}$ in relation R_k , that is, $\mathcal{X}_k(\mathbf{i}) = \sum_{\mathbf{j} \in R_k} [\mathbf{i} = \mathbf{j}]$; and let E be the set of equi-joins. Since the sketches are linear, the sketch of each frequency tensor can be computed in one pass over the relation. The results then follow directly from Theorem 4.1. $\square$

4.5.2 Triangle counting

Even for combinatorial problems for which tractable algorithms are known, ATNC can prove to be useful. Take, for instance, the task of counting triangles in graphs. Given the emergence of applications involving massive graphs, even polynomial-time algorithms can become prohibitively expensive. Since the triangle count can be formulated as $\text{tr}(AAA)$, with adjacency matrix A , it can also be approximated using ATNC. Efficient sketching-based algorithms for counting triangles have previously been explored, for example, in (Jowhari and Ghodsi, 2005). In fact, the AMS sketch has inspired several such efficient algorithms for graph problems (Henzinger et al., 1998; Feigenbaum et al., 2005a, 2004, 2005b). The result for triangle counting presented in Corollary 4.3 matches the space bound of the second estimator in (Jowhari and Ghodsi, 2005), but improves upon their time complexity of $O(mN \log 1/\delta)$. Moreover, the second estimator in (Jowhari and Ghodsi, 2005) needs 12-wise independent hash functions, whereas the presented method relies on at most 4-wise independent hash functions, which are significantly more efficient in practice.

Corollary 4.3. *For any directed graph G with n nodes, N edges, adjacency matrix A , and any $\epsilon, \delta > 0$, the number of triangles in G can be computed with at most ϵ absolute error in a single pass over the edges using Algorithm 4 in time $O((m \log m + N) \log 1/\delta)$ using $O(m \log 1/\delta)$ space with $m = \Omega(N^3/\epsilon^2)$ with probability at least $1 - \delta$.*

Proof. Let $\mathcal{X}_1, \mathcal{X}_2, \mathcal{X}_3 = A$ and $E = \{(1, 6), (2, 3), (4, 5)\}$. Then the result follows directly

from Theorem 4.1. Note that there is an additional N^3 term in the bound on m , in order to remove the relative term $\|\mathcal{X}\|_{\text{F}} = \|A\|_{\text{F}}^3 = N^{3/2}$ from Definition 4.4. $\square$

Bibliography

- Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003. ISSN 0022-0000. doi: [https://doi.org/10.1016/S0022-0000\(03\)00025-4](https://doi.org/10.1016/S0022-0000(03)00025-4). URL <https://www.sciencedirect.com/science/article/pii/S0022000003000254>. Special Issue on PODS 2001.
- Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3): 211–230, 2003.
- Charu C Aggarwal and Philip S Yu. A survey of synopsis construction in data streams. *Data streams: models and algorithms*, pages 169–207, 2007.
- Thomas D Ahle, Michael Kapralov, Jakob BT Knudsen, Rasmus Pagh, Ameya Velingker, David P Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 141–160. SIAM, 2020a. See also the extended arXiv preprint arXiv:1909.01410.
- Thomas Dybdahl Ahle, Jakob Tejs Bæk Knudsen, and Mikkel Thorup. The power of hashing with mersenne primes. *arXiv preprint arXiv:2008.08654*, 2020b.
- Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '06, page 557–563, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595931341. doi: 10.1145/1132516.1132597. URL <https://doi.org/10.1145/1132516.1132597>.
- Abdulrazzak Ali. Duplicates detection, 2022. URL <https://doi.org/10.7910/DVN/07LNDH>.
- Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM Symposium on Theory of computing (STOC)*, pages 20–29, 1996.
- Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 10–20, 1999.
- Itai Arad and Zeph Landau. Quantum computation and the evaluation of tensor networks. *SIAM Journal on Computing*, 39(7):3089–3121, 2010.

- Haim Avron, Huy Nguyen, and David Woodruff. Subspace embeddings for the polynomial kernel. *Advances in Neural Information Processing Systems (NeurIPS)*, 27, 2014.
- Shivnath Babu and Jennifer Widom. Continuous queries over data streams. *ACM SIGMOD Record*, 30(3):109–120, 2001.
- Ziv Bar-Yossef, TS Jayram, Ravi Kumar, D Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, pages 1–10, 2002.
- Albert-László Barabási. Network science. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1987):20120375, 2013.
- Alain Biem, Eric Bouillet, Hanhua Feng, Anand Ranganathan, Anton Riabov, Olivier Verscheure, et al. Ibm infosphere streams for scalable, real-time, intelligent transportation services. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1093–1104, 2010.
- Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
- Jacob C Bridgeman and Christopher T Chubb. Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of physics A: Mathematical and theoretical*, 50(22):223001, 2017.
- Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004.
- Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.
- Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.
- Michael Brusco, J Dennis Cradit, and Douglas Steinley. A comparison of 71 binary similarity coefficients: The effect of base rates. *Plos One*, 16(4):e0247751, 2021.
- Walter Cai, Magdalena Balazinska, and Dan Suciu. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *Proceedings of the 2019 International Conference on Management of Data*, pages 18–35, 2019.
- Carlo Vittorio Cannistraci, Gregorio Alanis-Lobato, and Timothy Ravasi. From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific Reports*, 3(1):1–14, 2013.
- Lei Cao, Qingyang Wang, and Elke A Rundensteiner. Interactive outlier exploration in big data streams. *Proceedings of the VLDB Endowment*, 7(13):1621–1624, 2014.
- Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data

- streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.
- Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thirty-fourth annual ACM Symposium on Theory of computing (STOC)*, pages 380–388, 2002.
- Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43, 1998.
- Chen Chen, Hongzhi Yin, Junjie Yao, and Bin Cui. Terec: A temporal recommender system over tweet stream. *Proceedings of the VLDB Endowment*, 6(12):1254–1257, 2013.
- Zhida Chen, Gao Cong, and Walid G Aref. Star: A distributed stream warehouse system for spatial data. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2761–2764, 2020.
- Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.
- Tobias Christiani. Dartminhash: Fast sketching for weighted sets. *arXiv preprint arXiv:2005.11547*, 2020.
- Ondřej Chum and Jiří Matas. Fast computation of min-hash signatures for image collections. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3077–3084. IEEE, 2012.
- Ondrej Chum, James Philbin, Andrew Zisserman, et al. Near duplicate image detection: Min-hash and tf-idf weighting. In *The British Machine Vision Conference (BMVC)*, volume 810, pages 812–815, 2008.
- Graham Cormode. Current trends in data summaries. *ACM SIGMOD Record*, 50(4):6–15, 2022.
- Graham Cormode. Applications of sketching and pathways to impact. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS '23*, page 5–10, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701276. doi: 10.1145/3584372.3589937. URL <https://doi.org/10.1145/3584372.3589937>.
- Graham Cormode and Minos Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st international conference on Very large Data Bases (VLDB)*, pages 13–24, 2005.
- Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.
- Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava.

- Finding hierarchical heavy hitters in data streams. In *Proceedings 2003 VLDB Conference*, pages 464–475. Elsevier, 2003.
- Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 12 2011. ISSN 1931-7883. doi: 10.1561/19000000004. URL <https://doi.org/10.1561/19000000004>.
- Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 271–280, 2007.
- Nur Nasuha Daud, Siti Hafizah Ab Hamid, Muntadher Saadoon, Firdaus Sahran, and Nor Badrul Anuar. Applications of link prediction in social networks: A review. *Journal of Network and Computer Applications*, 166:102716, 2020.
- Kyle Deeds. *Data-Aware Complexity Analysis and Program Optimization*. PhD thesis, University of Washington, 2025.
- Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 61–72, 2002.
- Yuxiao Dong, Qing Ke, Bai Wang, and Bin Wu. Link prediction based on local information. In *2011 International Conference on Advances in Social Networks Analysis and Mining*, pages 382–386. IEEE, 2011.
- Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. Selectivity estimation for range predicates using lightweight models. *Proceedings of the VLDB Endowment*, 12(9):1044–1057, 2019.
- Otmar Ertl. Bagminhash-minwise hashing algorithm for weighted sets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1368–1377, 2018.
- Jianqing Fan, Fang Han, and Han Liu. Challenges of big data analysis. *National Science Review*, 1(2):293–314, 2014.
- Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 531–543. Springer, 2004.
- Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3): 207–216, 2005a.
- Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. Graph distances in the streaming model: the value of space. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 5, pages 745–754, 2005b.

- Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985. ISSN 0022-0000. doi: [https://doi.org/10.1016/0022-0000\(85\)90041-8](https://doi.org/10.1016/0022-0000(85)90041-8). URL <https://www.sciencedirect.com/science/article/pii/0022000085900418>.
- Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete Mathematics & Theoretical Computer Science*, DMTCS Proceedings vol. AH, 2007 Conference on Analysis of Algorithms (AofA 07):10, Jan 2007. ISSN 1365-8050. doi: 10.46298/dmtcs.3545. URL <https://dmtcs.episciences.org/3545>.
- Sam Fletcher, Md Zahidul Islam, et al. Comparing sets of patterns with the jaccard index. *Australasian Journal of Information Systems*, 22, 2018.
- Valerio Freschi. A graph-based semi-supervised algorithm for protein function prediction from interaction maps. In *International Conference on Learning and Intelligent Optimization (LION)*, pages 249–258. Springer, 2009.
- Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. Tracking set-expression cardinalities over continuous update streams. *The VLDB Journal*, 13(4):354–369, 2004.
- Ross W Gayler. Vector symbolic architectures answer jackendoff’s challenges for cognitive neuroscience. In *International Conference on Cognitive Science (ICCS)*, pages 133–138, 2003.
- Nikos Giatrakos, Alexander Artikis, Antonios Deligiannakis, and Minos Garofalakis. Complex event recognition in the big data era. *Proceedings of the VLDB Endowment*, 10(12):1996–1999, 2017.
- Phillip B Gibbons and Yossi Matias. Synopsis data structures for massive data sets. *External Memory Algorithms*, 50:39–70, 1999.
- Anna C Gilbert, Yannis Kotidis, Shanmugavelayutham Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB*, volume 1, pages 79–88, 2001.
- Grove Karl Gilbert. Finley’s tornado predictions. *American Meteorological Journal. A Monthly Review of Meteorology and Allied Branches of Study (1884-1896)*, 1(5):166, 1884.
- Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.
- Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1093–1103, 2012.
- Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record*, 30(2):58–66, 2001.
- Emre Guney. Reproducible drug repurposing: When similarity does not suffice. volume 22, pages 132–143, 02 2017. doi: 10.1142/9789813207813_0014.

- Bikash Gyawali, Lucas Anastasiou, and Petr Knoth. Deduplication of scholarly documents using locality sensitive hashing and word embeddings. In *Proceedings of 12th Language Resources and Evaluation Conference*, pages 894–903. France European Language Resources Association, May 2020.
- Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, et al. Cardinality estimation in dbms: a comprehensive benchmark evaluation. *Proceedings of the VLDB Endowment*, 15(4):752–765, 2021.
- Reihaneh H Hariri, Erik M Fredericks, and Kate M Bowers. Uncertainty in big data analytics: survey, opportunities, and challenges. *Journal of Big Data*, 6(1):1–16, 2019.
- Mohammad Al Hasan and Mohammed J Zaki. A survey of link prediction in social networks. In *Social Network Data Analytics*, pages 243–275. Springer, 2011.
- Shohedul Hasan, Saravanan Thirumuruganathan, Jeess Augustine, Nick Koudas, and Gautam Das. Deep learning models for selectivity estimation of multi-attribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1035–1050, 2020.
- Mike Heddes, Igor Nunes, Pere Vergés, Denis Kleyko, Danny Abraham, Tony Givargis, Alexandru Nicolau, and Alexander Veidenbaum. Torchhd: An open source python library to support research on hyperdimensional computing and vector symbolic architectures. *Journal of Machine Learning Research*, 24(255):1–10, 2023. URL <http://jmlr.org/papers/v24/23-0300.html>.
- Mike Heddes, Igor Nunes, Tony Givargis, and Alex Nicolau. Convolution and cross-correlation of count sketches enables fast cardinality estimation of multi-join queries. *Proceedings of the ACM on Management of Data*, 2(3), May 2024. doi: 10.1145/3654932. URL <https://doi.org/10.1145/3654932>.
- Monika Henzinger, Bay-Wei Chang, Brian Milch, and Sergey Brin. Query-free news search. In *Proceedings of the 12th International Conference on World Wide Web (WWW)*, pages 1–10, 2003.
- Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, 50:107–118, 1998.
- Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. Deepdb: learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13(7):992–1005, 2020.
- Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:22118–22133, 2020.
- Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. Tencentrec: Real-time

- stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 227–238, 2015.
- Toshihide Ibaraki and Tiko Kameda. On the optimal nesting order for computing n-relational joins. *ACM Transactions on Database Systems (TODS)*, 9(3):482–502, 1984.
- Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998.
- Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing (STOC)*, pages 618–625, 1997.
- Sergey Ioffe. Improved consistent sampling, weighted minhash and l1 sketching. In *2010 IEEE International Conference on Data Mining (ICDM)*, pages 246–255. IEEE, 2010.
- Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. Compass: Online sketch-based query optimization for in-memory databases. In *Proceedings of the 2021 International Conference on Management of Data*, pages 804–816, 2021.
- Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New Phytologist*, 11(2): 37–50, 1912.
- T. S. Jayram, Ravi Kumar, and D. Sivakumar. The one-way communication complexity of hamming distance. *Theory of Computing*, 4(6):129–135, 2008. doi: 10.4086/toc.2008.v004a006. URL <https://theoryofcomputing.org/articles/v004a006>.
- William B Johnson, Joram Lindenstrauss, et al. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.
- Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 1972.
- Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics: 11th Annual International Conference, COCOON 2005 Kunming, China, August 16–19, 2005 Proceedings 11*, pages 710–716. Springer, 2005.
- Daniel M. Kane and Jelani Nelson. Sparser johnson-lindenstrauss transforms. *J. ACM*, 61(1), January 2014. ISSN 0004-5411. doi: 10.1145/2559902. URL <https://doi.org/10.1145/2559902>.
- 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*, pages 41–52, 2010.
- Pentti Kanerva. *Sparse distributed memory*. MIT Press, 1988.
- Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2): 139–159, 2009.

- Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *2016 IEEE 57th annual symposium on foundations of computer science (FOCS)*, pages 71–78. IEEE, 2016.
- Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. Learned cardinality estimation: An in-depth study. In *Proceedings of the 2022 International Conference on Management of Data*, pages 1214–1227, 2022.
- Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018.
- David Koslicki and Hooman Zabeti. Improving minhash via the containment index with applications to metagenomic analysis. *Applied Mathematics and Computation*, 354:206–215, 2019.
- Hai Lan, Zhifeng Bao, and Yuwei Peng. A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration. *Data Science and Engineering*, 6:86–101, 2021.
- Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In *2017 IEEE 58th annual symposium on foundations of computer science (FOCS)*, pages 633–638. IEEE, 2017.
- Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499*, 2021.
- Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015.
- Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters, 2008.
- Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge University Press, 2020.
- Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? a survey on approximate query processing. *Data Science and Engineering*, 3:379–397, 2018.
- Xiaoyun Li and Ping Li. Rejection sampling for weighted jaccard similarity revisited. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 4197–4205, 2021.
- Xin Li and Hsinchun Chen. Recommendation as link prediction: a graph kernel-based machine learning approach. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital Libraries*, pages 213–216, 2009.

- David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.
- Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128(2):261–318, 2020.
- Shuxin Liu, Xinsheng Ji, Caixia Liu, and Yi Bai. Extended resource allocation index for link prediction of complex network. *Physica A: Statistical Mechanics and its Applications*, 479:174–183, 2017.
- Yipeng Liu, Jiani Liu, Zhen Long, and Ce Zhu. Tensor sketch. *Tensor Computation for Data Analysis*, pages 299–321, 2022.
- Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.
- Linjian Ma and Edgar Solomonik. Fast and accurate randomized algorithms for low-rank tensor decompositions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 24299–24312. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/cbef46321026d8404bc3216d4774c8a9-Paper.pdf.
- Arvind V. Mahankali, David P. Woodruff, and Ziyu Zhang. Near-Linear Time and Fixed-Parameter Tractable Algorithms for Tensor Decompositions. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 79:1–79:23, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-309-6. doi: 10.4230/LIPIcs.ITCS.2024.79. URL <https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2024.79>.
- Osman Asif Malik and Stephen Becker. Low-rank tucker decomposition of large tensors using tensorsketch. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/45a766fa266ea2eb6b6680fa139d2a3d-Paper.pdf.
- Bradley Malin, Edoardo Airoldi, and Kathleen M Carley. A network analysis model for disambiguation of names in lists. *Computational & Mathematical Organization Theory*, 11(2):119–139, 2005.
- Nishad Manerikar and Themis Palpanas. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *Data & Knowledge Engineering*, 68(4):415–430, 2009.
- Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 141–150, 2007.

- Igor L Markov and Yaoyun Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.
- Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Computing Surveys (CSUR)*, 49(4):1–33, 2016.
- Jiří Matoušek. On variants of the johnson–lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008. doi: <https://doi.org/10.1002/rsa.20218>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/rsa.20218>.
- Christoph Mayer, Haozhe Zhang, Mahmoud Abo Khamis, Kyle Deeds, Dan Olteanu, and Dan Suciu. Corrbound: Cardinality estimation accounting for inter- and intra-relation correlations. *Proc. ACM Manag. Data*, 4(1), April 2026. doi: 10.1145/3786633. URL <https://doi.org/10.1145/3786633>.
- Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Trans. Database Syst.*, 31(3):1095–1133, September 2006. ISSN 0362-5915. doi: 10.1145/1166074.1166084. URL <https://doi.org/10.1145/1166074.1166084>.
- J. Misra and David Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982. ISSN 0167-6423. doi: [https://doi.org/10.1016/0167-6423\(82\)90012-0](https://doi.org/10.1016/0167-6423(82)90012-0). URL <https://www.sciencedirect.com/science/article/pii/0167642382900120>.
- Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proceedings of the VLDB Endowment*, 2(1):982–993, 2009.
- Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.
- Ryan Moughan. *Parallel Architectures for Hyperdimensional Computing*. PhD thesis, Master’s thesis. University of California at Berkeley, 2021.
- Magnus Müller. *Selected problems in cardinality estimation*. PhD thesis, University of Mannheim, 2022.
- Suphakit Niwattanakul, Jatsada Singthongchai, Ekkachai Naenudorn, and Supachanun Wanapu. Using of jaccard coefficient for keywords similarity. In *Proceedings of the International Multiconference of Engineers and Computer Scientists*, volume 6, pages 380–384, 2013.
- Alexander Novikov, Dmitrii Podoprikin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 28, 2015.
- Igor Nunes, Mike Heddes, Pere Vergés, Danny Abraham, Alex Veidenbaum, Alex Nicolau, and Tony Givargis. Dothash: Estimating set similarity metrics for link prediction and document deduplication. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD ’23, page 1758–1769, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599314. URL <https://doi.org/10.1145/3580305.3599314>.

- Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17(1):1–14, 2016.
- Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.
- Rafael Padilla, Sergio L Netto, and Eduardo AB Da Silva. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals, and Image Processing (IWSSIP)*, pages 237–242. IEEE, 2020.
- Rasmus Pagh. Compressed matrix multiplication. *ACM Transactions on Computation Theory (TOCT)*, 5(3):1–17, 2013.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 2019.
- Ninh Pham and Rasmus Pagh. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 239–247, 2013.
- Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. Trust management for the semantic web. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, pages 351–368, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39718-2.
- Stephen Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of Documentation*, 2004.
- Elina Robeva and Anna Seigal. Duality of graphical models and tensor networks. *Information and Inference: A Journal of the IMA*, 8(2):273–288, 2019.
- Kenneth H Rosen. *Discrete mathematics and its applications*. McGraw-Hill Higher Education, 2012.
- Gordon J Ross, Dimitris K Tasoulis, and Niall M Adams. Nonparametric monitoring of data streams for changes in location and scale. *Technometrics*, 53(4):379–389, 2011.
- Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1449–1463, 2016.
- Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9, 2021.
- Florin Rusu and Alin Dobra. Sketches for size of join estimation. *ACM Transactions on Database Systems (TODS)*, 33(3):1–46, 2008.
- Athar Sefid, Jian Wu, C Ge Allen, Jing Zhao, Lu Liu, Cornelia Caragea, Prasenjit Mitra, and C Lee Giles. Cleaning noisy and heterogeneous metadata for record linking across

- scholarly big datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 9601–9606, 2019.
- Yang Shi and Animashree Anandkumar. Higher-order count sketch: dimensionality reduction that retains efficient tensor operations. *arXiv preprint arXiv:1901.11261*, 2019.
- Anshumali Shrivastava. Simple and efficient weighted minwise hashing. *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 2016.
- Anshumali Shrivastava and Ping Li. In defense of minhash over simhash. In *Artificial Intelligence and Statistics*, pages 886–894. PMLR, 2014.
- Nisheeth Shrivastava, Chiranjeev Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.
- Statista. Worldwide visits to google.com from october 2023 to march 2024 (in billions), 2024. URL <https://www.statista.com/statistics/268252/web-visitor-traffic-to-googlecom/>.
- Statista. Volume of data or information created, captured, copied, and consumed worldwide from 2010 to 2029 (in zettabytes), 2025. URL <https://www.statista.com/statistics/871513/worldwide-data-created/>.
- Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.
- Michael Strube and Simone Paolo Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *AAAI*, volume 6, pages 1419–1424, 2006.
- Damian Szklarczyk, Annika L. Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda Tsankova Doncheva, John H. Morris, Peer Bork, Lars Juhl Jensen, and Christian von Mering. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic Acids Research*, 47:D607 – D613, 2019.
- Carlos HC Teixeira, Arlei Silva, and Wagner Meira Jr. Min-hash fingerprints for graph kernels: A trade-off among accuracy, efficiency, and compression. *Journal of Information and Data Management*, 3(3):227–227, 2012.
- Mikkel Thorup and Yin Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 615–624, 2004.
- David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P Chakkappen. Join size estimation subject to filter conditions. *Proceedings of the VLDB Endowment*, 8(12):1530–1541, 2015.
- Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

- Feiyu Wang, Qizhi Chen, Yuanpeng Li, Tong Yang, Yaofeng Tu, et al. Joinsketch: A sketch algorithm for accurate and unbiased inner-product estimation. *Proceedings of the ACM on Management of Data*, 1(1):1–26, 2023.
- Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. Are we ready for learned cardinality estimation? *Proceedings of the VLDB Endowment*, 14(9):1640–1654, 2021.
- Yining Wang, Hsiao-Yu Tung, Alexander J Smola, and Anima Anandkumar. Fast and guaranteed tensor decomposition via sketching. *Advances in neural information processing systems*, 28, 2015.
- Mark N. Wegman and J. Lawrence Carter. New classes and applications of hash functions. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 175–182, 1979. doi: 10.1109/SFCS.1979.26.
- Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.
- David P Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.
- Wei Wu, Bin Li, Ling Chen, Junbin Gao, and Chengqi Zhang. A review for weighted minhash algorithms. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 34(6):2553–2573, 2020a.
- Ziniu Wu, Amir Shaikhha, Rong Zhu, Kai Zeng, Yuxing Han, and Jingren Zhou. Bayescard: Revitalizing bayesian frameworks for cardinality estimation. *arXiv preprint arXiv:2012.14743*, 2020b.
- Naoya Yamanaka, Takeshi Ogita, Siegfried M Rump, and Shin’ichi Oishi. A parallel algorithm for accurate dot product. *Parallel Computing*, 34(6-8):392–410, 2008.
- Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid sketch: A sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment*, 10(11):1442–1453, 2017.
- Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. Neurocard: one cardinality estimator for all tables. *Proceedings of the VLDB Endowment*, 14(1):61–73, 2020.
- Wen-tau Yih and Christopher Meek. Improving similarity measures for short segments of text. In *AAAI*, volume 7, pages 1489–1494, 2007.
- Han Zhu, Mingsheng Long, Jianmin Wang, and Yue Cao. Deep hashing network for efficient similarity retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 30, 2016.
- Jianhan Zhu, Jun Hong, and John G Hughes. Using markov models for web site link pre-

diction. In *Proceedings of the Thirteenth ACM Conference on Hypertext and Hypermedia*, pages 169–170, 2002.

Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, et al. Flat: fast, lightweight and accurate method for cardinality estimation. *Proceedings of the VLDB Endowment*, 14(9):1489–1502, 2021.

Appendix A

Proofs

This appendix provides proofs for the theoretical results presented in the dissertation.

A.1 Proofs for Chapter 2

A.1.1 Intersection cardinality by dot product

Theorem A.1. *Consider arbitrary sets $A, B \subseteq S$. Let $\phi : S \rightarrow \mathbb{R}^{|S|}$ be any injective map of their elements to vectors in one of the orthonormal bases of $\mathbb{R}^{|S|}$, and let*

$$\mathbf{a} = \sum_{a \in A} \phi(a) \quad \text{and} \quad \mathbf{b} = \sum_{b \in B} \phi(b).$$

Then, $\mathbf{a} \cdot \mathbf{b} = |A \cap B|$.

Proof. By the definitions of $\mathbf{a}$ and $\mathbf{b}$,

$$\mathbf{a} \cdot \mathbf{b} = \left(\sum_{a \in A} \phi(a) \right) \cdot \left(\sum_{b \in B} \phi(b) \right)$$

Using the distributivity of the dot product over addition, we can rewrite the above equation as:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{a \in A} \sum_{b \in B} \phi(a) \cdot \phi(b)$$

Because ϕ is injective, i.e., $\phi(a) = \phi(b)$ if, and only if, $a = b$, and based on the orthogonal property of orthonormal vectors, $\phi(a) \cdot \phi(b) = 1$ if $a = b$, and zero otherwise. Then,

$$\mathbf{a} \cdot \mathbf{b} = \sum_{a \in A} \sum_{b \in B} [a = b] = \sum_{x \in A \cap B} 1 = |A \cap B|$$

□

A.1.2 Intersection cardinality estimate by dot product

Theorem A.2. *Consider arbitrary sets $A, B \subseteq S$ and any constant $d \in \mathbb{N}^+$. Let $\psi : S \rightarrow \mathbb{R}^d$ be a uniform random mapping of elements in S to unit vectors which are the points of a d -dimensional hypercube, and let*

$$\mathbf{a} = \sum_{a \in A} \psi(a) \quad \text{and} \quad \mathbf{b} = \sum_{b \in B} \psi(b).$$

Then, $\mathbb{E}[\mathbf{a} \cdot \mathbf{b}] = |A \cap B|$, and

$$\text{Var}(\mathbf{a} \cdot \mathbf{b}) = \frac{1}{d} (|A| |B| + |A \cap B|^2 - 2|A \cap B|)$$

Proof. By the definitions of $\mathbf{a}$ and $\mathbf{b}$,

$$\mathbf{a} \cdot \mathbf{b} = \left(\sum_{a \in A} \psi(a) \right) \cdot \left(\sum_{b \in B} \psi(b) \right)$$

Applying the linearity of expectation and distributivity of the dot product over addition, we have:

$$\mathbb{E}[\mathbf{a} \cdot \mathbf{b}] = \mathbb{E} \left[\sum_{a \in A} \sum_{b \in B} \psi(a) \cdot \psi(b) \right] = \sum_{a \in A} \sum_{b \in B} \mathbb{E}[\psi(a) \cdot \psi(b)]$$

Given that ψ maps uniformly to unit vectors which are the points of a d -dimensional hypercube, the elements of ψ are sampled uniformly at random from $\left\{ -\frac{1}{\sqrt{d}}, +\frac{1}{\sqrt{d}} \right\}$. When $a = b$,

$$\mathbb{E}[\psi(a) \cdot \psi(b)] = \mathbb{E}[\psi(a) \cdot \psi(a)] = \mathbb{E} \left[\sum_{i=1}^d \psi(a)_i^2 \right] = \sum_{i=1}^d \frac{1}{d} = 1$$

where the right-hand subscripts denote the dimension of the vector. And when $a \neq b$,

$$\mathbb{E}[\psi(a) \cdot \psi(b)] = \mathbb{E} \left[\sum_{i=1}^d \psi(a)_i \psi(b)_i \right] = \sum_{i=1}^d \mathbb{E}[\psi(a)_i \psi(b)_i] = 0$$

Thus, $\mathbb{E}[\psi(a) \cdot \psi(b)] = [a = b]$, which ensures that:

$$\mathbb{E}[\mathbf{a} \cdot \mathbf{b}] = \sum_{a \in A} \sum_{b \in B} [a = b] = \sum_{x \in A \cap B} 1 = |A \cap B|$$

Moreover, the variance of the estimator is obtained as follows, where $r_i(a, b) = \psi(a)_i \psi(b)_i$ and we use the definitions of $\mathbf{a}$, $\mathbf{b}$, and the dot product,

$$\text{Var}(\mathbf{a} \cdot \mathbf{b}) = \text{Var} \left(\sum_{i=1}^d \sum_{a \in A} \sum_{b \in B} r_i(a, b) \right)$$

Since each dimension of ψ is sampled independently from an identical distribution (i.i.d.),

$$\text{Var}(\mathbf{a} \cdot \mathbf{b}) = \sum_{i=1}^d \text{Var}\left(\sum_{a \in A} \sum_{b \in B} r_i(a, b)\right)$$

We then separate the equal from the non-equal pairs of a and b and note that when $a = b$ their elements are identical thus their outcome has no variance, this gives:

$$\begin{aligned} \text{Var}(\mathbf{a} \cdot \mathbf{b}) &= \sum_{i=1}^d \text{Var}\left(\sum_{x \in A \cap B} \psi(x)_i^2\right) + \text{Var}\left(\sum_{a \in A} \sum_{b \in B \setminus \{a\}} r_i(a, b)\right) \\ &= \sum_{i=1}^d \text{Var}\left(\sum_{a \in A} \sum_{b \in B \setminus \{a\}} r_i(a, b)\right) \end{aligned}$$

Using the property of linear combination of random variables we get,

$$\begin{aligned} \text{Var}(\mathbf{a} \cdot \mathbf{b}) &= \sum_{i=1}^d \sum_{a \in A} \sum_{b \in B \setminus \{a\}} \text{Var}(r_i(a, b)) \\ &\quad + \sum_{x \in A \setminus \{a\}} \sum_{y \in B \setminus \{x, b\}} \text{Cov}(r_i(a, b), r_i(x, y)) \end{aligned}$$

Lastly, we observe that the covariance is $\frac{1}{d^2}$ when $(a, b) = (y, x)$ giving the variance:

$$\begin{aligned} \text{Var}(\mathbf{a} \cdot \mathbf{b}) &= \sum_{i=1}^d \frac{1}{d^2} (|A| |B| + |A \cap B|^2 - 2|A \cap B|) \\ &= \frac{1}{d} (|A| |B| + |A \cap B|^2 - 2|A \cap B|) \end{aligned}$$

□

A.1.3 Adamic-Adar estimate by dot product

Theorem A.3. *Consider an arbitrary graph $G = (V, E)$ with nodes V and edges E . Take any constant $d \in \mathbb{N}^+$ and let $\Gamma(v) \subseteq V$ denote the neighbors of node $v \in V$. Let $\psi : V \rightarrow \mathbb{R}^d$*

be a uniform random mapping of nodes in V to unit vectors pointing to the corners of a d -dimensional hypercube, and let

$$\mathbf{u} = \sum_{x \in \Gamma(u)} \psi(x) \sqrt{\frac{1}{\log |\Gamma(x)|}} \quad \text{and} \quad \mathbf{v} = \sum_{y \in \Gamma(v)} \psi(y) \sqrt{\frac{1}{\log |\Gamma(y)|}}$$

Then, $\mathbb{E}[\mathbf{u} \cdot \mathbf{v}] = A(u, v)$.

Proof. By the definitions of $\mathbf{u}$ and $\mathbf{v}$,

$$\mathbf{u} \cdot \mathbf{v} = \left(\sum_{x \in \Gamma(u)} \psi(x) \sqrt{\frac{1}{\log |\Gamma(x)|}} \right) \cdot \left(\sum_{y \in \Gamma(v)} \psi(y) \sqrt{\frac{1}{\log |\Gamma(y)|}} \right)$$

Applying the linearity of expectation and distributivity of the dot product over addition, we have:

$$\begin{aligned} \mathbb{E}[\mathbf{u} \cdot \mathbf{v}] &= \mathbb{E} \left[\sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \left(\psi(x) \sqrt{\frac{1}{\log |\Gamma(x)|}} \right) \cdot \left(\psi(y) \sqrt{\frac{1}{\log |\Gamma(y)|}} \right) \right] \\ &= \sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \mathbb{E} \left[\left(\psi(x) \sqrt{\frac{1}{\log |\Gamma(x)|}} \right) \cdot \left(\psi(y) \sqrt{\frac{1}{\log |\Gamma(y)|}} \right) \right] \\ &= \sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \sqrt{\frac{1}{\log |\Gamma(x)|}} \sqrt{\frac{1}{\log |\Gamma(y)|}} \mathbb{E}[\psi(x) \cdot \psi(y)] \end{aligned}$$

From the previous proof we have that $\mathbb{E}[\psi(x) \cdot \psi(y)] = [x = y]$, giving:

$$\begin{aligned} \mathbb{E}[\mathbf{u} \cdot \mathbf{v}] &= \sum_{x \in \Gamma(u)} \sum_{y \in \Gamma(v)} \sqrt{\frac{1}{\log |\Gamma(x)|}} \sqrt{\frac{1}{\log |\Gamma(y)|}} [x = y] \\ &= \sum_{x \in \Gamma(u) \cap \Gamma(v)} \frac{1}{\log |\Gamma(x)|} = A(u, v) \end{aligned}$$

□

A.2 Proofs for Chapter 3

Proof for Theorem 3.3. By the definitions of X and $c_{k,j}$, with $I = I_0 \times \cdots \times I_{r-1}$, and using the linearity of expectation, we get:

$$\mathbb{E}[X] = \frac{1}{m} \sum_{j=0}^{m-1} \sum_{i \in I} \mathbb{E} \left[\prod_{k=0}^{r-1} \mathcal{F}_k(i) \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{j,\{u,v\}}(i_u) \right]$$

By construction of the sketches, the sign functions are independent across different joins and there are exactly two occurrences of each sign function, one for each end of a join edge. We can thus write:

$$\mathbb{E}[X] = \frac{1}{m} \sum_{j=0}^{m-1} \sum_{i \in I} \left(\prod_{k=0}^{r-1} \mathcal{F}_k(i) \right) \prod_{\{u,v\} \in E} \mathbb{E}[s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i_v)]$$

Since $\mathbb{E}[s(a)s(b)] = \mathbf{1}(a = b)$, we get the desired expectation.

For the variance, all the dimensions of the sketches are i.i.d., and since $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, for any $j \in [m]$, we have:

$$\text{Var}(X) = \frac{1}{m} \text{Var} \left(\prod_{k=0}^{r-1} c_{k,j} \right) \leq \frac{1}{m} \mathbb{E} \left[\left(\prod_{k=0}^{r-1} c_{k,j} \right)^2 \right]$$

By the definition of $c_{k,j}$ and the linearity of expectation, we have:

$$\begin{aligned} \text{Var}(X) &\leq \frac{1}{m} \sum_{i \in I} \sum_{i' \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \mathcal{F}_0(i') \cdots \mathcal{F}_{r-1}(i') \\ &\quad \mathbb{E} \left[\prod_{k=0}^{r-1} \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i'_u) \right] \end{aligned}$$

Taking into account that each sign function occurs twice (now four times since the value is squared), we obtain:

$$\begin{aligned} \text{Var}(X) &\leq \frac{1}{m} \sum_{i \in I} \sum_{i' \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \mathcal{F}_0(i') \cdots \mathcal{F}_{r-1}(i') \\ &\quad \prod_{\{u,v\} \in E} \mathbb{E}[s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i'_u) s_{j,\{u,v\}}(i_v) s_{j,\{u,v\}}(i'_v)] \end{aligned}$$

By the four-wise independence of the sign functions, the expected value is one if there are two equal pairs or if all values are equal, and zero otherwise. Therefore, we have that:

$$\begin{aligned} &\mathbb{E}[s_{j,\{u,v\}}(i_u) s_{j,\{u,v\}}(i'_u) s_{j,\{u,v\}}(i_v) s_{j,\{u,v\}}(i'_v)] \\ &= \mathbf{1}((i_u = i_v \wedge i'_u = i'_v) \vee (i_u = i'_u \neq i_v = i'_v) \vee (i_u = i'_v \neq i_v = i'_u)) \end{aligned}$$

For each join, there are three disjunctions which are conjoined over all the joins. By the distributivity property of conjunction over disjunction, there are thus $3^{|E|}$ disjunctions in total. Since the queries are acyclic, $|E| = r - 1$. The variance is bound by the sum over all disjunctions, where intersections are counted double. Using the Cauchy–Schwarz inequality, each disjunction itself is bound by the product of squared frequency norms. This gives the desired upper bound on the variance. $\square$

Proof for Theorem 3.4. By the definitions of X and S_k , with $I = I_0 \times \cdots \times I_{r-1}$, and the linearity of expectation, we have:

$$\mathbb{E}[X] = \sum_{j \in J} \sum_{i \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \mathbb{E} \left[\prod_{k=0}^{r-1} \mathbf{1}(H_k(i) = G_k(j)) \prod_{u \in \Omega(R_k)} \prod_{v \in \Gamma(u)} s_{\{u,v\}}(i_u) \right]$$

By the definitions of H_k and G_k , since the sign and bin functions are independent, and using

again the observation that each sign function occurs twice, we can write:

$$\mathbb{E}[X] = \sum_{i \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \left(\prod_{\{u,v\} \in E} \mathbf{1}(i_u = i_v) \right) \\ \sum_{j \in J} \mathbb{E} \left[\prod_{k=0}^{r-1} \mathbf{1} \left(\sum_{u \in \Omega(R_k)} h_{\Psi(u)}(i_u) - j_{\Psi(u)} \equiv 0 \pmod{m} \right) \right]$$

By isolating the case where all $i_u = i_v$, all the attribute values in the same graph component must be equal. Since each independent bin function is thus called with only one distinct value, the expected value of the bin functions is $m^{-(w-r+1)}$, which is exactly the reciprocal of $|J|$, giving the desired expectation.

For the variance, we have that $\text{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, where:

$$\mathbb{E}[X]^2 = \sum_{i \in I} \sum_{i' \in I} \left(\prod_{k=0}^{r-1} \mathcal{F}_k(i) \mathcal{F}_k(i') \right) \prod_{\{u,v\} \in E} \mathbf{1}(i_u = i_v \wedge i'_u = i'_v) \\ \mathbb{E}[X^2] = \sum_{i \in I} \sum_{i' \in I} \left(\prod_{k=0}^{r-1} \mathcal{F}_k(i) \mathcal{F}_k(i') \right) \left(\prod_{\{u,v\} \in E} \mathbb{E}[s_{\{u,v\}}(i_u) s_{\{u,v\}}(i'_u) s_{\{u,v\}}(i_v) s_{\{u,v\}}(i'_v)] \right) \\ \sum_{j \in J} \sum_{j' \in J} \mathbb{E} \left[\prod_{k=0}^{r-1} \mathbf{1}(H_k(i) = G_k(j)) \mathbf{1}(H_k(i') = G_k(j')) \right]$$

The expected value of the sign functions is stated in the proof for Theorem 3.3. If we consider only the first disjunction, then we get:

$$\mathbb{E}[X]^2 \sum_{j \in J} \sum_{j' \in J} \mathbb{E} \left[\prod_{k=0}^{r-1} \mathbf{1}(H_k(i) = G_k(j)) \mathbf{1}(H_k(i') = G_k(j')) \right]$$

which is equal to $\mathbb{E}[X]^2$, because when $i_u = i_v \wedge i'_u = i'_v$, all graph components must have one or two distinct values each. In the case of one distinct value, $j_q = j'_q$ for that graph component q . Thus, the expected value per graph component is either $1/m^2$ or $\mathbf{1}(j_q = j'_q)/m$, making the sums over J equal to 1. Now, let us consider everything but $\mathbb{E}[X]^2$ in $\mathbb{E}[X^2]$.

There must be at least one occurrence of either $i_u = i'_u \neq i_v = i'_v$ or $i_u = i'_v \neq i_v = i'_u$. Either way, $j_q = j'_q$ for that graph component q while there are still at least two distinct values. The sums over J is thus $\leq 1/m$. We then get that:

$$\begin{aligned}\mathbb{E}[X^2] &= \mathbb{E}[X]^2 + Y \implies \text{Var}(X) \leq \frac{1}{m} \mathbb{E}[X]^2 + Y \\ \text{Var}(X) &\leq \frac{1}{m} \sum_{i \in I} \sum_{i' \in I} \mathcal{F}_0(i) \cdots \mathcal{F}_{r-1}(i) \mathcal{F}_0(i') \cdots \mathcal{F}_{r-1}(i') \\ &\quad \prod_{\{u,v\} \in E} \mathbf{1}(i_u = i_v \wedge i'_u = i'_v) + \mathbf{1}(i_u = i'_u \neq i_v = i'_v) + \mathbf{1}(i_u = i'_v \neq i_v = i'_u)\end{aligned}$$

which is the same expression of the variance bound as the one for Theorem 3.3. The final steps of the variance bound are thus equivalent, resulting in the desired upper bound. $\square$

A.3 Proofs for Chapter 4

A.3.1 Proof of Lemma 4.5

Proof. We first use the definition of the discrete Fourier transform and the count sketch to obtain

$$\begin{aligned}(\overline{FCx})_k &= \overline{\sum_{j=1}^m e^{-i2\pi \frac{k-1}{m}(j-1)} \sum_{i=1}^n x_i s(i) [h(i) = j]} \\ &= \sum_{j=1}^m e^{i2\pi \frac{k-1}{m}(j-1)} \sum_{i=1}^n x_i s(i) [h(i) = j].\end{aligned}$$

This uses the following properties of the complex conjugate: $\overline{x+y} = \bar{x} + \bar{y}$, $\overline{xy} = \bar{x}\bar{y}$, $\overline{\exp(i)} = \exp(-i)$, and $\bar{x} = x$ when $x \in \mathbb{R}$. Using the definition of the complement count sketch, we have

$$(FC'x)_k = \sum_{l=1}^m e^{-i2\pi \frac{k-1}{m}(l-1)} \sum_{i=1}^n x_i s(i) [-h(i) + 2 \equiv l \pmod{m}].$$

Using a change of variable, setting $l = -j + 2$, we obtain

$$\begin{aligned} (FC'x)_k &= \sum_{j=1}^m e^{-i2\pi \frac{k-1}{m}(-j+2-1)} \sum_{i=1}^n x_i s(i) [-h(i) + 2 \equiv -j + 2 \pmod{m}] \\ &= \sum_{j=1}^m e^{i2\pi \frac{k-1}{m}(j-1)} \sum_{i=1}^n x_i s(i) [h(i) = j] = (\overline{FC}x)_k, \end{aligned}$$

which concludes the proof. $\square$

A.3.2 Proof of Lemma 4.6

Proof. First, note that every tensor can be written as a linear sum of the tensor product of standard basis vectors: $\mathcal{X}_k = \sum_{\mathbf{i}_k=1}^{\mathbf{n}_k} \mathcal{X}_k(\mathbf{i}_k) \times_{u \in V(k)} e_{i_u}$, where $\times_{u \in S}$ denotes the tensor product for each member u of S . Since $x_k = (\bullet_{u \in V(k)} FC_u) \text{vec}(\mathcal{X}_k)$, by linearity and the mixed-product identity $(A \bullet B)(x \otimes y) = (Ax) \circ (By)$,

$$x_k = \left(\bullet_{u \in V(k)} FC_u \right) \left(\sum_{\mathbf{i}_k=1}^{\mathbf{n}_k} \mathcal{X}_k(\mathbf{i}_k) \bigotimes_{u \in V(k)} e_{i_u} \right) = \sum_{\mathbf{i}_k=1}^{\mathbf{n}_k} \mathcal{X}_k(\mathbf{i}_k) \left(\circ_{u \in V(k)} FC_u e_{i_u} \right),$$

where $\circ_{u \in S}$ is the Hadamard product for each member u of S . This expresses the sketch x_k of tensor $\mathcal{X}_k$ as a sum over its entries. Note that $\bullet_{u \in V(k)} FC_u$ is the Fourier transform of a tensor sketch matrix and $\text{vec}(\mathcal{X}_k) = \sum_{\mathbf{i}_k=1}^{\mathbf{n}_k} \mathcal{X}_k(\mathbf{i}_k) \bigotimes_{u \in V(k)} e_{i_u}$. The estimate is then

obtained by combining the sketches with the Hadamard product as follows:

$$\hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E) = e_1^\top F^{-1}(x_1 \circ \cdots \circ x_p) \quad (\text{A.1})$$

$$= \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) e_1^\top F^{-1} \left(\bigcirc_{u \in [q]} FC_u e_{i_u} \right) \quad (\text{A.2})$$

$$= \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) e_1^\top F^{-1} \left(\left(\bigcirc_{(u,v) \in E} FC_u e_{i_u} \right) \circ \left(\bigcirc_{(u,v) \in E} FC_v e_{i_v} \right) \right) \quad (\text{A.3})$$

$$= \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) e_1^\top F^{-1} \left(\left(\bigcirc_{(u,v) \in E} FC_u e_{i_u} \right) \circ \left(\bigcirc_{(u,v) \in E} FC'_u e_{i_v} \right) \right) \quad (\text{A.4})$$

$$= \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) e_1^\top F^{-1} \left(\left(\bigcirc_{(u,v) \in E} FC_u e_{i_u} \right) \circ \overline{\left(\bigcirc_{(u,v) \in E} FC_u e_{i_v} \right)} \right) \quad (\text{A.5})$$

$$= \frac{1}{m} \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) \left\langle \bigcirc_{(u,v) \in E} FC_u e_{i_u}, \bigcirc_{(u,v) \in E} FC_u e_{i_v} \right\rangle \quad (\text{A.6})$$

$$= \frac{1}{m} \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) \left\langle FT \bigotimes_{(u,v) \in E} e_{i_u}, FT \bigotimes_{(u,v) \in E} e_{i_v} \right\rangle \quad (\text{A.7})$$

$$= \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) \left\langle T \bigotimes_{(u,v) \in E} e_{i_u}, T \bigotimes_{(u,v) \in E} e_{i_v} \right\rangle. \quad (\text{A.8})$$

Eq. (A.3) divides the count sketches and the complement count sketches into two groups. Note that for every contraction, a count sketch is assigned to one mode and its complement to the other, so there is an equal number of them. Eq. (A.5) uses Lemma 4.5 and Eq. (A.6) uses $e_1^\top F^{-1}(x \circ \bar{y}) = \frac{1}{m} \langle x, y \rangle$. This step of the proof requires each contraction to have one conjugated mode which is achieved using the complement count sketches. Eq. (A.7) uses the fact that Eq. (A.6) includes a Fourier transform of a tensor sketch. Eq. (A.8) uses the invariance of the inner product to a unitary transformation, given that F is a unitary matrix times $\sqrt{m}$. The expectation of the estimator is given by

$$\mathbb{E}[\hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E)] = \sum_{i=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) \mathbb{E} \left\langle T \bigotimes_{(u,v) \in E} e_{i_u}, T \bigotimes_{(u,v) \in E} e_{i_v} \right\rangle,$$

where

$$\mathbb{E} \left\langle T \bigotimes_{(u,v) \in E} e_{i_u}, T \bigotimes_{(u,v) \in E} e_{i_v} \right\rangle = \left\langle \bigotimes_{(u,v) \in E} e_{i_u}, \bigotimes_{(u,v) \in E} e_{i_v} \right\rangle = \prod_{(u,v) \in E} \langle e_{i_u}, e_{i_v} \rangle = \prod_{(u,v) \in E} [i_u = i_v].$$

The expected value and the variance bound follow directly from Lemma 4.3. The squared Frobenius norms for each tensor in the variance bound stem from the sum of tensor components. Note that in our proof, we do not assume anything about the structure of the tensor network. $\square$

A.3.3 Proof of Lemma 4.7

Proof. Since $\text{vec}(\mathcal{X}_i) = \mathbf{1}_n \otimes \mathbf{1}_n$ for $2 \leq i \leq q$, we use the convolution theorem $F(x \star y) = \overline{Fx} \circ Fy$ and the mixed-product identity $(A \bullet B)(x \otimes y) = (Ax) \circ (By)$ to obtain

$$y = e_1^\top F^{-1}((FC_1 \mathbf{1}) \circ \overline{(FC_1 \mathbf{1})} \circ (FC_2 \mathbf{1}) \circ \dots \circ \overline{(FC_{q-1} \mathbf{1})} \circ (FC_q \mathbf{1}) \circ (FC_q \mathbf{1})),$$

where the complex conjugate is taken over every other pair of count sketches. We can use the associativity of the Hadamard product to rearrange the expression so that all the conjugated terms are on one side. Note that $e_1^\top F^{-1}(\overline{x} \circ y) = \frac{1}{m} \mathbf{1}^\top (\overline{x} \circ y) = \frac{1}{m} \langle x, y \rangle$, and $\|x\|_2^2 = \langle x, x \rangle$, giving

$$\frac{1}{m} \langle (FC_1 \mathbf{1}) \circ \dots \circ (FC_q \mathbf{1}), (FC_1 \mathbf{1}) \circ \dots \circ (FC_q \mathbf{1}) \rangle = \frac{1}{m} \|(FC_1 \mathbf{1}) \circ \dots \circ (FC_q \mathbf{1})\|_2^2.$$

We now apply (Ahle et al., 2020a, Lemma 43), since $\|T(x^{\otimes q})\|_2^2 = \frac{1}{m} \|(FC_1 \mathbf{1}) \circ \dots \circ (FC_q \mathbf{1})\|_2^2$ because F^{-1} is a unitary matrix times $\frac{1}{\sqrt{m}}$. Lastly, note that $\|x^{\otimes q}\|_2^4 = \prod_{i=1}^{q+1} \|\mathcal{X}_i\|_F^2$, concluding the proof. $\square$

A.3.4 Proof of Lemma 4.8

Proof. We can assume that the TNC satisfies the WLOG assumptions from Section 4.4.1.

Then, by expanding $g(o)$ and the definition of the inner product, we have

$$\text{vec}(\mathcal{X}_o)^\top g(o) = \sum_{\mathbf{i}_o=1}^n \mathcal{X}_o(\mathbf{i}_o) \left(\bigotimes_{l_1 \in \Gamma(o)} \text{mat}(\mathcal{X}_{l_1}) g(l_1) \right) (\mathbf{i}_o),$$

where $\mathbf{i}_k$ contains all the indices of tensor $\mathcal{X}_k$. Let $\mathbf{o}_k$ contain all the indices associated with outgoing edges of tensor $\mathcal{X}_k$, and let $\mathbf{l}_i$ contain all the indices of all the tensors at a distance i from the root, also called the i th level of the tree. Note that $\mathbf{i}_o = \mathbf{l}_1$ since the indices of the root tensor are all the indices in the first level. Let $E(l)$ denote the subset of contractions that are part of the first l levels of the tree. We then obtain the following:

$$\text{vec}(\mathcal{X}_o)^\top g(o) = \sum_{\mathbf{l}_1=1}^n \sum_{\mathbf{l}_2=1}^n \mathcal{X}_o(\mathbf{i}_o) \left(\prod_{l_1 \in \Gamma(o)} \mathcal{X}_{l_1}(\mathbf{i}_{l_1}) g(l_1)(\mathbf{o}_{l_1}) \right) \prod_{(u,v) \in E(1)} [i_u = i_v],$$

where the indices in $\mathbf{i}_o$ are doubled with the introduction of $\mathbf{i}_{l_1}$. The added Iverson brackets ensures that the two indices are equal. We can then repeat this expansion of the recursion for all the w levels of the tree, giving us

$$\begin{aligned} \text{vec}(\mathcal{X}_o)^\top g(o) &= \sum_{\mathbf{l}_1=1}^n \sum_{\mathbf{l}_2=1}^n \cdots \sum_{\mathbf{l}_w=1}^n \mathcal{X}_o(\mathbf{i}_o) \left(\prod_{l_1 \in \Gamma(o)} \mathcal{X}_{l_1}(\mathbf{i}_{l_1}) \left(\prod_{l_2 \in \Gamma(l_1)} \mathcal{X}_{l_2}(\mathbf{i}_{l_2}) \cdots \right) \right) \prod_{(u,v) \in E} [i_u = i_v] \\ &= \sum_{\mathbf{i}=1}^n \mathcal{X}_1(\mathbf{i}_1) \cdots \mathcal{X}_p(\mathbf{i}_p) \prod_{(u,v) \in E} [i_u = i_v], \end{aligned}$$

where $\mathbf{i}$ contains all the indices since this is a full TNC, concluding the proof. $\square$

A.3.5 Proof of Lemma 4.9

Proof. The proof proceeds by first establishing the expectation, followed by the variance bound.

Expectation. Note that all the recursive sketch matrices $R_1, \dots, R_p$ are independent from each other. Then, by the law of total expectation, Lemma 4.4, and with $x_o = \text{vec}(\mathcal{X}_o)$,

$$\mathbb{E}[y] = \mathbb{E}[x_o^\top R_o^\top R_o \hat{g}(o)] = \mathbb{E}[\mathbb{E}[x_o^\top R_o^\top R_o \hat{g}(o) | \hat{g}(o)]] = x_o^\top \left(\bigotimes_{l \in \Gamma(o)} \mathbb{E}[\text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l)] \right).$$

It follows that $\mathbb{E}[x_o^\top R_o^\top R_o \hat{g}(o)] = x_o^\top g(o)$ since $\mathbb{E}[\text{mat}(\mathcal{X}_k) R_k^\top R_k \hat{g}(k) | \hat{g}(k)] = \text{mat}(\mathcal{X}_k) \hat{g}(k)$ for any k , which matches the result in Lemma 4.8.

Variance bound. By the law of total variance, Lemma 4.4, and with $x_o = \text{vec}(\mathcal{X}_o)$,

$$\begin{aligned} \text{Var}(\hat{\text{tc}}(\mathcal{X}_1 \times \dots \times \mathcal{X}_p, E)) &= \mathbb{E}[\text{Var}(x_o^\top R_o^\top R_o \hat{g}(o) | \hat{g}(o))] + \text{Var}(\mathbb{E}[x_o^\top R_o^\top R_o \hat{g}(o) | \hat{g}(o)]) \\ &= \left(\left(1 + \frac{8}{m} \right)^{2q_o} - 1 \right) \|\mathcal{X}_o\|_{\text{F}}^2 \mathbb{E}[\|\hat{g}(o)\|_2^2] + \text{Var}(x_o^\top \hat{g}(o)). \end{aligned}$$

For the norm of $\hat{g}$, with $M_l = \text{mat}(\mathcal{X}_l)$, we have that

$$\begin{aligned} \mathbb{E}[\|\hat{g}(k)\|_2^2 | \hat{g}(l) \forall l \in \Gamma(k)] &= \prod_{l \in \Gamma(k)} \mathbb{E}[\|\text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l)\|_2^2 | \hat{g}(l)] \\ &= \prod_{l \in \Gamma(k)} \sum_{i=1}^n \mathbb{E}[(M_l(i, :) R_l^\top R_l \hat{g}(l))^2 | \hat{g}(l)] \\ &= \prod_{l \in \Gamma(k)} \sum_{i=1}^n \text{Var}(M_l(i, :) R_l^\top R_l \hat{g}(l) | \hat{g}(l)) + \mathbb{E}[M_l(i, :) R_l^\top R_l \hat{g}(l) | \hat{g}(l)]^2. \end{aligned}$$

Using Lemma 4.4 and the Cauchy-Schwarz inequality we get the following bound:

$$\begin{aligned}\mathbb{E}[\|\hat{g}(k)\|_2^2|\hat{g}(l) \ \forall l \in \Gamma(k)] &\leq \prod_{l \in \Gamma(k)} \left(\left(1 + \frac{8}{m}\right)^{2q_l-2} - 1 \right) \|\mathcal{X}_l\|_F^2 \mathbb{E}[\|\hat{g}(l)\|_2^2] + \|\mathcal{X}_l\|_F^2 \mathbb{E}[\|\hat{g}(l)\|_2^2] \\ &= \prod_{l \in \Gamma(k)} \left(1 + \frac{8}{m}\right)^{2q_l-2} \|\mathcal{X}_l\|_F^2 \mathbb{E}[\|\hat{g}(l)\|_2^2].\end{aligned}$$

Going back to the remaining variance term, we have that

$$\begin{aligned}\text{Var}(x_o^\top \hat{g}(o)) &= \mathbb{E} \left[\left(x_o^\top \left(\bigotimes_{l \in \Gamma(o)} \text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l) \right) - x_o^\top \left(\bigotimes_{l \in \Gamma(o)} \text{mat}(\mathcal{X}_l) g(l) \right) \right)^2 \right] \\ &= \mathbb{E} \left[\left(x_o^\top \left(\bigotimes_{l \in \Gamma(o)} \text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l) - \text{mat}(\mathcal{X}_l) g(l) \right) \right)^2 \right] \\ &\leq \|\mathcal{X}_o\|_F^2 \prod_{l \in \Gamma(o)} \mathbb{E}[\|\text{mat}(\mathcal{X}_l) R_l^\top R_l \hat{g}(l) - \text{mat}(\mathcal{X}_l) g(l)\|_2^2] \quad (\text{Cauchy-Schwarz}) \\ &= \|\mathcal{X}_o\|_F^2 \prod_{l \in \Gamma(o)} \sum_{i=1}^n \mathbb{E}[(M_l(i, :) R_l^\top R_l \hat{g}(l) - M_l(i, :) g(l))^2] \\ &= \|\mathcal{X}_o\|_F^2 \prod_{l \in \Gamma(o)} \sum_{i=1}^n \text{Var}(M_l(i, :) R_l^\top R_l \hat{g}(l))\end{aligned}$$

We then apply the law of total variance to obtain:

$$\begin{aligned}\text{Var}(x_o^\top \hat{g}(o)) &\leq \|\mathcal{X}_o\|_F^2 \prod_{l \in \Gamma(o)} \sum_{i=1}^n \mathbb{E}[\text{Var}(M_l(i, :) R_l^\top R_l \hat{g}(l) | \hat{g}(l))] + \text{Var}(\mathbb{E}[M_l(i, :) R_l^\top R_l \hat{g}(l) | \hat{g}(l)]) \\ &\leq \|\mathcal{X}_o\|_F^2 \prod_{l \in \Gamma(o)} \left(\left(1 + \frac{8}{m}\right)^{2q_l-2} - 1 \right) \|\mathcal{X}_l\|_F^2 \mathbb{E}[\|\hat{g}(l)\|_2^2] + \sum_{i=1}^n \text{Var}(M_l(i, :) \hat{g}(l)).\end{aligned}$$

Combining the results thus far gives us

$$\begin{aligned}\text{Var}(y) &\leq \left(\left(1 + \frac{8}{m}\right)^{2q_o} - 1 \right) \|\mathcal{X}_o\|_F^2 \prod_{l \in \Gamma(o)} \left(1 + \frac{8}{m}\right)^{2q_l-2} \|\mathcal{X}_l\|_F^2 \mathbb{E}[\|\hat{g}(l)\|_2^2] \\ &\quad + \|\mathcal{X}_o\|_F^2 \prod_{l \in \Gamma(o)} \left(\left(1 + \frac{8}{m}\right)^{2q_l-2} - 1 \right) \|\mathcal{X}_l\|_F^2 \mathbb{E}[\|\hat{g}(l)\|_2^2] + \sum_{i=1}^n \text{Var}(M_l(i, :) \hat{g}(l)),\end{aligned}$$

The first term recursively aggregates all the tensor norms and $(1 + 8/m)$ coefficients by traversing the tensor diagram from the root to the leaves. In total, there are $q_o + \sum_{k=1: k \neq o}^p q_k - 1 = q - p + 1 = t$ of such $(1 + 8/m)$ coefficients. Note that, for each leaf of the tree, $\hat{g}(l) = 1$, which has zero variance. If we set $t' = t - q_o$, then this results in the following:

$$\begin{aligned} \text{Var}(y) &\leq \left(1 + \frac{8}{m}\right)^{2t} \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2 - \left(1 + \frac{8}{m}\right)^{2t'} \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2 \\ &\quad + \|\mathcal{X}_o\|_{\text{F}}^2 \prod_{l \in \Gamma(o)} \left(\left(1 + \frac{8}{m}\right)^{2q_l - 2} - 1 \right) \|\mathcal{X}_l\|_{\text{F}}^2 \mathbb{E}[\|\hat{g}(l)\|_2^2] + \sum_{i=1}^n \text{Var}(M_l(i, :)\hat{g}(l)), \end{aligned}$$

The final inequality is derived from the observation that the overcounting of the norms of the leaf tensors results in the cancellation of the last two terms. However, it should be noted that we are overcounting by no less than the product of the norms, thus ultimately yielding

$$\text{Var}(y) \leq \left(1 + \frac{8}{m}\right)^{2t} \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2 - \left(1 + \frac{8}{m}\right)^{2t'} \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2 + \left(1 + \frac{8}{m}\right)^{2t'} \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2 - \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2,$$

which equals to $\left(\left(1 + \frac{8}{m}\right)^{2t} - 1\right) \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2$, concluding the proof. $\square$

A.3.6 Proof of Theorem 4.1

Proof. Let $y = \hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E)$, then, since this is a full TNC,

$$\mathbb{E}[\|y - \text{tc}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E)\|_{\text{F}}^2] = \mathbb{E}[(y - \text{tc}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E))^2] = \text{Var}(y).$$

Note that $(1 + 8/m)^{2t} - 1 \leq \exp(16t/m) - 1 \leq 32t/m$ when $m \geq 16t$. Therefore, by **(1)** Lemma 4.9, let $m = \Omega(t/\epsilon^2)$; and **(2)** Lemma 4.6, let $m = \Omega(3^t/\epsilon^2)$ to obtain an estimator for full ATNC (see Definition 4.4) with constant success probability using the Chebyshev inequality. Note that $\prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}} = \|\mathcal{X}_1 \times \cdots \times \mathcal{X}_p\|_{\text{F}}$. We obtain success with high probability by selecting the median of $O(\log 1/\delta)$ independent and identically distributed

estimates with constant success probability using the Chernoff bound. $\square$

A.3.7 Proof of Corollary 4.1

Proof. Expanding the squared Frobenius norm of the error, with $\hat{\mathcal{Y}}$ and $\mathcal{Y}$ as the approximate and exact TNC, respectively, gives

$$\begin{aligned} & \mathbb{E} \left[\left\| \hat{\text{tc}}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E) - \text{tc}(\mathcal{X}_1 \times \cdots \times \mathcal{X}_p, E) \right\|_{\text{F}}^2 \right] \\ &= \sum_{\mathbf{r}=1}^n \mathbb{E} \left[\left(\hat{\mathcal{Y}}(\mathbf{r}) - \mathcal{Y}(\mathbf{r}) \right)^2 \right] \\ &= \sum_{\mathbf{r}=1}^n \text{Var} \left(\hat{\mathcal{Y}}(\mathbf{r}) \right). \end{aligned}$$

Let $\mathcal{X}_k(\mathbf{s}_k)$ denote the subtensor of $\mathcal{X}_k$ used to compute the entry $\hat{\mathcal{Y}}(\mathbf{r})$, with

$$\mathbf{s}_k = (\gamma(u) : u \in V(k)), \quad \text{and} \quad \gamma(u) = \begin{cases} : & \text{if } u \in K, \\ i_u & \text{otherwise,} \end{cases}$$

where $V(k)$ are the modes of $\mathcal{X}_k$ and K the contracted modes. Then, by **(1)** Lemma 4.9, let $m \geq 32t/(\epsilon^2\delta)$; and **(2)** Lemma 4.6, let $m \geq 3^t/(\epsilon^2\delta)$,

$$\sum_{\mathbf{r}=1}^n \text{Var} \left(\hat{\mathcal{Y}}(\mathbf{r}) \right) \leq \sum_{\mathbf{r}=1}^n \epsilon^2 \delta \prod_{k=1}^p \|\mathcal{X}_k(\mathbf{s}_k)\|_{\text{F}}^2 = \epsilon^2 \delta \prod_{k=1}^p \|\mathcal{X}_k\|_{\text{F}}^2.$$

By the Chebyshev inequality and Definition 4.4, this concludes the proof. $\square$

Appendix B

Supplementary algorithms

This appendix contains pseudocode for functions used by the main algorithms presented in the dissertation. We start with Algorithm 6, which computes the sign and row of the tensor sketch matrix for a given column (which corresponds to a tensor index). This function is used in Algorithm 7, which computes the sign and row of the recursive sketch matrix for a given column. This function, in turn, is used to simultaneously compute the sketch of a tensor and its matrix-vector product in Algorithm 8.

The tensor sketch matrix of order q is composed of q count sketch matrices, which in turn consist of q sign and row hash functions. An alternative definition of the tensor sketch matrix $T \in \mathbb{R}^{m \times n^q}$, specifies each component as $T(j, i) = s_1(i_1) \cdots s_q(i_q)[(h_1(i_1) + \cdots + h_q(i_q) - q + 1) \equiv j \pmod{m}]$, where $i = 1 + \sum_{k=1}^q (i_k - 1)n^{q-k}$ (the vectorization of tensor index $\mathbf{i}$). Based on this formulation, Algorithm 6 computes the sign and row of the tensor sketch matrix for a given column. This is achieved by multiplying the signs and summing the rows (modulo the sketch size), obtained by the count sketch hash functions for each mode.

Algorithm 7 computes the sign and row of the recursive sketch matrix for a given column. This is achieved by reducing a list of row indices, which are obtained by the row hash func-

Algorithm 6 Tensor sketch matrix sign and row by column.

input: index $\mathbf{i} = (i_1, \dots, i_q)$ and tensor sketch matrix $T \in \mathbb{R}^{m \times n^q}$, which is composed of independent count sketch matrices $C_1, \dots, C_q \in \mathbb{R}^{m \times n}$ that consist of sign and row hash functions s_k and h_k for all $k \in [q]$

output: $\sigma \in \{-1, 1\}$ and $b \in [m]$

```

1: function TENSORSKETCHHASH( $\mathbf{i}, T$ )
2:    $\sigma \leftarrow 1$ 
3:    $b \leftarrow 0$ 
4:   for  $k \in [q]$  do
5:      $\sigma \leftarrow \sigma \cdot s_k(i_k)$ 
6:      $b \leftarrow b + h_k(i_k)$ 
7:   end for
8:    $b \leftarrow ((b - q) \bmod m) + 1$ 
9:   return  $\sigma, b$ 
10: end function

```

tions of the count sketch matrices for each mode, in a binary tree fashion using Algorithm 6. At each step, the signs are accumulated through multiplication.

Algorithm 8 computes the size- m sketch of an order- q tensor $\mathcal{X}$ while simultaneously multiplying the resulting sketch matrix with a vector. This is an important function for our second method as it significantly improves upon the complexity of first computing the sketch followed by the matrix-vector product, which would take time $O(q \text{nnz}(\mathcal{X}) + m^2)$ and space $O(m^2)$, and improves it to just $O(q \text{nnz}(\mathcal{X}))$ time and $O(m)$ space.

Algorithm 7 Recursive sketch matrix sign and row by column.

input: index $\mathbf{i} = (i_1, \dots, i_q)$ and recursive sketch matrix $R \in \mathbb{R}^{m \times n^q}$, which is composed of independent count sketch matrices $C_1, \dots, C_q \in \mathbb{R}^{m \times n}$ that consist of sign and row hash functions s_k and h_k for all $k \in [q]$, and independent tensor sketch matrices $T_{l,1}, \dots, T_{l,l/2} \in \mathbb{R}^{m \times m^2}$ for $l \in \{2, 4, \dots, q/2, q\}$

output: $\sigma \in \{-1, 1\}$ and $b \in [m]$

```

1: function RECURSIVESKETCHHASH( $\mathbf{i}, R$ )
2:    $\sigma \leftarrow 1$ 
3:    $b \in \{0\}^q$ 
4:   for  $k \in [q]$  do
5:      $\sigma \leftarrow \sigma \cdot s_k(i_k)$ 
6:      $b_k \leftarrow h_k(i_k)$ 
7:   end for
8:   for  $l \in \{q, q/2, \dots, 4, 2\}$  do
9:     for  $k \in [l/2]$  do
10:       $s, h \leftarrow \text{TensorSketchHash}((b_{2k-1}, b_{2k}), T_{l,k})$ 
11:       $\sigma \leftarrow \sigma \cdot s$ 
12:       $b_k \leftarrow h$ 
13:    end for
14:  end for
15:  return  $\sigma, b_1$ 
16: end function

```

Algorithm 8 Combined tensor sketching and matrix-vector product.

input: order q tensor $\mathcal{X} \in \mathbb{R}^{n \times \dots \times n}$, count sketch matrix $C \in \mathbb{R}^{m \times n}$ that consists of sign and row hash functions s and h , recursive sketch matrix $R \in \mathbb{R}^{m \times n^{q-1}}$, and any vector $z \in \mathbb{R}^m$

output: $y = C \text{mat}(\mathcal{X}) R^\top z$

```

1: function SKETCHEDMATRIXVECTORPRODUCT( $C, \mathcal{X}, R, z$ )
2:    $y \in \{0\}^m$ 
3:   for each component  $\mathcal{X}(\mathbf{i})$  do
4:      $j \leftarrow h(i_1)$ 
5:      $\sigma, b = \text{RecursiveSketchHash}((i_2, i_3, \dots, i_q), R)$ 
6:      $y_j \leftarrow y_j + \mathcal{X}(\mathbf{i}) \cdot z_b \cdot \sigma \cdot s(i_1)$ 
7:   end for
8:   return  $y$ 
9: end function

```

Appendix C

Supplementary empirical results

This appendix presents additional empirical results. The results below provide more details to those interested in, for example, exploring the DotHash framework in future work.

The results in Figure C.1 show the time taken to compare the sketch of the sets created by each method in the link prediction datasets. For each of the methods and datasets, the figure shows the CPU and GPU times for different sketch dimensions d , in the case of DotHash and SimHash, and min-wise hash functions k for MinHash. The lines represent the mean over 5 runs of calculating the represented metric for all edges in the test set.

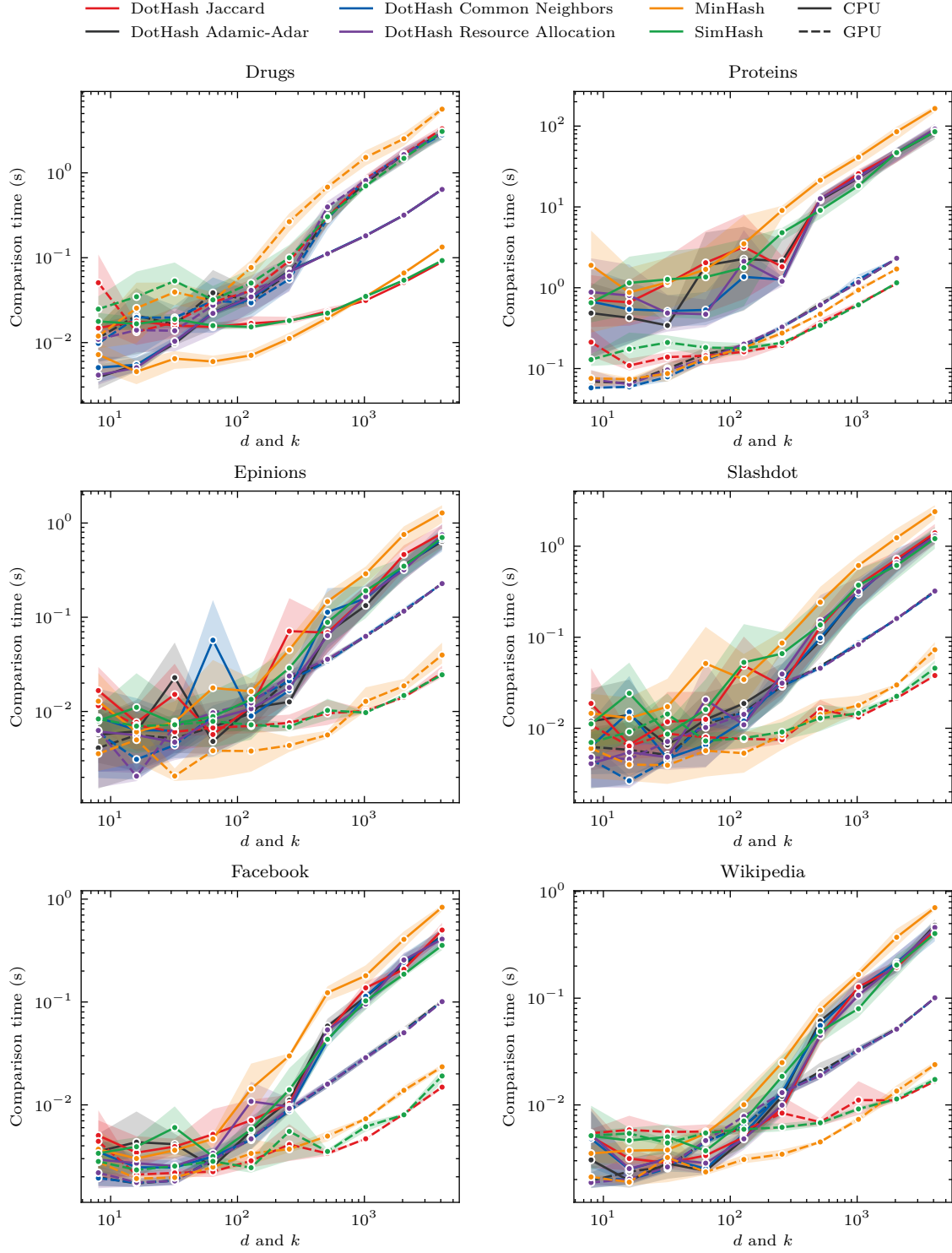


Figure C.1. Node signature comparison time in the link prediction tasks.