On Bilinear Techniques for Similarity Search and Boolean Matrix Multiplication

Matti Karppa
On Bilinear Techniques for Similarity Search and Boolean Matrix Multiplication

Matti Karppa

A doctoral dissertation completed for the degree of Doctor of Science (Technology) to be defended, with the permission of the Aalto University School of Science, at a public examination held at the lecture hall T2 of the school on 24 January 2020 at 12.

Aalto University
School of Science
Computer Science
Supervising professor
Professor Petteri Kaski, Aalto University, Finland

Preliminary examiners
Professor Alexandr Andoni, Columbia University, USA
Professor Oded Schwartz, Hebrew University of Jerusalem, Israel

Opponent
Professor Rasmus Pagh, IT University of Copenhagen, Denmark
Abstract

Algorithms are the art of efficient computation: it is by the power of algorithms that solving problems becomes feasible, and that we may harness the power of computing machinery. Efficient algorithms translate directly to savings in resources, such as time, storage space, and electricity, and thus money. With the end of the exponential increase in the computational power of hardware, the value of efficient algorithms may be greater than ever.

This thesis presents advancements in multiple fields of algorithms, related through the application of bilinear techniques. Functions that map elements from a pair of vector spaces to a third vector space with the property that they are linear in their arguments, or bilinear maps, are a ubiquitous and fundamental mathematical tool, the canonical example being the matrix multiplication. We address both the applications that make use of bilinear maps and the computation of the bilinear maps itself, Boolean matrix multiplication in particular.

In the field of similarity search, we improve on Valiant's randomized algorithm [FOCS 2012; J. ACM 2015] for finding correlated vectors by (i) presenting an improved sampling scheme that enables faster processing by using fast matrix multiplication, and (ii) derandomizing Valiant's algorithm. These results are mostly of theoretical nature since they rely on fast matrix multiplication.

We also present (iii) an adaptive prefix-assignment method for symmetry breaking. An instantiation of McKay's canonical extension framework [J. Algorithms 1998], the method produces a set of partial assignments with respect to a sequence of a prefix of variables in a system of constraints, such that all generated assignments are pairwise nonisomorphic. The method breaks the symmetries completely with respect to the prefix sequence, and can benefit from an auxiliary representation of symmetries in the form of a colored graph. We also provide an implementation that works as a preprocessor for Boolean satisfiability solvers, and show experimentally that the method is also of practical value and parallelizes well in a distributed computer cluster setting.

We address matrix multiplication by (iv) introducing a probabilistic extension of the notions of rank and border rank, and show that, under this notion, the structural tensor for 2×2 matrix multiplication has strictly lower probabilistic tensor rank and border rank than the deterministic rank. We use this fact to derive a randomized algorithm for multiplying two Boolean matrices that is asymptotically faster than Strassen's algorithm [Numer. Math. 1969].

Finally, (v) using the recent result of Karstadt and Schwartz [SPAA 2017], we implement Strassen's multiplication over the binary field in an alternative basis for a multiple-GPU shared-memory system. We evaluate the implementation with one-tebibit input, and show that it exceeds the theoretical peak performance of the elementary algorithm in terms of bit operations, and also offers substantial savings in energy consumption.

Keywords bilinear algorithms, matrix multiplication, similarity search, symmetry breaking

ISSN (printed) 1799-4934 ISSN (pdf) 1799-4942
Location of publisher Helsinki Location of printing Helsinki Year 2020
Tekijä
Matti Karppa

Väitöskirjan nimi
On Bilinear Techniques for Similarity Search and Boolean Matrix Multiplication

Julkaisija
Perustieteiden korkeakoulu

Yksikkö
Tietotekniikan laitos

Sarja
Aalto University publication series DOCTORAL DISSERTATIONS 9/2020

Tutkimusala
Algoritmit

Käsikirjoituksen pvm
30.12.2019

Väitöspäivä
24.01.2020

Väittelyluvan myöntämispäivä
17.12.2019

Kieli
Englanti

Monografia

Artikkeli

Essee

Väitämäärä


Työssä esitetään myös (iii) adaptiivinen prefiksinsijoitusmenetelmä symmetrian särkemiseen.


Työssä käsitellään matriisikertolaskua (iv) esittelemällä probabilistinen laajennus rankin ja border rankin käsitteille ja osoittamalla, että 2 × 2-matriisikertolaskukontorilla on aidosti pienempi probabilistinen rank ja border rank kuin deterministinen rank. Tämän tiedon avulla johdetaan kahden Boolean matriisin kertolaskuun satunnaistettu algoritmi, joka on asymptootisesti nopeampi kuin Strassenin algoritmi [Numer. Math. 1969].

Lopuksi (v) hyödyntämällä Karstadin ja Schwartzin tulosta [SPAA 2017] työssä implementoidaan Strassenin kertolasku binääriikunnan yli vaihtoehtoisessa kannassa usean GPU:n jaetun muistin järjestelmässä. Implementaation toimintaan arvioidaan yhden tebibilin syötteellä ja osoitetaan kokeellisesti, että se yllättää näiivin algoritmin teoreettisen huippusuuruuskyyvyn bittioraatioiden suhteen ja tarjoaa myös merkittäviä säästöjä energiankulutuksen suhteen.

Avainsanat bilinearialgoritm, matriisikertolasku, samankaltaisuus, symmetrinen särkeminen.

ISBN (painettu) 978-952-60-8915-7
ISBN (pdf) 978-952-60-8916-4

ISSN (painettu) 1799-4934
ISSN (pdf) 1799-4942

Julkaisupaikka Helsinkin
Painopaikka Helsinkin
Vuosi 2020

Sivumäärä 273

Preface

The doctoral thesis is the pinnacle of years of study. It concludes the journey that started years ago, and marks the transformation of a passive recipient of scientific information to one that actively contributes to the community.

This thesis has taken me four and a half years to finish, and it has taken its toll. The time has been characterized by feelings of insecurity and ineptitude. I am not quite sure if I would encourage my younger self to take on the task again, but I am grateful that it is over.

I thank Petteri Kaski, my thesis advisor, for the incredible skill he has shown with regard to theoretical computer science research; I do not think there are many people in this country who have such high standards for their research. I warmly thank my other collaborators, without whom the articles constituting this thesis could not have been written: Jukka Kohonen, Tommi Junntila, and Padraig Ó Catháin. I am also very grateful for the excellent statements by Professor Alexandr Andoni and Professor Oded Schwartz who worked as pre-examiners of this dissertation; I never expected to see so much praise from such highly regarded researchers in their respective fields. I also thank my friends who have helped make this possible: Eric, Sami, Hannu, Janne, Johannes, Jaakko, and all the others who I forgot to mention. And my wife Tuovi, for being there for me, despite all the trouble I made her and our son Linus put through.

I would like to thank the European Research Council for funding this research under the European Union’s Seventh Framework Programme (FP/2007–2013)/ERC Grant Agreement No. 338077, “Theory and Practice of Advanced Search and Enumeration”. I also thank Aalto Science-IT for providing computational resources that were used to carry out the experiments.

Before I began my work on the thesis, my advisor asked me if I was mad. At the time, I did not quite comprehend what he meant by that; now I do.

Espoo, December 27, 2019,

Matti Karppa
Contents

4.3 Symmetry breaking ........................................ 55
4.4 Adaptive prefix assignment .............................. 58
4.5 Symmetries of the matrix multiplication tensor and the Brent equations ...................................... 60
4.6 Constraint programming ................................. 61

5. Parallel and distributed processing 63
5.1 Overview of the parallel landscape ....................... 63
5.2 Parallel matrix multiplication ............................ 65
5.3 Distributed implementation of adaptive prefix assignment ... 68

6. Similarity search 69
6.1 Problem formulation ....................................... 69
6.2 Curse of dimensionality ................................... 72
6.3 Dimensionality reduction ................................... 74
6.4 Locality-sensitive hashing ................................. 75
6.5 Listing outlier correlations via fast matrix multiplication ... 77
6.6 Derandomizing the algorithm for outlier correlations ... 79

7. Conclusion 83

References 87

Errata 117

Publications 119
This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.


Author’s Contribution

Publication I: “A Faster Subquadratic Algorithm for Finding Outlier Correlations”

The paper was written jointly. In particular but without limiting to, the author rewrote the proof of the main theorem for the journal version of the paper and mostly wrote lower bound proof via the orthogonal vectors conjecture.

Publication II: “Explicit correlation amplifiers for finding outlier correlations in deterministic subquadratic time”

The paper was written jointly. In particular but without limiting to, the author wrote proofs to the application corollaries.


The paper was written jointly. The author wrote the parallelized implementation, performed the experiments, and in particular wrote the extension to value symmetry.

Publication IV: “Probabilistic tensors and opportunistic Boolean matrix multiplication”

The author worked on systematic computational work with tensor decompositions which served as the basis and led to the establishment of the concepts in question. The author also participated in the writing process.
Publication V: “Engineering Boolean matrix multiplication for multiple-accelerator shared-memory architectures”

The paper was written jointly. The author wrote the implementation, performed the experiments, and in particular wrote most of the description of the implementation and the experiments.
1. Introduction

The third industrial revolution, the shift in technology to digital electronics and computers powered by microprocessors, is often attributed to the exponential increase in computing power seen over the past half-century. Yet it is algorithms\(^1\), the art of efficient computation, that enables us to harness this power.

While algorithms have been known since antiquity, such as Euclid’s algorithm for Greatest Common Divisor (GCD) or the sieve of Eratosthenes for finding prime numbers, the Newton-Raphson iteration from the 17\(^{th}\) century, or even Gauss’\(^2\) original discovery of the Fast Fourier Transform (FFT)\(^3\) in the 19\(^{th}\) century\(^4\), vast advances were taken in the 20\(^{th}\) century with regard to the scalability for large datasets: sorting algorithms such as Quicksort\(^5\) and merge sort\(^\)\(^6\), binary search\(^7\), Karatsuba\(^8\) and Schönhage-Strassen\(^9\) algorithms for multiplying large integers, Strassen’s Fast Matrix Multiplication (FMM)\(^10\), and numerous other results. Fundamental research is carried out with computational tools such as the Metropolis-Hastings Monte Carlo algorithm\(^11\), the simplex algorithm\(^12\), and Viterbi algorithm\(^13\). Algorithms for error-correcting codes\(^14\), data compression\(^15\), and cryptography\(^16\), and cryptography\(^17\)\(^18\) are among essentials that allow us to use the Internet as we know it today. Boolean satisfiability (SAT) solvers based on Conflict-Driven Clause-Learning (CDCL)\(^19\), Support-Vector Machines (SVM)\(^20\), the back-propagation algorithm\(^21\), the Expectation-Maximization (EM) algorithm\(^22\), and various deep learning algorithms\(^23\) underlie the ongoing revolution of artificial intelligence.

Scalability, the property that the required amount of computation grows sufficiently slowly with the increase of input size, is what makes it possible to process datasets that number in the billions of entries\(^24\). A set of $10^9$ entries translates to an order of $10^{18}$ pairs; assuming we are able to process one million pairs of records a second, the difference in the runtime between a near-linear algorithm and a quadratic algorithm is between the orders of dozens of minutes and tens of thousands of years. The key application of algorithms is thus the

\(^1\)The word *algorithm* is derived from *Algorithmi*, the latinization of the name of the Persian mathematician Muhammad ibn Mūsā al-Khwārizmī (c. 780 – c. 850).
saving of resources – and making large-scale processing feasible to begin with –, with time being one of such resources; other resources include storage space and energy, and all of these equate to not only money but also effects on the environment and the society at large.

The design of algorithms may be more important than ever due to the end of Dennard scaling [154] and Moore’s law [175, 432]. We can no longer hope to simply get faster computer hardware by cramming ever smaller transistors on a silicon die; the laws of physics appear to stand in the way. Instead, we need to exploit parallel processing which may not be possible with pre-existing serial algorithms; rather, we must develop algorithms that are inherently parallelizable, that is, algorithms that by their very essence can be run in parallel.

In this thesis, we address bilinear techniques. Bilinear maps are one of the most fundamental mathematical tools that see frequent application in computer science. The canonical example of a bilinear map is the matrix multiplication: the operation $C = AB$ is linear in both of its operands. Scaling either of the operands by a constant is equal to scaling the product by the same constant, and if we substitute one of the operands by an additive expression, say $A = A' + A''$, we can either perform the addition first and compute the product second, or we may just as well compute two products and sum them later.

As a consequence, FMM algorithms recur often in this work. A key primitive operation in linear algebra, it is impossible to list its applications: a large body of work involving machine learning or computational science makes use of matrix multiplication in one way or another. Although not immediately obvious, FMM is also a very efficient tool for constructing faster algorithms for a myriad of problems, such as similarity search as presented in this thesis.

One of the important variants of the matrix multiplication is that of multiplying Boolean matrices, or Boolean Matrix Multiplication (BMM) over the Boolean semiring $\mathbb{B} = (0,1,\lor,\land)$. Figure 1.1 shows an example of the BMM. Figures 1.1a and 1.1b show the operand matrices, and Figure 1.1c shows the product matrix. Representing the one-zero-matrices as black and white images gives a graphical interpretation on the operation: we have a one (a white pixel) in the product exactly when, in the corresponding row on the left-hand operand and the corresponding column on the right-hand operand, we have a one in the same location. BMM also has numerous applications, such as the computation of the transitive closure [194, 181], context-free parsing [439], and triangle detection and counting [259, 24]. Yet, despite the apparent simplicity, the problem is known to be difficult: even the best work is computationally demanding, and despite considerable effort, the exact bounds on the computational complexity of the problem remain elusive (see Section 3.7).

An interesting notion in algorithm design is the fact that we may employ algorithms for designing algorithms. We can use a computationally expensive algorithm to discover mathematical structures that enable the construction of fast algorithms for the problem at hand; that is, we use two algorithms where one algorithm gives the other algorithm as output. Once the fast algorithm has
Introduction

Figure 1.1. An example of the BMM. Figure (a) shows the left-hand operand, (b) the right-hand operand, and (c) the product. Ones are represented by white pixels and zeros by black pixels. We have a one at coordinates \((i,j)\) in the product if and only if there exists a location \(k\) on the \(i^{th}\) row of the left-hand operand and the \(j^{th}\) column of the right-hand operand, such that they are both one. In other words, if the value at coordinates \((i,k), (k,j)\) is equal to one in left and right-hand operands, respectively.

been discovered, it can be employed indefinitely and need not be rediscovered.

As a practical example, it has been known for a long time that the computation of bilinear maps can be completely determined by the decomposition of the structural tensor associated with the problem \([105]\). Using algorithmic methods to find such decompositions thus directly yields algorithms for the problems. These algorithms can be applied with indifference as to how the decomposition was found; as soon as the decomposition is found, the algorithm will follow, regardless of whether it was discovered by a human being or a machine. SAT solvers provide one possible tool for doing this; it is possible to encode the requirements for the equation system as a SAT instance, the solution for which tells us whether a desired decomposition exists. This is complicated by the high amount of symmetry of such instances which requires us to break the symmetries in order to make it feasible to solve larger instances.

Similarity search is a key application in contemporary artificial intelligence. Similarity search concerns itself with the finding of inexact matches between pairs of objects in a database. Applications of similarity search are too numerous to list exhaustively, but include web clustering \([109]\), face recognition \([229]\), computational biology \([171]\), computational chemistry \([451]\), recommender systems \([92, 389]\), and natural language processing (NLP) \([436]\).

Figure 1.2 shows an example of a particularly easy-to-access variant of similarity search: the light bulb problem, introduced by L. Valiant in 1988 \([440]\). We are given \(n\) light bulbs blinking uniformly and independently at random, except for a single planted pair whose blinking is correlated. We observe whether the light bulbs are on or off at \(d\) discrete points in time, and are asked to identify the unique correlated pair. In Figure 1.2a, the observations are represented as a binary matrix where each row corresponds to the observations of a single light bulb at different time-points. In Figure 1.2b, the unique correlated pair is highlighted. The question of scalability is how identify this pair without
resorting to comparing all pairs of light bulbs which, as was pointed out earlier, is prohibitively expensive in the presence of a large amount of light bulbs.

In addition to this introduction, this thesis consists of five publications. Publications I and II address the problem of finding correlated pairs of vectors in a high-dimensional space, a variant of similarity search. Publication III discusses the problem of breaking symmetries. Publications IV and V concern themselves with the properties of tensors and matrix multiplication in particular. Publications I, II, and IV are theoretical in nature, Publication V is mostly focused on experimental work, and Publication III is a mixture of both theoretical and experimental work.

In Publication I, we contribute to similarity search literature by improving on an earlier algorithm [438] of G. Valiant by presenting an improved randomized sampling scheme that exploits a Cartesian structure of the samples, and uses fast matrix multiplication for rapid sampling. The result was published in ACM Transactions on Algorithms, with a preliminary version of the paper having been published in Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms, SODA 2016.

In Publication II, we remove the randomization in the sampling of the same earlier algorithm of [438] by using expander graph constructions [388] for constructing a family of functions we call correlation amplifiers: explicit functions that work to amplify the relative gap of dissimilarity between similar and dissimilar objects. A preliminary version of the publication was published in the Proceedings of the 24th European Symposium of Algorithms, ESA 2016. This thesis contains an extended version that has been submitted for review.

In Publication III, we give a novel algorithm for breaking symmetries with respect to a prefix sequence of variables. An instantiation of McKay’s canonical

---

Figure 1.2. An example of the light bulb problem. Figure (a) shows a binary data matrix where each row corresponds to a unique light bulb. The columns correspond to different time points. Figure (b) highlights the correlated pair.
extension framework [328], the algorithm is applicable to any problem whose symmetries can be encoded in the form of a colored graph, and the algorithm is highly parallelizable. We show experimentally the effectiveness of the algorithm and its applicability to distributed processing in a computer cluster by providing an implementation that can be used as a preprocessor for SAT instances. The paper has been accepted for publication in the Journal of Symbolic Computation. A preliminary version of the paper was published in the Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing, SAT 2017.

In Publication IV, we define the notion of probabilistic rank of tensors, and use it to derive a randomized Strassen-like algorithm for BMM that improves asymptotically on Strassen’s algorithm with the same-sized base tensor, bypassing existing optimality results. The publication appeared in the Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019).

Finally, in Publication V, we provide implementations of the elementary $O(n^3)$ matrix multiplication over the binary field and the Boolean semiring, and of the binary Strassen’s multiplication for shared-memory, multiple-accelerator systems. We evaluate the performance of our implementations experimentally on tebibit-sized input. In particular, we employ a recent result [282] that shows that, while leaving the number of multiplications unaffected, the number of additions in Strassen’s algorithm can be reduced from what was previously thought to be optimal by employing a change of basis on the operands. Our results show that using Strassen’s algorithm can exceed the theoretical peak performance of the elementary cubic algorithm on our target systems at reasonable input sizes. The paper has been submitted for review.

The remainder of the introduction is arranged as follows. Chapter 2 presents mathematical preliminaries necessary for the remaining chapters. Chapter 3 presents an introduction to bilinear algorithms and how algorithms for them can be derived from the decompositions of the associated structural tensors, and presents a review in the most important results on fast matrix multiplication, including their feasibility in practice, and discusses how randomization can help with BMM. Chapter 4 discusses the discovery of exact low-rank decompositions of tensors, and how symmetry reduction can be of use in the process. Chapter 5 introduces us to the contemporary importance of parallelizability of algorithms, and presents some of our key results in this regard. Chapter 6 discusses similarity search, in high dimensions in particular, and how our results improve on previous research. Finally, Chapter 7 concludes this introduction of the thesis.
2. Preliminaries

In this chapter, we set up the notation and mathematical preliminaries we need throughout the remainder of this introduction of the thesis. Section 2.1 sets up fundamental notation and definitions. Section 2.2 presents the basic definitions of tensor algebra. Section 2.3 introduces us to groups and symmetry. Section 2.4 presents the notation for graphs that we need. Finally, Section 2.5 introduces us to Boolean logic and the satisfiability problem in particular.

2.1 Notation and definitions

For a positive integer \( n \), let \([n] = \{1, 2, \ldots, n\}\). Throughout this thesis, we move tacitly between mixed-base representations of indices when an index may have an interpretation requiring it to be split in sub-indices. That is, instead of \( \ell \in [nm] \), we may write \((i, j) \in [n] \times [m] \) for \( \ell = (i - 1)m + j \), and in general, for \( \ell \in [n_1n_2 \cdots n_N] \) we may write \((i_1, i_2, \ldots, i_N) \in [n_1] \times [n_2] \times \cdots \times [n_N] \) for

\[
\ell = 1 + \sum_{k=1}^{N} (i_k - 1) \prod_{j=k+1}^{N} n_j.
\]

We assume that \( \mathbb{F} \) is a field, and \( \mathbb{F}_2 \) is the binary field over \([0, 1]\) with the ordinary addition and multiplication modulo 2.

We use the big-O notation [285, Section 1.2.11]. For \( f, g : \mathbb{N} \rightarrow \mathbb{N} \), we say \( f \) is \( \Theta(g(n)) \) if there exist positive constants \( M \) and \( n_0 \) such that, for all \( n \geq n_0 \), we have \( f(n) \leq Mg(n) \). We use \( \tilde{\Theta}(g(n)) \) to suppress polylogarithmic factors by defining that \( f \) is \( \tilde{\Theta}(g(n)) \) if there exists \( c > 0 \) such that \( f \) is \( \Theta(g(n)(\log n)^c) \). Similarly, we use the little-o: we say that \( f \) is \( o(g(n)) \) if, for every \( c > 0 \), there exists an \( n_0 \) such that, for all \( n \geq n_0 \), \( f(n) \leq cg(n) \). Conversely, we say \( f \) is \( \Omega(g(n)) \) if \( g \) is \( \Theta(f(n)) \).

For a logical proposition \( P \), we use the Iverson bracket notation \([P]\) to indicate a 1 if \( P \) is true and 0 otherwise. We use the Kronecker delta \( \delta_{ij} \) to indicate a 1 if \( i = j \) and 0 otherwise.

We follow [45] when discussing computational problems. We say that a problem is a decision problem if the problem poses a question that can be answered by YES
Preliminaries

or NO. We say that a decision problem is in \textbf{NP} (Non-deterministic Polynomial time) if there exists a polynomial-time\(^1\) Turing machine that can determine if a proposed solution to a decision problem is correct (YES) or not (NO); that is, there exist certificates for the problem that can be efficiently verified (see \cite[Definition 2.1]{45}). We say that a problem is \textbf{NP}-hard if there exists a polynomial-time reduction from any problem in \textbf{NP} to the problem in question; if a problem is simultaneously \textbf{NP}-hard and in \textbf{NP}, we say it is \textbf{NP}-complete (see \cite[Definition 2.7]{45}).

2.2 Tensors

We follow the terminology of Kolda and Bader \cite{287}. We say that a tensor \(\mathcal{T}\) of order \(N\) over (a fixed but arbitrary) field \(\mathbb{F}\) is a multidimensional array whose elements are in \(\mathbb{F}\) and are indexed by \(N\)-tuples of integers. Although it is possible to regard tensors as basis-independent multilinear maps from vector spaces to a vector space, we shall assume a fixed but arbitrary basis for the tensor. When \(\mathbb{F}\) is \(\mathbb{F}_2\), we say the tensor (or matrix or vector) is \textit{binary}. We call the \(N\) dimensions of the tensor \textit{modes} of the tensor. For \(i = 1, 2, \ldots, N\), let \(n_i\) be the length of mode \(i\) of the tensor \(\mathcal{T}\). We then say that \(\mathcal{T}\) has \textit{shape} \(n_1 \times n_2 \times \cdots \times n_N\). We denote the set of all such tensors by \(\mathbb{F}^{n_1 \times n_2 \times \cdots \times n_N}\).

We say that tensors of order one are \textit{vectors} and of order two are \textit{matrices}. We denote scalars by roman lowercase letters, such as \(s\), vectors by bold lowercase letters, such as \(\mathbf{x}\), matrices by bold uppercase letters, such as \(\mathbf{A}\), and higher-order tensors by bold uppercase Euler script letters, such as \(\mathbf{T}\). We say that the vector space composed of vectors of shape \(n\) has dimension \(n\), and denote \(\mathbb{F}^n\). We denote the particular elements by corresponding lowercase letters with the indices in subscript, such as \(t_{ijk}\). We sometimes use the arrow notation \(\vec{i}\) when referring to the index of an arbitrary but fixed element \(t_{\vec{i}}\) of a fixed but otherwise unspecified tensor of possibly unspecified order \(\mathcal{T}\). For a vector \(\mathbf{x}\) of length \(n\), we also use \(\mathbf{x}_{\vec{i}}\) for \(\vec{i} \in [n]^r\) to denote a sample of \(s\) elements (with possible repetition).

Whenever we mix vectors and matrices, such as in the case of matrix-vector multiplication, we tacitly treat the vectors as \textit{column vectors} of shape \(n \times 1\) and explicitly mark the transpose when \textit{row vectors} are required. We denote the identity matrix (of implicit shape) by \(\mathbf{I}\), the transpose of a matrix \(\mathbf{A}\) by \(\mathbf{A}^\top\), and the inverse of a full-rank square matrix by \(\mathbf{A}^{-1}\), satisfying \(\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}\).

With order-three tensors, we may take \textit{slices} of a tensor along some mode, that is, we fix the value of the mode and take the corresponding submatrix; we use this to create representations third-order tensors by listing the slices. For an

\(^1\)Assuming the size of the input is \(n\), there exists a constant \(c > 0\) such that the number of steps required is \(O(n^c)\) as \(n \to \infty\), that is, bounded by a polynomial of constant degree.

\(^2\)Alternatively, we could formulate this as there being an algorithm that can be computed by a \textit{non-deterministic Turing machine} in polynomial time. See \cite[Theorem 2.6]{45}.
Let \( F^n, F^m, F^p \) be vector spaces over \( F \) of dimension \( n, m, p \), respectively. We say that \( f : F^n \to F^m \) is a \textit{linear map} if \( f(x + y) = f(x) + f(y) \) and \( f(cx) = cf(x) \) for all \( x, y \in F^n \) and \( c \in F \). If \( p = 1 \), that is, the map is to scalars, we say \( f \) is a \textit{linear form}. We denote the set of linear forms of \( F^n \) by \( F^n^* \), and say \( F^n^* \) is the \textit{dual space} of \( F^n \). If we treat \( F^n \) as column vectors, then \( F^{n*} \) consists of row vectors. We say that a function \( f : F^n \times F^m \to F^p \) is a \textit{bilinear map} if \( f(x, y) \) is a linear map on \( x \) when \( y \) is held constant, and vice versa. When \( p = 1 \), that is, when the map is to scalars, we say that \( f \) is a \textit{bilinear form}. Furthermore, we say that a map \( f : F^n \times F^m \times F^p \to F \) is a \textit{trilinear form}.

In what follows, we are particularly interested in tensors of order three. Let \( T \in F^{n_1 \times n_2 \times \ldots \times n_N} \) and \( S \in F^{m_1 \times m_2 \times \ldots \times m_N} \) be tensors. We say that \( V = T \otimes S \in F^{n_1 \times n_2 \times \ldots \times n_N \times m_1 \times m_2 \times \ldots \times m_N} \) is the \textit{Kronecker product} of \( T \) and \( S \) such that for all \( k = 1, 2, \ldots, N \) and \( t_k = 1, 2, \ldots, n_k \) and \( m_k = 1, 2, \ldots, m_k \), we have

\[
u_{i_1j_1 \cdots i_N} = t_{i_1j_1 \cdots i_N} s_{i_1j_1 \cdots i_N}.
\]

For a positive integer \( k \), we say that \( T^\otimes k \) is the \( k \)th \textit{Kronecker power} of \( T \), satisfying

\[T^\otimes k = T \otimes T \otimes \cdots \otimes T . \]

With matrices \( A \) and \( B \) of shapes \( n \times t \) and \( m \times t \), respectively, we may use \( C = A \otimes B \) for the \textit{partial Kronecker product} along the first mode, for a matrix of shape \( nm \times t \), such that for each \( i \in [n], j \in [m], k \in [t] \), it holds that \( c_{i,j,k} = a_{i,k} b_{j,k} \).

Assuming the shapes of tensors \( T \) and \( S \) coincide, we define the \textit{inner product} as the sum of element-wise products, that is,

\[
\langle T, S \rangle = \sum_{i_1 = 1}^{n_1} \sum_{i_2 = 1}^{n_2} \cdots \sum_{i_N = 1}^{n_N} t_{i_1i_2 \cdots i_N} s_{i_1i_2 \cdots i_N} .
\]

We define the \textit{Frobenius norm} of a tensor as \( \|T\|_F = \sqrt{\langle T, T \rangle} \). For a vector \( x \in \mathbb{R}^d \) and a real number \( p \geq 1 \), we say that \( \|x\|_p = \sqrt[p]{} \sum_{i=1}^{d} |x_i|^p \) is the \( L_p \)-norm and similarly say that the \( L_p \)-norm of the difference of two vectors \( \|x - y\|_p \) is the \( L_p \)-metric. For the Euclidean norm \( L_2 \), the 2 may be omitted, and we write simply \( \|x\|_2 = \|x\| = \|x\| \). For binary vectors \( x, y \in \{0, 1\}^d \) or \( x, y \in \{-1, 1\}^d \), we write \( d_H(x, y) = \sum_{i=1}^{d} |x_i - y_i| \) for the \textit{Hamming distance} between the vectors.

We define the \textit{Hadamard product} of two tensors whose shapes coincide simply as the elementwise product, such that for \( V = T \ast S \), we have

\[
u_{i_1j_1 \cdots i_N} = t_{i_1j_1 \cdots i_N} s_{i_1j_1 \cdots i_N} .
\]

Let \( x^{(1)}, x^{(2)}, \ldots, x^{(N)} \) be vectors of length \( n_1, n_2, \ldots, n_N \), respectively. We define the \textit{vector outer product} \( x^{(1)} \odot x^{(2)} \) as the \( n_1 \times n_2 \) matrix \( Z \) satisfying \( z_{ij} = x^{(1)}_i x^{(2)}_j \),
and, in general, \( \mathcal{T} = \mathbf{x}^{(1)} \circ \mathbf{x}^{(2)} \circ \cdots \circ \mathbf{x}^{(N)} \) such that \( \mathcal{T} \) has shape \( n_1 \times n_2 \times \cdots \times n_N \) and \( t_{i_1,i_2,\ldots,i_N} = x_{i_1}^{(1)} x_{i_2}^{(2)} \cdots x_{i_N}^{(N)} \).

For \( \ell = 1, 2, \ldots, N \), let \( \mathbf{A}^{(\ell)} \) be an \( n_{\ell} \times r \) matrix. Let \( \mathbf{a}^{(\ell)}_i \) be the \( i \)th column of \( \mathbf{A}^{(\ell)} \). We define the Kruskal product as the tensor \( \mathcal{T} = [\mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)}] \) of order \( N \) and shape \( n_1 \times n_2 \times \cdots \times n_N \) such that

\[
\mathcal{T} = \sum_{j=1}^{r} \mathbf{a}^{(1)}_j \circ \mathbf{a}^{(2)}_j \circ \cdots \circ \mathbf{a}^{(N)}_j. \tag{2.1}
\]

We say that a tensor \( \mathcal{T} \) of order \( N \) is a rank-one tensor if there exist \( N \) vectors \( \mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_N \) such that \( \mathcal{T} = \mathbf{a}_1 \circ \mathbf{a}_2 \circ \cdots \circ \mathbf{a}_N \). Any tensor may be written as a sum of rank-one tensors. In general, we say that a tensor \( \mathcal{T} \) has rank \( r \), or \( \text{rk} \mathcal{T} = r \), if \( r \) is the least number of rank-one tensors that need to be summed to obtain \( \mathcal{T} \). If we arrange the vectors as the columns of the matrices \( \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)} \) in the Kruskal product of Equation (2.1), the matrices directly give us a rank decomposition of the tensor. As an exception to our rule, we sometimes use the Greek letters \( \alpha, \beta, \gamma \) to denote the constituent matrices of the rank decomposition of a third-order tensor. Kronecker products and powers satisfy the submultiplicativity property

\[
\text{rk} \mathcal{T} \otimes \mathcal{S} \leq \text{rk} \mathcal{T} \cdot \text{rk} \mathcal{S}, \quad \text{rk} \mathcal{T}^\circ \mathbf{s} \leq (\text{rk} \mathcal{T})^p. \tag{2.2}
\]

Following [406], let us define the polynomial representation for a \( a \times b \times c \) tensor \( \mathcal{T} \) as a trilinear form \( t : \mathbb{F}^a \times \mathbb{F}^b \times \mathbb{F}^c \to \mathbb{F} \). We can treat the tensor as an element of the polynomial ring \( t \in \mathbb{F}[\mathbf{x}, \mathbf{y}, \mathbf{z}] = \mathbb{F}[x_1, x_2, \ldots, x_a, y_1, y_2, \ldots, y_b, z_1, z_2, \ldots, z_c] \) if we take the elements \( x_1, x_2, \ldots, x_a, y_1, y_2, \ldots, y_b, z_1, z_2, \ldots, z_c \) of the input vectors as the indeterminates. Define \( \mathbb{F}_c = \mathbb{F}[c] \) as the univariate polynomial ring with the indeterminate \( c \).

For \( \ell = 1, 2, \ldots, r \), let \( \alpha_\ell \in \mathbb{F}[\mathbf{x}], \beta_\ell \in \mathbb{F}[\mathbf{y}], \) and \( \gamma_\ell \in \mathbb{F}[\mathbf{z}] \) be polynomial representations of linear forms, and let \( h \) be a positive integer. Consider

\[
\sum_{\ell=1}^{r} \alpha_\ell(c) \beta_\ell(c) \gamma_\ell(c) = c^h t + \Theta(c^{h+1}), \tag{2.3}
\]

where \( \alpha_\ell(c) \in \mathbb{F}_c[\mathbf{x}], \beta_\ell(c) \in \mathbb{F}_c[\mathbf{y}], \) and \( \gamma_\ell(c) \in \mathbb{F}_c[\mathbf{z}] \) are polynomials of order at most \( h \), that is, \( \alpha_\ell(c) = \sum_{i=0}^{h} c^i a^i_\ell \) (and likewise for \( \beta_\ell(c) \) and \( \gamma_\ell(c) \)). The minimal \( r \) of this kind \( r_k(t) \) is the approximate rank of order \( h \), satisfying \( \text{rk} \mathcal{T} = r_0(t) \geq r_1(t) \geq \cdots \). The minimum of these numbers is the border rank of \( \mathcal{T} \), denoted \( \text{rk} \mathcal{T} \). Similarly, we get a border rank decomposition from Equation (2.3), by substituting the corresponding value of \( h \) from the minimal decomposition.

A third rank variety that we need is the support rank of Cohn and Umans [133]. We say that the support of a tensor is the set of the coordinates of its non-zero elements, denoted \( \text{supp} \mathcal{T} \). The support rank of a tensor is the minimum rank over the space of tensors sharing the same support. That is,

\[
\text{rk}_s \mathcal{T} = \min_{\mathcal{T} : \text{supp} \mathcal{T} = \text{supp} \mathcal{T}'} \text{rk} \mathcal{T}'. \tag{2.4}
\]
The support border rank $\text{rk}_s T$ is defined analogously by replacing $\text{rk} T'$ in Equation (2.4) with the border rank $\text{rk}_b T'$.

We sometimes need to vectorize matrices or higher-order tensors. In such case, unless otherwise noted, we use the generalized row-major order, that is, assuming $\mathbf{v}$ is the vectorized version of $\mathbf{T}$, we have

$$v_\ell = v_{(i_1,i_2,...,i_N)} = v_{1+\sum_{k=1}^N(i_k-1)\prod_{j=k+1}^N n_j} = t_{i_1i_2...i_N}.$$  

We say that $S$ is a subtensor of $\mathcal{T}$, or $S \subseteq \mathcal{T}$, if $S$ and $\mathcal{T}$ agree in shape and underlying field, and it holds that $s_{ij} = t_{ij}$, or $s_{ij} = 0$ for all elements of $S$.

Finally, we say that two tensors $\mathcal{T}$ and $\mathcal{S}$ are (permutation) isomorphic if we can obtain one from the other by permuting the modes or the coordinates along the modes, and denote this by $\mathcal{T} \cong \mathcal{S}^3$. Importantly, the tensor rank and border rank are invariant with respect to isomorphism.

### 2.3 Symmetry

We follow Dixon and Mortimer [163]. Let $\Gamma$ be a finite group, and let $\Omega$ be a finite set. We say that $\Gamma$ acts on the domain $\Omega$, and denote the group action by exponential notation such that, for any $\gamma \in \Gamma$ and $X \in \Omega$, $X^\gamma \in \Omega$ is the element obtained by acting on $X$ (from the right) with $\gamma$. By group axioms, for all $\gamma, \beta \in \Gamma$ and $X \in \Omega$, we have $X^{\gamma \beta} = (X^\gamma)^\beta$. We write $\Lambda \leq \Gamma$ to denote that $\Lambda$ is a subgroup of $\Gamma$. For a subset $W \subseteq \Omega$, we use the shorthand notation $W^\gamma = \{w^\gamma : w \in W\}$ to indicate the elementwise action of $\gamma$.

For a finite set $\Omega$, we write $\text{Sym}(\Omega)$ for the group of all permutations of $\Omega$ with the composition of mappings as the group operation. A permutation $\pi \in \text{Sym}(\Omega)$ is an $r$-cycle if, for $r$ distinct elements $x_1, x_2, \ldots, x_r \in \Omega$, $\pi$ maps $x_i$ onto $x_{i+1}$ for $i = 1, 2, \ldots, r-1$, and $x_r$ onto $x_1$, and leaves all other points fixed. We denote the cycle as $(x_1 x_2 \cdots x_r)$. We denote particular permutations as the product of disjoint cycles, where by disjoint we mean that no two cycles move a common element (see [163, Section 1.2]). If we can express $\Gamma \leq \text{Sym}(\Omega)$ as the closure of composition of a finite set of permutations $\pi_1, \pi_2, \ldots, \pi_n \in \text{Sym}(\Omega)$, we say that the permutations generate the group $\Gamma$, are the generators of $\Gamma$, and write $\Gamma = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$.

For an object $X \in \Omega$, we write $X^\Gamma = \{X^\gamma : \gamma \in \Gamma\}$ for the orbit of $X$ (under the action of $\Gamma$) and $\Gamma_X = \{\gamma \in \Gamma : X^\gamma = X\} \leq \Gamma$ for the stabilizer subgroup of $X$. We also say that $\Gamma_X$ is the automorphism group of $X$ and write $\text{Aut}(X)$ when $\Gamma$ may be inferred from the context.

We say that two objects are isomorphic if they are on the same orbit, that is, $X, Y \in \Omega$ are on the same orbit if and only if there exists an isomorphism $\gamma \in \Gamma$ that satisfies $Y = X^\gamma$, and write $X \cong Y$. If we want to stress or disambiguate the group in question, we write $X \cong_{\Gamma} Y$. We say that an isomorphism from an

---

3For matrices, this corresponds to the transpose and the permutation of rows or columns.
object to itself is an automorphism. Interchangeably, we say that two objects are symmetric if they are isomorphic.

Supposing $\Gamma$ acts on a set $U$, we extend the action of $\Gamma$ to the elementwise action of $\Gamma$ on the subsets of $W \subseteq U$ by $W^\gamma = \{w^\gamma : w \in W\}$ for all $\gamma \in \Gamma$ and $W \subseteq U$. For two subsets of equal cardinality $X, Y \subseteq U$ we write $X \cong Y$ if there exists $\gamma \in \Gamma$ such that $Y = X^\gamma$.

The wreath product group $\text{Sym}(R) \wr \text{Sym}(U)$ consists of pairs $(\pi, \sigma)$ where $\pi \in \text{Sym}(U)$ is a permutation of $U$ and $\sigma : U \to \text{Sym}(R)$ associates a permutation $\sigma(u) \in \text{Sym}(R)$ with each element $u \in U$. The group operation is defined as $(\pi, \sigma)(\pi_1, \sigma_1) = (\pi \pi_1, \sigma \sigma_1)$, where $\pi = \pi_1 \pi_2$ and $\sigma(u) = \sigma_1(u^{\pi_2^{-1}})\sigma_2(u)$. An element $(\pi, \sigma) \in \text{Sym}(R) \wr \text{Sym}(U)$ acts on an element $(u, r) \in U \times R$ by $(u, r)^{(\pi, \sigma)} = (u^{\pi}, r^{\sigma(u^r)})$.

### 2.4 Graphs

We follow Diestel [160]. A graph $G = (V, E)$ is defined over a set of vertices $V$ and edges $E \subseteq V \times V$. The graph is said to be undirected if for all $(u, v) \in E$, also $(v, u) \in E$; otherwise the graph is said to be directed. If the graph is undirected and $(u, v) \in E$, we say $u$ and $v$ are adjacent. If the graph is directed, for every edge $(u, v) \in E$, we say $u$ is an initial vertex, and $v$ is a terminal vertex.

For an undirected graph, we write $N(u)$ for the set of vertices that are adjacent to $u$ and call the set the neighbors of $u$. We say that the number of neighbors, or degree $u = |N(u)|$, is the degree of $u$. We say that an undirected graph $\Delta$-regular if every vertex $u \in V$ satisfies $\deg u = \Delta$.

If the graph is directed, we say that the number of edges whose terminal vertex is $u$ is the in-degree of $u$; likewise, we say that the number of edges whose initial vertex is $u$ is the out-degree of $u$.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. If $V' \subseteq V$ and $E' \subseteq E$, we say $G'$ is a subgraph of $G$, write $G' \subseteq G$, and say $G$ contains $G'$.

We say that a non-empty graph $P = (V, E)$ of the form $V = \{x_0, x_1, \ldots, x_k\}$ and $E = \{(x_0, x_1), (x_1, x_2), \ldots, (x_{k-1}, x_k)\}$ where all $x_i$ are distinct, is a path of length $k$, and write $P = x_0x_1 \cdots x_k$. We say that the vertices $x_0$ and $x_k$ are linked by $P$. If $k \geq 3$, and $P = x_0 \cdots x_{k-1}$ is a path, we say that the graph $C$ obtained by adding the edge $x_{k-1}x_0$ to $P$ is a $k$-cycle. If the graph contains no cycles, we say it is acyclic.

Let $C$ be a set of colors. A coloring$^4$ is a function $c : V \to C$ that assigns a color to each vertex. If we assign each vertex of the same color to a same subset and then consider the set composed of all such subsets, the coloring induces a partition of the vertices.

$^4$The term coloring is often used for proper coloring that is subject to the additional constraint that there should be no edges between two vertices of the same color, that is, for all $u \neq v \in V$, if $c(u) = c(v)$, then $(u, v) \notin E$. 

20
Preliminaries

Two (colored) graphs $G = (V, E_G)$ and $H = (V, E_H)$ over the same set of vertices are isomorphic with respect to the symmetric group of the vertices if there exists a permutation of vertices $\pi \in \text{Sym}(V)$ such that the permuted graph $H^\pi = (V, E_H^\pi)$ where $E_H^\pi = \{(u^\pi, v^\pi) : (u, v) \in E_H\}$ is equal to $G$, and the partitions induced by the colorings are equal. When we say two graphs are isomorphic, we denote this particular form of isomorphism.

2.5 Boolean logic

We follow [85]. Let $\phi$ and $\theta$ be Boolean formulae over the set of variables $X = \{x_1, x_2, \ldots, x_n\}$. We say that formulae $x_i$ and $\neg x_i$ are literals. We denote logical conjunction, disjunction, negation, exclusive disjunction, implication, and equivalence by $\phi \land \theta$, $\phi \lor \theta$, $\neg \phi$, $\phi \oplus \theta$, $\phi \Rightarrow \theta$, and $\phi \iff \theta$, respectively, and obtain more complex formulae by combining these.

We denote the Boolean values true by 1 and false by 0. We say that a mapping $v : X \to \{1, 0\}$ from the variables to the Boolean values is a truth assignment. We denote the evaluation of a formula $\phi$ simply by $v(\phi) \in \{1, 0\}$. If $v(\phi) = 1$, we say that $v$ satisfies $\phi$, we say that $\phi$ is satisfiable. This gives rise to the Boolean satisfiability problem (SAT) that asks if there exists a truth assignment that satisfies the given formula. SAT is NP-complete, and a fundamental problem in computational complexity theory.

We say that a disjunction of literals is a clause. If the formula $\phi$ consists of a conjunction of clauses, we say $\phi$ is in Conjunctive Normal Form (CNF). We say that any two clauses are equal if they consist of equal sets of literals. We say that two CNF formulae $\phi$ and $\theta$ are equal if they consist of equal sets of clauses. An arbitrary formula can be transformed into CNF by performing the Tseitin transformation [434]. Roughly, this process works by enumerating all of the sub-expressions within the formula, assigning a unique auxiliary variable to each of them, and then applying a set of rules to output clauses for each sub-expression and conjunct them with the substitution rules. This increases the size of the formula by at most a linear factor. Therefore, we may consider a more restrictive version of SAT, the CNF-SAT, which requires the input formula to be in CNF. We use SAT and CNF-SAT interchangeably. Finally, if we restrict the number of literals in a clause to be at most $k$, we call this special case $k$-SAT.
3. **Bilinear algorithms and fast matrix multiplication**

This chapter presents the concept of *bilinear algorithms*, or algorithms that compute bilinear maps, and gives a short introduction to the developments in FMM algorithms. Section 3.1 defines the basic concepts of bilinear maps, bilinear algorithms, structural tensors, and self-reducibility. Section 3.2 defines arithmetic circuits and shows how they can be used for the construction of algorithms, and how self-reducibility helps us construct recursive algorithms. Section 3.3 recalls the elementary, cubic algorithm for matrix multiplication. Section 3.4 introduces us to the notion of matrix multiplication tensor, and how it can be of use for the design of FMM algorithms. Section 3.5 introduces Strassen’s FMM algorithm, the first asymptotically subcubic algorithm. Section 3.6 gives an overview of the post-Strassen development in asymptotics of matrix multiplication. In Section 3.7, we discuss variants of matrix multiplication that have been considered in the literature, such as rectangular matrix multiplication, partial matrix multiplication, and BMM. In Section 3.8, we take a look at the possibility of performing matrix multiplication faster-than-usual using the idea of alternative-basis multiplication of Karstadt and Schwartz [282] and present some improvements. In Section 3.9, we present some basic notions of randomized algorithms, introduce the notions of probabilistic tensors and probabilistic tensor (border) rank, and show how randomization can be used in conjunction with matrix multiplication. Finally, in Section 3.10, we discuss the feasibility of FMM techniques, that is, whether they can be applied in practice.

### 3.1 Bilinear maps, structural tensors, and self-reducibility

In the Section 2.2, we defined linear maps and forms, bilinear maps and forms, and trilinear forms. We noted that linear forms correspond to transposed vectors, and the form can be evaluated by computing a vector-vector product, or an inner product between two vectors. In the case of linear maps, there exists a matrix that completely characterizes the map. For example, any linear map \( f : \mathbb{F}^n \to \mathbb{F}^m \) can be completely characterized by a matrix \( \mathbf{A} \) of shape \( n \times m \), and for a given input vector \( \mathbf{x} \), the value of the map can be computed by computing \( \mathbf{y} = \mathbf{A}\mathbf{x} \).
There is a connection between linear maps and bilinear forms. For every bilinear form \( f : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F} \), there exists a corresponding \( n \times m \) matrix \( A \) that completely characterizes the function. The value of the function for given \( x, y \) may then be computed by \( f(x, y) = x^\top A y \). In the case of arbitrary bilinear maps \( f : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^p \), we may treat the evaluation of the map for given arguments as that of evaluating \( p \) bilinear forms. Correspondingly, there exist \( p \) matrices of shape \( n \times m \) that completely define the map. This gives rise to the notion of the structural tensor which we may form by considering the matrices corresponding to the \( p \) bilinear forms as the slices of an order-3 tensor.

**Definition 3.1.** The structural tensor of a bilinear map \( f : \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^p \) is the third-order tensor \( T \) of shape \( n \times m \times p \) whose elements can be used to evaluate the value \( z = f(x, y) \), for all \( x \in \mathbb{F}^n \) and \( y \in \mathbb{F}^m \), by computing

\[
z_k = \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ijk} x_i y_j \quad (3.1)
\]

for all \( k = 1, 2, \ldots, p \). The tensor is unique up to choice of basis.

Assume that \( \alpha, \beta, \gamma \) are matrices of shape \( n \times r, m \times r, \) and \( r \times p \), respectively, that satisfy \( T = [\alpha, \beta, \gamma] \). We can then write Equation (3.1) in the following form:

\[
z_k = \sum_{i=1}^{n} \sum_{j=1}^{m} t_{ijk} x_i y_j = \sum_{i=1}^{n} x_i \left( \sum_{\ell=1}^{r} \alpha_{i\ell} \beta_{j\ell} \gamma_{k\ell} \right) = \sum_{\ell=1}^{r} \gamma_{k\ell} \left( \sum_{i=1}^{n} \alpha_{i\ell} x_i \right) \left( \sum_{j=1}^{m} \beta_{j\ell} y_j \right), \quad (3.2)
\]

for all \( k = 1, 2, \ldots, p \). We can subdue the index \( k \) by rephrasing the set of equations defined by Equation (3.2) in matrix form as

\[
z = \gamma((\alpha^\top x) * (\beta^\top y)). \quad (3.3)
\]

We essentially treat the columns of \( \alpha \) and \( \beta \) as linear forms in \( \mathbb{F}^{n*} \) and \( \mathbb{F}^{m*} \), respectively, to perform a transformation into a space of dimension \( r \) where we perform the elementwise multiplication, and then return to \( \mathbb{F}^{p} \) by performing another transformation, this time with the rows of \( \gamma \).

Furthermore, let us note that there is a similar connection between bilinear maps and trilinear forms as there is between linear maps and bilinear forms: a trilinear form \( f : \mathbb{F}^n \times \mathbb{F}^m \times \mathbb{F}^p \to \mathbb{F} \) is completely characterized by a tensor \( T \) of shape \( n \times m \times p \), and can be evaluated for given input vectors \( x, y, z \) by computing

\[
\sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{p} t_{ijk} x_i y_j z_k.
\]

Let \( r \) be a positive integer, and \( u_\ell \in \mathbb{F}^{n*}, v_\ell \in \mathbb{F}^{m*}, w_\ell \in \mathbb{F}^{p} \), for \( \ell = 1, 2, \ldots, r \). We say that a bilinear algorithm of length \( r \) is a sequence of triples \((u_1, v_1, w_1; u_2, v_2, w_2; \ldots; u_r, v_r, w_r)\)
that computes \( f(x, y) = \sum_{\ell=1}^{r'} (u_\ell x)(v_\ell y)w_\ell \) (see for example [105, Chapter 14]; see also [296, 419]). In other words, for a given \( x \) and \( y \), the algorithm evaluates \( f(x, y) \). Assuming \( r \) is minimum, we have that \( r = \text{rk} \mathcal{I} \). Furthermore, it can be shown that this \( r \) differs at most by a factor 2 from the number of multiplications required for computing the map, and that multiplications suffice; the use of division does not help improve the runtime (see [428] and [105, Chapter 14]). The study of complexity of bilinear algorithms is thus that of efficiently evaluating the tensor-decomposition constructs of Equation (3.3).

An important property exhibited by some bilinear maps, notably matrix multiplication, is self-reducibility. We say that a family of tensors (and the corresponding bilinear maps) is self-reducible if it is closed under Kronecker products. That is, the Kronecker product of two tensors from the family is also in the family. This allows us to compose larger instances from smaller base tensors. The importance of this property comes from the fact that self-reducibility enables us to construct recursive algorithms for the problem, using smaller base cases.

### 3.2 Arithmetic circuits

Here we repeat the definitions from Publication IV. We say that a circuit is a directed acyclic graph whose nodes have in-degree zero or in-degree two. We call the nodes of in-degree zero input gates, and the nodes of in-degree two arithmetic gates. We employ two types of arithmetic gates: addition and multiplication gates. The addition gates compute a linear combination of their input, that is, there are scalars \( \alpha, \beta \in F \) corresponding to the gate, and on input \( x, y \), the gate outputs the value \( z = \alpha x + \beta y \). We treat some gates as output gates whose value can be read or used as input in circuit compositions. Furthermore, we say that a circuit is a sum–product–sum circuit if every directed path from an input gate to an output gate contains exactly one multiplication gate.
In general, the arithmetic circuits are not unique: there may be several quite different circuits that compute the same map. For a practical example, consider the following linear map

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = Ax. \quad (3.4)$$

Figure 3.1 shows two possible arithmetic circuits that implement the linear map of Equation (3.4). While both circuits implement the same map, the circuit (a) is less efficient than circuit (b) which reuses intermediate results in its computation.

Consider an arbitrary but fixed bilinear map $f : \mathbb{F}^n \times \mathbb{F}^m \rightarrow \mathbb{F}^p$ whose structural tensor is $\mathcal{F} = [\alpha, \beta, \gamma]$, giving a bilinear algorithm of length $r$. It follows from Equation (3.3) that there exists a sum–product–sum arithmetic circuit that implements the bilinear algorithm, such as the one in Figure 3.2. Here we use the dotted box as a shorthand for a number of addition nodes that implement the matrix-vector multiplication in Equation (3.3).

We say that the arithmetic complexity of a bilinear map is the minimum number of gates or arithmetic operations that are required to construct an arithmetic circuit that implements the map. In general, the arithmetic complexity of a bilinear map may be dominated by the number of additions; however, if the base tensor of a self-reducible family of tensors is sufficiently sparse, the arithmetic complexity is dominated by multiplications. Then, due to the fact that tensor rank is submultiplicative under Kronecker products (Equation (2.2)), we have an upper bound on the number of multiplications required in larger cases.
Bilinear algorithms and fast matrix multiplication

The meaning of self-reducibility becomes apparent when we consider compositions of arithmetic circuits. Let \( f : \mathbb{F}^{n_1 \times m_1} \rightarrow \mathbb{F}^{p_1 \times p_2} \) be a bilinear map obtained by composition from bilinear maps \( f_1 : \mathbb{F}^{n_1 \times m_1} \rightarrow \mathbb{F}^{p_1 \times p_2} \) and \( f_2 : \mathbb{F}^{n_2 \times m_2} \rightarrow \mathbb{F}^{p_1 \times p_2} \) by forming the structural tensor of \( f \) as the Kronecker product of the respective structural tensors \( T_1, T_2 \). That is, \( T = T_1 \otimes T_2 = [a_1, \beta_1, \gamma_1] \otimes [a_2, \beta_2, \gamma_2] \). Furthermore, let us assume that the lengths of the bilinear algorithms corresponding to \( f_1 \) and \( f_2 \) are \( r_1 \) and \( r_2 \), respectively. We then get the arithmetic circuit of Figure 3.3, yielding a bilinear algorithm of length \( r_1 r_2 \). The composition preserves the sum–product–sum circuit form: there are multiplications only in the middle of the composite circuit, preserving the structure of the components. Furthermore, considering the input vector shapes as \( n_1 n_2 \) and \( m_1 m_2 \), we may regard the composition as \( r_1 \) instances of the second map, and we could, in fact, simply insert \( r_1 \) copies of the second map in the circuit of the first map to obtain the composition. This recursive structure can be applied an arbitrary number of times, so the construction gives a direct upper bound on multiplicative complexity, the minimum number of multiplication gates required to construct an arithmetic circuit implementing the map. Finally, it should be noted that this does not preclude the possibility that larger problem instances may exhibit regularity that is not caught by the lower-order decompositions; as such, it may be possible to construct more efficient algorithms by using a larger base case. In particular, this is the case with matrix multiplication.

3.3 The elementary matrix multiplication algorithm

Matrix multiplication is one of the most fundamental primitive operations in linear algebra. The classical definition of matrix multiplication, usually taught to students in high school or in the introductory mathematics classes in higher education, defines the product of an \( s \times t \) matrix \( A \) by the \( t \times u \) matrix \( B \) as the \( s \times u \) matrix \( C \) such that every element \( c_{ij} \) of \( C \) is the inner product between the \( i \)th row of \( A \) and the \( j \)th column of \( B \). In other words,

**Definition 3.2.** For positive integers \( s, t, u \), *matrix multiplication* is the bilinear map \( f : \mathbb{F}^{s \times t} \times \mathbb{F}^{t \times u} \rightarrow \mathbb{F}^{s \times u} \), the the value of which \( C = f(A, B) = AB \) satisfies for all \( i = 1, 2, \ldots, s \) and \( j = 1, 2, \ldots, u \),

\[
c_{ij} = \sum_{k=1}^{t} a_{ik} b_{kj}, \tag{3.5}
\]

A direct implementation of Equation (3.5) gives the elementary algorithm for matrix multiplication. For square matrices, with \( s = t = u = n \), the algorithm takes \( O(n^3) \) arithmetic operations. More precisely, the algorithm requires exactly \( n^3 \) multiplications and exactly \( n^3 - n^2 \) additions.
Bilinear algorithms and fast matrix multiplication

Figure 3.3. An arithmetic circuit for computing the bilinear map whose structural tensor is given by the composition $T = \langle \alpha_1, \beta_1, \gamma_1 \rangle \otimes \langle \alpha_2, \beta_2, \gamma_2 \rangle$. This circuit retains the sum–product–sum form. Furthermore, the outer part of the circuit directly implements to the left-hand tensor, while the inner part consists of $r_1$ copies of a circuit implementing the right-hand tensor.

### 3.4 The matrix multiplication tensor

It follows from Definition (3.2) that we may view the computation of the matrix product as that of evaluating $su$ bilinear forms on vectors of length $t$. Accordingly, we may define a structural tensor for the map, following Definition 3.1.

**Definition 3.3.** For positive integers $s, t, u$, the **structural tensor of matrix multiplication**, denoted $\langle s, t, u \rangle$, is the structural tensor tensor of the bilinear map $\mathbb{F}^{s \times t} \times \mathbb{F}^{t \times u} \rightarrow \mathbb{F}^{s \times u}$ computing the matrix multiplication of $s \times t$ matrix by a $t \times u$ matrix.

Letting $m_{(i, k), (j, k'), (i', j')} (i, j, k, k')$ denote an element of $\langle s, t, u \rangle$, with $i, i' \in [s]$, $j, j' \in [t]$, and $k, k' \in [u]$, the tensor has the form:

$$m_{(i, k), (j, k'), (i', j')} = \langle i = i' \rangle \langle j = j' \rangle \langle k = k' \rangle.$$  \hspace{1cm} (3.6)

For example, $(2, 2, 2)$ is the following tensor:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \ \end{bmatrix}.$$  \hspace{1cm} (3.7)
In Equation (3.7), the tensor is presented as slices along the third mode, each slice corresponding to an element of the output matrix \( C \). Also, we note that the Equations (3.6) and (3.7) assume a transposed second operand \( B \).

The following properties hold for any \( \langle a, b, c \rangle \) and \( \langle d, e, f \rangle \) (see [406] for more details). Larger tensors may be built from smaller tensors by taking the Kronecker product (up to isomorphism)

\[
\langle ad, be, cf \rangle \cong \langle a, b, c \rangle \otimes \langle d, e, f \rangle.
\]  

It follows directly from Equation (3.8) that \( \langle a, b, c \rangle^{\otimes p} \cong \langle a^p, b^p, c^p \rangle \). This, in turn, yields the symmetrization property that enables us to construct a square-matrix multiplication tensor from an arbitrary-shaped tensor \( \langle s, t, u \rangle \),

\[
\langle s, t, u \rangle \otimes \langle t, u, s \rangle \otimes \langle u, s, t \rangle \cong \langle stu, stu, stu \rangle.
\]  

We also have that the rank of \( \langle s, t, u \rangle \) is cyclically symmetric [240],

\[
rk \langle s, t, u \rangle = rk \langle t, u, s \rangle = rk \langle u, s, t \rangle.
\]  

Equation (3.10) implies together with Equations (2.2) and (3.9) that

\[
rk \langle stu, stu, stu \rangle \leq (rk \langle s, t, u \rangle)^3.
\]  

\[2.5\text{ Strassen’s algorithm}\]

Strassen was the first to describe a subcubic algorithm for matrix multiplication [427]. He used the \( 2 \times 2 \) square matrix multiplication as the base case to develop a recursive algorithm that achieves an upper bound of \( \mathcal{O}(n^{\log_2 7}) \approx \mathcal{O}(n^{2.81}) \) in the asymptotic number of arithmetic operations. In other words, Strassen was the first to provide a rank decomposition that showed that the rank of \( (2, 2, 2) \) was at most 7.

We now give a brief description of Strassen’s algorithm. Assume \( A \) and \( B \) are \( n \times n \) matrices such that \( n \) is a positive power of two. Consider the \( \frac{n}{2} \times \frac{n}{2} \) submatrices of \( A \) and \( B \),

\[
A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.
\]
Define the following auxiliary matrices:

\[
M_1 = (A_{11} + A_{22})(B_{11} + B_{22}), \\
M_2 = (A_{21} + A_{22})B_{11}, \\
M_3 = A_{11}(B_{12} - B_{22}), \\
M_4 = A_{22}(B_{21} - B_{11}), \\
M_5 = (A_{11} + A_{12})B_{22}, \\
M_6 = (A_{21} - A_{11})(B_{11} + B_{12}), \\
M_7 = (A_{12} - A_{22})(B_{21} + B_{22}).
\]

(3.12)

Then, we obtain the \( \frac{n}{2} \times \frac{n}{2} \) result submatrices as follows:

\[
C_{11} = M_1 + M_4 - M_5 + M_7, \\
C_{12} = M_2 + M_4, \\
C_{21} = M_3 + M_5, \\
C_{22} = M_1 + M_3 - M_2 + M_6.
\]

(3.13)

Arranging the submatrices as an \( n \times n \) matrix \( C \), we see that the procedure is equivalent to computing the matrix product with the elementary algorithm:

\[
C = \begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} = \begin{bmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{21}B_{11} + A_{22}B_{21} \\
A_{11}B_{12} + A_{12}B_{22} & A_{21}B_{12} + A_{22}B_{22}
\end{bmatrix}.
\]

In the case \( n = 2 \), computing the matrix product with Equations 3.12 and 3.13 reduces the number of scalar multiplications required from 8 to 7; however, at the cost of increasing the number of additions from 4 to 18. For operands of higher powers of two, the multiplications in Equation 3.12 may be processed recursively, using the same procedure. This directly yields that the grand total number of multiplications required is \( 7^d \) for operands of size \( n = 2^d \). The exact number of additions required \( A(n) \) can be computed with the following recurrence:

\[
A(1) = 0, \quad A(n) = 18 \cdot \left( \frac{n}{2} \right)^2 + 7 \cdot A \left( \frac{n}{2} \right),
\]

(3.14)

which can be solved in closed form as \( A(n) = 6 \cdot \left( 7^\log_2 n - n^2 \right) \) or

\[
A(2^d) = 6 \cdot \left( 7^d - 4^d \right).
\]

(3.15)

Winograd [459] improved on the algorithm by reducing the number of additions to 15. We present here the explicit equations we used for the Strassen-Winograd
Bilinear algorithms and fast matrix multiplication

variant of Strassen’s algorithm in Publication IV (see also [93, pp. 45–46]):

\[ T_1 = A_{21} + A_{22} \]
\[ T_2 = A_{12} \]
\[ T_3 = A_{12} + A_{22} \]
\[ T_4 = A_{21} + T_3 \]
\[ T_5 = A_{11} + T_4 \]
\[ T_6 = A_{21} \]
\[ T_7 = A_{11} \]
\[ S_1 = B_{21} + B_{22} \]
\[ S_2 = B_{21} \]
\[ S_3 = B_{12} - B_{22} \]
\[ S_4 = B_{21} - S_3 \]
\[ S_5 = B_{12} \]
\[ S_6 = B_{11} - S_4 \]
\[ S_7 = B_{11} \]
\[ Q_1 = T_1 \cdot S_1 \]
\[ Q_2 = T_2 \cdot S_2 \]
\[ Q_3 = T_3 \cdot S_3 \]
\[ Q_4 = T_4 \cdot S_4 \]
\[ Q_5 = T_5 \cdot S_5 \]
\[ Q_6 = T_6 \cdot S_6 \]
\[ Q_7 = T_7 \cdot S_7 \]
\[ U_1 = Q_2 - Q_4 \]
\[ U_2 = U_1 - Q_3 \]
\[ U_3 = Q_3 - U_1 \]
\[ Q_5 \]
\[ Q_6 = T_6 \cdot S_6 \]
\[ Q_7 = T_7 \cdot S_7 \]
\[ C_{11} = Q_2 + Q_7 \]
\[ C_{12} = U_3 - Q_7 \]
\[ C_{21} = Q_2 + Q_7 \]
\[ C_{22} = U_3 - Q_7 \]

In the Strassen-Winograd case, the total number of multiplications is unchanged at \( 7^d \) for operands of size \( n = 2^d \). However, the Recurrence (3.15) changes to

\[ A(1) = 0, \quad A(n) = 15 \cdot \left( \frac{n}{2} \right)^2 + 7 \cdot A \left( \frac{n}{2} \right), \]

which implies a closed form analogue of Equation (3.15) of

\[ A(2^d) = 5 \cdot (7^d - 4^d). \]

It is known that the Strassen-Wingograd algorithm is optimal for computing a \( 2 \times 2 \) matrix product in both the number of multiplications [239, 459] and the number of additions [373, 103] if we operate in the standard basis (see Section 3.7).

3.6 Asymptotics of Fast Matrix Multiplication

In this section, we consider the upper bound of the runtime exponent of square matrix multiplication \( \omega = \omega(F) \). Assuming the operands are of shape \( n \times n \) and the underlying field is \( F \), we define \( \omega \) as follows (see, for example, [406, Equation 0.2] or [138]).

**Definition 3.4.** Let \( \omega_n(F) \) denote the base-\( n \) logarithm of the upper bound on the number of multiplications sufficient\(^1\) for computing an \( n \times n \) matrix product over the field \( F \). The exponent of square matrix multiplication is the value \( \omega = \omega(F) \) that satisfies

\[ \omega(F) = \inf \{ \omega_n(F) \mid n \geq 2 \}. \] (3.18)

That is, \( \omega \) states an asymptotic upper bound on the number of multiplications; that the matrix product can be computed using \( n^{\omega+o(1)} \) multiplications as \( n \) tends to infinity. There is a connection between \( \omega \) and \( \text{rk} \langle n, n, n \rangle \).

**Lemma 3.5** ([105, Proposition 15.1]). The value of \( \omega \) is determined by the rank of \( \langle n, n, n \rangle \), as

\[ \omega = \inf \{ \beta \mid \text{rk} \langle n, n, n \rangle = \Theta(n^\beta) \}. \]

\(^1\)That is, \( n^{\omega_n(F)} \) multiplications suffice.
Hence, Lemma 3.5 implies that the search for the existence of FMM algorithms reduces to that of finding upper bounds on the rank of the matrix multiplication tensor.

We have from Equation (3.11) and the Strassen framework described in Section 3.5 that a non-trivial upper bound for \( \mathrm{rk}(s, t, u) \) for fixed \( s, t, u \) directly yields an upper bound \( \omega \leq \frac{1}{\log(\omega)} \left( \frac{1}{\log(\mathrm{rk}(s, t, u))} \right) \), and an algorithm that multiplies \( n \times n \) matrices over \( \mathbb{F} \) in \( O \left( n \log \left( \frac{stu}{\omega} \right) \right) \) operations.

As explained in Section 3.5, Strassen [427] was the first to discover that \( \mathrm{rk}(2, 2, 2) \leq 7 \), implying \( \omega < 2.8074 \). It is now known that this bound on \( \mathrm{rk}(2, 2, 2) \) is tight [239, 459] and holds with equality. It is also known that the border rank [295] and support rank [89] of \( (2, 2, 2) \) equal exactly 7, that is, \( \mathrm{rk}(2, 2, 2) = 7 \), \( \mathrm{rks}(2, 2, 2) = 7 \) over any field, and \( \mathrm{rk}_s(2, 2, 2) = 7 \) over \( \mathbb{C} \). This means that the Strassen-Winograd algorithm is optimal\(^2\) for multiplying \( 2 \times 2 \) matrices over fields, and hence, faster algorithms require a larger base tensor.

Pan’s [363] technique of trilinear aggregation, built around a rank-143,640 decomposition of \( \langle 70, 70, 70 \rangle \), was the next step, reducing the upper bound to \( \omega < 2.7962 \). Pan’s refinement [365] on his techniques yielded an algorithm with an upper bound of \( \omega < 2.7734 \), for a base case of \( \langle 44, 44, 44 \rangle \).

Bini showed [86] that the existence of Arbitrary Precision Approximate (APA) algorithms implies the existence of exact algorithms of similar complexity. This makes the border rank of the matrix multiplication tensor an object of investigation, as such algorithms can be constructed using border rank decompositions. This line of research [406, 364, 393, 138] yielded an upper bound of \( \omega < 2.495548 \).

Finally, the foundations of the asymptotically-fastest currently known methods were laid by Strassen [429] with the introduction of the laser method which led to the result \( \omega < 2.376 \) of Coppersmith and Winograd [139]. This bound stood for two decades, until it was improved in a sequence of papers [149, 441], with the current record of \( \omega < 2.3728639 \) due to Le Gall [301].

In [132], Cohn and Umans present an alternative framework for FMM. They identify finite groups \( G \) such that matrix multiplication can be embedded into the group algebra \( \mathbb{C}[G] \) where the multiplication can be performed pointwise by a group-theoretic generalization of the FFT. In [131], Cohn, Kleinberg, Szegedy, and Umans apply the group-theoretic framework to groups with wreath product structure such that they achieve a running times of \( O(n^{2.48}) \) and \( O(n^{2.41}) \), which match earlier upper bounds proved by Strassen [429] and Coppersmith and Winograd [139] using different techniques, and show that even the \( \omega < 2.376 \) result of Coppersmith and Winograd [139] has a group-theoretic analogue. In [133], Cohn and Umans present a further generalization of their framework, along with the notion of the tensor support rank.

So far, there are no known results that would preclude the possibility that \( \omega \) might be 2, and this has indeed been conjectured by Coppersmith and Winograd\(^2\) in the sense that if we limit ourselves to a straight-line program that may only compute additions or multiplications of its previous operations; we bypass this in Publication IV with the help of randomization.
grad [139] and Cohn, Kleinberg, Szegedy, and Umans [131]. The best known lower bound for the rank of the matrix multiplication tensor is \( \text{rk}(n, n, n) \geq 3n^2 - 2\sqrt{n^7} - 3n \) due to Massarenti and Raviolo [321]. It is known, however, that current methods are unlikely to yield a quadratic result. In [23], Alon, Shpilka, and Umans showed that if the Erdős-Rado Sunflower Conjecture [173] is true, then Coppersmith and Winograd’s [139] formulation of the \( \omega = 2 \) conjecture fails.

The recent advances in the asymptotics [149, 441, 301] are based on analysis of higher powers of the Coppersmith-Winograd tensor [139]. However, Ambainis, Filmus, and Le Gall [30] showed that higher powers can do only so much; that no such algorithm can achieve runtime of \( O(n^{2.3078}) \).

Blasiak, Church, Cohn, Grochow, and Umans [90] showed that the Cohn-Umans [132] framework cannot achieve \( \omega = 2 \) with any finite abelian group with a bounded exponent. These results were extended to certain classes of non-abelian groups by Sawin [402] and Blasiak, Church, Cohn, Grochow, and Umans [91].

In [17], Alman and Vassilevska Williams provide a unifying framework, generalizing Strassen’s [429] Laser method, using the structural tensor of addition modulo integer \( q \), of which the Coppersmith-Winograd tensor is a \textit{monomial degradation}. They show that, within this framework, \( \omega = 2 \) cannot be proven for any fixed \( q \).

In [20], Alman and Vassilevska Williams present three general methods for analyzing such tensor constructions which they call, in the order of increasing generality, the Solar method which subsumes the laser method and the Cohn-Umans group-theoretic method, the Galactic method which considers monomial degenerations in general, and the Universal method which considers degenerations in general. They then show that the generalization of the Coppersmith-Winograd tensor cannot be used to prove \( \omega = 2 \), and present limitations on the group-theoretic approaches.

Further information regarding these fastest techniques, including the unified tensor-framework, may be found in the recent paper of Alman and Vassilevska Williams [20]. Pan [366] provides a more detailed survey on the history of the developments in FMM and the feasibility of these techniques.

### 3.7 Variants of matrix multiplication

In this section, we briefly review some variants of matrix multiplication. So far, we have mainly discussed square matrix multiplication, so the natural extension is to discuss \textit{rectangular matrix multiplication}, that is, allow the operands to be non-square. The natural counterpart of \( \omega \) in rectangular matrix multiplication is the constant \( \alpha \) that states that an \( n \times n^\alpha \) matrix can be multiplied by an \( n^\alpha \times n \) matrix using \( n^{2+o(1)} \) operations (see, for example, [300]). We note that \( \omega = 2 \) would imply \( \alpha = 1 \) and vice versa. The best known lower bound on \( \alpha \) is \( \alpha > 0.31389 \), due to Le Gall and Urrutia [302].
Another important variant is the APA matrix multiplication, the algorithms for which can be constructed from border-rank decompositions. Bini showed \cite{86} that such algorithms imply the existence of exact algorithms. The idea of arbitrary-precision approximation yielded an algorithm \cite{87} for computing the product of $2 \times 2$ matrices of form

$$AB = \begin{bmatrix} a & b \\ 0 & c \end{bmatrix}$$

with only 5 multiplications, based on a border-rank-5 decomposition of the associated partial matrix multiplication tensor.

Schönhage considered an alternative notion of partial matrix multiplication by allowing some of the elements of the operands to be zero \cite{406}. He used this as a technique for constructing FMM algorithm for arbitrary matrices.

In contrast to Bini’s and Schönhage’s results, we consider a more fine-grained variety of partial matrix multiplication in Publication IV. Instead of manipulating the elements of the product or the operands, we zero-out elements of the matrix multiplication tensor itself, probabilistically. For example, zeroing out any non-zero element of $(2,2,2)$ yields a rank-6 tensor. When computing an approximation of $C = AB$ with such a structural tensor, one of the product elements of the form $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j}$ lacks one of the summands. We show that, despite this, we can get a randomized algorithm that approximates the product sufficiently well, and is asymptotically faster than Strassen’s algorithm.

The FMM schemes, such as Strassen’s algorithm, assume that the algebra over which the matrices are defined is a ring. Yet another important variety of matrix multiplication problems is the BMM, or matrix multiplication over the Boolean algebra $B = (0,1, \lor, \land)$ which is a semiring. This makes techniques requiring additive cancellation inapplicable as such. However, it is possible to reduce the problem of Boolean multiplication to that of integer multiplication \cite{181}. There are also combinatorial algorithms for BMM; see the famous Four Russians algorithm \cite{44}, and for more recent work, for example, \cite{405, 309, 63}. In \cite{22, 405}, algorithms are presented for computing witnesses of the Boolean product, that is, for the product $C = AB$ over the Boolean algebra, the algorithm finds an index $k$ for every $c_{ij}$ such that $a_{ik} = b_{kj} = 1$.

The notion of support rank of Cohn and Umans \cite{133} may be used to establish that BMM can be performed in $n^{\omega_s + o(1)}$ operations where $\omega_s$ is the support-rank exponent satisfying $2 \leq \omega_s \leq \omega < 2.3728639$ and $\omega \leq \frac{2}{3} \omega_s - 1$, leaving open the possibility that matrix multiplication over a general field $\mathbb{F}$ has strictly higher asymptotic complexity than BMM.

BMM has some very important applications, such as the computation of the transitive closure \cite{194, 181}, context-free parsing \cite{439}, and triangle detection and counting \cite{259, 24}. Triangle detection is of particular importance,
since it has been shown to be equivalent to BMM under the notion of subcubic equivalences [458], and has many important applications (see the overview of applications in [435]; also see other work on triangle detection and counting [374, 88, 104, 326, 168]; see also the review [225]).

We have so far discussed only dense matrix-matrix multiplication and continue to do so. It is, however, important to note the existence of sparse matrix multiplication algorithms, and matrix-vector multiplication algorithms.

A matrix is sparse if most of its elements are zero. Instead of a continuous array, a sparse matrix is often more convenient to represent by listing the coordinates of its non-zero elements. The runtime of sparse matrix multiplication is generally dependent on the number of nonzero elements. For literature on fast sparse matrix multiplication, see Yuster and Zwick [465]; for the Boolean sparse matrix multiplication, see [32, 172]; and for distributed sparse matrix multiplication, see [57, 51, 61, 62].

In some applications, it is interesting to compute numerous matrix-vector products, possibly by fixing the matrix. That is, given an $n \times n$ matrix $M$, we may (possibly in an online setting) receive a linear number of $n$-vectors $x_1, x_2, \ldots, x_n$, and we need to output the vectors $Mx_1, Mx_2, \ldots, Mx_n$. Henzinger, Krinninger, Nanongkai, and Saranurak [232] conjecture that, under the Boolean algebra, there is no (randomized) truly subcubic\(^3\) algorithm for solving the problem. For arbitrary (and sufficiently large) fields, it is known that polynomial-space data structures necessarily require $\Omega(n^2/\log n)$ time per vector [127]. For literature on matrix-vector multiplication, see, for example, [453, 299, 113] in the context of online Boolean matrix-vector multiplication. In particular, parallel sparse matrix-vector multiplication has been addressed in the literature, especially in practical engineering work, see for example [71, 180, 241].

Finally, we conclude this section with some interesting related work. In [358], Pagh presented an interesting variant to the usual matrix multiplication, the compressed matrix multiplication, a randomized algorithm that uses 2-wise independent hash functions and FFT to compute an estimate of the matrix multiplication. Assuming the Frobenius norm of the product is sufficiently small, one can compute the matrix product exactly in $\tilde{O}(N + nb)$ time where $N$ is the number of nonzero entries in the operand matrices, and $b$ the number of nonzero entries in the product matrix, with high probability. The verification and correction of matrix product entries is also important, since we know that hardware failures cause nonnegligible amounts of erroneous computation [98]. Work in this field was pioneered by Freivalds [188]. In [200], Gasieniec, Levkopoulou, Lingas, Pagh, and Tokuyama present an algorithm for correcting up to $k$ erroneous entries in the product in time $\tilde{O}(n^2 + kn)$.

\(^3\)That is, an algorithm running in time $\mathcal{O}(n^{3-c})$ for some $c > 0$. 

35
3.8 Alternative-basis matrix multiplication

Although the Strassen-Winograd algorithm of Equations (3.16) is known to be optimal for computing the $2 \times 2$ matrix product, this optimality result only holds with respect to the *standard basis*. In 2017, Karstadt and Schwartz [282] showed that the Strassen-Winograd algorithm can be performed with only 12 additions by performing a change of basis on the data, thus bypassing Probert’s [373] and Bshouty’s [103] lower bounds. One useful feature of the result lies in the fact that, since the mapping that changes the basis is an involution, the change of basis needs to be done only once, after which an arbitrary number of matrix multiplications may be performed in series. Also, while the change of basis incurs additional cost, the underlying computational hardware may still be able to benefit from the lower arithmetic burden of the actual multiplication routine.

We employ a variant of the alternative basis result in Publication V. We present two particular basis changes that have either a *self-inversion* property or a *chaining* property. We start by describing the self-inversion variant. Let

\[
A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}
\]

be the left and right hand block matrices in the standard basis, respectively. The alternative basis with self-inversion is obtained by computing (recursively)

\[
\begin{align*}
A_{11}' &= A_{11}, & B_{11}' &= B_{11}, \\
A_{12}' &= A_{12}, & B_{12}' &= B_{12}, \\
A_{21}' &= A_{21}, & B_{21}' &= B_{21}, \\
A_{22}' &= A_{12} + A_{21} + A_{22}, & B_{22}' &= B_{21} - B_{12} + B_{22}.
\end{align*}
\]

The inverse transformation is obtained by

\[
\begin{align*}
C_{11} &= C_{11}', \\
C_{12} &= C_{22}' - C_{12}', \\
C_{21} &= C_{21}' + C_{22}', \\
C_{22} &= -C_{22}'.
\end{align*}
\]
In this basis, the Strassen-Winograd equations from Equation 3.16 become

\[
\begin{align*}
T'_1 &= A'_{11}, & S'_1 &= B'_{11}, \\
T'_2 &= A'_{12}, & S'_2 &= B'_{21}, \\
T'_3 &= A'_{21}, & S'_3 &= B'_{11} - B'_{22}, \\
T'_4 &= A'_{22}, & S'_4 &= B'_{22}, \\
T'_5 &= A'_{11} + A'_{22}, & S'_5 &= B'_{12}, \\
T'_6 &= A'_{22} - A'_{12}, & S'_6 &= B'_{12} + B'_{22}, \\
T'_7 &= A'_{22} - A'_{21}, & S'_7 &= B'_{21} - B'_{22}, \\
Q'_1 &= T'_1 \cdot S'_1 & C'_{11} &= Q'_1 + Q'_2, \\
Q'_2 &= T'_2 \cdot S'_2 & C'_{12} &= Q'_2 - Q'_5, \\
Q'_3 &= T'_3 \cdot S'_3 & C'_{21} &= Q'_3 + Q'_6, \\
Q'_4 &= T'_4 \cdot S'_4 & C'_{22} &= Q'_4 + Q'_7 - Q'_2 - Q'_6, \\
Q'_5 &= T'_5 \cdot S'_5, \\
Q'_6 &= T'_6 \cdot S'_6, \\
Q'_7 &= T'_7 \cdot S'_7.
\end{align*}
\]

This reduces the number of additions required to 12, which yields a tally of

\[
A(2^d) = 4 \cdot (7^d - 4^d)
\]

additions (cf. Equation (3.17)).

It follows directly that applying the basis transformation of Equation (3.19) on \(A\) and \(B\), computing \(C'\) according Equation (3.21), and performing the inverse transformation of Equation (3.20) to obtain \(C\), we get

\[
C = \begin{bmatrix}
C'_{11} & C'_{12} \\
C'_{21} & C'_{22}
\end{bmatrix}
= \begin{bmatrix}
A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22}
\end{bmatrix}.
\]

An alternative way to represent the operations carried out by the basis change is to consider the operation as a bilinear map. The system of equations then determine matrices that characterize the maps. If we vectorize the input matrices, we can efficiently evaluate operand matrices whose side lengths are higher powers of the base length by Kroneckering-up the matrices, and applying Yates’ algorithm [462]. The details of this procedure are presented in Publication V; here we only present an abstract, high-level description.

Let us denote the forward-transformation matrices for the left and right hand operand by \(\psi\) and \(\chi\), respectively. Denote the matrices corresponding to the sum equations of the left and right hand operands by \(\alpha\) and \(\beta\), respectively. Then, denote the aggregation equations that collect sums of the products for output by \(\gamma\). Finally, denote the inverse basis transformation by \(\phi\). In the case of \(2 \times 2\) square matrix multiplication, we have that \(\psi, \chi,\) and \(\phi\) have shape \(4 \times 4\), \(\alpha\) and \(\beta\) have shape \(7 \times 4\), and \(\gamma\) has shape \(4 \times 7\), assuming Strassen-like rank-7.
Bilinear algorithms and fast matrix multiplication

decomposition is used. Then, the vectorized computation of operands $\mathbf{x}$ and $\mathbf{y}$ becomes

$$
\phi \gamma((\alpha \psi \mathbf{x}) * (\beta \chi \mathbf{y})).
$$

The transformation matrices corresponding Equations (3.19), (3.20), and (3.21) are

$$
\psi = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1
\end{bmatrix}, \quad (3.23)
$$

$$
\chi = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & -1 & 1 & 1
\end{bmatrix}, \quad (3.24)
$$

$$
\phi = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & -1 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & -1
\end{bmatrix}, \quad (3.25)
$$

$$
\alpha = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}, \quad (3.26)
$$

$$
\beta = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & -1 \\
0 & 0 & 0 & 1
\end{bmatrix}, \quad \text{and} \quad (3.27)
$$
Bilinear algorithms and fast matrix multiplication

\[
\gamma = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 1 & 0 & -1 & 1
\end{bmatrix}.
\] (3.28)

In our implementation, presented in Publication V, we perform the operations over \( \mathbb{F}_2 \), meaning that no signs were present, so Equations (3.23) and (3.24) reduce to \( \psi = \chi \), yielding identical transformations for the left and right hand operands. Importantly, due to the structure of the transformations, the transformations can be computed in-place with only a part of the data needing to be moved in memory. Moreover, we have from Equation (3.23) and (3.25) that, over \( \mathbb{F}_2 \), \( \psi = \psi^{-1} \) and \( \phi = \phi^{-1} \). However, this form precludes us from chaining the multiplications in the alternative basis since the transformations are not inverses of each other. Due to this, we also provided another basis change in Publication V.

Let us denote the transformed operands in an alternative basis that allows for chaining by two primes, and the corresponding transformation matrices by single primes. We get the following transformation:

\[
\begin{align*}
A_{11}'' &= A_{11} & B_{11}'' &= B_{11}, \\
A_{12}'' &= A_{12} & B_{12}'' &= B_{12}, \\
A_{21}'' &= A_{12} + A_{21} + A_{22} & B_{21}'' &= B_{21} + B_{22} - B_{12}, \\
A_{22}'' &= A_{12} + A_{22} & B_{22}'' &= B_{22} - B_{12}.
\end{align*}
\] (3.29)

The inverse transformation is obtained by

\[
\begin{align*}
C_{11} &= C_{11}'', \\
C_{12} &= C_{12}'', \\
C_{21} &= C_{21}'' - C_{22}'', \\
C_{22} &= C_{22}'' - C_{12}''.
\end{align*}
\] (3.30)
In this basis, the Strassen-Winograd equations from Equation 3.16 become

\[
\begin{align*}
T''_1 & = A''_{11} \quad S''_1 = B''_{11}, \\
T''_2 & = A''_{12} \quad S''_2 = B''_{22} - B''_{21}, \\
T''_3 & = A''_{21} \quad S''_3 = B''_{21}, \\
T''_4 & = A''_{22} \quad S''_4 = B''_{22}, \\
T''_5 & = A''_{11} + A''_{21} \quad S''_5 = B''_{12}, \\
T''_6 & = A''_{12} - A''_{21} \quad S''_6 = B''_{12} + B''_{21}, \\
T''_7 & = A''_{22} - A''_{21} \quad S''_7 = B''_{21} - B''_{11}, \\
Q''_1 & = T''_1 \cdot S''_1 \quad R = Q''_2 + Q''_3 + Q''_5, \\
Q''_2 & = T''_2 \cdot S''_2 \quad C''_{11} = Q''_1 - Q''_2, \\
Q''_3 & = T''_3 \cdot S''_3 \quad C''_{12} = R + Q''_5, \\
Q''_4 & = T''_4 \cdot S''_4 \quad C''_{21} = R + Q''_7, \\
Q''_5 & = T''_5 \cdot S''_5 \quad C''_{22} = Q''_4 + Q''_5, \\
Q''_6 & = T''_6 \cdot S''_6, \\
Q''_7 & = T''_7 \cdot S''_7,
\end{align*}
\]

(3.31)

Applying the transformation of Equation 3.29, then computing the products according to Equation (3.31), and applying the inverse transformation of Equation 3.30 results through a direct computation as \( C = AB \). In matrix form, the equations can be defined as

\[
\begin{align*}
\psi' & = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \\
\chi' & = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 1 \\ 0 & -1 & 0 & 1 \end{bmatrix}, \\
\phi' & = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & -1 & 0 & 1 \end{bmatrix},
\end{align*}
\]

(3.32)

(3.33)

(3.34)
We observe that over $\mathbb{F}_2$, we have from Equations (3.32), (3.33) and (3.34) that $\psi = \chi = \phi^{-1}$, thus enabling chaining. Also, Equation (3.31) shows that the number of additions remains at 12. However, the cost is at the weight distribution of the matrices: comparing Equations (3.26), (3.27), and (3.28) to Equations (3.35), (3.36), and (3.37), we notice that the latter contain more non-zero entries. The runtime implications are discussed and verified empirically in Publication V.

The Equations (3.19), (3.20), (3.21), (3.29), (3.30), and (3.31) were derived over $\mathbb{F}_2$ using a SAT solver to minimize the length of a straight-line program computing the product, modeled as a Boolean circuit. For generality, the signs of the equations and the corresponding matrices were determined by a brute-force search.

### 3.9 Randomized algorithms and Boolean matrix multiplication using probabilistic tensors

We say that a randomized algorithm is any algorithm that makes use of a source of independently and uniformly sampled random bits, separate from the input, to make choices during the execution of the algorithm (for a very brief introduction to the topic, see [340] and the references therein; for textbooks, see [339, 335]).
Hence, the behavior of the algorithm may vary between consecutive runs. We say that an algorithm that does not depend on random bits is a deterministic algorithm.

Randomized algorithms that are guaranteed to always give the correct solution, but with runtime possibly dependent on the random bits, are called las Vegas algorithms. If erroneous output is allowed at a bounded probability, we call the algorithm a Monte Carlo algorithm. The probability of error in Monte Carlo algorithms can be made arbitrarily small by independent repetitions. Randomized algorithms can be used to obtain better performance than is known to be possible with the best known deterministic algorithms. Early successful examples of randomization include the Metropolis-Hastings algorithm [333, 227], the factoring of polynomials over finite fields [80], randomized selection [183], the verification of matrix products [188], hashing [110], primality testing [422, 423, 382], and pattern matching [280].

As was presented in Section 3.7, a concept related to BMM is that of witnesses of the Boolean product. Indeed, whenever we encounter a $c_{ij} = 1$ in the Boolean product, we know that there must be an index $k$, such that $a_{ik} = b_{kj} = 1$. We also know that if there is even one such witness, then we invariably have $c_{ij} = 1$, even if there are more such witnesses, so BMM is monotone in this sense: once we know during the computation there will be a one in the product, it can never become zero again. This suggests a question about whether we could use randomization to compute some of the ones more efficiently than than every one of them.

In Publication IV, we answer the question affirmatively by presenting a novel way of constructing randomized algorithms by the means of probabilistic tensors and, especially, the probabilistic extensions of tensor rank and border rank. The notion of probabilistic tensors is related to the notion of probabilistic polynomials [454, 1, 19], and Alman and Williams’ [18] notion of probabilistic matrices and probabilistic matrix rank. We give the definitions and important properties of probabilistic tensors and tensor rank from Publication IV here. We start by defining the concept of probabilistic tensors.

**Definition 3.6** (Probabilistic tensor Publication IV, Definition 1.1). A probabilistic tensor is a probability distribution over a space of tensors.

We need the following notion. Let $\mathcal{T}$ be a tensor and let $\tilde{\mathcal{T}}$ be a probabilistic tensor over the same space of tensors that contains $\mathcal{T}$. We say that $\tilde{\mathcal{T}}$ supports $\mathcal{T}$ entrywise with probability at least $p$ if for every entry in the tensor $\tilde{\mathcal{T}}$ it holds that the entry is either zero or agrees with the corresponding entry of $\mathcal{T}$, and agreement occurs with probability at least $p$. In other words, we have that for every $S \in \tilde{\mathcal{T}}$, we have $S \subseteq \mathcal{T}$ and $\Pr[s_{i} = t_{i}] \geq p$ for all $i$. 

Equation (3.38). We list the rank-one tensors, one tensor per row:

Let us now rewrite the rank-one tensors corresponding to the rows of $\alpha, \beta, \gamma$ in Equation (3.38). We list the rank-one tensors, one tensor per row:

\[
\begin{align*}
\alpha &= \begin{bmatrix}
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} & \\
\beta &= \begin{bmatrix}
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 \\
1 & 1 & -1 & -1 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} & \\
\gamma &= \begin{bmatrix}
0 & -1 & 0 & 1 \\
1 & -1 & -1 & 1 \\
0 & 0 & 1 & -1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\end{align*}
\] (3.38)

For a practical example, consider the following matrix representation of the Strassen-Winograd decomposition of $(2,2,2)$ from Equation (3.16).
The sum of the tensors listed in Equation (3.39) equals $\langle2,2,2\rangle$. In particular, the last tensor only contains exactly one non-zero element. Leaving this tensor out from the decomposition yields a rank-6 tensor as shown in Equation (3.40):

$$
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}.

Comparing Equation (3.40) to Equation (3.7), we see that only one element is missing, in the upper left hand corner. Thus, leaving one summand out of the eight in the matrix product, the tensor rank goes down. Also, by symmetry, we have that a similar decomposition exists for every non-zero in $\langle2,2,2\rangle$. We may thus represent $\langle2,2,2\rangle$ with a probabilistic tensor, consisting of every such subtensor (eight in total), with uniform distribution, such that any one subtensor is drawn with probability $p = 1/8$. Thus, this probabilistic tensor supports $\langle2,2,2\rangle$ entrywise with probability at least $7/8$.

We may now define the probabilistic tensor rank.

**Definition 3.7** (Probabilistic tensor rank [Publication IV, Definition 1.2]). The **probabilistic rank** of a nonzero tensor $T$ is the minimum of $E_{S \in \tilde{T}}|rkS|p^{-1}$ over all probabilistic tensors $\tilde{T}$ that support $T$ entrywise with probability at least $p > 0$. The probabilistic rank of a zero tensor is 0. We write $\tilde{rk}T$ for the probabilistic rank of $T$.

The probabilistic tensor rank extends to border rank.

**Definition 3.8** (Probabilistic border rank). The **probabilistic border rank** is defined by replacing the tensor rank $rkS$ in Definition 3.7 with the border rank $rkS$. We write $\tilde{rk}T$ for the probabilistic border rank of $T$.

It follows from the definition that $\tilde{rk}T \leq \tilde{rk}T \leq rkT$, and it is also shown in Publication IV that probabilistic rank and probabilistic border rank satisfy important properties of ordinary tensor rank and border rank, such as submultiplicativity under the Kronecker product.

We also show that the probabilistic rank of $\langle s, t, u \rangle$ satisfies $\tilde{rk}\langle s, t, u \rangle \geq (stu)^{2/3}$, and via the support-rank exponent $\omega_s$ of Cohn and Umans [133], $\omega_s \leq \inf \tau : \tilde{rk}\langle t, t, t \rangle = \Theta(t^\tau)$. For $\langle2,2,2\rangle$, we prove the following upper bound.

**Theorem 3.9** (Publication IV, Theorem 1.2). *Over any ring, it holds that*

$$
\tilde{rk}\langle2,2,2\rangle \leq 6 + \frac{6}{7} \quad \text{and} \quad \tilde{rk}\langle2,2,2\rangle \leq 6 + \frac{2}{3}.
$$

We show that the probabilistic tensor rank can be used to construct an algorithm for fast BMM as our main result.
**Theorem 3.10** (Publication IV, Theorem 1.1). For all integers $s, t, u \geq 2$ and any constant $\epsilon > 0$, there exists a randomized algorithm that with high probability multiplies two given $n \times n$ matrices over the Boolean algebra in

$$O(n^{(\log_{stu}(s,t,u)) + \epsilon})$$

operations in the field of scalars for $(s, t, u)$.

Theorems 3.9 and 3.10 imply an asymptotic runtime exponent of $\log_2(6 + 6/7) \approx 2.778$. This should be contrasted with Strassen’s asymptotic runtime exponent of $\log_2 7 \approx 2.807$. The exact number of arithmetic operations consists of $6^d$ multiplications, and the recurrence for the number of additions is

$$A(1) = 0, A(n) = 14 \cdot \left(\frac{n}{2}\right)^2 + 6 \cdot A \left(\frac{n}{2}\right),$$

yielding

$$A(2^d) = 7 \cdot (6^d - 4^d)$$

additions in closed form.

Due to the monotone property of BMM, the application of the rank-6 decomposition over $\mathbb{F}_2$ approximates the BMM product with one-sided error: erroneous ones can never occur in the product, but erroneous zeros can. Therefore, the probability of witnessing any single one can be boosted by performing independent repetitions and taking a disjunction of the resulting product matrices. Using the rank-6 decomposition as a base case, we get a witnessing probability of $p = (7/8)^d/2$ (from Publication IV, Theorem 1.5). Therefore, we can boost the probability of witnessing a one in the product to be at least $1 - 1/e$ by performing

$$R = \lceil 2 \cdot (8/7)^d \rceil$$

repetitions. In comparison, applying the standard rank-7 Strassen-Winograd decomposition requires $R = 2$ repetitions.

### 3.10 Feasibility of fast matrix multiplication

While there has been quite a remarkable development in the asymptotic complexity of matrix multiplication as described in Section 3.6, these developments are indeed asymptotic in nature. The big-O notation may hide very large constants and lower-order terms which brings the feasibility of these algorithms into question: do they actually provide any significant advantage over simpler algorithms on inputs of such size that they can fit in the memory of contemporary computers? Pan’s survey [366] addresses this issue, and, according to him, the asymptotically fastest feasible algorithm is the $O(n^{2.7734})$ algorithm by Pan himself [365]. Ballard, Benson, Druinsky, Lipshitz, and Schwartz [56] answer another critique sometimes presented with regard to FMM: numerical stability;
Bilinear algorithms and fast matrix multiplication

Table 3.1. This table lists the (exact) relative number of arithmetic operations when employing a particular FMM algorithm over the number of arithmetic operations used by the elementary (cubic) algorithm. Each of the rows displays a particular size of matrix multiplication problem where the matrices are square and whose sides have length \( n = 2^d \). The value in each column shows the ratio of the total number of arithmetic operations used by the particular algorithm, divided by the number of arithmetic operations used by the elementary algorithm. That is, the value is computed by summing the number of multiplications \( 7d \) and the number of additions varying between the different algorithms, and then dividing the total number of arithmetic operations by \( 2n^3 - n^2 \), the number of arithmetic operations used by the elementary algorithm. The column Str/Elem shows the ratio for Strassen’s algorithm with additions from Equation (3.15), the column SW/Elem shows the ratio for Strassen-Winograd algorithm with additions from Equation (3.17), and the column AB/Elem shows the ratio for the alternative-basis Strassen-Winograd with additions from Equation (3.22).

<table>
<thead>
<tr>
<th>( d )</th>
<th>( n )</th>
<th>Str/Elem</th>
<th>SW/Elem</th>
<th>AB/Elem</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>64</td>
<td>1.54</td>
<td>1.32</td>
<td>1.10</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>1.36</td>
<td>1.16</td>
<td>0.97</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>1.19</td>
<td>1.02</td>
<td>0.85</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>1.05</td>
<td>0.90</td>
<td>0.75</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>0.92</td>
<td>0.79</td>
<td>0.66</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>0.80</td>
<td>0.69</td>
<td>0.57</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>0.70</td>
<td>0.60</td>
<td>0.50</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
<td>0.62</td>
<td>0.53</td>
<td>0.44</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
<td>0.54</td>
<td>0.46</td>
<td>0.39</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
<td>0.47</td>
<td>0.40</td>
<td>0.34</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>0.41</td>
<td>0.35</td>
<td>0.30</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
<td>0.36</td>
<td>0.31</td>
<td>0.26</td>
</tr>
<tr>
<td>18</td>
<td>262144</td>
<td>0.32</td>
<td>0.27</td>
<td>0.23</td>
</tr>
<tr>
<td>19</td>
<td>524288</td>
<td>0.28</td>
<td>0.24</td>
<td>0.20</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>0.24</td>
<td>0.21</td>
<td>0.17</td>
</tr>
</tbody>
</table>

According to their error analysis, the feasible algorithms are indeed sufficiently stable to be useful.

Table 3.1 shows how large the matrices need to be before we see any hope of gaining improvement over the elementary algorithm in wall clock time by computing the ratio of the actual number of operations required by a particular algorithm to the number of operations required by the elementary algorithm. This shows that Strassen’s original algorithm requires a matrix of size \( 1024 \times 1024 \) for any gain. However, this can be improved to \( 512 \times 512 \) by taking Winograd’s alternative equations, and down to \( 128 \times 128 \) if we use the alternative basis, not including the operations required for performing the change of basis. However, this may give an overly optimistic view on the expected speedup because Strassen’s algorithm in all its variations requires non-linear access to the operand matrices while modern Central Processing Units (CPUs) have been optimized for linear access (see, for example, [255, 265]). However, it may be worthwhile to note that the recursive structure of Strassen’s algorithm makes
**Table 3.2.** This table compares the (exact) relative number of arithmetic operations used by different algorithms in the case of BMM. The columns $d$ and $n$ describe the side-length of the square matrix. The algorithms as described in the labels are the elementary algorithm (Elem), Strassen-Winograd (SW), alternative-basis Strassen-Winograd (AB), and the rank-6 decomposition (Six). The values in the columns show the ratio between the arithmetic operations as labeled. The arithmetic operation counts are computed as $2n^3 - n^2$ for the elementary algorithm, and $p^{-1}(M(2^d) + A(2^d))$ for the Strassen-like algorithms over $\mathbb{F}_2$; $M(2^d)$ is the number of multiplications ($7d$ for SW and AB, $6d$ for Six), $A(2^d)$ is the number of additions (Equation (3.17) for SW, Equation (3.22) for AB, and Equation (3.41) for Six), and $p$ is the probability for witnessing a fixed one in the product ($p = 1/2$ for SW and AB, $p = (7/8)^d/2$ for Six).

<table>
<thead>
<tr>
<th>$d$</th>
<th>$n$</th>
<th>Elem/SW</th>
<th>Elem/AB</th>
<th>Elem/Six</th>
<th>SW/Six</th>
<th>AB/Six</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1024</td>
<td>0.64</td>
<td>0.76</td>
<td>0.56</td>
<td>0.89</td>
<td>0.74</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>0.73</td>
<td>0.87</td>
<td>0.66</td>
<td>0.92</td>
<td>0.76</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>0.83</td>
<td>0.99</td>
<td>0.79</td>
<td>0.96</td>
<td>0.80</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
<td>0.95</td>
<td>1.14</td>
<td>0.88</td>
<td>0.93</td>
<td>0.78</td>
</tr>
<tr>
<td>14</td>
<td>16384</td>
<td>1.08</td>
<td>1.30</td>
<td>1.08</td>
<td>1.00</td>
<td>0.83</td>
</tr>
<tr>
<td>15</td>
<td>32768</td>
<td>1.24</td>
<td>1.48</td>
<td>1.25</td>
<td>1.01</td>
<td>0.84</td>
</tr>
<tr>
<td>16</td>
<td>65536</td>
<td>1.41</td>
<td>1.69</td>
<td>1.47</td>
<td>1.04</td>
<td>0.87</td>
</tr>
<tr>
<td>17</td>
<td>131072</td>
<td>1.61</td>
<td>1.94</td>
<td>1.66</td>
<td>1.03</td>
<td>0.86</td>
</tr>
<tr>
<td>18</td>
<td>262144</td>
<td>1.84</td>
<td>2.21</td>
<td>1.93</td>
<td>1.05</td>
<td>0.87</td>
</tr>
<tr>
<td>19</td>
<td>524288</td>
<td>2.11</td>
<td>2.53</td>
<td>2.27</td>
<td>1.08</td>
<td>0.90</td>
</tr>
<tr>
<td>20</td>
<td>1048576</td>
<td>2.41</td>
<td>2.89</td>
<td>2.72</td>
<td>1.13</td>
<td>0.94</td>
</tr>
<tr>
<td>21</td>
<td>2097152</td>
<td>2.75</td>
<td>3.30</td>
<td>3.09</td>
<td>1.12</td>
<td>0.94</td>
</tr>
<tr>
<td>22</td>
<td>4194304</td>
<td>3.15</td>
<td>3.77</td>
<td>3.69</td>
<td>1.17</td>
<td>0.98</td>
</tr>
<tr>
<td>23</td>
<td>8388608</td>
<td>3.59</td>
<td>4.31</td>
<td>4.25</td>
<td>1.18</td>
<td>0.98</td>
</tr>
<tr>
<td>24</td>
<td>16777216</td>
<td>4.11</td>
<td>4.93</td>
<td>4.98</td>
<td>1.21</td>
<td>1.01</td>
</tr>
</tbody>
</table>

It is easy to create hybrid algorithms where the elementary algorithm is used at smaller problem sizes. This is the approach we took in Publication V.

Table 3.2 shows a similar comparison in the case of BMM. Here, we compare the exact number of arithmetic operations required by different randomized algorithms over $\mathbb{F}_2$ to witness any fixed one in the product matrix with at least probability $1 - 1/e$. The elementary algorithm forms an exception since it is deterministic. For the Strassen-Winograd, in standard and alternative bases, the number of repetitions required is constant 2 (see Publication IV for details). For the rank-6 randomized algorithm, the number of repetitions is $\lceil 2/(7/8)^d \rceil$ (from Equation (3.42)).

It can be seen in Table 3.2 that, due to the need for repetitions, an input of size $n = 8192$ is needed before the alternative-basis Strassen-Winograd overtakes the elementary algorithm, and an even larger input of size $n = 16384$ before the rank-6 algorithm offers any gain over the elementary algorithm. However, asymptotics do eventually play out at $n = 16777216$ where the rank-6 algorithm finally overtakes the alternative-basis Strassen. Assuming we use one bit per
element and 8 bits per byte, it should be noted that a $16777216 \times 16777216$ binary matrix requires 32 TiB of storage, which is not at all an unfeasible amount of memory, at least in mass or distributed storage if not the DRAM of a single computer as of 2019.

Although the bulk of the practical work in the implementation of FMM presented in literature has addressed Strassen’s algorithm and its variants, it should be noted that implementations of Pan’s trilinear aggregation method [364] also exist [294, 276, 277], although these implementations are rather dated, which makes it difficult to predict if they are competitive on contemporary hardware.
4. Finding low-rank decompositions of tensors

In this chapter, we review methods for finding (exact) low-rank decompositions for tensors. In Section 4.1, we present Brent equations, the solutions to which yield the desired decompositions. In Section 4.2, we present Alternating Least Squares (ALS), a popular methodology for finding such decompositions, and particularly elaborate on Smirnov’s [418] approach to ALS. In Section 4.3, we present the problem of breaking symmetries (in constraint programs), and review the symmetry breaking literature. In Section 4.4, we present our theoretical main result from Publication III, an instantiation of McKay’s canonical extension method that can be used for breaking the symmetries or isomorph-free listing of combinatorial objects. In Section 4.5, we return to the matrix multiplication tensor and Brent equations, and review their symmetries, and show how the preceding methodology can be used for isomorph-free enumeration of subtensors. Finally, in Section 4.6, we describe how constraint programming methods, assisted by symmetry breaking, can be used to discover exact tensor decompositions.

4.1 Overview and Brent equations

Let \( \mathcal{J} \) be a third order tensor. The act of finding a tensor decomposition \( a, \beta, \gamma \) such that \( \mathcal{J} = [a, \beta, \gamma] \) is equivalent to solving a set of cubic equations called Brent equations as per [97]. For \( (s,t,u) \), the Brent equations are defined as a set of \( s^2t^2u^2 \) equations defined for \( i, i' = 1,2,\ldots,s; \; j, j' = 1,2,\ldots,u; \; k, k' = \ldots,u \) as

\[
\sum_{\ell=1}^{r} a_{i,k},t \beta_{k,j},t \gamma_{j,i} = t(i,k'),(j,j'),(i',i) = \delta_{ii'}\delta_{jj'}\delta_{kk'}.
\]

The equations look similar even when the target tensor \( \mathcal{J} \) is arbitrary. For simplicity, we shall drop the double-indexing and consider a general \( a \times b \times c \) tensor \( \mathcal{J} \).

**Definition 4.1.** The Brent Equations of length \( r \) for the target tensor \( \mathcal{J} \) of shape \( a \times b \times c \), is the set of \( abc \) cubic equations seeking to decompose \( \mathcal{J} \) into a sum of
Finding low-rank decompositions of tensors

$r$ rank-one tensors. For $i = 1, 2, \ldots, a; j = 1, 2, \ldots, b; k = 1, 2, \ldots, c$, the equation is of form

$$\sum_{\ell=1}^{r} a_{i\ell} \beta_{j\ell} \gamma_{k\ell} = t_{ijk}$$

where $\alpha, \beta, \gamma$ are matrices of shape $a \times r$, $b \times r$, $c \times r$, respectively, and satisfy $[\alpha, \beta, \gamma] = \mathcal{T}$.

Observe that the least number $r$ for which the system remains solvable is equal to $\text{rk} \mathcal{T}$. Since the system for $(2,2,2)$ consists of 64 equations, we give a smaller explicit example from Publication III.

**Example 4.2** (Publication III, Example 17). Consider the following system of eight cubic equations over 24 variables taking values modulo 2:

\[
\begin{align*}
x_{11}y_{11}z_{11} + x_{12}y_{12}z_{12} + x_{13}y_{13}z_{13} &= 0, \\
x_{21}y_{11}z_{11} + x_{22}y_{12}z_{12} + x_{23}y_{13}z_{13} &= 0, \\
x_{11}y_{21}z_{21} + x_{12}y_{22}z_{22} + x_{13}y_{23}z_{23} &= 0, \\
x_{21}y_{21}z_{21} + x_{22}y_{22}z_{22} + x_{23}y_{23}z_{23} &= 1, \\
x_{11}y_{21}z_{21} + x_{12}y_{22}z_{22} + x_{13}y_{23}z_{23} &= 1, \\
x_{11}y_{21}z_{21} + x_{12}y_{22}z_{22} + x_{13}y_{23}z_{23} &= 1, \\
x_{21}y_{21}z_{21} + x_{22}y_{22}z_{22} + x_{23}y_{23}z_{23} &= 1, \\
x_{21}y_{21}z_{21} + x_{22}y_{22}z_{22} + x_{23}y_{23}z_{23} &= 1.
\end{align*}
\]

This system seeks to decompose a $2 \times 2 \times 2$ tensor (whose elements appear on the right hand sides of the equations) into a sum of three rank-one tensors.

In general, finding such decompositions is very difficult. Indeed, Håstad showed [226] that finding the rank of an arbitrary tensor is NP-hard over $\mathbb{Q}$ and NP-complete over finite fields. There are some computational tools that may be employed, nevertheless. Already Brent [97] used an Alternating Least Squares (ALS) approach to find an approximate solution over the rationals. A similar approach was adopted by Johnson and McLoughlin [269] who used it to find a rank-23 decomposition of $(3,3,3)$. We will discuss the work by Smirnov [418] in Section 4.2 that includes the rounding of rational-valued solutions into integer solutions. Benson and Ballard [76] also used a similar approach for automatically generating families of FMM algorithms.

It is also possible to use Constraint Programming (CP) methods, for example, by reducing the system of equations from Definition 4.1 to a CNF-SAT problem, by using the Tseitin transformation for example. We will discuss this approach in Section 4.6. While at first glance, this approach may seem intractable, the solving of larger instances is made possible by breaking the symmetry in the instances.
As was pointed out in Section 3.2, the decompositions directly yield bilinear algorithms. Thus, as related work, although not seeking new base tensor decompositions, Huang, Rice, Matthew, and van de Geijn [242] presented interesting work regarding the generation of multiple-layered FMM algorithms.

Other work includes [431, 304, 96, 82, 353]. All of these have in common that they only address special cases, such as limited length of modes, matrix slice structure, or symmetric tensors.

Finally, while we restrict ourselves to the search for exact tensor decompositions, it should be noted that extensive literature exists for finding approximate low-rank decompositions of tensors, as these have applications in functional Magnetic Resonance Imaging (fMRI) [69], the ranking of semantic web data [187], electroencephalography (EEG) [136], and signal processing and machine learning [414], among others. In addition to Kolda and Bader’s [287], Comon’s [135], and Sidiropoulos et al.’s [414] surveys, see, for example, [150, 467, 169, 355, 354, 372, 189, 202, 424].

4.2 Alternating least squares

We describe an alternating least squares approach for finding integer-valued tensor decompositions following Smirnov [418]. For other similar ALS work, see [97, 269, 76]; see also the orthogonalized ALS of Sharan and Valiant [411] that offers some provable convergence guarantees with respect to finding its factors or the constituent rank-1 tensors. While a description is contained within [418], we describe a variant of Smirnov’s approach in a way that is easy to reimplement.

Let \( T \) be a target tensor of shape \( a \times b \times c \) for which we wish to find a decomposition \( \alpha, \beta, \gamma \) where \( \alpha, \beta, \gamma \) have shapes \( a \times r, b \times r, c \times r \), respectively. Let us begin with a high-level overview. We start by finding a rational solution that approximately minimizes the least-squares objective function

\[
f'(\alpha, \beta, \gamma) = \left( \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} \left( \sum_{\ell=1}^{r} \alpha_{i\ell} \beta_{j\ell} \gamma_{k\ell} - t_{ijk} \right)^2 \right)^{1/2}.
\]

After an approximate solution is found, we seek to round it to an exact integer solution that matches \( T \) by adding regularization terms to the objective function (4.1):

\[
f(\alpha, \beta, \gamma) = \left( \sum_{i=1}^{a} \sum_{j=1}^{b} \sum_{k=1}^{c} \left( \sum_{\ell=1}^{r} \alpha_{i\ell} - \alpha_{i\ell}^{\text{exact}} \right)^2 + \sum_{\ell=1}^{r} \left( \beta_{j\ell} - \beta_{j\ell}^{\text{exact}} \right)^2 + \sum_{\ell=1}^{r} \left( \gamma_{k\ell} - \gamma_{k\ell}^{\text{exact}} \right)^2 \right)^{1/2}.
\]

(4.2)

The regularization terms \( \alpha, \beta, \gamma \) softly enforce the values to be in a certain interval.
Algorithm 1: Alternating least-squares iteration. During each iteration, two of the input matrices are held constant, and one of the input matrices is updated; this is repeated three times. The nested elementwise min and max functions form the regularization term from the \( R \) matrix passed as argument. The constant parameter \( \epsilon \) controls the strength of regularization.

```plaintext
1: procedure ITERATE(\( T, \alpha, \beta, \gamma, R_\alpha, R_\beta, R_\gamma, \epsilon \))
2:   Fix \( \beta \) and \( \gamma \)
3:   \( \alpha \leftarrow \max(\min(\alpha, R_\alpha), -R_\alpha) \)
4:   \( \beta \leftarrow \max(\min(\beta, R_\beta), -R_\beta) \)
5:   \( \gamma \leftarrow \max(\min(\gamma, R_\gamma), -R_\gamma) \)
6:   Reshape and permute \( T \) to have the shape \( a \times bc \)
7:   \( A \leftarrow \beta \otimes \gamma \)
8:   \( a \leftarrow (TA + \epsilon a)(A^\top A + \epsilon I)^{-1} \)
9:   Repeat the steps above for computing \( \beta \), but fix \( \alpha \) and \( \gamma \) instead
10:  Repeat the steps above for computing \( \gamma \), but fix \( \alpha \) and \( \beta \) instead
11:  return \( \alpha, \beta, \gamma \)
12: end procedure
```

We now provide a more detailed view. The higher-order objective function is transformed into a linear one by holding two of the three indeterminate matrices \( \alpha, \beta, \gamma \) constant and minimizing with respect to the remaining matrix in a round-robin fashion. Also observe that \( f'(\alpha, \beta, \gamma) = f(\alpha, \beta, \gamma) \) if we let the regularization terms be all zero, so we only need to consider the objective function (4.2) in our analysis. A single iteration is presented in Algorithm 1.

We now justify the computation of the iteration in Algorithm 1.

Lemma 4.3. Algorithm 1 correctly computes the least-squares iteration.

Proof. Consider the system of equations

\[
\nabla f(\alpha, \beta, \gamma) = 0. \tag{4.3}
\]

Without loss of generality, let us assume \( \beta \) and \( \gamma \) are held constant, and solve the equation for \( \alpha \). We may view \( a \) as an \( ar \times 1 \) column vector. Consequently, we may treat \( f(\alpha, \beta, \gamma) \) as a \( ar \times 1 \) column vector. Writing the partial derivative with respect to \( a_{iw} \) where \( i \in [a] \) and \( u \in [r] \), we get from Equations (4.2) and (4.3) that for a single equation in the system, we may write

\[
\frac{\partial}{\partial a_{iw}} \sum_{j=1}^{b} \sum_{k=1}^{c} \beta_{ju} \gamma_{ku} \left( t_{ijk} - \sum_{\ell=1}^{r} a_{i\ell} \beta_{j\ell} \gamma_{k\ell} \right) + \epsilon (a_{iw} - a_{iw}) = 0. \tag{4.4}
\]

In matrix form, we may write the system of equations as follows,

\[
(M + \epsilon I)\alpha = J(\beta \otimes \gamma) + \epsilon a, \tag{4.5}
\]
Algorithm 2 The procedure for finding the rational approximate solution. The alternating least squares iteration is repeated until the cost function reaches a sufficiently small value. The target tensor $\mathcal{T}$ is supplied as input.

1: procedure $\text{RATIONALALS}(\mathcal{T})$
2: Randomly initialize $\alpha, \beta, \gamma$
3: Initialize the cost $c$ to a huge value
4: while the cost $c$ exceeds a threshold do
5: for a constant number of iterations do
6: $\alpha, \beta, \gamma \leftarrow \text{ITERATE}(\mathcal{T}, \alpha, \beta, \gamma, 0, 0, 0)$
7: end for
8: $c \leftarrow \|\mathcal{T} - \|\alpha, \beta, \gamma\|_F^2$
9: end while
10: return $\alpha, \beta, \gamma$
11: end procedure

where

$$
m_{i'u,i'\ell} = \sum_{j=1}^b \sum_{k=1}^c \beta_{j\ell} \gamma_{k\ell} \beta_{ju} \gamma_{ku} [i = i']$$

$$= \sum_{j=1}^b \sum_{k=1}^c (\beta \otimes \gamma)_{jk,\ell} (\beta \otimes \gamma)_{j,ku} [i = i']$$

$$= \|i = i'\|((\beta \otimes \gamma)^T(\beta \otimes \gamma))_{\ell u}. \quad (4.6)$$

Hence, denoting $A = \beta \otimes \gamma$, we get the least squares solution for $\alpha$ from Equations (4.5) and (4.6) by

$$\alpha = (\mathcal{T}A + \epsilon A)(A^T A + \epsilon I)^{-1}. \quad \square$$

Consider now Algorithm 2. The procedure works by initializing the target matrices to random values (for example, Gaussian-distributed random values), and then applies the $\text{ITERATE}$ procedure until the cost of the objective function becomes sufficiently small.

Algorithm 3 describes how the approximate rational solution can be rounded into an exact integer solution by fixing one element at the time. In the beginning, all elements are unfixed and allowed to change arbitrarily. The algorithm uses the same iteration subroutine, but, at the end of each iteration, if the rounded versions of $\alpha, \beta, \gamma$ do not provide the correct answer, the largest elements in absolute value of $\alpha, \beta, \gamma$ are fixed. Fixing is done by setting the regularization terms with a clipping function that maps the values to the range $[-m, m]$ such that values outside the interval are penalized. Here $m, \epsilon > 0$ are constants that control the rounding process. For example, if we sought a decomposition with coefficients in $\{-1, 0, 1\}$, the choice of $m = 1$ would be reasonable. The constant $\epsilon$
Algorithm 3 The algorithm for rounding the rational-valued solution into an integer-valued one. It is assumed that the rational solution is passed as input, along with the parameters \( m, \epsilon > 0 \). Then, the largest elements in the matrices are fixed, one at a time, and regularization is enforced in the interval \([-m, m]\) by a clipping function. The cost is computed by using the rounded versions of the decomposition matrices. The function \( \text{ROUND} \) performs elementwise rounding to the nearest integer value. Also observe that it is possible for the algorithm to fail; there is no guarantee that an integer-valued solution is reached.

```plaintext
1: procedure INTEGERALS(\( \alpha, \beta, \gamma, m \))
2:   \( R_\alpha, R_\beta, R_\gamma \leftarrow \infty \)
3:   while there are unfixed entries do
4:     Update \( \epsilon \)
5:     for a constant number of iterations do
6:       \( \alpha, \beta, \gamma \leftarrow \text{ITERATE}(\mathcal{J}, \alpha, \beta, \gamma, R_\alpha, R_\beta, R_\gamma, \epsilon) \)
7:     end for
8:     \( c \leftarrow \| \mathcal{J} - [\text{ROUND}(\alpha), \text{ROUND}(\beta), \text{ROUND}(\gamma)] \|_F \)
9:     if \( c = 0 \) then
10:        return \( \text{ROUND}(\alpha), \text{ROUND}(\beta), \text{ROUND}(\gamma) \)
11:     else
12:        Find an absolutely largest element \( i \) of \( \alpha \) that has not been fixed
13:        \( (R_\alpha)_i \leftarrow m \)  \( \triangleright \) Fix an element
14:        Repeat the steps above for \( \beta \) and \( R_\beta \), and \( \gamma \) and \( R_\gamma \)
15:     end if
16:   end while
17: fail  \( \triangleright \) Rounding failed
18: end procedure
```

determines the strength of regularization. The practical implementation sets the value of a matrix \( R_\alpha \) (and \( R_\beta \) and \( R_\gamma \)) of the same shape as \( \alpha \) (and \( \beta \) and \( \gamma \)) to \( m \); initially no regularization is used. The actual regularization matrix \( \alpha \) is constructed from \( R_\alpha \) (and \( \beta \) from \( R_\beta \) and \( \gamma \) from \( R_\gamma \)) by using the nested elementwise min and max functions in Algorithm 1 as a clipping function that enforces the values to be in the given range. The parameter \( \epsilon \) that controls the strength of regularization is held fixed during the execution of the iteration of Algorithm 1; the update rule in Algorithm 3 is applied upon fixing an entry. Experimentally, a reasonable choice for the update rule would be to multiplicatively increase the value of \( \epsilon \), so that the more elements have been fixed, the more regularization is applied. One possible rule that we experimented with was

\[
\epsilon \leftarrow \epsilon \exp \frac{\ln 0.1 - \ln 0.001}{\max\{ar, br, cr\}}.
\]

The problem with this approach is that, at each iteration, the cost may only decrease, so it is possible for the procedure to become stuck at a local minimum. This means that several restarts may be required before an integer solution is
reached. However, due to the formulation, it is easy to do many runs in parallel, for example, on Graphics Processing Unit (GPU) hardware which alleviates the problem to some degree. Unfortunately, the algorithm gives no theoretical guarantees about convergence. However, the results presented in Publication IV were founded by a systematic exploration of tensor decompositions with a GPU implementation of an algorithm similar to the one described in this section. This led to the discovery of several interesting subtensor decompositions, the properties of which were then theoretically justified. Also, with this methodology, Smirnov [418] has been able to produce several Strassen-like matrix multiplication algorithms of variable lengths, using different base tensors. He also extends the approach to that of finding border rank decompositions by rounding the coefficients of Laurent polynomials. In [419], Smirnov presents a border-rank-46 decomposition for \( \langle 4, 4, 4 \rangle \).

4.3 Symmetry breaking

We say that a problem exhibits symmetry if, for example, it is possible to permute the variables associated with an instance such that the instance is preserved. Such cases arise naturally in CP, SAT for example, when considering systems of graphs (see, for example, [204]), or enumerating the subtensors of a fixed parent tensor (see Section 4.5). For (colored) graphs, it may be possible to exchange the vertices or the colors whilst preserving the structure of the graph, and in the case of tensors, it may be possible to exchange the modes whilst preserving the value of any given element. In the case of SAT, it may be possible to exchange literals whilst preserving the satisfiability, the number of solutions, or even the exact solutions.

Knowing the symmetries of the instance of a combinatorial search problem enables us to break or remove the symmetry, that is, restrict our search to only one example from each isomorphism class since all instances within a class are essentially equal. This may amount to an exponential reduction in the size of the search space by removing a large number of dead-end branches in the search tree. Here on, we consider the problem of listing nonisomorphic instances of the problem by fixing variables in the problem instance, under the action of the symmetry group for the problem.

Let us continue with a practical example.

**Example 4.4** (Publication III). Consider the following CNF-SAT instance:

\[(x_1 \vee x_2) \wedge (x_1 \vee \neg x_3 \vee \neg x_5) \wedge (x_2 \vee \neg x_4 \vee \neg x_6).
\]

The symmetries of this instance are generated by the following two permutations: \((x_1, x_2)(x_3, x_4)(x_5, x_6)\) and \((x_4, x_6)\). Hence, the automorphism group of the system consists of all the permutations of variables given by

\[
\Gamma = \{(x_1, x_2)(x_3, x_4)(x_5, x_6), (x_4, x_6)\}.
\]
In particular with respect to application in SAT and CSP (Constraint Satisfaction Problem), it is possible to classify the existing literature into static and dynamic, and complete and partial symmetry breaking. Static methods [142, 25, 28, 383, 26, 233, 158] consider the constraint system independently and modify it to account for the symmetries. Dynamic methods [102, 75, 208, 176, 185, 305, 55, 205, 376, 380, 397, 403, 27, 73, 159, 157] are interleaved with the solver system and modify the behavior of the solver on-the-fly, typically by rejecting branches of the search tree. Complete symmetry breaking methods [142, 394, 233] exhaustively enumerate all isomorphism classes and remove all symmetry from the problem. However, since this may be intractable, partial methods [142, 325, 25, 376, 28, 267, 303] have been developed which aim for a more favorable time or space tradeoff by leaving some residual symmetry in the problem.

It is possible to distinguish between semantic and syntactic symmetry [398]. Semantic symmetry refers to those symmetries that are intrinsic to the problem at hand and independent of the representation. Syntactic symmetries arise from the algebraic representation of the constraint system. Syntactic symmetries may be viewed as a subgroup of the semantic symmetries; that is, syntactic symmetries always preserve semantics, but not vice versa; alternatively, we may view this as the symmetries of the problem being obfuscated by its algebraic encoding. In the CP community, semantic and syntactic symmetries are sometimes called constraint and solution symmetries, respectively [130].

In Publication III, we are looking for a functional solution that generates partial assignments, or maps a subset of variables to values. Thus, we may distinguish between variable and value symmetry, depending on which elements we permute, and consider the mixture of the two. In a general case, we may apply different permutations for the images of different variables; that is, we may view the permutation as having wreath-product structure, consisting of a variable permutation and a number of value permutations (see Publication III, Section 5). Some authors only look at a more limited form of value symmetry where the values are permuted irrespective of the variables, for example [377, 381]. See also [378, 379] for work addressing special cases such as where all variables of the constraint program are enforced to take differing values, or where the solution is expected to be surjective, that is, requiring that all values occur in the solution.

It is well-known that the detection of symmetries is reducible to GRAPH ISOMORPHISM [141] (see also [52]). The problem is defined as follows.

**Problem 4.5 (GRAPH ISOMORPHISM).** Given two colored graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with colorings $c_1 : V_1 \to C$ and $c_2 : V_2 \to C$, does there exist a color and edge-preserving bijection $f : V_1 \to V_2$ such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$, and $c_1(u) = c_1(v)$ if and only if $c_2(f(u)) = c_2(f(v))$?

It is known that GRAPH ISOMORPHISM is in NP but it is not known if it is NP-complete [199]. Observe that two graphs can only be isomorphic if they
have an equal number of vertices and edges, and their colorings are compatible, that is, the colorings induce partitions of equal size, partwise. Practical work on symmetry detection thus employs tools such as GAP [198], nauty [327, 329], saucy [146] or bliss [273].

A classical way to exploit symmetry in SAT/CSP is by adding constraints that prevent symmetrical subproblems from being probed during the search [375]. These constraints are commonly called Symmetry-Breaking Predicates (SBPs) [142, 398] that enforce only certain solutions, and remove some or all of the symmetry. Typically, this is done by deriving lexicographic leader (lex leader) constraints from the generators of the automorphism group [142, 315, 28, 207, 445, 398]. Other SBPs include [182, 192, 412]. In [26], SBPs for pseudo-Boolean formulae are described. Unfortunately, the SBP-based approach may be intractable for complete symmetry breaking in many cases due to the size of required constraints (see, for example, [315, 446]).

A certain variant of symmetry breaking is conditional symmetry breaking [206, 445], also known as local symmetry [74, 72] (as opposed to global symmetry or unconditional symmetry). That is, we seek symmetries in a subproblem that are induced by some condition; the subproblem may be obtained, for example, by fixing a partial assignment. Such subproblems arise naturally during a systematic search for a solution to a CSP.

Some of the more recent work includes Itzhakov and Codish [260] whose SBPs are powerful enough to completely break symmetries in search problems on small (10-vertex) graphs, independent of the particular instance in question. In [129], Codish, Gange, Itzhakov, and Stuckey extend this technique by adopting techniques from the algorithm of nauty [327, 329] to create efficient SBPs. Further work on this line of research, trying to keep SBPs small in the context of graph search and matrix model problems, is provided by Codish, Ehlers, Gange, Itzhakov, and Stuckey [128]. Heule [233] presents very compact SBPs that completely break symmetries for graphs of order up to five. In [158], Devriendt, Bogaerts, Bruynooghe, and Denecker present a tool called BreakID for static symmetry breaking, improving on an earlier tool called Shatter [28]. Devriendt, Bogaerts, and Bruynooghe [157] present a dynamic method called Symmetric Explanation Learning, extending earlier work from [159].

The isomorphism-free enumeration of combinatorial objects is a related problem. In particular, we will describe and apply McKay's canonical extension method [328, 329] in Section 4.4. Other possible frameworks in this field include Read's orderly algorithm [385], and the homomorphism principle of Kerber and Laue [283]. For other work on the canonical labeling of graphs, see Babai and Luks [54]. For more recent work on the canonical labeling and isomorphism of graphs, Babai [53] gives a quasipolynomial algorithm for the graph isomorphism problem. Grohe, Neuen, and Schweitzer [219] improve on Babai's work and provide an algorithm for graph isomorphism on graphs with \( n \) vertices and bounded degree \( d \) that runs in time \( \mathcal{O}(n^{\text{polylog}d}) \). Lokshtanov, Pilipczuk,
Pilipczuk, and Saurabh [312] give a fixed-parameter tractable algorithm for canonical labeling and isomorphism testing of graphs of bounded treewidth, which was improved on by Grohe, Neuen, Schweitzer, and Wiebking [220] who additionally provide a canonizing algorithm. For more general work, for example, Jefferson, Jonauskyte, Pfeiffer, and Waldecker [266] give an algorithm for finding canonical images in arbitrary groups, and, in [409], Schweitzer and Wiebking define a general notion of combinatorial object and provide a unified framework for designing canonizing algorithms for such objects.

4.4 Adaptive prefix assignment

In this section, we describe concisely the main result of Publication III which is an instantiation of McKay’s canonical extension method [328]. The method works by generating partial assignments\(^2\) from a set of variables \(U\) to a set of values \(R\), one variable at a time following a prefix sequence. As output, we obtain a set of partial assignments corresponding to each isomorphism class with respect to group \(\Gamma \leq \text{Sym}(U)\). We start by giving a very brief overview of McKay’s method, and then present some particularities of the method of Publication III.

The main idea of McKay’s canonical extension method [328] is to list objects without repeating their isomorphs by extending seeds to objects, in a procedure that can be iterated inductively. A more extensive exposition with proofs is given in Publication III, Section 3. For now, let \(\Omega\) be the set of objects and \(\Sigma\) the set of seeds, and let \(\Gamma\) act on \(\Omega\) and \(\Sigma\). We say that \(X \in \Omega\) extends \(S \in \Sigma\), and write \(X \in S\), when we can build \(X\) from \(S\). Observe that we may view \(e\) as a relation \(e \subseteq \Omega \times \Sigma\). We will assume that (i) \(e\) is a union of orbits of \(\Gamma\), that is, \(e^{\Gamma} = e\), and (ii) for every object \(X \in \Omega\) there exists a seed \(S \in \Sigma\) such that \(X \in S\).

We require the notion of canonical labeling map to proceed.

**Definition 4.6** (Canonical labeling map). A function \(\kappa : \Omega \to \Gamma\) is a canonical labeling map for the action of \(\Gamma\) on \(\Omega\) if, for all \(X, Y \in \Omega\), it holds that \(X \equiv Y\) implies \(X^{\kappa(X)} = Y^{\kappa(Y)}\).

We say that \(X^{\kappa(X)}\) is the canonical form of \(X\); all isomorphic objects have identical forms. With the help of a canonical labeling map, we may define a canonical extension map \(M : \Omega \to \Sigma\). We require that for all \(X \in \Omega\), \(X \in M(X)\), and that all isomorphic objects have isomorphic canonical extensions.

Let \(e(S) = \{X \in \Omega : X \in S\}\) be the set of objects that extend \(S\). We will consider the following procedure (P): Let \(S \in \Sigma\) be given as input. Iterate over all \(X \in e(S)\). Perform zero or more isomorph rejection tests on \(X\) and \(S\). If the tests indicate we should accept \(X\), visit \(X\).

The procedure (P) defines a recursive depth-first search. A careful choice of tests ensures that (P) visits every isomorphism class of objects in \(\Omega\) exactly\(^2\) for a concrete example, these may be viewed as partial truth assignments in the binary case.
Finding low-rank decompositions of tensors

once (Publication III, Lemmata 1–4). The canonical extension map $M$ can be constructed from $\kappa$ by identifying the seed of the canonical form of $X$, and then constructing the canonical extension (see Publication III, Lemma 5). This concludes the abstract description of McKay’s method.

We will now present our instantiation of the method. Let $R$ be a finite set where the variables in $U$ take values. For a subset $W \subseteq U$, we say that a partial assignment of values to $W$ is a mapping $X : W \rightarrow R$. An element $\gamma$ of the group $\Gamma$ acts on $X$ by $X^\gamma : W^\gamma \rightarrow R$ where $X^\gamma(u) = X(u^{-1})$ for all $u \in W^\gamma$. We write $X = W$ for the underlying set of variables assigned by $X$.

The prefix sequence is prescribed by $k$ distinct elements of $U$, $u_1, u_2, \ldots, u_k$. We denote the subprefixes by $U_j = \{u_1, u_2, \ldots, u_j\}$ for $j = 0, 1, \ldots, k$. We define $\Omega_j$ to be the set of partial assignments $X : W \rightarrow R$ where $X \simeq U_j$. We say $X \in \Omega_j$ is normalized if $X \simeq U_j$.

Assuming the availability of the canonical labeling map $\kappa : \Omega_j \rightarrow \Gamma$, we use canonical extension to construct exactly one object from each orbit of $\Gamma$ on $\Omega_j$ from exactly one seed from each orbit of $\Gamma$ on $\Omega_j - 1$, for each $j = 1, 2, \ldots, k$.

For a fixed $j = 1, 2, \ldots, k$, the particular procedure we consider is (P’) defined as follows: Let normalized $S \in \Omega_j - 1$ be given as input. For each $p \in \text{Aut}(U_j - 1)$ and each $r \in R$, construct the assignment $X : U_j - 1 \cup \{p\} \rightarrow R$ defined by $X(p) = r$ and $X(u) = S(u)$ for all $u \in U_j - 1$. Perform isomorph rejection tests (Publication III, Lemmata 10 and 11) on $X$ and $S$. If the tests accept, visit the normalized $X$.

It is shown in Publication III that (P) is a correct instantiation of the procedure (P) in McKay’s framework. The algorithm is general in nature and makes few assumptions about the objects that are being operated on and is not limited to SAT or particular CSP instances, as long as the symmetries can be encoded as colored graphs. This is particularly useful if the constraint encoding of the objects obfuscates the semantic symmetries, such as in the case of Example 4.2 (see Figure 4.1 and Section 4.5). Due to the guarantees of McKay’s method, the algorithm achieves complete symmetry breaking with respect to the prefix sequence. Furthermore, the depth-first search structure is such that the subtrees of the search tree do not depend on one another, enabling efficient parallelization. Parallelization is considered more in Section 5.3.

We also provide an implementation called reduce$^3$ that uses the nauty$^{[327, 329]}$ canonical labeling software to provide access to the canonical labeling maps. The software can use user-provided graphs as representation of the symmetries, or generate them from CNF-SAT input.

It is possible to extend this method to account for value symmetry, that is, we also consider the cases where the values assigned to individual variables are allowed to change. This makes the underlying group structure more complicated.

$^3$https://github.com/mkarppa/reduce
Finding low-rank decompositions of tensors

Figure 4.1. Symmetries of \((2,2,2)\). Each variable vertex corresponds to one non-zero element in the tensor. Determining if two subtensors are symmetric is reduced to GRAPH ISOMORPHISM when we add two differently-colored vertices corresponding to values 1 and 0, and connect these vertices to exactly those variable vertices that get the desired value.

as it assumes wreath-product structure. Analysis is provided in Publication III, Section 5. However, value symmetries are not included in our implementation.

4.5 Symmetries of the matrix multiplication tensor and the Brent equations

In this section, we describe the coordinate-wise symmetries of the matrix multiplication tensor and its subtensors with an explicit example of \((2,2,2)\). We also give a partial description of the symmetries of the Brent equations.

In the general case, the matrix multiplication tensor \(\langle s, t, u \rangle\) has \(stu\) non-zero elements. The symmetries arise from the permutation of modes of the tensor. That is, any two subtensors \(S, T \subseteq \langle s, t, u \rangle\) are symmetric if and only if there exist a permutation of the coordinates of the elements that preserves ones and zeros in their locations.

For a particular example, let us look at \((2,2,2)\) in Equation (3.7). The symmetries of subtensors are captured by the graph in Figure 4.1. Each non-zero element of \((2,2,2)\) is in bijective correspondence with a unique vertex as labeled in the graph. It is possible to determine if two subtensors are isomorphic by reduction to GRAPH ISOMORPHISM by adding two differently-colored value vertices to the graph, corresponding to values 1 and 0, and connecting them to the corresponding variable vertices. That is, if we have a subtensor \(T \subseteq \langle 2,2,2\rangle\) such that \(t_{111} = 1\), we connect the value vertex 1 to the variable vertex 111 in the corresponding graph.

Feeding the graph of Figure 4.1 into adaptive prefix assignment software implementing the method described in Section 4.4 with a full prefix sequence of 8 variables (in any order), we obtain exactly 22 representatives, one from each isomorphism class. The representatives are shown as binary images in Table 4.1.
Table 4.1. The 22 isomorphism classes of subtensors of \((2,2,2)\) arranged by rank. The images encode the binary values with 0 as black and 1 as white. The slices have been taken along the third mode, and stacked horizontally. Each image shows exactly one representative from each isomorphism class.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Representatives from each class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Now consider the system of equations in Example 4.2. The size of the automorphism group of this particular system is 12, and may be justified as follows: the order of summation may be permuted arbitrarily (3! permutations), and the \(y\) and \(z\) variables can be interchanged (2! permutations), yielding a total of \(2!3! = 12\) permutations. This can be verified computationally using the reduce software by feeding the graph of Figure 4.2 as input.

In general, the size of the automorphism group of a system of Brent equations depends on the structure of the target tensor, as the number of interchangeable sets of variables depends on whether the different modes are compatible in length, and whether the right-hand sides of the equations agree with one another. As a concrete, analytical example, the size of the automorphism group for a system seeking rank-\(r\) decomposition of \((n,n,n)\) is at least \(3!r!(n!)^3\): the variables \(x, y, z\) may be interchanged freely (3! permutations), the order of summation may be permuted freely (\(r!\) permutations), and the elements along each mode can be permuted freely (\(n!\) permutations per mode for a total of \((n!)^3\) permutations).

### 4.6 Constraint programming

A system of Brent equations can be converted to a constraint program. We consider the special case where we seek to solve the system over \(\mathbb{F}_2\), and, for this purpose, convert the system into a SAT instance.

Over \(\mathbb{F}_2\), each cubic equation is of the form \(\sum x \cdot y \cdot z = b\) where \(b \in \{0, 1\}\), the multiplication corresponds to an AND, and the summation to a XOR. The Boolean formula corresponding to the system is therefore a conjunction of a XOR of conjunctions (if \(b = 1\)) or their negation (if \(b = 0\)). If the system is satisfiable, a solution to the system can be directly read from a satisfying truth-assignment; otherwise, it can be determined with certainty that the system has no solution.
Figure 4.2. A graph encoding the symmetries in Example 4.2. Observe that there are nodes of five colors. One color corresponds to variables, one to multiplications, one to additions, one to value 0, and one for value 1. The image is from Publication III.

Let us consider the SAT encoding of Example 4.2.

**Example 4.7.** The system of equations in Example 4.2 can be encoded as the following SAT instance:

\[
\neg((x_{11} \land y_{11} \land z_{11}) \lor (x_{12} \land y_{12} \land z_{12}) \lor (x_{13} \land y_{13} \land z_{13}))
\land \neg((x_{21} \land y_{11} \land z_{11}) \lor (x_{22} \land y_{12} \land z_{12}) \lor (x_{23} \land y_{13} \land z_{13}))
\land \neg((x_{11} \land y_{11} \land z_{21}) \lor (x_{12} \land y_{12} \land z_{22}) \lor (x_{13} \land y_{13} \land z_{23}))
\land ((x_{21} \land y_{11} \land z_{21}) \lor (x_{22} \land y_{12} \land z_{22}) \lor (x_{23} \land y_{13} \land z_{23}))
\land ((x_{11} \land y_{21} \land z_{11}) \lor (x_{12} \land y_{22} \land z_{12}) \lor (x_{13} \land y_{23} \land z_{13}))
\land ((x_{21} \land y_{21} \land z_{11}) \lor (x_{22} \land y_{22} \land z_{12}) \lor (x_{23} \land y_{23} \land z_{13}))
\land ((x_{11} \land y_{21} \land z_{21}) \lor (x_{12} \land y_{22} \land z_{22}) \lor (x_{13} \land y_{23} \land z_{23}))
\land ((x_{21} \land y_{21} \land z_{21}) \lor (x_{22} \land y_{22} \land z_{22}) \lor (x_{23} \land y_{23} \land z_{23})).
\]

The system can be converted into CNF by applying the Tseitin transformation [434] with linear overhead.

As described in Section 4.5, these systems have a considerable amount of symmetry. For matrix multiplication tensors, simply the factor of \(r!\) in the size of the automorphism group may be huge, the encoding in CNF-SAT obfuscates the symmetries, and the Tseitin transformation introduces auxiliary variables that may cause trouble for the solver. Fortunately, it is possible to apply domain knowledge of the original problem formulation, and use explicit graph constructions (such as in Figure 4.2) with the adaptive prefix assignment method as a preprocessor to reduce the amount of symmetry. This makes larger problem instances tractable, and the system may be solved using standard SAT solvers. For example, the ranks of the subtensors in Table 4.1 were obtained by trying different values of \(r = 1, 2, \ldots\) until the smallest value of \(r\) for which the system was satisfiable was found.
In this chapter, we discuss the necessity and implications of parallel processing. Section 5.1 gives an overview to the landscape of parallel processing. In Section 5.2, we give an overview of the literature on the parallelization of matrix multiplication, and discuss our results from Publication V where we parallelize Strassen’s matrix multiplication in an alternative basis for multiple-GPU shared-memory systems. In Section 5.3, we discuss the distributed MPI implementation of our adaptive prefix assignment method in Publication III.

5.1 Overview of the parallel landscape

Over the past 60 years, we have seen an exponential increase in the computing power of computers. Historically, there have been two laws that have governed this increase: Dennard scaling [155] and Moore’s law [336, 337]. Dennard scaling roughly states that as transistors (and consequently microprocessors) become smaller, their power-density remains constant. Moore’s law states that the number of transistors in integrated circuits doubles every two years. These laws have enabled us to build microprocessors using ever smaller transistors, providing computing capability without needing more power. Unfortunately, Dennard scaling failed around 2005 (see, for example, [154]), and we seem to be facing the end of Moore’s law (see, for example, [175, 432]), due to limits imposed by the laws of physics.

This change has necessitated the industry to seek more computing power by exploiting parallelism, as it no longer seems to be possible to increase computing power substantially by simply cramming more transistors into microprocessors [48]. For the past decade, we have been witnessing the transition from individual fast general-purpose CPUs to multiple CPUs, and, at one extreme, to the thousands of compute cores on-board GPUs. As an example of the state of art, as of 2018, the Intel Skylake family of CPUs, produced with a 14-nanometer lithography process, offer 2–28 cores per chip [256], and each Nvidia Tesla V100 GPU contains 5120 cores [351]. The industry expects a slowdown in the shrink-
Parallel and distributed processing

ing of the lithography process, with 7 nm predicted to be available in 2019, and 1 nm process predicted to be in production only in 2033 [245].

There are several ways to classify hardware parallelism in microprocessors (see the textbook by Hennessy and Patterson for a more detailed introduction into these topics [231]). A typical distinction would be a hierarchy of instruction-level parallelism (ILP), data-level parallelism (DLP), thread-level parallelism (TLP), and request-level parallelism (RLP). Another well-known classification is Flynn’s [184] taxonomy of microprocessors where they are classified depending on whether they make use of a single or multiple instruction and data streams into the following classes: single-instruction stream, single-data stream (SISD), single-instruction stream, multiple-data stream (SIMD), and multiple-instruction stream, multiple-data stream (MIMD).

ILP refers to the possibility that sequential instructions may not be dependent on one another which allows for the simultaneous execution of multiple instructions. The exploitation of ILP has been a mainstream approach by virtually all relevant microprocessors since the 1980s. Approaches to exploiting ILP include instruction pipelining, superscalar execution, out of order and speculative execution, and branch prediction. With respect to Flynn’s taxonomy, these approaches are applicable to all processors. While ILP is a very attractive form of parallelism from a programmer’s point of view, these techniques may be of limited use beyond a certain point, an observation that was already made by Wall in 1991 [443].

DLP occurs when the same operation is applied multiple times on different data. In terms of Flynn’s taxonomy, SIMD processors are useful in this case. General-purpose CPUs have seen the introduction of ever-wider vector instructions\(^1\) for performing elementary operations on multiple operands simultaneously. At the same time, GPUs have been introduced for general-purpose computing; GPUs are composed of a great number of SIMD processors, and excel at such computations.

TLP and RLP present higher levels of parallelism where the control of execution differs between different instances, whether they be threads or processes, thus requiring MIMD processors. The difference between the two has to do with whether the memory is shared or not; we may view RLP as distributed computing occurring possibly on entirely different physical compute nodes, whereas TLP may be viewed as requiring access to the same resources. Contemporary multi-core CPUs excel at TLP, and, at one extreme, RLP can be seen in warehouse-scale computers (WSC, see [65]) where entirely independent requests are satisfied on hundreds of thousands of servers at a time.

Since parallelism is required for increased efficiency in resource usage, both time and energy, it necessitates us to consider the particularities of parallel processing further.

\(^1\)For example, Intel has extended their x86 instruction set with MultiMedia eXtension (MMX) in 1997, Streaming SIMD Extensions (SSE) in 1999, and Advanced Vector eXtensions (AVX) in 2011. As of 2019, the newest iteration of AVX is the AVX-512 and includes the support of 512-bit wide vector registers.
processing when designing new algorithms [48]. Since it seems that we can no longer expect to obtain exponential speedup from newer processors, we need to address the issues in software. This presents new challenges as we need to address heterogeneous computing of mixed CPU and GPU systems with entirely different kinds of processors numbering in the thousands on individual systems and with several systems possibly interconnected over network. Central problems include the latency and limited bandwidth for transferring data; communication may be more expensive than raw computing. Parallelism also has its own theoretical limitations, such as Amdahl’s law [31] which sets limitations on how great a speedup may be obtained on a program dependent on the amount of non-parallelizable code.

Algorithmic considerations are also required to ensure the reliability of High-Performance Computing (HPC). Reliability is a great challenge to achieving exascale computing\(^2\), due to hardware failures and bugs caused by the increased complexity of software [420]. For a concrete example, assuming the mean time to failure for a processor is 25 years, in a system of 100,000 processors this would imply a mean time between failures of only two hours [66]. Errors corrupting memory are commonplace [66, 426], and are caused by a number of factors, including phenomena outside human control, such as cosmic rays.

### 5.2 Parallel matrix multiplication

Due to the limitations of local memory and simply the attempt to reduce total wall clock time, there has been interest in performing matrix multiplication in a parallel or distributed setting (whether shared or distributed memory). The main challenge in parallel matrix multiplication is the cost of communication which easily exceeds the arithmetic cost [60], and it is precisely the cost of communication that makes it difficult to scale the matrix multiplication algorithms for large computer clusters. In what follows, we will assume we are considering a system of \(P\) processors with \(M\) words of memory each, for computing \(n \times n\) square matrix multiplication.

There are several families of distributed algorithms for the elementary \(\Theta(n^3)\) matrix multiplication with different communication and data layout strategies. The simplest is probably the family of 2D algorithms where the data is laid out equally across the \(P\) processors without any replication in a \(\sqrt{P} \times \sqrt{P}\) grid. That is, each submatrix lies in the fast memory of a unique processor. It is known that, in this setting, the elementary algorithm requires \(\Omega(n^2/\sqrt{P})\) words of communication [237, 258], a result which was extended by Ballard, Demmel, Holtz, and Schwartz [59] to include a large class of algorithms in numerical linear algebra (see also Pagh and Stöckel [362] who generalize Hong and Kung’s [237] result to sparse matrices). This lower bound was already achieved by Cannon [108].

\(^2\)Computing systems capable of exceeding one exaFLOPS (Floating-point Operations Per Second) performance.
we allow \( \sqrt{P} \) copies of input/output matrices to be spread along a cubic grid, the family of 3D algorithms, a lower bound of \( \Omega(n^2/p^{2/3}) \) words of communication holds [258]. This is achieved by [151, 5, 271, 4]. Solomnik and Demmel [421] present a family of algorithms which they call 2.5D matrix multiplication algorithms that variate between the 2D and 3D cases, achieving the known lower bounds. Other older work includes [83, 122, 203, 121]. See also the paper by Schatz, van de Geijn, and Poulson [404] that addresses the systematic design of parallel matrix multiplication algorithms.

The idea of parallelizing Strassen’s algorithm is not new [316, 217, 291, 156, 244]. Ballard, Demmel, Holtz, and Schwartz showed [60] that a parallel implementation of a Strassen-like algorithm\(^3\) with arithmetic complexity \( \Theta(n^{\omega_0}) \) requires that at least \( \Omega(n^{\omega_0}/(PM^{\omega_0/2-1})) \) words are communicated during computation.

In [58], Ballard, Demmel, Holtz, Lipshitz, and Schwartz present an algorithm which they call Communication-Avoiding Parallel Strassen (or CAPS) that reaches this lower bound and is thus communication-optimal. Roughly speaking, the algorithm works by following the Strassen recursion steps in parallel. The first \( k \) steps (depending on the amount of local memory available) are called Depth-First Steps (DFS) where each of the seven subproblems is processed sequentially, among all processors. After \( k \) steps, the subproblem size becomes sufficiently small such that each processor may process the remaining subproblem independently, thus each seven subproblems are processed in parallel. These steps are called Breadth-First Steps (BFS). They show that a more complicated interleaved arrangement of the different kinds of steps can reduce the amount of communication by at most a constant factor, so this simple approach is essentially optimal. In [311], Lipshitz, Ballard, Demmel, and Schwartz show that CAPS performs very well in practice in a distributed-memory supercomputer environment.

In a shared-memory setting, Benson and Ballard [76] present a code-generating tool that automatically produces parallel FMM algorithms, accounting for the shape of the rectangular matrix, following the ideas of CAPS, including a hybrid approach of combining BFS and DFS steps. Huang, Rice, Matthews, and van de Geijn [242] also present an automated approach for generating parallel FMM algorithms for shared-memory systems. In both cases, the results are practical and actually implemented, and are experimentally shown to be useful.

Finally, energy efficiency is important, as the energy consumed translates directly to costs, both monetary and environmental – this is not merely a hardware issue but also an algorithmic issue. Demmel, Gearhart, Lipshitz, and Schwartz [152] present a distributed matrix multiplication algorithm that achieves perfect strong scaling in terms of energy; that is, by using all available memory, the algorithm can be executed on multiple nodes without a significant increase in the total amount of energy spent.

\(^3\)An algorithm that shares the recursive structure of Strassen’s algorithm but possibly uses a different base tensor.
Parallel and distributed processing

In Publication V, we address the parallelization of a variant of Strassen’s algorithm in the alternative basis [282] for binary matrices in a heterogeneous shared-memory computing environment, and compare the empirical performance on three systems: the Nvidia DGX-1 [350] with eight Nvidia Tesla V100 SXM2 accelerators [351], and systems with four Nvidia Tesla P100 accelerators [349] and four Nvidia Tesla K80 accelerators [348].

In our work, we draw on the ideas of CAPS to design an implementation that is aggressively parallelizable on the GPU architecture. At the heart, is a finely crafted kernel for performing the elementary $64 \times 64$ binary matrix multiplication over a single warp$^4$ of 32 threads. For larger inputs, the base block is $128 \times 128$ where a single layer of Strassen recursion is performed and seven elementary $64 \times 64$ multiplications are performed in parallel, using the register files of the GPU streaming multiprocessors only without any intermediate memory accesses. Larger inputs are processed in parallel by performing two levels of Strassen recursion at once in the registries. Since applying the Strassen recursion, the transforming of $4 \times 4$ subarrays into $7 \times 7$ subarrays by a twofold application of Equation (3.16), takes a large amount of auxiliary memory, in the largest cases that can still fit completely in the memory of the GPU, the Strassen steps are carried out sequentially. One can see here an analogy to the BFS and DFS steps of CAPS.

In our experiments, we consider up to one-tebibit input matrices. Such binary matrices take 128 GiB of memory each, and, as such, cannot fit in the GPU memory of 16 GiB per accelerator. Therefore, we apply the structural tensor decomposition to create sufficiently small input matrices by XORing the appropriate submatrix blocks in host DRAM, perform the multiplication on the GPU, and then integrate the result by XORing in the host DRAM. While this procedure is easily parallelized, we found that the host DRAM bandwidth becomes a bottleneck. Nevertheless, we were able to achieve a speedup of over 2.2: a parallelized elementary implementation took 1880 seconds to process the one-tebibit input, whereas the Strassen-like, alternative-basis algorithm performed in 821 seconds. In terms of effective bit operations per second$^5$, we saw an increase from 1.23 Pbop/s to 2.8 effective Pbop/s. As to energy consumption, we saw a decrease of total energy from 1.83 kWh to 0.80 kWh, which translates to a decrease from 2.85 pJ/bop to 1.25 pJ/effective bop.

$^4$Warp is Nvidia terminology which is similar to a thread of 32 SIMD operations in CPU terms. See, for example, [231].

$^5$The number of bit operations a elementary algorithm would have to perform to compute the product in the time, that is, for computing $n \times n$ product in time $t$, the effective bit operations are computed by $(2n^3 - n^2)/t$.
5.3 Distributed implementation of adaptive prefix assignment

In Publication III, we present an implementation of the adaptive prefix assignment method (see Section 4.4) that works in a distributed computer cluster environment. The implementation is written in C++ [257] and works using the OpenMPI [196] implementation of the Message Passing Interface (MPI) [332, 356, 221].

The procedure (P') in question is, in essence, a depth-first search. The search begins from an empty assignment, and the assignment is extended level by level in depth-first order. Once the leaf-level of the recursion tree (determined by the parameter $k$) is reached, if the assignment corresponding to the leaf passes the test, the partial assignment is returned. Then, the search backtracks and continues at a higher level. In a sequential implementation, a natural way to do this is by the means of a stack that stores unvisited partial assignments. When a partial assignment is popped from the stack, the tests are performed, and if the assignment is accepted, it is normalized and all child assignments (all possible extensions) are generated and stored on the stack.

Any partial assignment in the search tree is independent of its siblings; no communication is required between the different branches of the recursion, except to maintain information about which branches have not yet been visited. This enables very high-level parallelization of the algorithm: whenever we encounter a new accepted partial assignment and generate its possible child extensions, we can redistribute the child tasks to other nodes in the computer cluster. Any node in the computer cluster only needs the initial graph to initialize itself, and the only information that needs to be conveyed over the network during computation are the partial assignments from which the search resumes since no exchange of information is required to determine if the assignment should be accepted or not.

We implemented work sharing in a very straightforward way, such that one of the nodes works as a master node and other nodes as worker nodes. The master node maintains a remote stack of unvisited assignments and divides work to workers. We also enabled a partitioning scheme to require that certain workers only work with assignments of prescribed length. Already this simple approach was was sufficient to show that a near-linear speedup is obtainable in practice with our algorithm.
6. Similarity search

In this chapter, we review literature on similarity search and present our results from Publications I and II. In Section 6.1, we formulate the problem with variations. Section 6.2 introduces the curse of dimensionality that makes the problem hard. Section 6.3 presents dimensionality reduction methods for coping with the curse of dimensionality. In Section 6.4, we review literature on Locality-Sensitive Hashing (LSH), a widely used framework for similarity search in high dimensions. In Section 6.5, we present how to list outlier correlations using fast matrix multiplication, the key result from Publication I. Finally, in Section 6.6, we introduce the notion of derandomization and describe how we derandomized the algorithm for outlier correlations, the main result of Publication II.

6.1 Problem formulation

Similarity search is the search for objects that are similar in a database. The numerous applications include genome-wide association studies [320, 447], document duplicate detection and web clustering [99, 101, 100, 319, 109, 307, 308], face recognition [229], computational biology [171], computational chemistry [451], recommender systems [147, 92, 389], NLP [436], and content-based image search [466].

A number of measures for similarity exist, particularly metrics, such as Hamming distance, Euclidean distance, Earth-Mover Distance (EMD), edit distance, and distances between probability distributions such as Kullbeck-Leibler divergence.

The problem of similarity search can be formulated as querying a database for a match to a query object, or as batch processing where pairwise matches are sought between two databases of objects. Typically, we would deal with vectors of fixed length (say, d), the components of which could have continuous (for example, vectors in \( \mathbb{R}^d \)) or discrete values (for example, in \( \{0, 1\}^d \)). A specific variant of such similarity search is the Nearest Neighbor Search (NNS).
**Problem 6.1** (Nearest Neighbor Search). Given a dataset $P$ of $n$ points in a metric space $(X, D)$ with distance function $D$, build a data structure that, given a query point $q \in X$, can be used to find $\arg\min_{p \in P} D(p, q)$.

In other words, the data structure should be applicable to find a point that is at least as close to the query point as any inequal point in the dataset. In older literature, this is also known as the post office problem [286, 195, 125, 126]. A variant of NNS is the $(c, r)$-Approximate Near Neighbor (ANN) problem.

**Problem 6.2** ($(c, r)$-Approximate Near Neighbor [37, Definition 1.1]). Given a dataset $P$ of $n$ points in a metric space $(X, D)$ with distance function $D$, build a data structure that, given a query point $q \in X$, if there exists a point $p \in P$ such that $D(p, q) \leq r$, then return any point $p' \in P$ such that $D(p', q) \leq cr$.

In other words, in the approximate version, we require that there is only one-sided approximation error: no point is returned if no point exists within the specified $r$-ball around the query point; however, if such a point exists, we allow the returned point to be just outside the ball, within the larger ball of radius $cr$. Problem 6.2 is also known as point location in equal balls in older literature (for example, [254]).

Early work on the approximate version of the problem includes Bern [81]. This formulation is used particularly in the context of LSH (see Section 6.4). In many cases, we deal with randomized algorithms; in such cases, if we were rigorous, the definition of Problem 6.2 ought to be refined to include a failure probability, but we tacitly ignore this parameter (see, for example, [37, Definition 1.2]). The failure probability enables Monte Carlo algorithms that may fail to report any point at small probability, despite there being a database point within the $r$-ball around the query point. Finally, as an interesting connection, Indyk showed [251] that, over the $L_\infty$ metric and $c < 3$, the superset query or partial matching problem (see [286, Section 6.5] and [392, 390]) is reducible to $(c, r)$-ANN.

An important variation to the theme is the offline version of the nearest neighbor search, the closest pair problem [37]. Given a pointset $P$ in a metric space, find the pair of points that minimizes their distance. This problem can be solved with a datastructure for NNS. Like the online version, the problem also has an approximate analog, the $(c, r)$-Approximate Close Pair problem.

**Problem 6.3** ($(c, r)$-Approximate Close Pair [37, Definition 5.1]). Given a dataset $P \subseteq X$ of $n$ points in a metric space $(X, D)$ with distance function $D$, if there exist distinct points $p^*, q^* \in X$ with $D(p^*, q^*) \leq r$, find a pair of distinct points $p, q \in P$ such that $D(p, q) \leq cr$.

Like in the case of $(c, r)$-ANN, Problem 6.3 should be augmented with a failure probability to enable Monte Carlo algorithms which we tacitly ignore for the time being (in [37], success probability of 2/3 is used).

Furthermore, the offline version can be varied to that of batch queries. That is, given a set of points, for each point, list the point closest to it. In an all-to-all case,
this corresponds to performing $n$ NNS queries simultaneously. The problem can be viewed as either monochromatic or bichromatic: in the monochromatic case, we want to match points to other points within the pointset; in a bichromatic case, we are given two sets of points and want to match points from one set to points in the other set.

In Publications I and II, we consider the following variant of the problem.

**Problem 6.4 (Outlier Correlations (Publication II, Problem 1)).** Given as input two sets $X, Y \subseteq \{-1,1\}^d$ with $|X| = |Y| = n$ and two thresholds, the outlier threshold $\rho > 0$ and the background threshold $\tau < \rho$, output all outlier pairs $(x, y) \in X \times Y$ with $|\langle x, y \rangle| \geq \rho d$, subject to the assumption that at most $q$ of the pairs $(x, y) \in X \times Y$ satisfy $|\langle x, y \rangle| > \tau d$.

Problem 6.4 is a bichromatic batch formulation on $\pm 1$-vectors. It suffices to address only the bichromatic case since there exist a reduction from a monochromatic instance with only multiplicative polylogarithmic cost in $n$ (see Publication I, Remark 1.2). If we scale the vectors by the dimension, the problem addresses angular distance (or cosine similarity) of the vectors on the unit sphere; that is, the angle between two vectors. There is also a straightforward connection to Hamming distance via $\langle x, y \rangle = d - 2d_H(x, y)$. Furthermore, if we allow for randomization, continuous-valued vectors on the Euclidean hypersphere can be quantized into $\pm 1$-valued vectors by random hyperplanes with an expected loss of $\lambda \rightarrow 1 - \frac{2}{\pi} \arccos \lambda$ for the parameters $\lambda = \rho, \tau$ (see [438, Algorithm 4]). The difference in the parameters $\rho$ and $\tau$ quantifies the gap between background “noise” and the interesting outliers.

In database literature, a similar problem formulation comes in the form of the similarity join [361], and particularly the inner product variety [318, 401, 43, 119, 10, 361, 324, 124]. The similarity join seeks to list the pairs (of rows from two tables in a database) that are similar with respect to some similarity measure; that is, such that they are close.

Finally, a particular variety of the closest pair problem to consider is the Light Bulb problem due to L. Valiant [440] (see also [438, Definition 1.1]).

**Problem 6.5 (Light Bulb (Publication II, Problem 6)).** Given as input a parameter $0 < \rho < 1$, and a set of $n$ vectors in $\{-1,1\}^d$ such that one planted pair of vectors has inner product at least $\rho d$ in absolute value, and all other $n - 2$ vectors are chosen independently and uniformly at random, find the planted pair among the $n$ vectors.

Instances of Problem 6.5 are random, but fundamentally it can be solved as if it were an instance of the Hamming closest pair problem; with a sufficiently large $d = \Omega(\log n)$, the planted pair is the one with highest correlation, or equivalently, smallest distance, with high probability. The name of the problem comes from the idea of observing $n$ light bulbs for whether they are on or off at $d$ discrete points in time, with all but one pair of light bulbs blinking independently at random. Problem 6.5 can also be seen as the problem of learning a parity function of weight 2 in the presence of noise.
Similarity search

Problems 6.1, 6.2, and 6.3 assume that the points lie in a metric space. The most commonly studied metric spaces are probably vector spaces equipped with Euclidean (or $L_2$) distance, $L_1$ or taxicab distance, or Hamming distances, or $L_\infty$ distance, Minkowski metrics, the angular distance (or equivalently cosine similarity), EMD, set similarity metrics, Hausdorff metrics, distances derived from color histograms, the edit distance, the inner product, the Frechet distance, the Mahalanobis distance, and the Kullback-Leibler divergence.

For an overview, see for example Shakhnarovich, Darrel, and Indyk, and the recent survey on high-dimensional NN by Andoni, Indyk, and Razenshteyn. For the relationship between different, closely related proximity problems, see for example [210, 455].

6.2 Curse of dimensionality

For on-line query problems (Problems 6.1 and 6.2), the immediate solution is a linear scan through the database. For off-line problems (Problems 6.3, 6.4, 6.5), the obvious solution is to compute all possible pairwise distances which takes a quadratic time. It is thus natural to seek algorithms that take $O(n^{1-\epsilon})$ or $O(n^{2-\epsilon})$ time for some constant $\epsilon > 0$ for query and batch problems, respectively, without taking too much space for the data structures that need to be built.

Querying for exact matches is easy: we can simply sort our database of $n$ vectors in lexical order in $O(dn \log n)$ time, and search the sorted database for an exact match in logarithmic time. The problem becomes considerably more difficult when we seek inexact matches, as the entire geometry of the problem changes, as was already observed by Minsky and Papert [334, Section 12.7.8].

Efficient solutions are known to exist for low dimensions $d = O(1)$, in practice up to smallish constants (for example, up to 10 in [448]). The solutions are typically based on space-partitioning data structures [77, 190, 223, 195, 70, 78, 463, 7, 79, 197, 262, 368] (see also [120, 399]), such as Voronoi diagrams [50, 167, 222, 13, 12], and enable $O(\log n)$ query times [125, 330]. Approximate approaches include both randomized [46, 126] and deterministic [47, 114] with logarithmic query times. In [343], an algorithm is presented that requires queries to be made sufficiently close to pre-existing points. In [290], a deterministic and dynamic data structure supporting updates in addition to queries is presented for bounded dimensions. Quite recently, Abdelkader, Arya, da Fonseca, and Mount [2] give an algorithm for $(1+\epsilon, r)$-ANN under various non-Euclidean
metrics that offers a query time of $\mathcal{O}(\log \frac{n}{\epsilon})$ and requires $\mathcal{O}((n/e^{d/2})\log(1/\epsilon))$ storage.

However, abandoning the assumption of constant dimensionality, the data structures used by such algorithms tend to scale exponentially in $d$ (for example, [284]). The dependence on the underlying dimension of the points and the degeneration into sequential search is known as the curse of dimensionality. The curse of dimensionality is an intrinsic property of high-dimensional geometry; the qualitative meaning of distance becomes unintuitive as the contrast between small and large distances decays, pairwise distances tend to become strongly concentrated, and almost everything is orthogonal in high dimensions [84, 234, 6, 186].

Already Minsky and Papert [334, Section 12.7] posed the tradeoff between query time and space requirement as an open problem for Hamming and Euclidean distances, and conjectured that a large portion of the database must invariably be read. Further research has shown that such tradeoffs do indeed exist [278, 293, 39, 123], and subject to some assumptions, the conjecture seems to hold since some of these tradeoffs are tight (especially [39] which is tight for data-dependent LSH; see Section 6.4).

Under certain assumptions, it seems that the curse of dimensionality is inevitable. Assuming the cell-probe model of Yao [461], Borodin, Ostrovsky, and Rabani [94] showed that NNS requires either superpolynomial storage space or the worst-time query takes linear time; Barkol and Rabani [64] and Jayram, Khot, and Rabani [263] improved on this lower bound. Chakrabarti and Regev [112] showed that with a polynomial number of cells of size $d^{O(1)}$, the query time is $\Omega(\log log d / \log log log d)$ for the approximate version of the problem. Dubiner showed that his $\tilde{O}(n^{2/(\rho+1)})$ algorithm for the light bulb problem is essentially optimal for a large class of algorithms, with an optimality-implication for LSH. Andoni, Indyk, and Pătraşcu showed that, for Hamming $(1+\epsilon)$-ANN, constant query time implies $n^{\Omega(1/\epsilon^2)}$ space [36]. Further lower-bounds are also known that assert the optimality of known LSH approaches [338, 352, 34, 39, 115] (see also Section 6.4).

The overall problem has a connection to the Strong Exponential Time Hypothesis (SETH) [246, 106] and the Orthogonal Vectors Conjecture (OVC). SETH asserts the non-existence of an $\tilde{O}(2^{ck})$-time algorithm for the $k$-SAT problem with $c < 1$ as $k \to \infty$. The ORTHOGONAL VECTORS problem asks if, given two sets of $n$ vectors of dimensionality $d$, we can find one vector from each set such that the two vectors are orthogonal. OVC then asserts that there does not exist $\delta > 0$ and a randomized algorithm that would solve ORTHOGONAL VECTORS for all $c \geq 1$ with $d \leq c \log n$ in time $\tilde{O}(n^{2-\delta})$. It is known from a reduction to ORTHOGO-

1 Subject to mild assumptions about the data and its distribution.
2 This applies to a wide range of metrics, but the effect is particularly pronounced with the usual $L_p$-norms.
3 This is also known as a generalization of Hopcroft’s problem, originally stated in the following form: “Given a set of $n$ points and $n$ lines in the plane, does any point lie on a line?” [174, 456]
NAL VECTORS that unless SETH is false, there cannot be an $\tilde{O}(n^{2-\epsilon})$ algorithm for OUTLIER CORRELATIONS with $\epsilon$ bounded from below by a positive constant as $|\rho - \tau| \to 0$ [247, 452, 456]. Furthermore, Ahle, Pagh, Razenshteyn, and Silvestri [10] showed a conditional lower bound for the existence of a subquadratic algorithm for the Inner Product Similarity Join (IPS-join) with a sufficiently large approximation factor, or equivalently a subquadratic algorithm for OUTLIER CORRELATIONS, on the assumption of OVC; that is, if there were such an algorithm, then OVC (and subsequently SETH) would be false. See [457] for more on complexity results conditioned on SETH, 3-SUM, and All-Pairs Shortest Paths (APSP), among others.

The existence of efficient general ANN data structures in high dimensions is addressed, for example, by Andoni, Nguyen, Nikolov, Razenshteyn, and Waingarten [41] who show the existence of data structures for $\text{poly}(\log \log n)$-approximate NNS with sublinear query time and essentially linear space in the case of symmetric norms; however, the result does not generalize to arbitrary norms. This suggests that the metric structure of the space in question may have connections to the difficulty of the problem.

### 6.3 Dimensionality reduction

An attempt can be made to mitigate the curse of dimensionality by dimensionality reduction since it is not the nominal dimensionality of the points that matters but the structure of the point set [288]. A key result in this field is the Johnson-Lindenstrauss (JL) Lemma [270]. Lemma 6.6 gives one possible formulation (following [3]).

**Lemma 6.6** (Johnson-Lindenstrauss [270, 3]). For any $\epsilon > 0$ and positive integer $d$, for every set of $n$ points $P \subseteq \mathbb{R}^d$, there exists $f : \mathbb{R}^d \to \mathbb{R}^k$ such that for all $u, v \in P$

$$(1 - \epsilon)\|u - v\|^2 \leq \|f(u) - f(v)\|^2 \leq (1 + \epsilon)\|u - v\|^2,$$

such that $k = \mathcal{O}(\epsilon^{-2} \log n)$.

Lemma 6.6 states that any point set in a high-dimensional Euclidean space can be embedded in a lower-dimensional Euclidean space, such that the distances are preserved up to $(1 + \epsilon)$-approximation error, with dimensionality at most $\mathcal{O}(\epsilon^{-2} \log n)$. This can be done efficiently in practice [3, 306, 278], including derandomized constructions [170, 274, 143]. Using ideas inspired by the FFT, Ailon and Chazelle present a variant of the transform that can be computed in $\tilde{O}(d \log d + \min(d \epsilon^{-2} \log n, \epsilon^{-2} \log^3 n))$ time, called Fast Johnson-Lindenstrauss Transform (FJLT). Lemma 6.6 is known to be tight up to constant factors [264], and, in fact, Larsen and Nelson [297, 298] show that there are point sets that require $\tilde{O}(\epsilon^{-2} \log n)$ dimensions for the JL guarantees to hold, thus showing that Lemma 6.6 is optimal. Although Lemma 6.6 itself only addresses Euclidean spaces, it can be applied to other metrics as well by embedding them in Euclidean
Similarity search

spaces [310, 170]. However, it has been shown that there is no analog of the JL Lemma for Hamming spaces [98], so different approaches must be taken to reduce dimensionality (see, for example, [292] for an approach in Hamming space).

A number of other results regarding the embeddings of metric spaces into normed spaces of small dimension with small distortion are known, for example, into $(\mathbb{R}^d, L_\infty)$ with $d = \tilde{O}(n^{1/\rho})$ [322], for embedding general $n$-point metric spaces into Hilbert spaces [95], for embedding Hausdorff metric spaces into $(\mathbb{R}^d, L_\infty)$, for explicit embedding of $L_2$ into $L_1$ [253], and for embedding continuous-valued data into binary vectors (values in $\{0, 1\}$ or $\{-1, 1\}$) [464]. Interestingly, in terms of the related notion of sketching, particularly in the context of estimating the distance between points in a metric space, Andoni, Krauthgamer, and Razenshteyn show [38] that normed spaces admit efficient sketches if and only if they can be embedded into $L_1 - \epsilon$ (or $L_1$ if the norm is closed under sum-product).

There exists a number of other readily-available dimensionality reduction techniques, such as Principal Component Analysis (PCA, [193]), that can be applied, for example, in the context of image analysis, but which may fail to give theoretical guarantees with respect to the distortions incurred by the transformation. An extensive literature exists covering areas such as discretization into similarity-preserving (random) binary codes [433, 346, 211, 347], and compression of a high-dimensional space into products of quantized subspaces [268].

### 6.4 Locality-sensitive hashing

A key application of the Johnson-Lindenstrauss Lemma (Lemma 6.6) is the breakthrough result of Indyk and Motwani [254, 224] (see also Indyk’s thesis [250]) that established the LSH line of research for $(c, r)$-ANN. LSH is a framework that is theoretically grounded and applicable in practice [209, 228, 147, 319, 430, 34, 371]. Let us define the notion (following [37]).

**Definition 6.7** (Locality-Sensitive Hashing (LSH, [37, Definition 2.5])). Given a metric space $(X, D)$, scale $r > 0$, approximation $c > 1$, and a finite set $U$, a distribution $\mathcal{H}$ of maps $h : X \rightarrow U$ is called $(r, cr, p_1, p_2)$-sensitive if the following hold for any $x, y \in X$: (i) if $D(x, y) \leq r$, then $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \geq p_1$, and (ii) if $D(x, y) > cr$, then $\Pr_{h \in \mathcal{H}}[h(x) = h(y)] \leq p_2$. The distribution $\mathcal{H}$ is called an LSH family and has quality $\rho = \rho(\mathcal{H}) = \frac{\log 1/p_1}{\log 1/p_2}$.

Indyk and Motwani use an LSH family to construct a number of hash tables as the data structure. Their algorithm is randomized and the properties of the LSH family provide control on error. If the underlying space has dimensionality $d$, the Indyk-Motwani algorithm achieves roughly $\tilde{O}(dn^\rho)$ query time, and requires $\tilde{O}(n^{1+\rho} + dn)$ space where the value of $\rho$ depends on the underlying metric.
For $L_1$, Indyk and Motwani achieve $\rho = 1/c$ by a random projection onto a single coordinate as the hash function. It is known that for the $L_1$ (and Hamming) metric $\rho \geq \frac{1}{c} - o(1)$ [338, 352], so the Indyk-Motwani construction is essentially optimal. A similar approach is adopted by Kushilevitz, Ostrovsky, and Rabani [292] who also meet the bound for the Hamming metric.

The Indyk-Motwani construction can be used for $L_2$ as well by embedding the Euclidean space onto the $L_1$-space. However, better runtimes are achievable by using dedicated constructions for the Euclidean space (and other $L_p$-norms). Datar, Immorlica, Indyk, and Mirrokni [148] achieve $\rho < 1/c$ for $L_p$-spaces with $p \in (1,10)$. In [33], Andoni and Indyk improved this to $\rho = 1/c^2 + o(1)$ for the Euclidean metric. This is now known to be optimal [338, 352]. In the same paper [33], Andoni and Indyk also presented a lattice-based LSH which is known to match the same $\rho$ [115].

These constructions are all data-independent. This means that the particular geometric properties of the point set in question are not employed when constructing the data structure. Other data-independent LSH-families include [99, 101, 116]. The lower-bound results for data-independent LSH [338, 352] may be overcome by making use of the geometric information regarding the point set, that is, by making the construction data-dependent. In a series of papers, Andoni, Indyk, Laarhoven, Nguyen, Razenshteyn, and Waingarten [35, 42, 39] developed a framework for data-dependent LSH that achieves $\rho \leq \frac{1}{2c^2-1} + o(1)$ for the Euclidean distance and $\rho \leq \frac{1}{c^2} + o(1)$ for the Hamming distance. Although the Spherical LSH algorithm of [42] is applied on vectors in the unit sphere, the algorithm works on a general Euclidean space by reduction. In [34], Andoni, Indyk, Laarhoven, Razenshteyn, and Schmidt describe Cross-polytope LSH, a practical data-dependent variant of the Spherical LSH. In [39], Andoni, Laarhoven, Razenshteyn, and Waingarten describe an optimal tradeoff between query time and space, establishing (with [43]) that the upper bounds obtained on $\rho$ are essentially tight. In [40], Andoni, Naor, Nikolov, Razenshteyn, and Waingarten present a data-dependent LSH data structure for a general $d$-dimensional normed space in the cell-probe [461] model.

The algorithms above are of the Monte Carlo variety: they may fail to report anything even if there is a point in the $r$-ball around the query point, and there is no efficient way to determine if the result is correct. Such query instances are known as false negatives. While general deterministic polynomial-time algorithms are not known for constructing a data structure with zero error for $(c,r)$-ANN [37], some steps towards reducing the dependence on random bits have been taken. Building on Kushilevitz, Ostrovsky, and Rabani’s work [292], Indyk [249] presented a deterministic algorithm for $(c = 3+\epsilon,r)$-ANN that avoids false negatives for any $\epsilon > 0$ with $d(1/\epsilon)^{O(1)}$ query time and $dn(1/\epsilon)^{O(1)}$ space. In the same paper, he presented a Las Vegas algorithm for $(c,r)$-ANN that is allowed to return a don’t-know symbol, hence avoiding false negatives. Indyk also presented a deterministic 3-approximate algorithm for the $L_\infty$ metric [251]. Pagh [359, 360] and Ahle [8] presented a Las Vegas analog of [254] that avoids
false negatives, achieving query time of $dn^{1/c+o(1)}$ and space $dn^{1+1/c+o(1)}$ for arbitrary $c > 1$ (see [371] for practical results over the Hamming metric). In [215], similar false-negative-avoiding work is presented in the context of distance-sensitive Bloom filters. Pacuk, Sankowski, Wegrzycki, and Wygocki reported on an LSH algorithm avoiding false negatives for an $L_p$ metric with $p \in [1, \infty)$ with sublinear query time (for $L_2$) if $c = \omega(\sqrt{d})$. This was improved on by Sankowski and Wygocki [400], and Wygocki [460]. Finally, very recently, Wei [449], building on aforementioned work, presents a Las Vegas version of the data-dependent LSH, matching the asymptotic runtimes and tradeoffs of Andoni, Laarhoven, Razenshteyn, and Waingarten [42, 39].

One of the shortcomings of the traditional LSH framework is the existence of a number of parameters, the optimal values for which may be difficult to predict. This is addressed in [67] by Bawa, Condie, and Ganesan who present a technique called LSH Forest that allows the traversal of LSH trees, automatically tuning the parameters. In [164], Dong, Wang, Josephson, Charikar, and Li provide a performance model for predicting the performance of the multi-probe LSH [317]. In [417], Slaney, Lifshits, and He present an algorithm for optimizing the LSH parameters. In [5], Ahle, Aumüller, and Pagh present a parameter-free variant of the LSH for reporting points within a radius of a query point.

A number of variations and generalizations of the LSH framework have been presented, such as Panigrahy’s [367] entropy-based hashing, the multi-probe LSH [317] that improves on the space requirement, Distance-Sensitive Hashing (DSH) [49] which generalizes the LSH from a pair of probabilities controlling the collision probability of “near” and “far” points to a soft probability as a function of distance between the objects (including the possibility of asymmetric hashing functions), Locality-Sensitive Filters (LSF) [68, 293, 123], Asymmetric LSH (ALSH) [413], Parallel LSH (PLSH) [430], and Cross-polytope LSH [34].

### 6.5 Listing outlier correlations via fast matrix multiplication

In Publications I and II, we consider the OUTLIER CORRELATIONS problem, and its special case, LIGHT BULB (Problems 6.4 and 6.5, respectively). Both of these problems are off-line batch versions of similarity search, and we explicitly addressed binary-valued input. Following G. Valiant [438], we also consider PARITY WITH NOISE in Publications I and II, but omit the details here.

As was mentioned in Section 6.1, LIGHT BULB is due to L. Valiant [440]. Early work on the problem includes that of Paturi, Rajasekaran, and Reif [369]. Other work on the problem include Dubiner’s bucketing coding [166], and May and Ozerov’s algorithm for decoding random linear codes [323]. Alman and Williams [19, 16] use probabilistic polynomials to construct a randomized algorithm that computes the batch version of the Hamming NNS, that is, the exact Hamming distance to the nearest neighbor for every point in the set, in truly subquadratic time for $d = \Theta(\log n)$. 
Our work builds on the work of G. Valiant \cite{438}. In Publication I, we improve on Valiant’s randomized algorithm, and, in Publication II, we derandomize Valiant’s algorithm. We will now briefly describe Valiant’s algorithm, and our contribution.

Valiant’s algorithm works in four phases: (i) amplification of inner products by tensoring, (ii) aggregating the vectors into buckets, (iii) approximate detection by matrix multiplication, and (iv) exact brute force listing. We will now describe each of these phases separately.

Let \( x, y \in \{-1,1\}^d \) be arbitrary data vectors from the two input sets \( X, Y \) of size \( n \). We may view the sets \( X, Y \) as \( \{-1,1\} \times n \times d \) matrices. In phase (i), Valiant applies the following identity: \( \langle x, y \rangle^p = \langle x^\otimes p, y^\otimes p \rangle \). That is, we observe that computing the inner product between the the \( p \)th Kronecker powers of the vectors is equal to amplifying the inner product to the \( p \)th power. That is, assuming the inner product (or correlation) is normalized to the range \([-1,1]\), the inner product decays in absolute value, but with smaller values approaching zero much faster.

As a downside, the length of the vectors grows to \( d^p \) which is prohibitively long; Valiant handles this by taking a uniformly random sample of size \( s \) for a sufficiently small \( s \). It follows from the Hoeffding bound \cite{236} that the expected value of the inner product of the downsampled vectors is sufficiently concentrated that the full vectors need not be computed.

In phase (ii), Valiant partitions the two input sets of vectors into \( n/t \) buckets of size \( t \). A random sign \( \pm 1 \) is assigned for each vector in the bucket, and the vectors are then aggregated, that is, summed after multiplying with the sign. We may consider the output of this stage as two \( s \times (n/t) \) matrices \( \tilde{X}, \tilde{Y} \). In phase (iii), these two matrices are simply multiplied together \( C = \tilde{X}^\top \tilde{Y} \). Due to concentration properties, the entry \( c_{ij} \) in the product matrix is small with high probability unless one of the outlier pairs is found in the \( i \)th and \( j \)th buckets of \( X \) and \( Y \), respectively. Thus, in phase (iv), Valiant simply computes all \( t^2 \) pairwise inner products between the vectors in the buckets whose values in the product matrix exceed a threshold, and lists the outliers.

As a result, Valiant obtains an algorithm that outputs the outlier pairs with probability \( 1 - o(1) \) in time
\[
\tilde{O}(qdn^{1/o} + n^{5/6 + o\log, \rho}) \leq \Theta(qdn^{0.7} + n^{1.62 + 3\log, \rho}),
\]
where \( o < 2.3728639 \) is the exponent of matrix multiplication (see Section 3.6), and as a corollary, for \( d = \Omega(\log n / \rho^2) \), solves LIGHT BULB for any constant \( \epsilon > 0 \) in time
\[
\tilde{O}(n^{5/6 + \epsilon}) \leq \Theta(n^{1.62}).
\]

Valiant also gives further corollaries for the closest pair problem, learning parities and \( k \)-juntas with noise, and Disjunctive Normal Form (DNF) formulae\(^4\).

Our main contribution in Publication I is a more efficient sampling scheme for phase (i) in Valiant’s algorithm which enables us to further use FMM. Instead of

\(^4\)For further discussion about the relationship between these problems and reductions, see \cite{179}.
taking a uniform sample over the \( d^p \) entries in the vector, we use sampling with a Cartesian structure. That is, if we let the multiset \( I \) to be the set of indices (drawn from \([d^p]\)) that we preserve in our sample, we will assume that the multiset has the structure of \( I = I_1 \times I_2 \) which will then enable us to construct our subsampled vector by performing \( s^{1/2} \times t \times s^{1/2} \) matrix products \( n/t \) times (see Publication I, Section 4.2 for details). We show that this sampling scheme preserves sufficient concentration properties, while the use of FMM makes the procedure faster. As a result, we obtain a general-purpose theorem with a tradeoff parameter that can be used to generate a family of algorithms for OUTLIER CORRELATIONS. As corollaries, we improve on Valiant’s runtime in Equation (6.1) and obtain randomized algorithms with running times of

\[
\tilde{O} \left( n^{1 - 2\omega/3} + qdn^{2(1 - \log_2 \rho)} \right),
\]

and

\[
\tilde{O} \left( n^{2\omega(1 - \log_2 \rho)} + qdn^{2\omega(1 - \log_2 \rho)} \right),
\]

where \( \alpha > 0.31389 \) is the exponent for quadratic rectangular matrix multiplication (see Section 3.7). In the case of LIGHT BULB, we improve on the runtime in Equation (6.2) to

\[
\tilde{O} \left( n^{\frac{2\omega + \epsilon}{\rho - \frac{\epsilon}{\pi}} + \frac{n\omega - \frac{\epsilon}{\pi}}{2}} \right) \leq \tilde{O}(n^{1.582}).
\]

### 6.6 Derandomizing the algorithm for outlier correlations

Derandomization refers to the process of removing the dependence on the random bits in a randomized algorithm; that is, turning the algorithm into a deterministic one. In other words, we seek a deterministic construction for selecting a suitable choice of the random bits, without sacrificing too much time. This may be useful, for instance, to make guarantees of success and runtime; the former is of particular importance in the case of Monte Carlo algorithms whereby it may be impossible to tell if the result of the algorithm is correct or not.

Derandomization is closely tied to the research of Pseudo-Random Generators (PRGs; for a survey, see [437], for some recent work, see, for example, [331, 118]). PRGs seek to construct a long sequence of bits from a small source of random bits (the seed), such that the sequence can be used to fool some class of random functions. That is, it should be impossible for an observer to use a statistical test to distinguish between actual random output and output generated with the pseudorandom sequence, up to some constant error.

One tool for derandomization is the explicit construction of expander graphs. Informally, expander graphs [238] are graphs that are sparse but well-connected; that is, there is a sequence of (regular) graphs whose degree grows slowly, but every not-too-large set of vertices has a large neighborhood [437]. It can be
shown by a probabilistic argument that regular random graphs make good expanders. However, for the purposes of derandomization, we are interested in explicit constructions, graphs that can be constructed deterministically. The best possible such graphs with explicit constructions are Ramanujan graphs [313, 238] that satisfy the Alon-Boppana bound [21, 345, 191]. Expander graphs have seen applications in error-correcting codes [416], compressed sensing [261], cryptographic hashing [117], and computational complexity [162, 387], among others. For examples of derandomization work using similar expander-based approach, see for example Murtagh, Reingold, Sidford, and Vadhan [341] who give a $\tilde{O}(\log n)$-space derandomized algorithm for solving undirected Laplacian systems [425, 370] using the derandomized graph-squaring construction of Rozenman and Vadhan [395] (cf. the randomized $\tilde{O}(\log n)$-space result of Doron, Le Gall, and Ta-Shma [165]).

Other possible tools for derandomization include $(n, k)$-universal sets\(^5\) [342], the method of conditional probabilities\(^6\) [22], exhaustive search of a probability space with limited independence [314, 22], and superimposed codes\(^7\) [248].

In Publication II, we derandomize Valiant’s algorithm [438] by using expander graphs to construct objects called correlation amplifiers.

**Definition 6.8** (Correlation amplifier, Publication II, Definition 2). Let $d$, $D$, and $P$ be positive integers with $p$ even, and let $0 \leq \tau \leq 1$ and $\gamma \geq 1$. A function $f : \{-1,1\}^d \to \{-1,1\}^D$ is a correlation amplifier with parameters $(d, D, p, \tau, \gamma)$ if for all pairs of vectors $x, y \in \{-1,1\}^d$, we have (i) if $|\langle x, y \rangle| < \tau d$, then $|\langle f(x), f(y) \rangle| \leq (\tau \gamma)^p D$, and (ii) if $|\langle x, y \rangle| \geq \tau d$, then $(\langle x, y \rangle)^\gamma d^{p/2} D \leq \langle f(x), f(y) \rangle \leq (\gamma \langle x, y \rangle)^\gamma d^{p/2} D$.

That is, a correlation amplifier is a deterministic function that ensures that sufficiently large inner products are amplified to the $p$th power and small inner products remain small with a control on error provided by the parameter $\gamma$.

For $x, y \in \{-1,1\}^d$, and positive integers $p, s$, Valiant approximates the $p$th power of the inner product $\langle x \otimes^p y, x \otimes^p y \rangle = \langle x, y \rangle^p$ by drawing uniformly at random a sample of $s$ coordinates, since the Kroneckered-up vectors of length $d^p$ would be prohibitively long. For a coordinate sample $\bar{i} \in [d^p]^s$, he then computes the values at these coordinates $\bar{x} = (x \otimes^p)_{\bar{i}}$ (and similarly for $\bar{y}$), and then approximates $\langle x, y \rangle^p \approx \langle \bar{x}, \bar{y} \rangle$.

We construct our correlation amplifiers by replacing Valiant’s uniform random sample with an explicit sample. The sample is determined by constructing an expander graph with the zig-zag product construction of Reingold, Vadhan, and Wigderson [388], and then applying repeated squaring to construct the final sample iteratively. One can see an analog to the gradually increasing independence paradigm [274, 214, 111, 213] in PRGs; it is possible to view this

---

\(^5\)Collections of $F_2^n$-vectors such that a restriction to any set of $k$ indices guarantees that all $2^k$ possible distinct vectors appear in the restriction.

\(^6\)The random bits are fixed by an iterative search of the probability space for good choices through bisection, fixing a choice at every iteration [22].

\(^7\)Sets of binary vectors such that no vector of the set is contained in the bitwise OR of a small number of others [248].
scheme as one that fools the uniform sampling scheme. The reason we use the zig-zag product construction instead of actual Ramanujan graphs (which would provide a stronger control on the concentration properties of the sample) lies in the properties of repeated squaring that require a more fine-grained family of expanders than is possible with the known Ramanujan constructions.

As a result, we obtain an explicit family of correlation amplifiers. For constants $0 < c < 1$, $0 < \delta < \alpha$, and $C > 60$, the derandomized version of Valiant’s algorithm runs in time

$$\tilde{O}(n^{2 - \frac{0.99(\alpha - \delta)}{4c + 1}} + q n^{\delta + \frac{1.99(\alpha - \delta)}{4c + 1}}).$$  \hspace{1cm} (6.5)$$

While the exponent of $n$ in Equation (6.5) is strictly below 2, it should be noted that it is not quite as good as that given by Equations (6.3) or (6.4), since we have that the exponent is at least 1.998. For LIGHT BULB, we obtain a corollary that, for a constants $0 < \delta < \alpha$, $C > 60$, and $\kappa > 1$, runs in time

$$\tilde{O}(n^{2 - \frac{0.99(1 - 1/\kappa)(\alpha - \delta)}{4c + 1}}).$$

Finally, it should be noted that, in the case of LIGHT BULB in low dimensions ($d = O(\log n)$), the work in Publications I and II has been improved on by Alman [15] who applies the polynomial method of algorithm design of [19, 16] in combination with our and Valiant’s techniques to obtain a simple-to-prove randomized algorithm with matching runtime. He also uses the random bits in the instance itself to derive a derandomized algorithm that, for constant $c > 0$, runs in time $\tilde{O}(n^{2^{\omega(3+c)}}) = \tilde{O}(n^{1.582})$ when $d = \tilde{O}(\log n)$, matching the runtime of the randomized algorithm in Publication I.
7. Conclusion

We have presented advances in several fields of algorithms: (i) a faster method for finding outlier correlations, (ii) a derandomized method for finding outlier correlations, (iii) a symmetry breaking method with implementation for solving systems of hard constraints applicable in the automated design of bilinear algorithms among other things, (iv) the notion of probabilistic tensors and a new algorithm for Boolean matrix multiplication, and an (v) implementation of Strassen’s FMM in alternative basis for multiple-GPU shared-memory systems.

For the (i) faster method of finding outlier correlations, we have improved on earlier work by Valiant [338] by replacing Valiant’s uniform sampling scheme with one that has a Cartesian structure. This enables faster sampling by using FMM more than before. We have also (ii) presented a family of functions called correlation amplifiers that map vectors to higher dimensions, such that the absolute inner product between the two vectors is amplified to a constant power; that is, Definition 6.8 says that if the vectors have a small inner product in absolute value, the amplified vectors have a bounded absolute inner product, and if the absolute inner product is sufficiently large, the amplified inner product is retained up to error. We have presented an explicit construction that allows the efficient and deterministic computation of the correlation amplifiers. In other words, this offers a deterministic sampling scheme that gives theoretical guarantees and subquadratic runtime for the underlying algorithm.

As to symmetry, (iii) the adaptive prefix-assignment scheme we proposed, based on McKay’s canonical extension framework [328], can break symmetry in constraint programs, and break the symmetries completely with respect to a prescribed prefix sequence. This can even be done in the presence of a symmetry-obfuscating constraint encoding by supplying an auxiliary colored graph. We also provide an implementation that works by using the nauty library for canonical labeling of graphs. We evaluated our implementation empirically, and found it effective, and suitable for distributed processing in a computer cluster environment.

Finally, we have addressed matrix multiplication by (iv) presenting the theoretical notion of probabilistic tensors and probabilistic tensor rank, and shown the usefulness of these concepts by deriving a randomized algorithm for BMM
that is asymptotically faster than Strassen’s algorithm [427]. We have also (v) employed the change of basis of Karstadt and Schwartz [282] that enables us to perform binary FMM using a Strassen-like construct on shared-memory multi-GPU systems, such that our implementation exceeds the theoretical peak performance of the system that the elementary algorithm could possibly provide, and show experimentally that we obtain a speedup of over 2 in comparison to the elementary algorithm, over both the binary field and the Boolean semiring.

In this introduction, we have also presented how the different fields addressed in the publications are connected. FMM is a central tool in our similarity search results, following Valiant [438]. The FMM algorithms, on the other hand, are built from tensor decompositions, the search for which can be aided by symmetry breaking. Also, symmetry breaking can help on its own for the enumeration of nonisomorphic subtensors, which was central in the formulation of the notion of the probabilistic tensors. Finally, we have also given considerations for the efficient use of resources in automated design and implementation of bilinear algorithms, including energy as well as time, and efficient parallelizability.

There are still several open problems, and a lot of work still remains to be done. For example, the fact that we restricted our analysis to ±1-valued vectors in Publications I and II was motivated by the notion of decoherence: we avoid variations in the absolute value between different components of the vectors. This makes our analysis simpler, and enables faster sampling, as we need not worry about large components getting lost. However, this also means that we do not know if we can extend the Cartesian sampling to continuous-valued vectors. Also, the application of the method to continuous-valued vectors requires randomized discretization. Could we do away with this, so as to enable deterministic discretization for use with our derandomization techniques? Also, as a more general problem, we still do not know if it is possible to reach linear scaling in the general closest pair problem.

An open problem regarding the adaptive-prefix assignment method of Publication III is the choice of good prefix. In our experiments, we used our combinatorial understanding to craft the prefix by hand. However, for a widespread and practical adoption of such methodology, an automated choice of the prefix should be done. Is there a theoretically-founded general way to do this, without having to understand the specifics of the problem at hand? Also, the communication scheme in our implementation was only preliminary; it remains to be seen if a more advanced communication protocol can scale the system to warehouse-scale computers of hundreds of thousands nodes.

The probabilistic tensors of Publication IV are likely to offer an area of research that may very well yield some fruit. For example, we only proved an upper bound of the probabilistic rank for the matrix multiplication tensor with the (2,2,2) base case. The subtensors of larger matrix multiplication tensors await investigation.

Finally, the engineering work of Publication V showed that Strassen’s algorithm can indeed be practical on contemporary heterogeneous computer systems.
The bottleneck of the system was the limited memory bandwidth on the CPU side; it would be interesting to see how well a similar setup would scale if the CPU side was stronger. Also, such considerations at the moment precluded the possibility of efficient application of the probabilistic framework with a rank-6 multiplication. A system with better bandwidth and more GPU memory could possibly make use of the framework, and show that the probabilistic BMM is, in fact, practical on heterogeneous architectures.
References


References


References


References


References


References


References


References


References


References


104
References


References


References


References


Errata

Publication I

The restated Corollaries 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6 should read Corollary 1.8, 1.9, 2.2, 2.4, 2.6, and 2.7, respectively.