Computing Generalized Convolutions Faster Than Brute Force

Barış Can Esmer
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ariel Kulik
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Dániel Marx
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Philipp Schepper
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Karol Węgrzycki
Saarland University, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarbrücken, Germany

Abstract

In this paper, we consider a general notion of convolution. Let D be a finite domain and let D^n be the set of n-length vectors (tuples) of D. Let $f\colon D \times D \to D$ be a function and let $\oplus_f$ be a coordinate-wise application of f. The f-Convolution of two functions $g, h\colon D^n \to \{-M, \dots, M\}$ is

$$(g \circledast_f h)(\mathbf{v}) := \sum_{\substack{\mathbf{v}_g, \mathbf{v}_h \in D^n \\ \text{s.t. } \mathbf{v} = \mathbf{v}_g \oplus_f \mathbf{v}_h}} g(\mathbf{v}_g) \cdot h(\mathbf{v}_h)$$

for every $\mathbf{v} \in D^n$. This problem generalizes many fundamental convolutions such as Subset Convolution, XOR Product, Covering Product or Packing Product, etc. For arbitrary function f and domain D we can compute f-Convolution via brute-force enumeration in $\widetilde{\mathcal{O}}(|D|^{2n} \cdot \mathrm{polylog}(M))$ time.

Our main result is an improvement over this naive algorithm. We show that f-Convolution can be computed exactly in $\widetilde{\mathcal{O}}((c \cdot |D|^2)^n \cdot \mathrm{polylog}(M))$ for constant $c := 5/6$ when D has even cardinality. Our main observation is that a *cyclic partition* of a function $f\colon D \times D \to D$ can be used to speed up the computation of f-Convolution, and we show that an appropriate cyclic partition exists for every f.

Furthermore, we demonstrate that a single entry of the f-Convolution can be computed more efficiently. In this variant, we are given two functions $g, h\colon D^n \to \{-M, \dots, M\}$ alongside with a vector $\mathbf{v} \in D^n$ and the task of the f-Query problem is to compute integer $(g \circledast_f h)(\mathbf{v})$. This is a generalization of the well-known Orthogonal Vectors problem. We show that f-Query can be computed in $\widetilde{\mathcal{O}}(|D|^{\frac{\omega}{2}n} \cdot \mathrm{polylog}(M))$ time, where $\omega \in [2, 2.373)$ is the exponent of currently fastest matrix multiplication algorithm.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Algorithm design techniques

Keywords and phrases Generalized Convolution, Fast Fourier Transform, Fast Subset Convolution

Digital Object Identifier 10.4230/LIPIcs.IPEC.2022.12

Related Version *Full Version*: `https://arxiv.org/abs/2209.01623` [22]

Funding Research supported by the European Research Council (ERC) consolidator grant No. 725978 SYSTEMATICGRAPH and the project TIPEA (grant No. 850979).

Acknowledgements We would like to thank Karl Bringmann and Jesper Nederlof for useful discussions. Barış Can Esmer and Philipp Schepper are part of Saarbrücken Graduate School of Computer Science, Germany.

1 Introduction

Convolutions occur naturally in many algorithmic applications, especially in the exact and parameterized algorithms. The most prominent example is a subset convolution procedure [23, 37], for which an efficient $\widetilde{\mathcal{O}}(2^n \cdot \mathrm{polylog}(M))$ time algorithm for subset convolution dates back to Yates [40] but in the context of exact algorithms it was first used by Björklund et al. [6].[1] Researchers considered a plethora of other variants of convolutions, such as: Cover Product, XOR Product, Packing Product, Generalized Subset Convolution, or Discriminantal Subset Convolution [6, 8, 7, 10, 35, 21, 11]. These subroutines are crucial ingredients in the design of efficient algorithms for many exact and parameterized algorithms such as Hamiltonian Cycle, Feedback Vertex Set, Steiner Tree, Connected Vertex Cover, Chromatic Number, Max k-Cut or Bin Packing [20, 10, 41, 28, 5, 39]. These convolutions are especially useful for dynamic programming algorithms on tree decompositions and occur naturally during join operations (e.g., [35, 20, 34]). Usually, in the process of algorithm design, the researcher needs to design a different type of convolution from scratch to solve each of these problems. Often this is a highly technical and laborious task. Ideally, we would like to have a single tool that can be used as a blackbox in all of these cases. This motivates the following ambitious goal in this paper:

Goal: Unify convolution procedures under one general umbrella.

Towards this goal, we consider the problem of computing f-*Generalized Convolution* (f-Convolution) introduced by van Rooij [34]. Let D be a finite domain and let D^n be the n length vectors (tuples) of D. Let $f\colon D \times D \to D$ be an arbitrary function and let $\oplus_f$ be a coordinate-wise application of the function f.[2] For two functions $g, h\colon D^n \to \mathbb{Z}$ the f-Convolution, denoted by $(g \circledast_f h)\colon D^n \to \mathbb{Z}$, is defined for all $\mathbf{v} \in D^n$ as

$$(g \circledast_f h)(\mathbf{v}) \coloneqq \sum_{\substack{\mathbf{v}_g, \mathbf{v}_h \in D^n \\ \text{s.t. } \mathbf{v} = \mathbf{v}_g \oplus_f \mathbf{v}_h}} g(\mathbf{v}_g) \cdot h(\mathbf{v}_h).$$

Here we consider a standard $\mathbb{Z}(+, \cdot)$ ring. Through the paper we assume that M is the absolute value of the maximum integer given on the input.

In the f-Convolution problem the functions $g, h\colon D^n \to \{-M, \dots, M\}$ are given as an input and the output is the function $(g \circledast_f h)$. Note, that the input and output of the f-Convolution problem consist of $3 \cdot |D|^n$ integers. Hence it is conceivable that f-Convolution could be solved in $\widetilde{\mathcal{O}}(|D|^n \cdot \mathrm{polylog}(M))$. Such a result for arbitrary f would be a real breakthrough in how we design parameterized algorithms. So far, however, researchers have focused on characterizing functions f for which f-Convolution can be solved in $\widetilde{\mathcal{O}}(|D|^n \cdot \mathrm{polylog}(M))$ time. In [34] van Rooij considered specific instances of this setting, where for some constant $r \in \mathbb{N}$ the function f is defined as either (i) standard addition: $f(x, y) \coloneqq x + y$, or (ii) addition with a maximum: $f(x, y) \coloneqq \min(x + y, r - 1)$, or (iii) addition modulo r, or (iv) maximum: $f(x, y) \coloneqq \max(x, y)$. Van Rooij [34] showed that for these special cases the f-Convolution can be solved in $\widetilde{\mathcal{O}}(|D|^n \cdot \mathrm{polylog}(M))$ time. His results allow the function f to differ between coordinates. A recent result regarding generalized

[1] We use $\widetilde{\mathcal{O}}(\cdot)$ notation to hide polylogarithmic factors. We assume that M is the maximum absolute value of the integers on the input.

[2] We provide a formal definition of $\oplus_f$ in Section 2.

Discrete Fourier Transform [32] can be used in conjunction with Yates algorithm [40] to compute f-Convolution in $\widetilde{\mathcal{O}}(|D|^{\omega \cdot n/2} \cdot \mathrm{polylog}(M))$ time when f is a finite-group operation and ω is the exponent of the currently fastest matrix-multiplication algorithms.[3] To the best of our knowledge these are the most general settings where convolution has been considered so far.

Nevertheless, for an arbitrary function f, to the best of our knowledge the state-of-the-art for f-Convolution is a straightforward quadratic time enumeration.

Question 1: Is the naive $\widetilde{\mathcal{O}}(|D|^{2n} \cdot \mathrm{polylog}(M))$ algorithm for f-Convolution optimal?

Similar questions were studied from the point of view of the Fine-Grained Complexity. In that setting the focus is on convolutions with sparse representations, where the input size is only the size of the support of the functions g and h. It is conjectured that even subquadratic algorithms are highly unlikely for these representations [19, 25]. However, these lower bounds do not answer Question 1, because they are highly dependent on the sparsity of the input.

1.1 Our Results

We provide a positive answer to Question 1 and show an exponential improvement (in n) over a naive $\widetilde{\mathcal{O}}(|D|^{2n} \cdot \mathrm{polylog}(M))$ algorithm for every function f.

▶ **Theorem 1.1** (Generalized Convolution). *Let D be a finite set and $f\colon D \times D \to D$. There is an algorithm for f-Convolution with the following running time $\widetilde{\mathcal{O}}\left(\left(\frac{5}{6} \cdot |D|^2\right)^n \cdot \mathrm{polylog}(M)\right)$ when $|D|$ is even, or $\widetilde{\mathcal{O}}\left(\left(\frac{5}{6} \cdot |D|^2 + \frac{1}{6} \cdot |D|\right)^n \cdot \mathrm{polylog}(M)\right)$ when $|D|$ is odd.*

Observe that the running time obtained by Theorem 1.1 improves upon the brute-force for every $|D| \geq 2$. Our technique works in a more general setting when $g\colon L^n \to \mathbb{Z}$ and $h\colon R^n \to \mathbb{Z}$ and $f\colon L \times R \to T$ for arbitrary domains L, R and T (see Section 2 for the exact running time dependence).

Our Technique: Cyclic Partition. Now, we briefly sketch the idea behind the proof of Theorem 1.1. We say that a function is k-cyclic if it can be represented as an addition modulo k (after relabeling the entries of the domain and image). These functions are somehow simple, because as observed in [34, 33] f-Convolution can be computed in $\widetilde{\mathcal{O}}(k^n \cdot \mathrm{polylog}(M))$ time if f is k-cyclic. In a nutshell, our idea is to partition the function $f\colon D \times D \to D$ into cyclic functions and compute the convolution on these parts independently.

More formally, a cyclic minor of the function $f\colon D \times D \to D$ is a (combinatorial) rectangle $A \times B$ with $A, B \subseteq D$ and a number $k \in \mathbb{N}$ such that f restricted to A, B is a k-cyclic function. The cost of the cyclic minor (A, B, k) is $\mathrm{cost}(A, B) := k$. A cyclic partition is a set $\{(A_1, B_1, k_1), \ldots, (A_m, B_m, k_m)\}$ of cyclic minors such that for every $(a, b) \in D \times D$ there exists a unique $i \in [m]$ with $(a, b) \in A_i \times B_i$. The cost of the cyclic partition $\mathcal{P} = \{(A_1, B_1, k_1), \ldots, (A_m, B_m, k_m)\}$ is $\mathrm{cost}(\mathcal{P}) := \sum_{i=1}^{m} k_i$. See Figure 1.1 for an example of a cyclic partition.

Our first technical contribution is an algorithm to compute f-Convolution when the cost of a cyclic partition is small.

[3] This observation was brought to our attention by Jesper Nederlof [27].

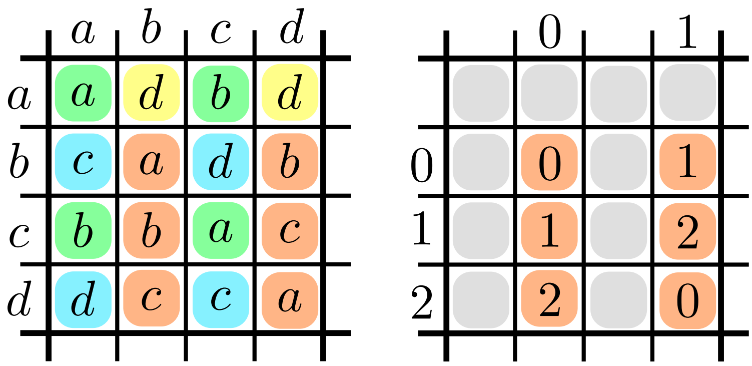


Figure 1.1 Left figure illustrates exemplar function $f\colon D \times D \to D$ over domain $D \coloneqq \{a, b, c, d\}$. We highlighted a cyclic partition with red, blue, yellow and blue colors. Each color represents a different minor of f. On the right figure we demonstrate that the red-highlighted minor can be represented as addition modulo 3 (after relabeling $a \mapsto 0$, $b \mapsto 1$ and $c \mapsto 2$). Hence the red minor has cost 3. The reader can further verify that green and blue minors have cost 2 and yellow minor has cost 1, hence the cost of that particular partition is $3 + 2 + 2 + 1 = 8$.

▶ **Lemma 1.2** (Algorithm for f-Convolution). *Let D be an arbitrary finite set, $f\colon D \times D \to D$ and let $\mathcal{P}$ be the cyclic partition of f. Then there exists an algorithm which given $g, h\colon D^n \to \mathbb{Z}$ computes $(g \circledast_f h)$ in $\widetilde{\mathcal{O}}((\mathrm{cost}(\mathcal{P})^n + |D|^n) \cdot \mathrm{polylog}(M))$ time.*

The idea behind the proof of Lemma 1.2 is as follows. Based on the partition $\mathcal{P}$, for any pair of vectors $\mathbf{u}, \mathbf{w} \in D^n$, we can define a type $\overline{p} \in [m]^n$ such that $(\mathbf{u}_i, \mathbf{w}_i) \in A_{\overline{p}_i} \times B_{\overline{p}_i}$ for every $i \in [n]$. Our main idea is to go over each type $\overline{p}$ and compute the sum in the definition of f-Convolution only for pairs $(\mathbf{v}_g, \mathbf{v}_h)$ that have type $\overline{p}$. In order to do this, first we select the vectors $\mathbf{v}_g$ and $\mathbf{v}_h$ that are compatible with this type $\overline{p}$. For instance, consider the example in Figure 1.1. Whenever $\overline{p}_i$ refers to, say, the red-colored minor, then we consider $\mathbf{v}_g$ only if its i-th coordinate is in $\{b, c, d\}$ and consider $\mathbf{v}_h$ only if its i-th coordinate is in $\{b, d\}$. After computing all these vectors $\mathbf{v}_g$ and $\mathbf{v}_h$, we can transform them according to the cyclic minor at each coordinate. Continuing our example, as the red-colored minor is 3-cyclic, we can represent the i-th coordinate of $\mathbf{v}_g$ and $\mathbf{v}_h$ as $\{0, 1, 2\}$ and then the problem reduces to addition modulo 3 at that coordinate. Therefore, using the algorithm of van Rooij [34] for cyclic convolution we can handle all pairs of type $\overline{p}$ in $\widetilde{\mathcal{O}}((\prod_{i=1}^{n} k_{\overline{p}_i}) \cdot \mathrm{polylog}(M))$ time. As we go over all m^n types $\overline{p}$ the sum of m^n terms is

$$\sum_{\overline{p} \in [m]^n} \left(\prod_{i=1}^{n} k_{\overline{p}_i} \right) = \left(\sum_{i=1}^{m} k_i \right)^n = \mathrm{cost}(\mathcal{P})^n.$$

Hence, the overall running time is $\widetilde{\mathcal{O}}(\mathrm{cost}(\mathcal{P})^n \cdot \mathrm{polylog}(M))$. This running time evaluation ignores the generation of the vectors given as input for the cyclic convolution algorithm. The efficient computation of these vectors is nontrivial and requires further techniques that we explain in Section 3.

It remains to provide the low-cost cyclic partition of an arbitrary function f.

▶ **Lemma 1.3.** *For any finite set D and any function $f\colon D \times D \to D$ there is a cyclic partition $\mathcal{P}$ of f such that $\mathrm{cost}(\mathcal{P}) \le \frac{5}{6}|D|^2$ when $|D|$ is even, or $\mathrm{cost}(\mathcal{P}) \le \frac{5}{6}|D|^2 + \frac{1}{6}|D|$ when $|D|$ is odd.*

For the sake of presentation let us assume that $|D|$ is even. In order to show Lemma 1.3, we partition D into pairs $A_1, \dots, A_k$ where $k \coloneqq |D|/2$ and consider the restrictions of f to $A_j \times D$ one by one. This allows us to encode f on $A_j \times D$ as a directed graph G with $|D|$

edges and $|D|$ vertices. We observe that for certain classes of subgraphs (i.e., paths, out-stars, in-stars, and cycles) there is a corresponding cyclic minor. Our goal is to partition this graph G into such subgraphs in a way that the total cost of the resulting cyclic partition is small. Following this argument, the proof of Lemma 1.3 becomes a graph theoretic analysis. The proof of Lemma 1.3 is included in Section 4. Our method applies for more general functions $f\colon L \times R \to T$, where domains L, R, T can be different and have arbitrary cardinality. We note that a weaker variant of Lemma 1.3 in which the guarantee is $\mathrm{cost}(\mathcal{P}_f) \le \frac{7}{8}|D|^2$ is easier to attain.

Efficient Algorithm for Convolution Query. Our next contribution is an efficient algorithm to query a single value of f-Convolution. In the f-Query problem, the input is $g, h\colon D^n \to \mathbb{Z}$ and a single vector $\mathbf{v} \in D^n$. The task is to compute a value $(g \circledast_f h)(\mathbf{v})$. Observe that this task generalizes[4] the fundamental problem of Orthogonal Vectors. We show that computing f-Query is much faster than computing the full output of f-Convolution.

▶ **Theorem 1.4** (Convolution Query). *For any finite set D and function $f\colon D \times D \to D$ there is a $\widetilde{\mathcal{O}}(|D|^{\omega \cdot n/2} \cdot \mathrm{polylog}(M))$ time algorithm for the f-Query problem.*

Here $\widetilde{\mathcal{O}}(n^{\omega} \cdot \mathrm{polylog}(M))$ is the time needed to multiply two $n \times n$ integer matrices with values in $\{-M, \dots, M\}$ and currently $\omega \in [2, 2.373)$ [2]. Note, that under the assumption that two matrices can be multiplied in the linear in the input time (i.e., $\omega = 2$) then Theorem 1.4 runs in the nearly-optimal $\widetilde{\mathcal{O}}(|D|^n \cdot \mathrm{polylog}(M))$ time. Theorem 1.4 is significantly faster than Theorem 1.1 even when we plug-in the naive algorithm for matrix multiplication (i.e., $\omega = 3$). The proof of Theorem 1.4 is inspired by an interpretation of the f-Query problem as counting length-4 cycles in a graph.

1.2 Related Work

Arguably, the problem of computing the Discrete Fourier Transform (DFT) is the prime example of convolution-type problems in computer science. Cooley and Tukey [18] proposed the fast algorithm to compute DFT. Later, Beth [4] and Clausen [17] initiated the study of generalized DFTs whose goal has been to obtain a fast algorithm for DFT where the underlying group is arbitrary. After a long line of works (see [31] for the survey), the currently best algorithm for generalized DFT concerning group G runs in $\mathcal{O}(|G|^{\omega/2+\epsilon})$ operations for every $\epsilon > 0$ [32].

A similar technique to ours was introduced by Björklund et al. [9]. The paper gave a characterization of lattices that admit a fast zeta transform and a fast Möbius transform. Their paper used the notion of *covering pairs*, which is similar to cyclic partitions used in this paper but with a completely different goal.

From the lower-bounds perspective to the best of our knowledge only a naive $\Omega(|D|^n)$ lower bound is known for f-Convolution (as this is the output size). We note that known lower bounds for different convolution-type problems, such as $(\min, +)$-convolution [19, 25], $(\min, \max)$-convolution [13], min-witness convolution [26], convolution-3SUM [14] or even skew-convolution [12] cannot be easily adapted to f-Convolution as the hardness of these problems comes primarily from the ring operations.

The Orthogonal Vector problem is related to the f-Query problem. In the Orthogonal Vector problem we are given two sets of n vectors $A, B \subseteq \{0, 1\}^d$ and the task is to decide if there is a pair $a \in A$, $b \in B$ such that $a \cdot b = 0$. In [38] it was shown that no $n^{2-\epsilon} \cdot 2^{o(d)}$

[4] It is a special case with $D = \{0, 1\}$, $\mathbf{v} = 0^n$ and $f(x, y) = x \cdot y$

algorithm for Orthogonal Vectors is possible for any $\epsilon > 0$ assuming SETH [36]. The currently best algorithm for Orthogonal Vectors run in $n^{2-1/\mathcal{O}(\log(d)/\log(n))}$ time [1, 15], $\mathcal{O}(n \cdot 2^{cd})$ for some constant $c < 0.5$ [30], or $\mathcal{O}(|{\downarrow}A| + |{\downarrow}B|)$ [7] (where $|{\downarrow}F|$ is the total number of vectors whose support is a subset of the support of input vectors).

1.3 Organization

In Section 2 we provide the formal definitions of the problems alongside the general statements of our results. In Section 3 we give an algorithm for f-Convolution that uses a given cyclic partition. In Section 4 we show that for every function $f\colon D \times D \to D$ there exists a cyclic partition of low cost. In Section 5 we conclude the paper and discuss future work.

In Appendix A we give an algorithm for f-Query and prove Theorem 1.4. In Appendix C and Appendix B we include the missing proofs.

2 Preliminaries

Throughout the paper, we use Iverson bracket notation, where for the logic expression P, the value of $[\![P]\!]$ is 1 when P is true and 0 otherwise. For $n \in \mathbb{N}$ we use $[n]$ to denote $\{1, \dots, n\}$. Through the paper we denote vectors in bold, for example, $\mathbf{q} \in \mathbb{Z}^k$ denotes a k-dimensional vector of integers. We use subscripts to denote the entries of the vectors, e.g., $\mathbf{q} \coloneqq (\mathbf{q}_1, \dots, \mathbf{q}_k)$.

Let L, R and T be arbitrary sets and let $f\colon L \times R \to T$ be an arbitrary function. We extend the definition of such an arbitrary function f to vectors as follows. For two vectors $\mathbf{u} \in L^n$ and $\mathbf{w} \in R^n$ we define

$$\mathbf{u} \oplus_f \mathbf{w} \coloneqq (f(\mathbf{u}_1, \mathbf{w}_1), \dots, f(\mathbf{u}_n, \mathbf{w}_n)).$$

In this paper, we consider the f-Convolution problem with a more general domain and image. We define it formally as follows:

▶ **Definition 2.1** (f-Convolution). *Let L, R and T be arbitrary sets and let $f\colon L \times R \to T$ be an arbitrary function. The f-Convolution of two functions $g\colon L^n \to \mathbb{Z}$ and $h\colon R^n \to \mathbb{Z}$, where $n \in \mathbb{N}$, is the function $(g \circledast_f h)\colon T^n \to \mathbb{Z}$ defined by*

$$(g \circledast_f h)(\mathbf{v}) \coloneqq \sum_{\mathbf{u} \in L^n,\ \mathbf{w} \in R^n} [\![\mathbf{v} = \mathbf{u} \oplus_f \mathbf{w}]\!] \cdot g(\mathbf{u}) \cdot h(\mathbf{w})$$

for every $\mathbf{v} \in T^n$.

As before the operations are taken in the standard $\mathbb{Z}(+,\cdot)$ ring and M is the maximum absolute value of the integers given on the input.

Now, we formally define the input and output to the f-Convolution problem.

▶ **Definition 2.2** (f-Convolution Problem (f-Convolution)). *Let L, R and T be arbitrary finite sets and let $f\colon L \times R \to T$ be an arbitrary function. The f-Convolution Problem is the following.*
Input: *Two functions $g\colon R^n \to \{-M, \dots, M\}$ and $h\colon L^n \to \{-M, \dots, M\}$.*
Task: *Compute $g \circledast_f h$.*

Our main result stated in the most general form is the following.

▶ **Theorem 2.3.** *Let $f\colon L \times R \to T$ such that L, R and T are finite. There is an algorithm for the f-*Convolution *problem with $\widetilde{\mathcal{O}}(c^n \cdot \operatorname{polylog}(M))$ time, where*

$$c := \begin{cases} \frac{|L|}{2} \cdot \frac{4\cdot|R|+|T|}{3} & \textit{if } |L| \textit{ is even} \\ \frac{|L|-1}{2} \cdot \frac{4\cdot|R|+|T|}{3} + |R| & \textit{otherwise.} \end{cases}$$

Theorem 1.1 is a corollary of Theorem 2.3 by setting $L = R = T = D$.

The proof of Theorem 2.3 utilizes the notion of *cyclic partition.* For any $k \in \mathbb{N}$, let $\mathbb{Z}_k = \{0, 1, \dots, k-1\}$. We say a function $f\colon A \times B \to C$ is *k-cyclic* if, up to a relabeling of the sets A, B and C, it is an addition modulo k. Formally, $f\colon A \times B \to C$ is *k-cyclic* if there are $\sigma_A\colon A \to \mathbb{Z}_k$, $\sigma_B\colon B \to \mathbb{Z}_k$, and $\sigma_C\colon \mathbb{Z}_k \to C$ such that

$$\forall a \in A,\ b \in B: \quad f(a,b) = \sigma_C\left(\sigma_A(a) + \sigma_B(b) \mod k\right).$$

We refer to the functions σ_A, σ_B and σ_C as the *relabeling* functions of f.

The *restriction* of $f\colon L \times R \to T$ to $A \subseteq L$ and $B \subseteq R$ is the function $g\colon A \times B \to T$ defined by $g(a,b) = f(a,b)$ for all $a \in A$ and $b \in B$. We say (A, B, k) is a *cyclic minor* of $f\colon L \times R \to T$ if the restriction of f to A and B is a k-cyclic function.

A *cyclic partition* of $f\colon L \times R \to T$ is a set of minors $\mathcal{P} = \{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$ such that (A_i, B_i, k_i) is a cyclic minor of f and for every $(a,b) \in L \times R$ there is a unique $1 \le i \le m$ such that $(a,b) \in A_i \times B_i$. The cost of the cyclic partition is $\operatorname{cost}(\mathcal{P}) = \sum_{i=1}^m k_i$.

Theorem 2.3 follows from the following lemmas.

▶ **Lemma 3.1** (Algorithm for Generalized Convolution)**.** *Let L, R and T be finite sets. Also, let $f\colon L \times R \to T$ be a function and let $\mathcal{P}$ be a cyclic partition of f. Then there is an $\widetilde{\mathcal{O}}((\operatorname{cost}(\mathcal{P})^n + |L|^n + |R|^n + |T|^n) \cdot \operatorname{polylog}(M))$ time algorithm for f-*Convolution.

▶ **Lemma 4.1.** *Let $f\colon L \times R \to T$ where L, R and T are finite sets. Then there is a cyclic partition $\mathcal{P}$ of f such that $\operatorname{cost}(\mathcal{P}) \le \frac{|L|}{2} \cdot \frac{4\cdot|R|+|T|}{3}$ when $|L|$ is even, and $\operatorname{cost}(\mathcal{P}) \le |R| + \frac{|L|-1}{2} \cdot \frac{4\cdot|R|+|T|}{3}$ when $|L|$ is odd.*

The proof of Lemma 3.1 is included in Section 3 and proof of Lemma 4.1 is included in Section 4. The proof of Lemma 3.1 uses an algorithm for Cyclic Convolution.

▶ **Definition 2.4** (Cyclic Convolution)**.** *The* Cyclic Convolution *problem is the following.*
Input: *Vector $\mathbf{r} \in \mathbb{N}^k$ and functions $g, h\colon Z \to \{-M, \dots, M\}$ where $Z = \mathbb{Z}_{\mathbf{r}_1} \times \dots \times \mathbb{Z}_{\mathbf{r}_k}$.*
Task: *Compute the function $g \odot h\colon Z \to \mathbb{Z}$ defined by*

$$(g \odot h)(\mathbf{v}) = \sum_{\mathbf{u},\mathbf{w} \in Z} \left(\prod_{i=1}^{k} [\![\mathbf{u}_i + \mathbf{w}_i = \mathbf{v}_i \mod \mathbf{r}_i]\!] \right) \cdot g(\mathbf{u}) \cdot h(\mathbf{w}).$$

Van Rooij [33] showed that Cyclic Convolution can be solved in $\widetilde{\mathcal{O}}\left(\left(\prod_{i=1}^k \mathbf{r}_i\right) \cdot \operatorname{polylog}(M)\right)$ time. However, his algorithm relies on finding an appropriate large prime p. In order to circumvent the discussion on how such a prime can be found efficiently and deterministically, we can use multiple smaller primes and the Chinese Reminder Theorem. We include the details in Appendix B.

▶ **Theorem 2.5** (Cyclic Convolution)**.** *There is an $\widetilde{\mathcal{O}}\left((\prod_{i=1}^k \mathbf{r}_i) \cdot \operatorname{polylog}(M)\right)$ algorithm for the* Cyclic Convolution *problem.*

3 Generalized Convolution

In this section we prove Lemma 3.1.

▶ **Lemma 3.1** (Algorithm for Generalized Convolution). *Let L, R and T be finite sets. Also, let $f\colon L \times R \to T$ be a function and let $\mathcal{P}$ be a cyclic partition of f. Then there is an $\widetilde{\mathcal{O}}((\mathrm{cost}(\mathcal{P})^n + |L|^n + |R|^n + |T|^n) \cdot \mathrm{polylog}(M))$ time algorithm for f-*Convolution.

Throughout the section we fix L, R and T, and $f\colon L \times R \to T$ to be as in the statement of Lemma 3.1. Additionally, fix a cyclic partition $\mathcal{P} = \{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$. Furthermore, let $\sigma_{A,i}$, $\sigma_{B,i}$ and $\sigma_{C,i}$ be the relabeling functions of the cyclic minor (A_i, B_i, k_i) for every $i \in [m]$. We assume the labeling functions are also fixed throughout this section.

In order to describe our algorithm for Lemma 3.1, we first need to establish several technical definitions.

▶ **Definition 3.2** (Type). *The* type *of two vectors $\mathbf{u} \in L^n$ and $\mathbf{w} \in R^n$ is the unique vector $\overline{p} \in [m]^n$ for which $\mathbf{u}_i \in A_{\overline{p}_i}$ and $\mathbf{w}_i \in B_{\overline{p}_i}$ for all $i \in [n]$.*

Observe that the type of two vectors is well defined as $\mathcal{P}$ is a cyclic partition. For any type $\overline{p} \in \{1, \dots, m\}^n$ we define

$$L_{\overline{p}} \coloneqq A_{\overline{p}_1} \times \cdots \times A_{\overline{p}_n}, \qquad R_{\overline{p}} \coloneqq B_{\overline{p}_1} \times \cdots \times B_{\overline{p}_n}, \qquad Z_{\overline{p}} \coloneqq \mathbb{Z}_{k_{\overline{p}_1}} \times \cdots \times \mathbb{Z}_{k_{\overline{p}_n}}$$

to be vector domains restricted to type $\overline{p}$. For any type $\overline{p}$ we introduce relabeling functions on its restricted domains. The relabeling functions of $\overline{p}$ are the functions $\boldsymbol{\sigma}^L_{\overline{p}}\colon L_{\overline{p}} \to Z_{\overline{p}}$, $\boldsymbol{\sigma}^R_{\overline{p}}\colon R_{\overline{p}} \to Z_{\overline{p}}$, and $\boldsymbol{\sigma}^T_{\overline{p}}\colon Z_{\overline{p}} \to T^n$ defined as follows:

$$\begin{aligned}
\boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{v}) &\coloneqq \big(\sigma_{A,\overline{p}_1}(\mathbf{v}_1), \dots, \sigma_{A,\overline{p}_n}(\mathbf{v}_n)\big) && \forall \mathbf{v} \in L_{\overline{p}},\\
\boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{v}) &\coloneqq \big(\sigma_{B,\overline{p}_1}(\mathbf{v}_1), \dots, \sigma_{B,\overline{p}_n}(\mathbf{v}_n)\big) && \forall \mathbf{v} \in R_{\overline{p}},\\
\boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q}) &\coloneqq \big(\sigma_{C,\overline{p}_1}(\mathbf{q}_1), \dots, \sigma_{C,\overline{p}_n}(\mathbf{q}_n)\big) && \forall \mathbf{q} \in Z_{\overline{p}}.
\end{aligned}$$

Our algorithm heavily depends on constructing the following projections.

▶ **Definition 3.3** (Projection of a Function). *The projection of a function $g\colon L^n \to \mathbb{Z}$ with respect to the type $\overline{p} \in [m]^n$, is the function $g_{\overline{p}}\colon Z_{\overline{p}} \to \mathbb{Z}$ defined as*

$$g_{\overline{p}}(\mathbf{q}) \coloneqq \sum_{\mathbf{u} \in L_{\overline{p}}} [\![\boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u}) = \mathbf{q}]\!] \cdot g(\mathbf{u}) \qquad \text{for every } \mathbf{q} \in Z_{\overline{p}}.$$

Similarly, the projection $h_{\overline{p}}\colon Z_{\overline{p}} \to \mathbb{Z}$ of a function $h\colon R^n \to \mathbb{Z}$ with respect to the type $\overline{p} \in [m]^n$ is defined as

$$h_{\overline{p}}(\mathbf{q}) \coloneqq \sum_{\mathbf{w} \in R_{\overline{p}}} [\![\boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{w}) = \mathbf{q}]\!] \cdot h(\mathbf{w}) \qquad \text{for every } \mathbf{q} \in Z_{\overline{p}}.$$

The projections are useful due to the following connection with $g \circledast_f h$.

▶ **Lemma 3.4.** *Let $g\colon L^n \to \mathbb{Z}$ and $h\colon R^n \to \mathbb{Z}$, then for every $\mathbf{v} \in T^n$ it holds that:*

$$(g \circledast_f h)(\mathbf{v}) = \sum_{\overline{p} \in [m]^n} \sum_{\mathbf{q} \in Z_{\overline{p}}} [\![\boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q}) = \mathbf{v}]\!] \cdot (g_{\overline{p}} \odot h_{\overline{p}})(\mathbf{q}),$$

where $g_{\overline{p}} \odot h_{\overline{p}}$ is the cyclic convolution of $g_{\overline{p}}$ and $h_{\overline{p}}$.

We give the proof of Lemma 3.4 in Section 3.1. It should be noted that the naive computation of the projection functions of g and h with respect to all types $\overline{p}$ is significantly slower than the running time stated in Lemma 3.1. To adhere to the stated running time we use a dynamic programming procedure for the computations, as stated in the following lemma.

▶ **Lemma 3.5.** *There exists an algorithm which given a function $g\colon L^n \to \{-M,\dots,M\}$ returns the set of its projections, $\{g_{\overline{p}} \mid \overline{p} \in [m]^n\}$, in time $\widetilde{\mathcal{O}}\left((\operatorname{cost}(\mathcal{P})^n + |L|^n) \cdot \operatorname{polylog}(M)\right)$.*

▶ Remark 3.6. Analogously, we can also construct every projection of a function $h\colon R^n \to \{-M,\dots,M\}$ in $\widetilde{\mathcal{O}}\left((\operatorname{cost}(\mathcal{P})^n + |R|^n) \cdot \operatorname{polylog}(M)\right)$ time.

The proof of Lemma 3.5 in given in Appendix C.

Our algorithm for f-Convolution (see Algorithm 1 for the pseudocode) is a direct implication of Lemma 3.4 and Lemma 3.5. First, the algorithm computes the projections of g and h with respect to every type $\overline{p}$. Subsequently, the cyclic convolution of $g_{\overline{p}}$ and $h_{\overline{p}}$ is computed efficiently as described in Theorem 2.5. Finally, the values of $(g \circledast_f h)$ are reconstructed by the formula in Lemma 3.4.

Algorithm 1 Cyclic Partition Algorithm for the f-Convolution problem.

Setting: Finite sets L, R and T, $f\colon L \times R \to T$ and a cyclic partition $\mathcal{P}$ of f, of size m.

Input: $g\colon L^n \to \{-M,\dots,M\}$, $h\colon R^n \to \{-M,\dots,M\}$

1 Construct the projections of g and h w.r.t $\overline{p}$, for all $\overline{p} \in [m]^n$ ▷ Lemma 3.5

2 For every $\overline{p} \in [m]^n$ compute $\mathsf{c}_{\overline{p}} = g_{\overline{p}} \odot h_{\overline{p}}$ ▷ Cyclic convolutions (Definition 2.4)

3 Define $\mathsf{r}\colon T^n \to \mathbb{Z}$ by

$$\mathsf{r}(\mathbf{v}) = \sum_{\overline{p}\in[m]^n} \sum_{\mathbf{q}\in Z_{\overline{p}} \text{ s.t. } \boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q})=\mathbf{v}} \mathsf{c}_{\overline{p}}(\mathbf{q}) \qquad \text{for all } \mathbf{v} \in T^n.$$

4 **return** r

Proof of Lemma 3.1. Observe that Algorithm 1 returns $\mathsf{r}\colon T^n \to \mathbb{Z}$ such that for every $\mathbf{v} \in T^n$ it holds that

$$\mathsf{r}(\mathbf{v}) = \sum_{\overline{p}\in[m]^n} \sum_{\substack{\mathbf{q}\in Z_{\overline{p}} \\ \text{s.t. } \boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q})=\mathbf{v}}} \mathsf{c}_{\overline{p}}(\mathbf{q}) = \sum_{\overline{p}\in[m]^n} \sum_{\mathbf{q}\in Z_{\overline{p}}} [\![\boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q}) = \mathbf{v}]\!] \cdot (g_{\overline{p}} \odot h_{\overline{p}})(\mathbf{q}) = (g \circledast_f h)(\mathbf{v}),$$

where the last equality is by Lemma 3.4. Thus, the algorithm returns $(g \circledast_f h)$ as required. It therefore remains to bound the running time of the algorithm.

By Lemma 3.5, Line 1 of Algorithm 1 runs in time $\widetilde{\mathcal{O}}((\operatorname{cost}(\mathcal{P})^n + |L|^n + |R|^n) \cdot \operatorname{polylog}(M))$. By Theorem 2.5, for any type $\overline{p} \in [m]^n$ the computation of $g_{\overline{p}} \odot h_{\overline{p}}$ in Line 2 runs in time $\widetilde{\mathcal{O}}((\prod_{i=1}^n k_{\overline{p}_i}) \cdot \operatorname{polylog}(M))$. Thus the overall running time of Line 2 is $\widetilde{\mathcal{O}}\left((\sum_{\overline{p}\in[m]^n} \prod_{i=1}^n k_{\overline{p}_i}) \cdot \operatorname{polylog}(M)\right)$.

Finally, observe that the construction of r in Line 3 can be implemented by initializing r to be zeros and iteratively adding the value of $\mathsf{c}_{\overline{p}}(\mathbf{q})$ to $\mathsf{r}(\sigma^T_{\overline{p}}(\mathbf{q}))$ for every $\overline{p} \in [m]^n$ and $\mathbf{q} \in Z_{\overline{p}}$. The required running time is thus $\widetilde{\mathcal{O}}(|T|^n \cdot \operatorname{polylog}(M))$ for the initialization and $\widetilde{\mathcal{O}}\left((\sum_{\overline{p}\in[m]^n} |Z_{\overline{p}}|) \cdot \operatorname{polylog}(M)\right) = \widetilde{\mathcal{O}}\left((\sum_{\overline{p}\in[m]^n} \prod_{i=1}^n k_{\overline{p}_i}) \cdot \operatorname{polylog}(M)\right)$ for the addition operations. Thus, the overall running time of Line 3 is

$$\widetilde{\mathcal{O}}\left(\left(|T|^n + \sum_{\overline{p}\in[m]^n} \prod_{i=1}^n k_{\overline{p}_i}\right) \cdot \operatorname{polylog}(M)\right).$$

Combining the above, with $\sum_{\overline{p}\in[m]^n} \prod_{i=1}^n k_{\overline{p}_i} = \left(\sum_{i=1}^m k_i\right)^n = (\text{cost}(\mathcal{P}))^n$ means that the running time of Algorithm 1 is

$$\widetilde{\mathcal{O}}\left((|T|^n + |R|^n + |L|^n + \text{cost}(\mathcal{P})^n) \cdot \text{polylog}(M)\right)$$

This concludes the proof of Lemma 3.1. ◀

3.1 Properties of Projections

In this section we provide the proof for Lemma 3.4. The proof of Lemma 3.4 uses the following definitions of coordinate-wise addition with respect to a type $\overline{p}$.

▶ **Definition 3.7** (Coordinate-wise Addition Modulo for Type). *For any $\overline{p} \in [m]^n$ we define a coordinate-wise addition modulo as*

$$\mathbf{q} +_{\overline{p}} \mathbf{r} \coloneqq \left((\mathbf{q}_1 + \mathbf{r}_1 \mod k_{\overline{p}_1}), \ldots, (\mathbf{q}_n + \mathbf{r}_n \mod k_{\overline{p}_n})\right) \qquad \textit{for every } \mathbf{q}, \mathbf{r} \in Z_{\overline{p}}.$$

Proof of Lemma 3.4. By Definition 2.1 it holds that:

$$(g \circledast_f h)(\mathbf{v}) = \sum_{\mathbf{u}\in L^n, \mathbf{w}\in R^n} [\![\mathbf{v} = \mathbf{u} \oplus_f \mathbf{w}]\!] \cdot g(\mathbf{u}) \cdot h(\mathbf{w}). \tag{3.1}$$

Recall that the type of every two vectors $(\mathbf{u}, \mathbf{w}) \in L^n \times R^n$ is unique and $[m]^n$ contains all possible types and hence, we can rewrite (3.1) as

$$(g \circledast_f h)(\mathbf{v}) = \sum_{\overline{p}\in[m]^n} \sum_{\mathbf{u}\in L_{\overline{p}}, \mathbf{w}\in R_{\overline{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot [\![\mathbf{v} = \mathbf{u} \oplus_f \mathbf{w}]\!] \tag{3.2}$$

By the properties of the relabeling functions, we get

$$\begin{aligned}
&= \sum_{\overline{p}\in[m]^n} \sum_{\mathbf{u}\in L_{\overline{p}}, \mathbf{w}\in R_{\overline{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot [\![\mathbf{v} = \boldsymbol{\sigma}^T_{\overline{p}}\left(\boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u}) +_{\overline{p}} \boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{w})\right)]\!] \\
&= \sum_{\overline{p}\in[m]^n} \sum_{\mathbf{q}\in Z_{\overline{p}}} \sum_{\mathbf{u}\in L_{\overline{p}}, \mathbf{w}\in R_{\overline{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot [\![\mathbf{v} = \boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q})]\!] \cdot [\![\mathbf{q} = \boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u}) +_{\overline{p}} \boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{w})]\!] \\
&= \sum_{\overline{p}\in[m]^n} \sum_{\substack{\mathbf{q}\in Z_{\overline{p}} \\ \text{s.t. } \boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q})=\mathbf{v}}} \sum_{\mathbf{u}\in L_{\overline{p}}, \mathbf{w}\in R_{\overline{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot [\![\mathbf{q} = \boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u}) +_{\overline{p}} \boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{w})]\!].
\end{aligned}$$

Observe that we can *partition* $L_{\overline{p}}$ (respectively $R_{\overline{p}}$) by considering the inverse images of $\mathbf{r} \in Z_{\overline{p}}$ under $\boldsymbol{\sigma}^L_{\overline{p}}$ (respectively $\boldsymbol{\sigma}^R_{\overline{p}}$), i.e. $L_{\overline{p}} = \biguplus_{\mathbf{r}\in Z_{\overline{p}}} \{\mathbf{u} \in L_{\overline{p}} \mid \boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u}) = \mathbf{r}\}$. Hence, for every $\overline{p} \in [m]^n$ and $\mathbf{q} \in Z_{\overline{p}}$ it holds that

$$\begin{aligned}
&\sum_{\mathbf{u}\in L_{\overline{p}}, \mathbf{v}\in R_{\overline{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot [\![\mathbf{q} = \boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u}) +_{\overline{p}} \boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{w})]\!] \\
&= \sum_{\mathbf{r},\mathbf{s}\in Z_{\overline{p}}} \sum_{\mathbf{u}\in L_{\overline{p}}, \mathbf{w}\in R_{\overline{p}}} g(\mathbf{u}) \cdot h(\mathbf{w}) \cdot [\![\mathbf{q} = \mathbf{r} +_{\overline{p}} \mathbf{s}]\!] \cdot [\![\mathbf{r} = \boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u})]\!] \cdot [\![\mathbf{s} = \boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{w})]\!] \\
&= \sum_{\mathbf{r},\mathbf{s}\in Z_{\overline{p}}} [\![\mathbf{q} = \mathbf{r} +_{\overline{p}} \mathbf{s}]\!] \left(\sum_{\mathbf{u}\in L_{\overline{p}}} [\![\mathbf{r} = \boldsymbol{\sigma}^L_{\overline{p}}(\mathbf{u})]\!] \cdot g(\mathbf{u})\right) \cdot \left(\sum_{\mathbf{w}\in R_{\overline{p}}} [\![\mathbf{s} = \boldsymbol{\sigma}^R_{\overline{p}}(\mathbf{w})]\!] \cdot h(\mathbf{w})\right) \\
&= \sum_{\mathbf{r},\mathbf{s}\in Z_{\overline{p}}} [\![\mathbf{q} = \mathbf{r} +_{\overline{p}} \mathbf{s}]\!] \cdot g_{\overline{p}}(\mathbf{r}) \cdot h_{\overline{p}}(\mathbf{s}) \\
&= (g_{\overline{p}} \odot h_{\overline{p}})(\mathbf{q}).
\end{aligned} \tag{3.3}$$

By plugging (3.3) into (3.2) we get

$$(g \circledast_f h)(\mathbf{v}) = \sum_{\overline{p}\in[m]^n} \sum_{\substack{\mathbf{q}\in Z_{\overline{p}} \\ \text{s.t. } \boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q})=\mathbf{v}}} (g_{\overline{p}} \odot h_{\overline{p}})(\mathbf{q}) = \sum_{\overline{p}\in[m]^n} \sum_{\mathbf{q}\in Z_{\overline{p}}} [\![\boldsymbol{\sigma}^T_{\overline{p}}(\mathbf{q}) = \mathbf{v}]\!] \cdot (g_{\overline{p}} \odot h_{\overline{p}})(\mathbf{q}),$$

as required. ◀

4 Existence of Low-Cost Cyclic Partition

In this section we prove Lemma 4.1.

▶ **Lemma 4.1.** *Let $f\colon L \times R \to T$ where L, R and T are finite sets. Then there is a cyclic partition $\mathcal{P}$ of f such that* $\mathrm{cost}(\mathcal{P}) \le \frac{|L|}{2} \cdot \frac{4\cdot|R|+|T|}{3}$ *when $|L|$ is even, and* $\mathrm{cost}(\mathcal{P}) \le |R| + \frac{|L|-1}{2} \cdot \frac{4\cdot|R|+|T|}{3}$ *when $|L|$ is odd.*

We first consider the special case when $|L| = 2$. Later we reduce the general case to this scenario and use the result as a black-box.

As a warm-up we construct a cyclic partition of cost at most $\frac{7}{8}|D|^2$ assuming that $L = R = T = D$ and that $|D|$ is even. For this, we first partition D into pairs $d_1^{(i)}, d_2^{(i)}$ where $i \in [|D|/2]$ and show for each such pair that f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and D has a cyclic partition of cost at most $\frac{7}{4}|D|$. The union of these cyclic partitions forms a cyclic partition of f with cost at most $\frac{|D|}{2} \cdot \frac{7}{4}|D|$.

To construct the cyclic partition for a fixed $i \in [|D|/2]$, we find a maximal number r of pairwise disjoint pairs $e_1^{(j)}, e_2^{(j)} \in D$ such that $|\{f(d_a^{(i)}, e_b^{(j)}) \mid a, b = 1, 2\}| \le 3$ for each $j \in [r]$, i.e. for each j at least one of the four values repeats. With this assumption, f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and $\{e_1^{(j)}, e_2^{(j)}\}$ is either a cyclic minor of cost at most 3 or can be decomposed into 3 trivial cyclic minors of the total cost at most 3. We claim that $r \ge |D|/4$. Indeed, assume that there are fewer than $|D|/4$ such pairs, i.e. $r < |D|/4$. Let $\overline{D}$ denote the $|D| - 2 \cdot r > |D|/2$ remaining values in D. As the set $\{f(d_a^{(i)}, d) \mid d \in \overline{D}, a = 1, 2\}$ can only contain at most $|D|$ values, we can find another pair $e_1^{(r+1)}, e_2^{(r+1)}$ with the above constraints. Note that f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and $\overline{D}$ can be decomposed into at most $2|\overline{D}|$ trivial minors. Hence, the cyclic partition for f restricted to $\{d_1^{(i)}, d_2^{(i)}\}$ and D has cost at most

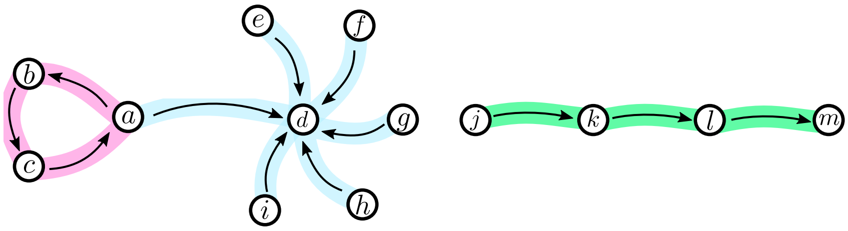
$$3r + 2 \cdot |\overline{D}| \le 3 \cdot \frac{|D|}{4} + 2 \cdot \frac{|D|}{2} \le \frac{7}{4}|D|.$$

4.1 Special Case

In this section, we prove the following lemma that is a special case of Lemma 4.1.

▶ **Lemma 4.2.** *If $f\colon L \times R \to T$ with $|L| = 2$, then there is a cyclic partition $\mathcal{P}$ of f such that* $\mathrm{cost}(\mathcal{P}) \le (4|R| + |T|)/3$.

To construct the cyclic partition we proceed as follows. First we define, for a function f, the representation graph G_f. Next we show that if this graph has a special structure, which we later call *nice*, then we can easily find a cyclic partition for the function f. Afterwards we decompose (the edges of) an arbitrary representation graph G_f into nice structures and then combine the cyclic partitions coming from these parts to a cyclic partition for the original function f.



	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}
ℓ_0	a	b	c	a	e	f	g	h	i	j	k	l
ℓ_1	b	c	a	d	d	d	d	d	d	k	l	m

Figure 4.1 Example of the construction of a representation graph from the function f to obtain a cyclic partition. We put an edge between vertices u and v if there is an r_i with $u = f(\ell_0, r_i)$ and $v = f(\ell_1, r_i)$. We highlight the partition of the graph into cycle (red), in-star (blue) and path (green). The cost of this cyclic partition is $3 + 7 + 4 = 14$.

▶ **Definition 4.3.** *Let $f : L \times R \to T$ be a such that $|L| = 2$ with $L = \{\ell_0, \ell_1\}$.*

We define a function $\lambda_f : R \to T \times T$ with $\lambda_f : r \mapsto (f(\ell_0, r), f(\ell_1, r))$ as the edge mapping *of f.*

We define a directed graph G_f (which might have loops) with vertex set $V(G_f) \coloneqq T$ and edge set $E(G_f) \coloneqq \{\lambda_f(r) \mid r \in R\}$. We say that G_f is the representation graph *of f.*

We say that a representation graph G_f is nice *if it is a cycle, a path (potentially with only one edge), an in-star, or an out-star.*[5]

Let $E' \subseteq E(G_f)$ be a subset of edges inducing the subgraph G' of G_f. With $T' \coloneqq V(G')$ and $R' \coloneqq \{r \in R \mid \lambda_f(r) \in E'\}$, we define $f' : L \times R' \to T'$ as the restriction of f such that the representation graph of f' is G'. Formally, $f'(\ell, r) = f(\ell, r)$ for all $\ell \in L$ and $r \in R'$. We say that f' is the function represented by *G' or E', respectively.*

A decomposition of a directed graph G is a set C of edge-disjoint subgraphs of G, such that each edge belongs to exactly one set in C. For ease of notation, we identify the set of edges E' with the induced graph G'. For a graph G, the *line graph* $L(G)$ is a graph where the set of edges $E(G)$ is the vertex set of $L(G)$ and $e_1, e_2 \in V(L(G))$ are adjacent in $L(G)$ if edges e_1, e_2 share an endpoint in graph G.

The following observation follows directly from the previous definition.

▶ **Observation 4.4.** *Let $G_1, \ldots, G_p$ be decomposition of the graph G_f into p subgraphs, let f_i be the function represented by G_i, and let $\mathcal{P}_i$ be a cyclic partition of f_i. Then $\mathcal{P} = \bigcup_{i \in [p]} \mathcal{P}_i$ is a cyclic partition of f with cost $\operatorname{cost}(\mathcal{P}) = \sum_{i \in [p]} \operatorname{cost}(\mathcal{P}_i)$.*

Cyclic Partitions Given Nice Representation Graphs. As a next step, we show that functions admit cyclic partitions if the representation graph is nice. Afterwards we show how to decompose (the representation) graphs into nice (representation) graphs. Finally, we combine these results to obtain a cyclic partition for the original function f. See Figure 4.1 for an example.

▶ **Lemma 4.5.** *Let $f : L \times R \to T$ be a function such that G_f is nice. Then f has a cyclic partition of cost at most $|T|$.*

[5] A star graph where either all edges are directed to the central vertex or all edges are directed away from it, respectively.

Proof. By definition, a nice graph is either a path, a cycle or an in-star or out-star. We handle each case separately in the following. Let $L = \{\ell_0, \ell_1\}$.

G_f is a cycle. We first define the relabeling functions of f to show that f is $|T|$-cyclic.
For the elements in L, let $\sigma_L : L \to \mathbb{Z}_2$ with $\sigma_L(\ell_i) = i$. To define σ_R and σ_T, fix an arbitrary $t_0 \in T$. Let $t_1, \ldots, t_{|T|}$ be the elements in T with $t_{|T|} = t_0$ such that, for all $i \in \mathbb{Z}_{|T|}$, there is some $r_i \in R$ with $\lambda_f(r_i) = (t_i, t_{i+1})$.[6] Note that these r_i exist since G_f is a cycle. Using this notation, we define $\sigma_T : \mathbb{Z}_{|T|} \to T$ with $\sigma_T(i) = t_i$, for all $i \in \mathbb{Z}_{|T|}$. For the elements in R we define $\sigma_R : R \to \mathbb{Z}_{|R|}$ with $\sigma_R(r) = i$ whenever $\lambda_f(r) = (t_i, t_{i+1})$ for some i.
It is easy to check that f can be seen as addition modulo $|T|$. Indeed, let $j \in \{0, 1\}$ and $r \in R$ with $\lambda_f(r) = (t_i, t_{i+1})$. Then we get

$$\sigma_T(\sigma_L(\ell_j) + \sigma_R(r) \mod |T|) = \sigma_T(j + i \mod |T|) = t_{j+i \mod |T|} = f(\ell_j, r_i) = f(\ell_j, r).$$

Thus, f is $|T|$-cyclic and $\{(L, R, |T|)\}$ is a cyclic partition of f.

G_f is a path. Similarly to the previous case, f can be represented as addition modulo $|T|$. As the proof is essentially identical to the cyclic case, we omit the details here.

G_f is a star. We only consider the out-star as the in-star follows symmetrically by swapping the roles of ℓ_0 and ℓ_1. We define the following cyclic partition $\mathcal{P}$ as

$$\mathcal{P} \coloneqq \{(\{\ell_0\}, R, 1)\} \cup \{(\{\ell_1\}, \{r\}, 1) \mid r \in R\}.$$

Note that every $(\ell, r) \in L \times R$ appears in exactly one minor of $\mathcal{P}$. Hence, $\mathcal{P}$ is indeed a cyclic partition. Next, we observe that each minor contains exactly one element of T. Thus, no addition is needed and hence f is 1-cyclic for each minor of $\mathcal{P}$. Thus the cost of each minor of $\mathcal{P}$ is 1.
By the structure of G_f, the cost of cyclic partition $\mathcal{P}$ is $|R| + 1 = |T|$. ◀

Decomposition into Nice Graphs. We first decompose the graph into cycles and acyclic components. The later parts are then decomposed further using the next two results.

▷ Claim 4.6. Every directed graph G can be decomposed into cycles and acyclic graphs.

Proof. We remove an arbitrary cycle from the graph and add it as a new component to the decomposition. We repeat this procedure until the graph is acyclic. ◁

Next, we decompose the acyclic graph further. First, we decompose it into pairs of edges that share at least one endpoint.

▷ Claim 4.7. Every directed graph $G = (V, E)$ whose undirected version is connected can be decomposed into $\lfloor |E|/2 \rfloor$ pairs of edges which share (at least) one endpoint and, if and only if $|E|$ is odd, one additional single edge.

Proof. If the graph has an odd number of edges, then we find one edge e, such that the removal of e does not disconnect the graph (except for maybe one isolated vertex). Next, we include this edge into the decomposition and apply the procedure for the case when the number of edges is even.

Chartrand et al. [16, Theorem 1] showed that if a graph G has an even number of edges, then there is a perfect matching in the line graph $L(G)$ of G. This perfect matching directly gives us a partition of the edges of G into $\lfloor |E|/2 \rfloor$ pairs which share at least one endpoint. ◁

[6] Note that there might be multiple $r \in R$ with $\lambda_f(r) = (t_i, t_{i+1})$.

Next, we present a different way to decompose the graph into nice structures.

▷ Claim 4.8. Every directed acyclic graph $G = (V, E)$ can be decomposed into at most $|V| - 1$ out-stars.

Proof. The sets of out-going edges from each vertex form a partition of all edges of the graph G. Moreover, each such non-empty set of edges describes an out-star. As in every directed acyclic graph, there is at least one sink vertex, there are at most $|V| - 1$ such out-stars. ◁

Combining the Results. Finally, we are ready to combine the above results and prove Lemma 4.2.

Proof of Lemma 4.2. We first use Claim 4.6 to decompose G_f into c cycles $C_1, \dots, C_c$ and d connected, acyclic graphs $G_1, \dots, G_d$.

For each C_i, Lemma 4.5 gives us a cyclic partition $\mathcal{P}'_i$ for the associated function with cost at most $|E(C_i)| = |V(C_i)|$. For the remaining components, we use the following claim.

▷ Claim 4.9. For each G_i, there is a cyclic partition $\mathcal{P}_i$ for the function represented by G_i with the cost at most

$$\mathrm{cost}(\mathcal{P}_i) \leq \frac{4|E(G_i)| + |V(G_i)|}{3}. \tag{4.1}$$

Proof. Fix some $i \in [d]$ in the following. We show the claim by considering two cases. For ease of notation, let $E_i = E(G_i)$ and $V_i = V(G_i)$.

In the case when $2|V_i| \geq |E_i| + 3$, we decompose the graph G_i via Claim 4.7. This decomposes G_i into $\lfloor |E_i|/2 \rfloor$ pairs of edges that share an endpoint (plus an extra edge when $|E_i|$ is odd). Observe that a pair of edges that share an endpoint is either a directed path, an in-star, or an out-star. Hence, by Lemma 4.5 each pair contributes a cost of 3 to the cyclic partition. Therefore, by Observation 4.4, the function represented by G_i has a cyclic partition with a cost at most $3|E_i|/2$ if $|E_i|$ is even and with cost at most $3(|E_i| - 1)/2 + 2$ if $|E_i|$ is odd. As the latter bound is the larger one, it can be easily checked that the claimed bound for the cyclic partition follows.

It remains to analyze case $2|V_i| < |E_i| + 3$. Here, we use Claim 4.8 to decompose the graph G_i into out-stars. By Observation 4.4 and as there is at least one sink vertex, there is a cyclic partition of the function represented by G_i with cost at most $|E_i| + |V_i| - 1$. By the assumption that $2|V_i| \leq |E_i| + 3$, the cost of the cyclic partition is bounded by $(4|E_i| + |V_i|)/3$ which settles (4.1). ◁

With the notation from the claim and by Observation 4.4, we define the cyclic partition $\mathcal{P}$ for f as

$$\mathcal{P} \coloneqq \bigcup_{i \in [c]} \mathcal{P}'_i \cup \bigcup_{i \in [d]} \mathcal{P}_i.$$

Because the G_is are connected components of G_f after the removal of $C_1 \dots C_c$, they are vertex-disjoint and it holds that $\sum_{i \in [d]} |V(G_i)| \leq |V(G_f)|$. Moreover, by Lemma 4.5 the cost of each cycle C_i is $|E(C_i)|$. Hence, we get

$$\mathrm{cost}(\mathcal{P}) \leq \sum_{i \in [c]} |E(C_i)| + \sum_{i \in [d]} \frac{4|E(G_i)| + |V(G_i)|}{3} \leq \frac{4|E(G_f)| + |V(G_f)|}{3}.$$

Because $|E(G_f)| \leq |R|$ and $|V(G_f)| = |T|$ the cost of this cyclic partition is bounded which finishes the proof. ◀

4.2 General Case

Now we have everything ready to prove the main result of this section.

Proof of Lemma 4.1. We first handle the case when $|L|$ is even. We partition L into $\lambda = |L|/2$ sets $L_1, \ldots, L_\lambda$ consisting of exactly two elements. We use Lemma 4.2 to find a cyclic partition $\mathcal{P}_i$ for each $f_i \colon L_i \times R \to T$. By definition of the cyclic partition, $\mathcal{P} = \bigcup_{i \in [\lambda]} \mathcal{P}_i$ is a cyclic partition for f, hence it remains to analyze the cost of $\mathcal{P}$.

Observe that for each G_i we have that $|V_i| \le |T|$ and $|E_i| \le |R|$. By the definition of the cost of the cyclic partition, we immediately get that

$$\mathrm{cost}(\mathcal{P}) \le \sum_{i=1}^{\lambda} \mathrm{cost}(\mathcal{P}_i) \le \lambda \cdot \frac{4 \cdot |R| + |T|}{3}.$$

If $|L|$ is odd, then we remove one element ℓ from L and let $L_0 = \{\ell\}$. There is a trivial cyclic partition $\mathcal{P}_0$ for $f_0 \colon L_0 \times R \to T$ of cost at most $|R|$. Then we use the above procedure to find a cyclic partition $\mathcal{P}'$ for the restriction of f to $L \setminus \{\ell\}$ and R. Hence, setting $\mathcal{P} = \mathcal{P}_0 \cup \mathcal{P}'$ gives a cyclic partition for f with cost

$$\mathrm{cost}(\mathcal{P}) \le \mathrm{cost}(\mathcal{P}_0) + \mathrm{cost}(\mathcal{P}') \le |R| + \left\lfloor \frac{|L|}{2} \right\rfloor \left(\frac{4 \cdot |R| + |T|}{3} \right). \quad ◀$$

▶ Remark 4.10. If $|L|$ and $|R|$ are both even, one can easily achieve the following cost

$$\min \left(\frac{L}{2} \cdot \frac{4 \cdot |R| + |T|}{3}, \frac{R}{2} \cdot \frac{4 \cdot |L| + |T|}{3} \right)$$

by swapping the role of L and R and considering the function $f' \colon R \times L \to T$ with $f'(r, \ell) = f(\ell, r)$ for all $\ell \in L$ and $r \in R$.

5 Conclusion and Future Work

In this paper, we studied the f-Convolution problem and demonstrated that the naive brute-force algorithm can be improved for every $f \colon D \times D \to D$. We achieve that by introducing a *cyclic partition* of a function and showing that there always exists a cyclic partition of bounded cost. We give an $\widetilde{\mathcal{O}}((c|D|^2)^n \cdot \mathrm{polylog}(M))$ time algorithm that computes f-Convolution for $c := 5/6$ when $|D|$ is even.

The cyclic partition is a very general tool and potentially it can be used to achieve greater improvements for certain functions f. For example, in multiple applications (e.g., [20, 34, 24, 29]) the function f has a cyclic partition with a single cyclic minor. Nevertheless, in our proof we only use cyclic minors where one dimension is at most 2. We suspect that better results can be obtained by considering larger minors.

We leave several open problems. Our algorithm offers an exponential (in n) improvement over a naive algorithm for domains D of constant size. Can we hope for an $\widetilde{\mathcal{O}}(|D|^{(2-\epsilon)n} \cdot \mathrm{polylog}(M))$ time algorithm for f-Convolution for some $\epsilon > 0$? We are not aware of any lower bounds, so in principle even an $\widetilde{\mathcal{O}}(|D|^n \cdot \mathrm{polylog}(M))$ time algorithm is plausible.

Ideally, we would expect that the f-Convolution problem can be solved in $\widetilde{\mathcal{O}}((|L|^n + |R|^n + |T|^n) \cdot \mathrm{polylog}(M))$ for any function $f \colon L \times R \to T$. In Figure 5.1 we include three examples of functions that are especially difficult for our methods.

Finally, we gave an $\widetilde{\mathcal{O}}(|D|^{\omega \cdot n/2} \cdot \mathrm{polylog}(M))$ time algorithm for f-Query problem. For $\omega = 2$ this algorithm runs in almost linear-time, however for the current bound $\omega < 2.373$ our algorithm runs in time $\widetilde{\mathcal{O}}(|D|^{1.19n} \cdot \mathrm{polylog}(M))$. Can f-Query be solved in $\widetilde{\mathcal{O}}(|D|^n \cdot \mathrm{polylog}(M))$ time without assuming $\omega = 2$?

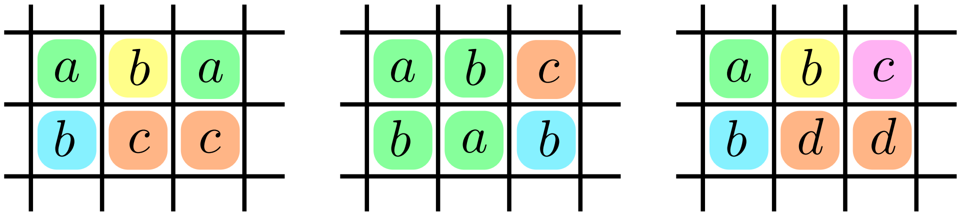


Figure 5.1 Here are three concrete examples of functions f for which we expect that the running times for f-Convolution should be $\widetilde{\mathcal{O}}(3^n \cdot \text{polylog}(M))$, $\widetilde{\mathcal{O}}(3^n \cdot \text{polylog}(M))$ and $\widetilde{\mathcal{O}}(4^n \cdot \text{polylog}(M))$. However, the best cyclic partitions for this functions have costs 4, 4 and 5 (the partitions are highlighted appropriately). This implies that the best running time, which may be attained using our techniques are $\widetilde{\mathcal{O}}(4^n \cdot \text{polylog}(M))$, $\widetilde{\mathcal{O}}(4^n \cdot \text{polylog}(M))$ and $\widetilde{\mathcal{O}}(5^n \cdot \text{polylog}(M))$.

References

1 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More Applications of the Polynomial Method to Algorithm Design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.

2 Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

3 Michael A. Bennett, Greg Martin, Kevin O'Bryant, and Andrew Rechnitzer. Explicit bounds for primes in arithmetic progressions. *Illinois J. Math.*, 62(1-4):427–532, 2018. doi:10.1215/ijm/1552442669.

4 Thomas Beth. *Verfahren der schnellen Fourier-Transformation: die allgemeine diskrete Fourier-Transformation–ihre algebraische Beschreibung, Komplexität und Implementierung*, volume 61. Teubner, 1984.

5 Andreas Björklund and Thore Husfeldt. The Parity of Directed Hamiltonian Cycles. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 727–735. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.83.

6 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier Meets Möbius: Fast Subset Convolution. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM, 2007. doi:10.1145/1250790.1250801.

7 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting Paths and Packings in Halves. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2009. doi:10.1007/978-3-642-04128-0_52.

8 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Covering and packing in linear space. *Inf. Process. Lett.*, 111(21-22):1033–1036, 2011. doi:10.1016/j.ipl.2011.08.002.

9 Andreas Björklund, Thore Husfeldt, Petteri Kaski, Mikko Koivisto, Jesper Nederlof, and Pekka Parviainen. Fast Zeta Transforms for Lattices with Few Irreducibles. *ACM Trans. Algorithms*, 12(1):4:1–4:19, 2016. doi:10.1145/2629429.

10 Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009. doi:10.1137/070683933.

11 Cornelius Brand. Discriminantal subset convolution: Refining exterior-algebraic methods for parameterized algorithms. *Journal of Computer and System Sciences*, 129:62–71, 2022. doi:10.1016/j.jcss.2022.05.004.

12 Karl Bringmann, Nick Fischer, Danny Hermelin, Dvir Shabtay, and Philip Wellnitz. Faster Minimization of Tardy Processing Time on a Single Machine. *Algorithmica*, 84(5):1341–1356, 2022. doi:10.1007/s00453-022-00928-w.

13 Karl Bringmann, Marvin Künnemann, and Karol Węgrzycki. Approximating APSP without scaling: equivalence of approximate min-plus and exact min-max. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 943–954, 2019.

14 Timothy M. Chan and Qizheng He. Reducing 3SUM to Convolution-3SUM. In Martin Farach-Colton and Inge Li Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 1–7. SIAM, 2020. doi:10.1137/1.9781611976014.1.

15 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021. doi:10.1145/3402926.

16 Gary Chartrand, Albert D Polimeni, and M James Stewart. The existence of 1-factors in line graphs, squares, and total graphs. In *Indagationes Mathematicae (Proceedings)*, volume 76, pages 228–232. Elsevier, 1973.

17 Michael Clausen. Fast generalized Fourier transforms. *Theoretical Computer Science*, 67(1):55–63, 1989.

18 James W Cooley and John W Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

19 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michal Włodarczyk. On Problems Equivalent to $(\min, +)$-Convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019. doi:10.1145/3293465.

20 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michał Pilipczuk, Johan M. M. van Rooij, and Jakub Onufry Wojtaszczyk. Solving Connectivity Problems Parameterized by Treewidth in Single Exponential Time. *ACM Trans. Algorithms*, 18(2):17:1–17:31, 2022. doi:10.1145/3506707.

21 Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*, 411(40-42):3701–3713, 2010. doi:10.1016/j.tcs.2010.06.018.

22 Barış Can Esmer, Ariel Kulik, Dániel Marx, Philipp Schepper, and Karol Węgrzycki. Computing Generalized Convolutions Faster Than Brute Force, 2022. doi:10.48550/ARXIV.2209.01623.

23 Philip Hall. A contribution to the theory of groups of prime-power order. *Proceedings of the London Mathematical Society*, 2(1):29–95, 1934.

24 Falko Hegerfeld and Stefan Kratsch. Solving Connectivity Problems Parameterized by Treedepth in Single-Exponential Time and Polynomial Space. In Christophe Paul and Markus Bläser, editors, *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*, volume 154 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.STACS.2020.29.

25 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the Fine-Grained Complexity of One-Dimensional Dynamic Programming. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.21.

26 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic Triangles, Intermediate Matrix Products, and Convolutions. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 53:1–53:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.53.

27 Jesper Nederlof. personal communication, 2022.

28 Jesper Nederlof, Jakub Pawlewicz, Céline M. F. Swennenhuis, and Karol Węgrzycki. A Faster Exponential Time Algorithm for Bin Packing With a Constant Number of Bins via Additive Combinatorics. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021*, pages 1682–1701. SIAM, 2021. doi:10.1137/1.9781611976465.102.

29 Jesper Nederlof, Michał Pilipczuk, Céline M. F. Swennenhuis, and Karol Węgrzycki. Hamiltonian Cycle Parameterized by Treedepth in Single Exponential Time and Polynomial Space. In Isolde Adler and Haiko Müller, editors, *Graph-Theoretic Concepts in Computer Science – 46th International Workshop, WG 2020, Leeds, UK, June 24-26, 2020, Revised Selected Papers*, volume 12301 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2020. doi:10.1007/978-3-030-60440-0_3.

30 Jesper Nederlof and Karol Węgrzycki. Improving Schroeppel and Shamir's Algorithm for Subset Sum via Orthogonal Vectors. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1670–1683. ACM, 2021. doi:10.1145/3406325.3451024.

31 Daniel N Rockmore. Recent progress and applications in group FFTs. In *Computational noncommutative algebra and applications*, pages 227–254. Springer, 2004.

32 Chris Umans. Fast Generalized DFTs for all Finite Groups. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 793–805. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00052.

33 Johan M. M. van Rooij. Fast Algorithms for Join Operations on Tree Decompositions. In Fedor V. Fomin, Stefan Kratsch, and Erik Jan van Leeuwen, editors, *Treewidth, Kernels, and Algorithms – Essays Dedicated to Hans L. Bodlaender on the Occasion of His 60th Birthday*, volume 12160 of *Lecture Notes in Computer Science*, pages 262–297. Springer, 2020. doi:10.1007/978-3-030-42071-0_18.

34 Johan M. M. van Rooij. A Generic Convolution Algorithm for Join Operations on Tree Decompositions. In Rahul Santhanam and Daniil Musatov, editors, *Computer Science – Theory and Applications – 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 – July 2, 2021, Proceedings*, volume 12730 of *Lecture Notes in Computer Science*, pages 435–459. Springer, 2021. doi:10.1007/978-3-030-79416-3_27.

35 Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In Amos Fiat and Peter Sanders, editors, *Algorithms – ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer, 2009. doi:10.1007/978-3-642-04128-0_51.

36 Virginia Vassilevska-Williams. On Some Fine-Grained Questions in Algorithms and Complexity. In *Proceedings of the International Congress of Mathematicians (ICM 2018)*, pages 3447–34, 2018.

37 Louis Weisner. Abstract theory of inversion of finite series. *Transactions of the American Mathematical Society*, 38(3):474–484, 1935.

38 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.

39 Michał Włodarczyk. Clifford Algebras Meet Tree Decompositions. *Algorithmica*, 81(2):497–518, 2019. doi:10.1007/s00453-018-0489-3.

40 Frank Yates. The design and analysis of factorial experiments. *Technical Communication No. 35.*, 1937.

41 Or Zamir. Breaking the 2^n Barrier for 5-Coloring and 6-Coloring. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 113:1–113:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.113.

A Querying a Generalized Convolution

In this section, we prove Theorem 1.4. The main idea is to represent the f-Query problem as a matrix multiplication problem, inspired by a graph interpretation of f-Query.

Let D be an arbitrary set and $f\colon D \times D \to D$. We assume D and f are fixed throughout this section. Let $g, h\colon D^n \to \{-M, \dots, M\}$ and $\mathbf{v} \in D^n$ be a f-Query instance. We use $\mathbf{a} \| \mathbf{b}$ to denote the concatenation of $\mathbf{a} \in D^m$ and $\mathbf{b} \in D^k$. That is $(\mathbf{a}_1, \dots, \mathbf{a}_m) \| (\mathbf{b}_1, \dots, \mathbf{b}_k) = (\mathbf{a}_1, \dots, \mathbf{a}_m, \mathbf{b}_1, \dots, \mathbf{b}_k)$. If we assume that n is even, then, for a vector $\mathbf{v} \in D^n$, let $\mathbf{v}^{(\mathrm{high})}, \mathbf{v}^{(\mathrm{low})} \in D^{n/2}$ be the unique vectors such that $\mathbf{v}^{(\mathrm{high})} \| \mathbf{v}^{(\mathrm{low})} = \mathbf{v}$. Indeed, to achieve this assumption let n be odd, fix an arbitrary $d \in D$, and define $\widetilde{g}, \widetilde{h}\colon D^{n+1} \to \{-M, \dots, M\}$ as $\widetilde{g}(\mathbf{u}_1, \dots \mathbf{u}_{n+1}) = [\![\mathbf{u}_{n+1} = d]\!] \cdot g(\mathbf{u}_1, \dots \mathbf{u}_n)$ and $\widetilde{h}(\mathbf{u}_1, \dots \mathbf{u}_{n+1}) = [\![\mathbf{u}_{n+1} = d]\!] \cdot h(\mathbf{u}_1, \dots \mathbf{u}_n)$ for all $\mathbf{u} \in D^{n+1}$. It can be easily verified that $(g \circledast_f h)(\mathbf{v}) = (\widetilde{g} \circledast_f \widetilde{h})(\mathbf{v} \| (f(d, d)))$. Thus, we can solve the f-Query instance $\widetilde{g}$, $\widetilde{h}$ and $\mathbf{v} \| (f(d, d))$ and obtain the correct result.

We first provide the intuition behind the algorithm and then formally show the existence.

Intuition. We define a directed multigraph G where the vertices are partitioned into four layers $\mathrm{L}^{(\mathrm{high})}$, $\mathrm{L}^{(\mathrm{low})}$, $\mathrm{R}^{(\mathrm{low})}$, and $\mathrm{R}^{(\mathrm{high})}$. Each of these sets consists of $|D|^{n/2}$ vertices representing every vector in $D^{n/2}$. For ease of notation, we use the vectors to denote the associated vertices; furthermore, the intuition assumes g and h are non-negative. The multigraph G contains the following edges:

- $g(\mathbf{w} \| \mathbf{x})$ parallel edges from $\mathbf{w} \in D^{n/2}$ in $\mathrm{L}^{(\mathrm{high})}$ to $\mathbf{x} \in D^{n/2}$ in $\mathrm{L}^{(\mathrm{low})}$.
- One edge from $\mathbf{x} \in D^{n/2}$ in $\mathrm{L}^{(\mathrm{low})}$ to $\mathbf{y} \in D^{n/2}$ in $\mathrm{R}^{(\mathrm{low})}$ if and only if $\mathbf{x} \oplus_f \mathbf{y} = v^{(\mathrm{low})}$.
- $h(\mathbf{z} \| \mathbf{y})$ parallel edges from $\mathbf{y} \in D^{n/2}$ in $\mathrm{R}^{(\mathrm{low})}$ to $\mathbf{z} \in D^{n/2}$ in $\mathrm{R}^{(\mathrm{high})}$.
- One edge from $\mathbf{z} \in D^{n/2}$ in $\mathrm{R}^{(\mathrm{high})}$ to $\mathbf{w} \in D^{n/2}$ in $\mathrm{L}^{(\mathrm{high})}$ if and only if $\mathbf{w} \oplus_f \mathbf{z} = v^{(\mathrm{high})}$.

In the formal proof, we denote the adjacency matrix between $\mathrm{L}^{(\mathrm{high})}$ and $\mathrm{L}^{(\mathrm{low})}$ by W, between $\mathrm{L}^{(\mathrm{low})}$ and $\mathrm{R}^{(\mathrm{low})}$ by X, between $\mathrm{R}^{(\mathrm{low})}$ and $\mathrm{R}^{(\mathrm{high})}$ by Y, and between $\mathrm{R}^{(\mathrm{high})}$ and $\mathrm{L}^{(\mathrm{high})}$ by Z. See Figure A.1 for an example of this construction.

Let $\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}$ be vertices in $\mathrm{L}^{(\mathrm{high})}$, $\mathrm{L}^{(\mathrm{low})}$, $\mathrm{R}^{(\mathrm{low})}$, and $\mathrm{R}^{(\mathrm{high})}$. It can be observed that if $(\mathbf{w} \| \mathbf{x}) \oplus_f (\mathbf{y} \| \mathbf{z}) \neq \mathbf{v}$, then G does not contain any cycle of the form $\mathbf{w} \to \mathbf{x} \to \mathbf{y} \to \mathbf{z} \to \mathbf{w}$ as one of the edges $(\mathbf{x}, \mathbf{y})$ or $(\mathbf{z}, \mathbf{w})$ is not present in the graph. Conversely, if $(\mathbf{w} \| \mathbf{x}) \oplus_f (\mathbf{y} \| \mathbf{z}) = \mathbf{v}$, then one can verify that there are $g(\mathbf{w} \| \mathbf{x}) \cdot h(\mathbf{z} \| \mathbf{y})$ cycles of the form $\mathbf{w} \to \mathbf{x} \to \mathbf{y} \to \mathbf{z} \to \mathbf{w}$. We therefore expect that $(g \circledast_f h)(\mathbf{v})$ is the number of cycles in G that start at some $\mathbf{w} \in D^{n/2}$ in $\mathrm{L}^{(\mathrm{high})}$, have length four, and end at the same vertex $\mathbf{w}$ in $\mathrm{L}^{(\mathrm{high})}$ again.

Formal Proof. We use the notation $\mathsf{Mat}_{\mathbb{Z}}(D^{n/2} \times D^{n/2})$ to refer to a $|D|^{n/2} \times |D|^{n/2}$ matrix of integers where we use the values in $D^{n/2}$ as indices. The *transition matrices* of g, h and $\mathbf{v}$ are the matrices $W, X, Y, Z \in \mathsf{Mat}_{\mathbb{Z}}(D^{n/2} \times D^{n/2})$ defined by

$$
\begin{aligned}
W_{\mathbf{w},\mathbf{x}} &\coloneqq g(\mathbf{w} \| \mathbf{x}) && \forall \mathbf{w}, \mathbf{x} \in D^{n/2}\\
X_{\mathbf{x},\mathbf{y}} &\coloneqq [\![\mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\mathrm{low})}]\!] && \forall \mathbf{x}, \mathbf{y} \in D^{n/2}\\
Y_{\mathbf{y},\mathbf{z}} &\coloneqq h(\mathbf{z} \| \mathbf{y}) && \forall \mathbf{y}, \mathbf{z} \in D^{n/2}\\
Z_{\mathbf{z},\mathbf{w}} &\coloneqq [\![\mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\mathrm{high})}]\!] && \forall \mathbf{z}, \mathbf{w} \in D^{n/2}
\end{aligned}
$$

Recall that the *trace* $\mathrm{tr}(A)$ of a matrix $A \in \mathsf{Mat}_{\mathbb{Z}}(m \times m)$ is defined as $\mathrm{tr}(A) \coloneqq \sum_{i=1}^{m} A_{i,i}$. The next lemma formalizes the correctness of this construction.

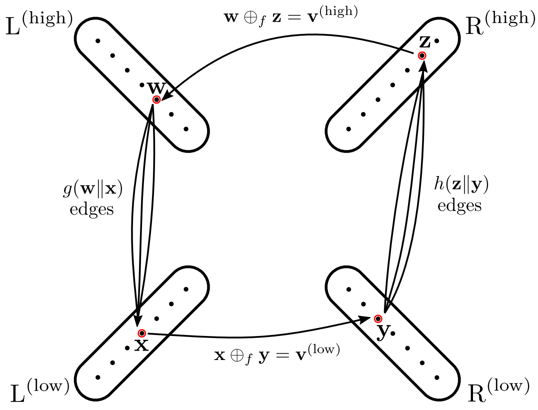


Figure A.1 Construction of the directed multigraph G. Each vertex in a layer corresponds to the vector in $D^{n/2}$. We highlighted 4 vectors $\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}$ each in a different layer. Note that the number of 4 cycles that go through all four $\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}$ is equal to $g(\mathbf{w}\|\mathbf{x}) \cdot h(\mathbf{z}\|\mathbf{y})$. The total number of directed 4-cycles in this graph corresponds to the value $(g \circledast_f h)(\mathbf{v})$ and $\mathrm{tr}(W \cdot X \cdot Y \cdot Z)$.

▶ **Lemma A.1.** *Let $n \in \mathbb{N}$ be an even number, $g, h \colon D^n \to \mathbb{Z}$ and $\mathbf{v} \in D^n$. Also, let $W, X, Y, Z \in \mathsf{Mat}_{\mathbb{Z}}(D^{n/2} \times D^{n/2})$ be the transition matrices of g, h and $\mathbf{v}$. Then,*

$$(g \circledast_f h)(\mathbf{v}) = \mathrm{tr}(W \cdot X \cdot Y \cdot Z).$$

Proof. For any $\mathbf{w}, \mathbf{y} \in D^{n/2}$ it holds that,

$$(W \cdot X)_{\mathbf{w},\mathbf{y}} = \sum_{\mathbf{x} \in D^{n/2}} W_{\mathbf{w},\mathbf{x}} \cdot X_{\mathbf{x},\mathbf{y}} = \sum_{\mathbf{x} \in D^{n/2}} [\![\mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\mathrm{low})}]\!] \cdot g(\mathbf{w}\|\mathbf{x}). \tag{A.1}$$

Similarly, for any $\mathbf{y}, \mathbf{w} \in D^{n/2}$ it holds that,

$$(Y \cdot Z)_{\mathbf{y},\mathbf{w}} = \sum_{\mathbf{z} \in D^{n/2}} Y_{\mathbf{y},\mathbf{z}} \cdot Z_{\mathbf{z},\mathbf{w}} = \sum_{\mathbf{z} \in D^{n/2}} [\![\mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\mathrm{high})}]\!] \cdot h(\mathbf{z}\|\mathbf{y}). \tag{A.2}$$

Therefore, for any $\mathbf{w} \in D^{n/2}$,

$$\begin{aligned}
(W \cdot X \cdot Y \cdot Z)_{\mathbf{w},\mathbf{w}} &= \sum_{\mathbf{y} \in D^{n/2}} (W \cdot X)_{\mathbf{w},\mathbf{y}} \cdot (Y \cdot Z)_{\mathbf{y},\mathbf{w}} \\
&= \sum_{\mathbf{y} \in D^{n/2}} \left(\sum_{\mathbf{x} \in D^{n/2}} [\![\mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\mathrm{low})}]\!] \cdot g(\mathbf{w}\|\mathbf{x}) \right) \left(\sum_{\mathbf{z} \in D^{n/2}} [\![\mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\mathrm{high})}]\!] \cdot h(\mathbf{z}\|\mathbf{y}) \right) \\
&= \sum_{\mathbf{x},\mathbf{y},\mathbf{z} \in D^{n/2}} [\![\mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\mathrm{low})}]\!] \cdot [\![\mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\mathrm{high})}\ ]\!] \cdot g(\mathbf{w}\|\mathbf{x}) \cdot h(\mathbf{z}\|\mathbf{y}) \\
&= \sum_{\mathbf{x},\mathbf{y},\mathbf{z} \in D^{n/2}} [\![(\mathbf{w}\|\mathbf{x}) \oplus_f (\mathbf{z}\|\mathbf{y}) = \mathbf{v}^{(\mathrm{high})}\|\mathbf{v}^{(\mathrm{low})}]\!] \cdot g(\mathbf{w}\|\mathbf{x}) \cdot h(\mathbf{z}\|\mathbf{y}),
\end{aligned}$$

where the second equality follows by (A.1) and (A.2). Thus,

$$\begin{aligned}\operatorname{tr}(W \cdot X \cdot Y \cdot Z) &= \sum_{\mathbf{w} \in D^{n/2}} (W \cdot X \cdot Y \cdot Z)_{\mathbf{w},\mathbf{w}} \\ &= \sum_{\mathbf{w} \in D^{n/2}} \sum_{\mathbf{x},\mathbf{y},\mathbf{z} \in D^{n/2}} [\![(\mathbf{w} \| \mathbf{x}) \oplus_f (\mathbf{z} \| \mathbf{y}) = \mathbf{v}]\!] \cdot g(\mathbf{w} \| \mathbf{x}) \cdot h(\mathbf{z} \| \mathbf{y}) \\ &= \sum_{\mathbf{u},\mathbf{t} \in D^n} [\![\mathbf{u} \oplus_f \mathbf{t} = \mathbf{v}]\!] \cdot g(\mathbf{u}) \cdot h(\mathbf{t}) \\ &= (g \circledast_f h)(\mathbf{v}). \qquad \blacktriangleleft\end{aligned}$$

Now we have everything ready to give the algorithm for f-Query.

Proof of Theorem 1.4. The algorithm for solving f-Query works in two steps:

1. Compute the transition matrices W, X, Y, and Z of g, h and $\mathbf{v}$ as described above.
2. Compute and return $\operatorname{tr}(W \cdot X \cdot Y \cdot Z)$.

By Lemma A.1 this algorithm returns $(g \circledast_f h)(\mathbf{v})$. Computing the transition matrices in Step 1 requires $\widetilde{\mathcal{O}}(|D|^n \cdot \operatorname{polylog}(M))$ time. Observe the maximal absolute values of an entry in the transition matrices is M. The computation of $W \cdot X \cdot Y \cdot Z$ in Step 2 requires three matrix multiplications of $|D|^{n/2} \times |D|^{n/2}$ matrices, which can be done in $\widetilde{\mathcal{O}}((|D|^{n/2})^\omega \cdot \operatorname{polylog}(M))$ time. Thus, the overall running time of the algorithm is $\widetilde{\mathcal{O}}(|D|^{\omega \cdot n/2} \cdot \operatorname{polylog}(M))$. ◀

B Proof of Theorem 2.5

In this section we prove Theorem 2.5. We let M be the absolute value of largest integer on the output of functions $g\colon L^n \to \mathbb{Z}$ and $h\colon R^n \to \mathbb{Z}$. We let $K := \prod_{i=1}^n \mathbf{r}_i$. We crucially rely on the following result by van Rooij [33].

▶ **Theorem B.1** ([33, Lemma 3]). *Let p denote a prime such that in the field $\mathbb{F}_p$, the $\mathbf{r}_i$-th root of unity exists for each $i \in [n]$. For two given functions $g, h\colon \mathbb{Z}_{\mathbf{r}_1} \times \cdots \times \mathbb{Z}_{\mathbf{r}_n} \to \mathbb{Z}$, we can compute their cyclic convolution modulo p (that is, return a function ϕ such that $\phi(\mathbf{q}) = (g \odot h)(\mathbf{v}) \mod p$ for every $\mathbf{q} \in \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_n}$) in time $\mathcal{O}(K \log(Kp))$ (assuming a $\mathbf{r}_i$-th primitive root of unity ω_i in the field $\mathbb{F}_p$ is given for all $i \in [n]$).*

The basic idea behind the proof is to compute $g \odot h$ modulo p_i for a sufficiently large number of distinct small primes p_i. If $\prod_i p_i > 2^n$, then the values of $g \odot h$ can be uniquely recovered using the Chinese Remainder Theorem.

▶ **Theorem B.2** (Chinese Remainder Theorem). *Let $m_1, \ldots, m_\ell$ denote a sequence of integers that are pairwise coprime and define $M := \prod_{i \in [\ell]} m_i$. Also let $0 \le a_i < m_i$ for all $i \in [\ell]$. Then there is a unique number $0 \le s < M$ such that*

$$s \equiv a_i \pmod{m_i}$$

for all $i \in [\ell]$. Moreover, there is an algorithm that, given $m_1, \ldots, m_\ell$ and $a_1, \ldots, a_\ell$, computes the number s in time $\mathcal{O}((\log M)^2)$.

Let $m := \lceil \log(3 \cdot |L|^n \cdot |R|^n \cdot M^2) \rceil$. We compute the list of the first m primes $p_1 < \cdots < p_m$ such that $p_i \equiv 1 \pmod{K}$ for all $i \in [m]$. By the Prime Number Theorem for Arithmetic Progressions (see, e.g., [3]) we get that $p_m = \mathcal{O}(\varphi(K) \cdot m \cdot \log m)$ where φ denotes Euler's totient function. In particular, $p_m = \mathcal{O}(K \cdot m \cdot \log m)$ because $\varphi(K) \le K$. Since prime testing can be done in polynomial time, we can find the sequence $p_1, \ldots, p_m$ in time $\mathcal{O}(K \cdot m \cdot (\log m)^c)$ for some constant c.

Next, for every $i \in [m]$ and $j \in [n]$, we compute a $\mathbf{r}_j$-th root of unity in $\mathbb{F}_{p_i}$ as follows. First observe that such a root of unity exists since $\mathbf{r}_j$ divides $p_i - 1$. For every $i \in [m]$ we first find the prime factors $q_{i,1}, \dots, q_{i,\ell_i}$ of $p_i - 1$ by iterating over every number in $\mathbb{Z}_{p_i}$ and checking if it is both prime and divides $p_i - 1$. This can be done in time $\mathcal{O}(p_i \cdot \mathrm{polylog}\ p_i)$. Next, we simply iterate over all elements $x \in \mathbb{F}_{p_i}$ and test whether a given element x is a $\mathbf{r}_j$-th root of unity in time $(\log p_i)^{\mathcal{O}(1)}$. So overall, computing all roots of unity for every p_i ($i \in [m]$) can be done in time

$$\sum_{i=1}^{m} \left(\mathcal{O}(p_i \cdot \mathrm{polylog}\ p_i) + \sum_{j=1}^{n} p_i \cdot (\log p_i)^{\mathcal{O}(1)} \right) = m \cdot n \cdot p_m \cdot (\log p_m)^{\mathcal{O}(1)} = K \cdot (n+m+\log K)^{\mathcal{O}(1)}.$$

Now, for every $i \in [m]$ and $\mathbf{q} \in \mathbb{Z}_{\mathbf{r}_1} \times \cdots \times \mathbb{Z}_{\mathbf{r}_n}$, we compute

$$(g \odot h)(\mathbf{q})^{(i)} \coloneqq (g \odot h)(\mathbf{q}) \pmod{p_i}$$

using Theorem B.1 in time $\mathcal{O}(m \cdot K \cdot \log(K \cdot p_m)) = K \cdot (m + \log K)^{\mathcal{O}(1)}$.

Finally, we can recover $(g \odot h)(\mathbf{q})$ for every $\mathbf{q}$ by the Chinese Remainder Theorem in time $\mathcal{O}(K \cdot m^2)$. Note that $\prod_{i\in[m]} p_i > 2^m \geq M$ which implies that all numbers are indeed uniquely recovered. In total, this achieves the desired running time. ◀

C Proof of Lemma 3.5

The idea is to use a dynamic programming algorithm loosely inspired by Yates algorithm [40].

Define $X^{(\ell)} = \left\{ (\overline{p}, \mathbf{q}) \;\middle|\; \overline{p} \in [m]^\ell,\ \mathbf{q} \in \mathbb{Z}_{\overline{p}_1} \times \cdots \times \mathbb{Z}_{\overline{p}_\ell} \right\}$ for every $\ell \in \{0, \dots, n\}$. We use $X^{(\ell)}$ to define a dynamic programming table $\mathsf{DP}^{(\ell)} \colon X^{(\ell)} \times L^{n-\ell} \to \mathbb{Z}$ for every $\ell \in \{0, \dots n\}$ by:

$$\mathsf{DP}^{(\ell)}[(\overline{p}_1, \dots, \overline{p}_\ell), (\mathbf{q}_1, \dots, \mathbf{q}_\ell)][\mathbf{t}_{\ell+1}, \dots, \mathbf{t}_n] \coloneqq \sum_{\substack{\mathbf{t}_1 \in A_{\overline{p}_1} \\ \cdots \\ \mathbf{t}_\ell \in A_{\overline{p}_\ell}}} \left(\prod_{i=1}^{\ell} [\![\sigma_{\overline{p}_i}(\mathbf{t}_i) = \mathbf{q}_i]\!] \right) \cdot g(\mathbf{t}_1, \dots, \mathbf{t}_n).$$

The tables $\mathsf{DP}^{(0)}, \mathsf{DP}^{(1)}, \dots, \mathsf{DP}^{(n)}$ are computed consecutively where the computation of $\mathsf{DP}^{(\ell)}$ relies on the values of $\mathsf{DP}^{(\ell-1)}$ for any $\ell \in [n]$. Observe that $g_{\overline{p}}(\mathbf{q}) = \mathsf{DP}^{(n)}[(\overline{p}_1, \dots, \overline{p}_n), (\mathbf{q}_1, \dots, \mathbf{q}_n)][\varepsilon]$ for every $\overline{p}$ and $\mathbf{q}$, which means that computing $\mathsf{DP}^{(n)}$ is equivalent to computing the projection functions $g_{\overline{p}}$ of g for every type $\overline{p}$.[7]

It holds that $\mathsf{DP}^{(0)}[\varepsilon, \varepsilon][\mathbf{t}] = g(\mathbf{t})$. Hence, $\mathsf{DP}^{(0)}$ can be trivially computed in $|L|^n$ time. We use the following straightforward recurrence to compute $\mathsf{DP}^{(\ell)}$:

$$\mathsf{DP}^{(\ell)}[(\overline{p}_1, \dots, \overline{p}_\ell), (\mathbf{q}_1, \dots, \mathbf{q}_\ell)][\mathbf{t}_{\ell+1}, \dots, \mathbf{t}_n] = \sum_{\mathbf{t}_\ell \in A_{\overline{p}_\ell}} [\![\sigma_{\overline{p}_\ell}(\mathbf{t}_\ell) = \mathbf{q}_\ell]\!] \cdot \mathsf{DP}^{(\ell-1)}[(\overline{p}_1, \dots, \overline{p}_{\ell-1}), (\mathbf{q}_1, \dots, \mathbf{q}_{\ell-1})][\mathbf{t}_\ell, \dots, \mathbf{t}_n]. \tag{C.1}$$

A dynamic programming algorithm which computes $\mathsf{DP}^{(n)}$ can be easily derived from (C.1) and the formula for $\mathsf{DP}^{(0)}$. The total number of states in the dynamic programming table $\mathsf{DP}^{(\ell)}$ is

$$\left(\sum_{\overline{p} \in [m]^\ell} (k_{\overline{p}_1} \cdot \ldots \cdot k_{\overline{p}_\ell}) \right) \cdot |L|^{n-\ell} = (k_1 + \cdots + k_m)^\ell \cdot |L|^{n-\ell} = \mathrm{cost}(\mathcal{P})^\ell \cdot |L|^{n-\ell}.$$

This is bounded by $\mathrm{cost}(\mathcal{P})^n + |L|^n$ for every $\ell \in [n]$. To transition between states we spend polynomial time per entry because we assume that $|L| = \mathcal{O}(1)$. Hence, we can compute $g_{\overline{p}}$ for every $\overline{p}$ in $\widetilde{\mathcal{O}}((\mathrm{cost}(\mathcal{P})^n + |L|^n) \cdot \mathrm{polylog}(M))$ time. ◀

[7] We use ε to denote the vector of length 0.