Algorithmica (2024) 86:334–366
https://doi.org/10.1007/s00453-023-01176-2

Computing Generalized Convolutions Faster Than Brute Force

Barış Can Esmer[1] · Ariel Kulik[1] · Dániel Marx[1] · Philipp Schepper[1] · Karol Węgrzycki[2]

Received: 13 January 2023 / Accepted: 20 September 2023 / Published online: 6 October 2023

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 $\widetilde{\mathcal{O}}(|D|^{2n} \cdot \operatorname{polylog}(M))$ time. Our main result is an improvement over this naive algorithm. We show that f-Convolution can be computed exactly in $\widetilde{\mathcal{O}}((c \cdot |D|^2)^n \cdot \operatorname{polylog}(M))$ for constant $c := 3/4$ 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

✉ Barış Can Esmer
baris-can.esmer@cispa.de

Ariel Kulik
ariel.kulik@cispa.de

Dániel Marx
marx@cispa.de

Philipp Schepper
philipp.schepper@cispa.de

Karol Węgrzycki
wegrzycki@cs.uni-saarland.de

1 CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

2 Max Planck Institute for Informatics, Saarland University, Saarbrücken, Germany

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 $\omega \in [2, 2.372)$ is the exponent of currently fastest matrix multiplication algorithm.

Keywords Generalized Convolution · Fast Fourier Transform · Fast Subset Convolution · Orthogonal Vectors

Mathematics Subject Classification Theory of computation · Parameterized complexity and exact algorithms · Theory of computation · Algorithm design techniques

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 [22, 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, Discriminantal Subset Convolution, Trimmed Subset Convolution or Lattice-based Convolution [6–8, 10, 11, 20, 24, 35]. 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 [5, 10, 19, 28, 39, 41]. These convolutions are especially useful for dynamic programming algorithms on tree decompositions and occur naturally during join operations (e.g., [19, 34, 35]). 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

[1] We use $\widetilde{\mathcal{O}}(x) = x \cdot \mathrm{polylog}(x)$ 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 Sect. 2.

$g, h \colon D^n \to \mathbb{Z}$ 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

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

Here we consider the 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 $\widetilde{\mathcal{O}}(|D|^n \cdot \operatorname{polylog}(M))$. 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 $\widetilde{\mathcal{O}}(|D|^n \cdot \operatorname{polylog}(M))$ 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: $f(x, y) := x + y$, or (ii) addition with a maximum: $f(x, y) := \min(x + y, r - 1)$, or (iii) addition modulo r, or (iv) maximum: $f(x, y) := \max(x, y)$. Van Rooij [34] showed that for these special cases the f-Convolution can be solved in $\widetilde{\mathcal{O}}(|D|^n \cdot \operatorname{polylog}(M))$ time. His results allow the function f to differ between coordinates. A recent result regarding generalized Discrete Fourier Transform [32] can be used in conjunction with Yates's algorithm [40] to compute f-Convolution in $\widetilde{\mathcal{O}}(|D|^{\omega \cdot n/2} \cdot \operatorname{polylog}(M))$ 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 $\widetilde{\mathcal{O}}(|D|^{2n} \cdot \operatorname{polylog}(M))$ 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 [18, 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 $\widetilde{\mathcal{O}}(|D|^{2n} \cdot \operatorname{polylog}(M))$ algorithm for every function f.

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

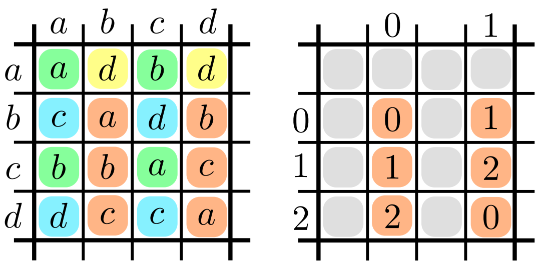


Fig. 1 Left figure illustrates exemplar function $f: D \times D \to D$ over domain $D := \{a, b, c, d\}$. We highlighted a cyclic partition with red, blue, yellow and blue colors. Each color represents a different minor of f. On the right figure we demonstrate that the red-highlighted minor can be represented 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 $3 + 2 + 2 + 1 = 8$ (Color figure online)

Theorem 1.1 (Generalized Convolution) *Let D be a finite set and $f: D \times D \to D$. There is an algorithm for f-*Convolution *with the following running time* $\widetilde{\mathcal{O}}\left(\left(\frac{3}{4} \cdot |D|^2\right)^n \cdot \text{polylog}(M)\right)$ *when $|D|$ is even, or* $\left(\left(\frac{3}{4} \cdot |D|^2 + \frac{1}{4} \cdot |D|\right)^n\right)$ *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 $g: L^n \to \mathbb{Z}$ and $h: R^n \to \mathbb{Z}$ and $f: L \times R \to T$ for arbitrary domains L, R and T (see Sect. 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 [33, 34] f-Convolution can be computed in $\widetilde{\mathcal{O}}(k^n \cdot \text{polylog}(M))$ time if f is k-cyclic. In a nutshell, our idea is to partition the function $f: D \times D \to D$ into cyclic functions and compute the convolution on these parts independently.

More formally, a cyclic minor of the function $f: D \times D \to D$ 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 $\text{cost}(A, B) := k$. A cyclic partition is a set $\{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$ 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 $\mathcal{P} = \{(A_1, B_1, k_1), \dots, (A_m, B_m, k_m)\}$ is $\text{cost}(\mathcal{P}) := \sum_{i=1}^{m} k_i$. See Fig. 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.

Lemma 1.2 (Algorithm for f-Convolution) *Let D be an arbitrary finite set, $f: D \times D \to D$ and let $\mathcal{P}$ be the cyclic partition of f. Then there exists an algorithm which given $g, h: D^n \to \mathbb{Z}$ computes $(g \circledast_f h)$ in $\widetilde{\mathcal{O}}((\text{cost}(\mathcal{P})^n + |D|^n) \cdot \text{polylog}(M))$ 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 $\boldsymbol{p} \in [m]^n$ such that $(\mathbf{u}_i, \mathbf{w}_i) \in$

$A_{\boldsymbol{p}_i} \times B_{\boldsymbol{p}_i}$ for every $i \in [n]$. Our main idea is to go over each type $\boldsymbol{p}$ and compute the sum in the definition of f-CONVOLUTION only for pairs $(\mathbf{v}_g, \mathbf{v}_h)$ that have type $\boldsymbol{p}$. In order to do this, first we select the vectors $\mathbf{v}_g$ and $\mathbf{v}_h$ that are compatible with this type $\boldsymbol{p}$. For instance, consider the example in Fig. 1. Whenever $\boldsymbol{p}_i$ 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 $\boldsymbol{p}$ in $\widetilde{\mathcal{O}}((\prod_{i=1}^{n} k_{\boldsymbol{p}_i}) \cdot \text{polylog}(M))$ time. As we go over all m^n types $\boldsymbol{p}$ the sum of m^n terms is

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

Hence, the overall running time is $\widetilde{\mathcal{O}}(\text{cost}(\mathcal{P})^n \cdot \text{polylog}(M))$. 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 Sect. 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 $f : D \times D \to D$ there is a cyclic partition $\mathcal{P}$ of f such that* $\text{cost}(\mathcal{P}) \le \frac{3}{4}|D|^2$ *when $|D|$ is even, or* $\text{cost}(\mathcal{P}) \le \frac{3}{4}|D|^2 + \frac{1}{4}|D|$ *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 $k{:=}|D|/2$ and consider the restrictions of f to $A_j \times D$ one by one. Intuitively, we partition the $D \times D$ table describing f into pairs of rows and give a bound on the cost of each pair. This partition allows us to encode f on $A_j \times D$ as a directed graph G with $|D|$ edges and $|D|$ vertices. We observe that directed cycles and directed paths can be represented as cyclic minors. Our goal is to partition 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 Sect. 4. We also give an example which suggests that the constant $\frac{3}{4}$ in Lemma 1.3 cannot be improved further while using the partition of D into arbitrary pairs (see Lemma 4.16).

Our method applies for more general functions $f : L \times R \to T$, 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 $\text{cost}(\mathcal{P}_f) \le \frac{7}{8}|D|^2$ is easier to attain (see Sect. 4).

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 $g, h : D^n \to \mathbb{Z}$ 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* $f : D \times D \to D$ *there is a* $\widetilde{\mathcal{O}}(|D|^{\omega \cdot n/2} \cdot \text{polylog}(M))$ *time algorithm for the* f-QUERY *problem.*

Here $\widetilde{\mathcal{O}}(m^{\omega} \cdot \text{polylog}(M))$ is the time needed to multiply two $m \times m$ integer matrices with values in $\{-M, \dots, M\}$ and currently $\omega \in [2, 2.372)$ [2, 21]. 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 $\widetilde{\mathcal{O}}(|D|^n \cdot \text{polylog}(M))$ time. Theorem 1.4 is significantly faster than Theorem 1.1, which can be used to solve f-QUERY in time $\widetilde{\mathcal{O}}\left(\left(\frac{3}{4} \cdot |D|^2\right)^n \cdot \text{polylog}(M)\right)$ when $|D|$ is even, or $\widetilde{\mathcal{O}}\left(\left(\frac{3}{4} \cdot |D|^2 + \frac{1}{4} \cdot |D|\right)^n \cdot \text{polylog}(M)\right)$ when $|D|$ is odd. This holds true even if 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 [17] proposed the fast algorithm to compute DFT. Later, Beth [4] and Clausen [16] 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.

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 [18, 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 Vectors problem is related to the f-QUERY problem. In the Orthogonal Vectors 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 there is no algorithm with a running time of $n^{2-\epsilon} \cdot 2^{o(d)}$ for the Orthogonal Vectors problem for any $\epsilon > 0$, assuming SETH [36]. The currently best algorithm for Orthogonal Vectors runs in time $n^{2-1/\mathcal{O}(\log(d)/\log(n))}$ [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).

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

1.3 Organization

In Sect. 2 we provide the formal definitions of the problems alongside the general statements of our results. In Sect. 3 we give an algorithm for f-Convolution that uses a given cyclic partition. In Sect. 4 we show that for every function $f\colon D \times D \to D$ there exists a cyclic partition of low cost. Finally, in Sect. 5 we give an algorithm for f-Query and prove Theorem 1.4. In Sect. 6 we conclude the paper and discuss future work.

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 $\{1, \ldots, n\}$. 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., $\mathbf{q}{:=}(\mathbf{q}_1, \ldots, \mathbf{q}_k)$.

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

$$\mathbf{u} \oplus_f \mathbf{w}{:=}(f(\mathbf{u}_1, \mathbf{w}_1), \ldots, f(\mathbf{u}_n, \mathbf{w}_n)).$$

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

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

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 $g\colon R^n \to \{-M, \ldots, M\}$ and $h\colon L^n \to \{-M, \ldots, M\}$.
Task: Compute $g \circledast_f h$.

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

Theorem 2.3 *Let $f: L \times R \to T$ 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*

$$c := \begin{cases} \frac{|L|}{2} \cdot \left(|R| + \frac{|T|}{2}\right) & \text{if } |L| \text{ is even} \\ \frac{|L|-1}{2} \cdot \left(|R| + \frac{|T|}{2}\right) + |R| & \text{otherwise.} \end{cases}$$

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 $f: A \times B \to C$ is *k-cyclic* if, up to a relabeling of the sets A, B and C, it is an addition modulo k. Formally, $f: A \times B \to C$ is *k-cyclic* if there are $\sigma_A: A \to \mathbb{Z}_k$, $\sigma_B: B \to \mathbb{Z}_k$, and $\sigma_C: \mathbb{Z}_k \to C$ 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. For example, a constant function $f: A \times B \to \{0\}$ defined by $f(a,b) = 0$ for all $(a,b) \in A \times B$ is 1-cyclic.

The *restriction* of $f: L \times R \to T$ to $A \subseteq L$ and $B \subseteq R$ is the function $g: A \times B \to T$ 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 $f: L \times R \to T$ if the restriction of f to A and B is a k-cyclic function.

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

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

Definition 2.4 (Cyclic Convolution) Let $k \in \mathbb{N}$ and $\mathbf{r} \in \mathbb{N}^k$. Also, let $g, h: Z \to \mathbb{N}$ be two functions where $Z = \mathbb{Z}_{\mathbf{r}_1} \times \cdots \times \mathbb{Z}_{\mathbf{r}_k}$. The Cyclic Convolution of g and h is the function $(g \odot h): Z \to \mathbb{N}$ defined by

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

for every $\mathbf{v} \in Z$.

For any $K \subseteq \mathbb{N}$ we define the K-Cyclic Convolution Problem in which we restrict the entries of the vector $\mathbf{r}$ in Definition 2.4 to be in K.

Definition 2.5 (K-Cyclic Convolution Problem) For any $K \subseteq \mathbb{N}$ the K-Cyclic Convolution Problem is defined as follows.
Input: Integers $k, M \in \mathbb{N}$, a vector $\mathbf{r} \in \mathbb{N}^k$ such that $\mathbf{r}_j \in K$ for every $j \in [k]$ and two functions $g, h \colon Z \to \{-M, \ldots, M\}$ where $Z = \mathbb{Z}_{\mathbf{r}_1} \times \cdots \times \mathbb{Z}_{\mathbf{r}_k}$.
Task: Compute the Cyclic Convolution $g \odot h \colon Z \to \mathbb{Z}$.

Van Rooij [33] claimed that the $\mathbb{N}$-Cyclic Convolution Problem can be solved in $\widetilde{\mathcal{O}}\left(\left(\prod_{i=1}^{k} \mathbf{r}_i\right) \cdot \operatorname{polylog}(M)\right)$ time. However, for his algorithm to work it must be given an appropriate large prime p and several primitives roots of unity in $\mathbb{F}_p$. We are unaware of a method which deterministically finds such a prime and roots while retaining the running time. To overcome this obstacle we present an algorithm for the K-Cyclic Convolution Problem when $K \subseteq \mathbb{N}$ is a fixed finite set. Our solution uses multiple smaller primes and the Chinese Reminder Theorem. We include the details in Appendix A.

Theorem 2.6 (K-Cyclic Convolution) *For any finite set $K \subseteq \mathbb{N}$, there is an $\widetilde{\mathcal{O}}\left(\left(\prod_{i=1}^{k} \mathbf{r}_i\right) \cdot \operatorname{polylog}(M)\right)$ algorithm for the K-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 $\widetilde{\mathcal{O}}((\operatorname{cost}(\mathcal{P})^n + |L|^n + |R|^n + |T|^n) \cdot \operatorname{polylog}(M))$ 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 $\mathcal{P} = \{(A_1, B_1, k_1), \ldots, (A_m, B_m, k_m)\}$. 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 $\boldsymbol{p} \in [m]^n$ for which $\mathbf{u}_i \in A_{\boldsymbol{p}_i}$ and $\mathbf{w}_i \in B_{\boldsymbol{p}_i}$ 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 $\boldsymbol{p} \in \{1, \ldots, m\}^n$ we define

$$L_{\boldsymbol{p}} := A_{\boldsymbol{p}_1} \times \cdots \times A_{\boldsymbol{p}_n}, \quad R_{\boldsymbol{p}} := B_{\boldsymbol{p}_1} \times \cdots \times B_{\boldsymbol{p}_n}, \quad Z_{\boldsymbol{p}} := \mathbb{Z}_{k_{\boldsymbol{p}_1}} \times \cdots \times \mathbb{Z}_{k_{\boldsymbol{p}_n}}$$

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

$$\begin{aligned}
\boldsymbol{\sigma}_{\boldsymbol{p}}^L(\mathbf{v}) &:= \left(\sigma_{A,\boldsymbol{p}_1}(\mathbf{v}_1), \dots, \sigma_{A,\boldsymbol{p}_n}(\mathbf{v}_n)\right) && \forall \mathbf{v} \in L_{\boldsymbol{p}},\\
\boldsymbol{\sigma}_{\boldsymbol{p}}^R(\mathbf{v}) &:= \left(\sigma_{B,\boldsymbol{p}_1}(\mathbf{v}_1), \dots, \sigma_{B,\boldsymbol{p}_n}(\mathbf{v}_n)\right) && \forall \mathbf{v} \in R_{\boldsymbol{p}},\\
\boldsymbol{\sigma}_{\boldsymbol{p}}^T(\mathbf{q}) &:= \left(\sigma_{C,\boldsymbol{p}_1}(\mathbf{q}_1), \dots, \sigma_{C,\boldsymbol{p}_n}(\mathbf{q}_n)\right) && \forall \mathbf{q} \in Z_{\boldsymbol{p}}.
\end{aligned}$$

Our algorithm heavily depends on constructing the following projections.

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

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

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

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

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:*

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

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

We give the proof of Lemma 3.4 in Sect. 3.1. It should be noted that the naive computation of the projection functions of g and h with respect to all types $\boldsymbol{p}$ 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, $\{g_{\boldsymbol{p}} \mid \boldsymbol{p} \in [m]^n\}$, in time $((\mathrm{cost}(\mathcal{P})^n + |L|^n))$.*

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

The proof of Lemma 3.5 in given in Sect. 3.1.

Our algorithm for f-Convolution (see Algorithm 1 for the pseudocode) is a direct implication of Lemmas 3.4 and 3.5. First, the algorithm computes the projections of g

Algorithm 1: Cyclic Partition Algorithm for the f-CONVOLUTION problem

```
Setting: Finite sets $L$, $R$ and $T$, $f: L \times R \to T$ and a cyclic partition $\mathcal{P}$ of $f$, of size $m$.
Input: $g: L^n \to \{-M, \dots, M\}$, $h: R^n \to \{-M, \dots, M\}$
1 Construct the projections of $g$ and $h$ w.r.t $\boldsymbol{p}$, for all $\boldsymbol{p} \in [m]^n$        ▷ Lemma 3.5
2 For every $\boldsymbol{p} \in [m]^n$ compute $\mathsf{c}_{\boldsymbol{p}} = g_{\boldsymbol{p}} \odot h_{\boldsymbol{p}}$        ▷ Cyclic convolutions (Definition 2.4)
3 Define $\mathsf{r}: T^n \to \mathbb{Z}$ by
      $\mathsf{r}(\mathbf{v}) = \sum_{\boldsymbol{p} \in [m]^n} \sum_{\mathbf{q} \in Z_{\boldsymbol{p}} \text{ s.t. } \boldsymbol{\sigma}_{\boldsymbol{p}}^T(\mathbf{q}) = \mathbf{v}} \mathsf{c}_{\boldsymbol{p}}(\mathbf{q})$    for all $\mathbf{v} \in T^n$.
4 return $\mathsf{r}$
```

and h with respect to every type $\boldsymbol{p}$. Subsequently, the cyclic convolution of $g_{\boldsymbol{p}}$ and $h_{\boldsymbol{p}}$ is computed efficiently as described in Theorem 2.6. Finally, the values of $(g \circledast_f h)$ are reconstructed by the formula in Lemma 3.4.

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

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

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 $\widetilde{\mathcal{O}}((\text{cost}(\mathcal{P})^n + |L|^n + |R|^n) \cdot \text{polylog}(M))$. Define $K = \{k \mid (A, B, k) \in \mathcal{P}\} = \{k_1, \dots, k_m\}$ be different costs of cyclic minors in $\mathcal{P}$. By Theorom 2.6, for any type $\boldsymbol{p} \in [m]^n$ the computation of $g_{\boldsymbol{p}} \odot h_{\boldsymbol{p}}$ in Line 2 is an instance of K-CYCLIC CONVOLUTION PROBLEM which can be solved in time $\widetilde{\mathcal{O}}((\prod_{i=1}^n k_{\boldsymbol{p}_i}) \cdot \text{polylog}(M))$. Thus the overall running time of Line 2 is $\widetilde{\mathcal{O}}\left((\sum_{\boldsymbol{p} \in [m]^n} \prod_{i=1}^n k_{\boldsymbol{p}_i}) \cdot \text{polylog}(M)\right)$.

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 $\mathsf{c}_{\boldsymbol{p}}(\mathbf{q})$ to $\mathsf{r}(\boldsymbol{\sigma}_{\boldsymbol{p}}^T(\mathbf{q}))$ for every $\boldsymbol{p} \in [m]^n$ and $\mathbf{q} \in Z_{\boldsymbol{p}}$. The required running time is thus $\widetilde{\mathcal{O}}(|T|^n \cdot \text{polylog}(M))$ for the initialization and $\widetilde{\mathcal{O}}\left((\sum_{\boldsymbol{p} \in [m]^n} |Z_{\boldsymbol{p}}|) \cdot \text{polylog}(M)\right) = \left((\sum_{\boldsymbol{p} \in [m]^n} \prod_{i=1}^n k_{\boldsymbol{p}_i})\right)$ for the addition operations. Thus, the overall running time of Line 3 is

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

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

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

This concludes the proof of Lemma 3.1. □

3.1 Properties of Projections

In this section we provide the proofs for Lemmas 3.4 and 3.5. The proof of Lemma 3.4 uses the following definitions of coordinate-wise addition with respect to a type $\boldsymbol{p}$.

Definition 3.7 (Coordinate-wise addition modulo for type) For any $\boldsymbol{p} \in [m]^n$ we define a coordinate-wise addition modulo as

$$\mathbf{q} +_{\boldsymbol{p}} \mathbf{r} := \big((\mathbf{q}_1 + \mathbf{r}_1 \mod k_{\boldsymbol{p}_1}), \ldots, (\mathbf{q}_n + \mathbf{r}_n \mod k_{\boldsymbol{p}_n})\big) \quad \text{for every } \mathbf{q}, \mathbf{r} \in Z_{\boldsymbol{p}}.$$

Proof of Lemma 3.4 By Definition 2.1 it holds that:

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

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

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

By the properties of the relabeling functions, we get

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

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

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

$$= \sum_{\mathbf{r},\mathbf{s}\in Z_p} [\![\mathbf{q} = \mathbf{r} +_p \mathbf{s}]\!] \left(\sum_{\mathbf{u}\in L_p} [\![\mathbf{r} = \boldsymbol{\sigma}^L_{\boldsymbol{p}}(\mathbf{u})]\!] \cdot g(\mathbf{u}) \right) \cdot \left(\sum_{\mathbf{w}\in R_p} [\![\mathbf{s} = \boldsymbol{\sigma}^R_{\boldsymbol{p}}(\mathbf{w})]\!] \cdot h(\mathbf{w}) \right)$$

$$= \sum_{\mathbf{r},\mathbf{s}\in Z_p} [\![\mathbf{q} = \mathbf{r} +_p \mathbf{s}]\!] \cdot g_{\boldsymbol{p}}(\mathbf{r}) \cdot h_{\boldsymbol{p}}(\mathbf{s})$$

$$= (g_{\boldsymbol{p}} \odot h_{\boldsymbol{p}})(\mathbf{q}). \tag{3.3}$$

By plugging (3.3) into (3.2) we get

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

$$= \sum_{\boldsymbol{p}\in[m]^n} \sum_{\mathbf{q}\in Z_p} [\![\boldsymbol{\sigma}^T_{\boldsymbol{p}}(\mathbf{q}) = \mathbf{v}]\!] \cdot \left(g_{\boldsymbol{p}} \odot h_{\boldsymbol{p}}\right)(\mathbf{q}),$$

as required. □

Proof of Lemma 3.5 The idea is to use a dynamic programming algorithm loosely inspired by Yates's algorithm [40].

Define $X^{(\ell)} = \left\{(\boldsymbol{p}, \mathbf{q}) \mid \boldsymbol{p} \in [m]^\ell,\ \mathbf{q} \in \mathbb{Z}_{\boldsymbol{p}_1} \times \cdots \times \mathbb{Z}_{\boldsymbol{p}_\ell}\right\}$ for every $\ell \in \{0, \ldots, n\}$. 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 $\ell \in \{0, \ldots n\}$ by:

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

The tables $\mathsf{DP}^{(0)}, \mathsf{DP}^{(1)}, \ldots, \mathsf{DP}^{(n)}$ 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 $g_{\boldsymbol{p}}(\mathbf{q}) = \mathsf{DP}^{(n)}[(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n), (\mathbf{q}_1, \ldots, \mathbf{q}_n)][\varepsilon]$ for every $\boldsymbol{p}$ and $\mathbf{q}$, which means that computing $\mathsf{DP}^{(n)}$ is equivalent to computing the projection functions $g_{\boldsymbol{p}}$ of g for every type $\boldsymbol{p}$.[5]

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 $\mathsf{DP}^{(\ell)}$:

$$\mathsf{DP}^{(\ell)}[(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_\ell), (\mathbf{q}_1, \ldots, \mathbf{q}_\ell)][\mathbf{t}_{\ell+1}, \ldots, \mathbf{t}_n]$$
$$= \sum_{\mathbf{t}_\ell\in A_{\boldsymbol{p}_\ell}} [\![\sigma_{\boldsymbol{p}_\ell}(\mathbf{t}_\ell) = \mathbf{q}_\ell]\!] \cdot \mathsf{DP}^{(\ell-1)}[(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{\ell-1}), (\mathbf{q}_1, \ldots, \mathbf{q}_{\ell-1})][\mathbf{t}_\ell, \ldots, \mathbf{t}_n]. \tag{3.4}$$

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

A dynamic programming algorithm which computes $\mathsf{DP}^{(n)}$ can be easily derived from (3.4) and the formula for $\mathsf{DP}^{(0)}$. The total number of states in the dynamic programming table $\mathsf{DP}^{(\ell)}$ is

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

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 $g_{\boldsymbol{p}}$ for every $\boldsymbol{p}$ in $\widetilde{\mathcal{O}}((\mathrm{cost}(\mathcal{P})^n + |L|^n)\cdot \mathrm{polylog}(M))$ time. □

4 The Existence of a 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* $\mathrm{cost}(\mathcal{P}) \le \frac{|L|}{2}\cdot(|R|+\frac{|T|}{2})$ *when $|L|$ is even, and* $\mathrm{cost}(\mathcal{P}) \le |R| + \frac{|L|-1}{2}\cdot(|R|+\frac{|T|}{2})$ *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 $\frac{|D|}{2}\cdot\frac{7}{4}|D| = \frac{7}{8}|D|^2$.

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 $|\{f(d_a^{(i)}, e_b^{(j)}) \mid a, b \in \{1,2\}\}| \le 3$ for each $j \in [r]$, i.e. for each j at least one of the four values $f(d_1^{(i)}, e_1^{(j)})$, $f(d_1^{(i)}, e_2^{(j)})$, $f(d_2^{(i)}, e_1^{(j)})$, $f(d_2^{(i)}, e_2^{(j)})$ 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 $\{f(d_a^{(i)}, d) \mid d \in \overline{D}, a \in \{1,2\}\}$ 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: $|L| = 2$

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

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

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.

Definition 4.3 (Graph Representation) Let $f: L \times R \to T$ be such that $|L| = 2$ with $L = \{\ell_0, \ell_1\}$.

We say a function $\lambda_f: R \to T \times T$ with $\lambda_f: r \mapsto (f(\ell_0, r), f(\ell_1, r))$ is the *edge mapping* of f. We say that a directed graph G_f (which might have self-loops) with vertex set $V(G_f):=T$ and edge set $E(G_f):=\{\lambda_f(r) \mid r \in R\}$ is the *representation graph* of f.

We say that the representation graph G_f is *nice* if G_f is a directed cycle or a directed path (potentially with a single edge).

As a next step we define the restriction of a function based on a subgraph of the corresponding representation graph.

Definition 4.4 (Restriction of f) Let $f: L \times R \to T$ be a function such that $|L| = 2$ and let G_f be the representation graph of f. Let $E' \subseteq E(G_f)$ be a given subset of edges inducing the subgraph G' of G_f.

Based on E' (and thus, G') we define a new function f' in the following and say that f' is the *function represented by* G' or E'.

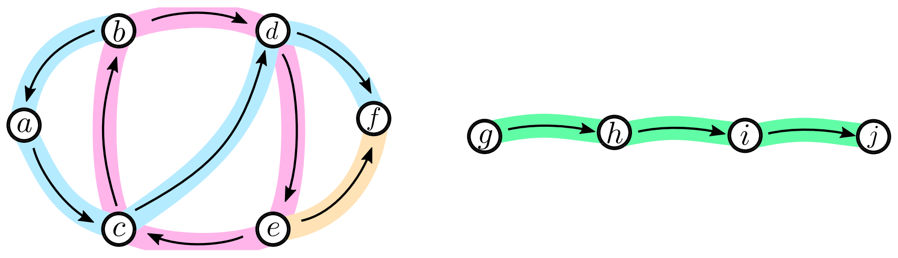
With $T':=V(G')$ and $R':=\{r \in R \mid \lambda_f(r) \in E'\}$, we define $f': L \times R' \to T'$ as the *restriction* of f such that the representation graph of f' is G'. Formally, we set $f'(\ell, r):=f(\ell, r)$ for all $\ell \in L$ and $r \in R'$.

A decomposition of a directed graph G is a family $\mathcal{F}$ of edge-disjoint subgraphs of G, such that each edge belongs to exactly one subgraph in $\mathcal{F}$. The following observation follows directly from the previous definition.

Observation 4.5 *Let $\{G_1, \ldots, G_k\}$ be a decomposition of the graph G_f into k subgraphs, let f_i be the function represented by G_i, and let $\mathcal{P}_i$ be a cyclic partition of f_i.*

Then $\mathcal{P} = \bigcup_{i \in [k]} \mathcal{P}_i$ is a cyclic partition of f with cost $\operatorname{cost}(\mathcal{P}) = \sum_{i \in [k]} \operatorname{cost}(\mathcal{P}_i)$.

Cyclic Partitions Using Nice Representation Graphs As a next step, we show that functions admit cyclic partitions if the representation graph is nice. We extend these results to functions with arbitrary representation graphs by decomposing these graphs into nice subgraphs. Finally, we combine these results to obtain a cyclic partition for the original function f (Fig. 2).



	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	b	c	c	d	d	e	e	g	h	i
ℓ_1	c	a	d	b	d	e	f	c	f	h	i	j

Fig. 2 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 an example decomposition of the edges into a cycle with 4 vertices (highlighted red) and three paths with 5, 2 and 4 vertices (highlighted blue, yellow and green respectively). The cost of this cyclic partition is $4 + 5 + 2 + 4 = 15$ (Color figure online)

Lemma 4.6 *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 $|T| = |V(G_f)|$.*

Proof By definition, a nice graph is either a cycle or a path. 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 $t_1, \dots, t_{|T|}$ be the elements in T with $t_{|T|} = t_0$ such that, for all $j \in \mathbb{Z}_{|T|}$, there is some $r_j \in R$ with $\lambda_f(r_j) = (t_j, t_{j+1})$.[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 $\sigma_T(j) = t_j$, for all $j \in \mathbb{Z}_{|T|}$. For the elements in R we define $\sigma_R : R \to \mathbb{Z}_{|R|}$ with $\sigma_R(r) = j$ whenever $\lambda_f(r) = (t_j, t_{j+1})$ for some j. It is easy to check that f can be seen as addition modulo $|T|$. Indeed, let $i \in \{0, 1\}$ and $r \in R$ with $\lambda_f(r) = (t_j, t_{j+1})$. Then we get

$$\begin{aligned}&\sigma_T(\sigma_L(\ell_i) + \sigma_R(r) \bmod |T|) = \sigma_T(i + j \bmod |T|)\\ &= t_{(i+j \bmod |T|)} = f(\ell_i, r_j) = f(\ell_i, r).\end{aligned}$$

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 $|T|$. The proof is essentially identical to the cyclic case and we include it for completeness. Let $\sigma_L : L \to \mathbb{Z}_2$ with $\sigma_L(\ell_i) = i$. Let $t_0, \dots, t_{|T|-1}$ be the elements of T such that, for all $j \in \mathbb{Z}_{|T|-1}$, there exist $r_j \in R$ with $\lambda_f(r_j) = (t_j, t_{j+1})$. Since G_f is a path, such r_j's must exist. We let $\sigma_T(j) = t_j$ for every $j \in \mathbb{Z}_{|T|}$. We define

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

$\sigma_R\colon R \to \mathbb{Z}_{|R|}$ with $\sigma_R(r) = j$ whenever $\lambda_f(r) = (t_j, t_{j+1})$ for some j. Now, we verify that f can be interpreted as addition modulo $|T|$. Consider $i \in \{0, 1\}$ and $r \in R$ with $\lambda_f(r) = (t_j, t_{j+1})$ for some $j \in \mathbb{Z}_{|T|-1}$. Observe that $j < |T| - 1$, hence $t_{j+1 \bmod |T|} = t_{j+1}$. Therefore, we get

$$\sigma_T(\sigma_L(\ell_i) + \sigma_R(r) \bmod |T|) = \sigma_T(i + j \bmod |T|)$$
$$= t_{(i+j \bmod |T|)} = f(\ell_i, r_j) = f(\ell_i, r).$$

Hence, f is $|T|$-cyclic with cyclic partition $\{(L, R, |T|)\}$.

□

In the next step, we decompose arbitrary graphs into nice subgraphs. To present our decomposition we need to introduce the following notation related to the degree of vertices.

Definition 4.7 (Sources, Sinks and Middle Vertices) Let $G = (V, E)$ be a directed graph. We denote by $\mathrm{indeg}(v)$ the *in-degree* of v, i.e., the number of edges terminating at v, and by $\mathrm{outdeg}(v)$ the *out-degree* of v, i.e., the number of edges starting at v.

We partition V into the three sets $V_{\mathsf{src}}(G)$, $V_{\mathsf{mid}}(G)$, and $V_{\mathsf{snk}}(G)$ defined as follows:

- Set $V_{\mathsf{src}}(G)$ contains all *source* vertices of G, that is, vertices with no incoming edges (i.e., $\mathrm{indeg}(v) = 0$). This includes all isolated vertices.
- Set $V_{\mathsf{mid}}(G)$ contains all *middle* vertices of G, that is vertices with incoming and outgoing edges (i.e., $\mathrm{indeg}(v), \mathrm{outdeg}(v) \geq 1$).
- Set $V_{\mathsf{snk}}(G)$ contains the (remaining) *sink* vertices of G, that is, vertices with incoming but no outgoing edges (i.e., $\mathrm{indeg}(v) \geq 1$ and $\mathrm{outdeg}(v) = 0$).

We additionally introduce the notion of *deficiency* which we use in the following proofs.

Definition 4.8 (Deficiency) Let $G = (V, E)$ be a directed graph. For all $v \in V$, we denote by $\mathrm{defi}(v) := \max\{\mathrm{outdeg}(v) - \mathrm{indeg}(v), 0\}$ the *deficiency* of v.

We define $\mathrm{Defi}(G) := \sum_{v \in V} \mathrm{defi}(v)$ as the *total deficiency* of the graph G.

We omit the graph G from the notation if it is clear from the context.

We use the deficiency to decompose the acyclic graphs into paths.

Lemma 4.9 *Every directed graph G can be decomposed into* $\mathrm{Defi}(G)$ *paths and an arbitrary number of cycles.*

Proof We construct the decomposition $\mathcal{F}$ of G as follows. In the first phase, we exhaustively find a directed cycle C in G. We add cycle C to the decomposition $\mathcal{F}$ and remove the edges of C from G. We continue the above procedure until graph G becomes acyclic. Next, in the second phase we exhaustively find a directed maximum length path P (note that P may be a single edge). We add P to the decomposition $\mathcal{F}$ and remove the edges of P from G. We repeat the second phase until the graph G becomes edgeless.

This concludes the construction of decomposition $\mathcal{F}$. For correctness observe that the above procedure always terminates because in each step we decrease the number of edges of G. Moreover, at the end of the above procedure $\mathcal{F}$ is a decomposition of G that consists only of paths and cycles.

We are left to show that the number of paths in $\mathcal{F}$ is exactly $\text{Defi}(G)$. Note that deleting a cycle in G does not change the value of $\text{Defi}(G)$, hence the first phase of the procedure does not influence $\text{Defi}(G)$ and we can assume that G is acyclic.

Next, we show that deleting a maximum length path from an acyclic graph decrements its deficiency by exactly 1. This then conclude the proof, because in the second phase of the procedure the deficiency of G decreases from $\text{Defi}(G)$ down to 0, which means that exactly $\text{Defi}(G)$ maximum length paths were added to $\mathcal{F}$.

Let P be a maximum length, directed path in the acyclic graph G. Let $s, t \in V(G)$ be the starting and terminating vertices of path P. Path P must start at a vertex with a positive deficiency, because otherwise P could have been extended at the start which would contradict the fact that P is of maximum length. Similarly, since P is of maximum length it must terminate in a sink vertex. Hence $\text{defi}(s) > 0$ and $\text{defi}(t) = 0$. Moreover, every vertex $v \in P \setminus \{s, t\}$ has exactly one incoming and one outgoing edge in P. Therefore, in the graph $G \setminus P$ the contribution to the total deficiency decreased only in the vertex s and only by 1. This means that $\text{Defi}(G) = \text{Defi}(G \setminus P) + 1$ which concludes the proof. □

Now we combine Lemmas 4.6 and 4.9 to show Lemma 4.10.

Lemma 4.10 *Let $f : L \times R \to T$ be a function with $|L| = 2$ and let G_f be the representation graph of f. Then, there exists a cyclic partition $\mathcal{P}$ for f with* $\text{cost}(\mathcal{P}) \leq |E(G_f)| + \text{Defi}(G_f)$.

Proof First, use Lemma 4.9 to decompose the graph into cycles and $\text{Defi}(G_f)$ paths. Then, for each of these paths and cycles, use Lemma 4.6 to obtain the cyclic minor. By Observation 4.5, these minors form a cyclic partition for the function represented by G_f. Let $\mathcal{P}$ be the resulting cyclic partition.

It remains to analyze the cost of the cyclic partition $\mathcal{P}$. By construction, each cyclic minor in $\mathcal{P}$ corresponds to a path or a cycle (possibly of length 1). By Lemma 4.6 the cost of a path or a cycle is the number of vertices it contains. Thus, for a path, the cost is equal to the number of edges plus one, and for a cycle the cost is equal to the number of edges. Hence, the cost of $\mathcal{P}$ is bounded by the number of edges of G_f plus the number of paths in the decomposition. The latter is precisely $\text{Defi}(G_f)$ by Lemma 4.9. □

Cyclic Partitions Using a Direct Construction In the following, we use a different method to construct a cyclic partition of the function f. Instead of decomposing the graph into nice subgraphs, we directly construct a partition and bound its cost.

Lemma 4.11 *Let $f : L \times R \to T$ be a function with $|L| = 2$ and let G_f be the representation graph of f. Then, there is a cyclic partition $\mathcal{P}$ of f with* $\text{cost}(\mathcal{P}) \leq |V(G_f)| + |V_{\text{mid}}(G_f)|$.

Proof For each $\ell \in L$, we use a single cyclic minor. Let $L = \{\ell_0, \ell_1\}$. For $i \in \{0, 1\}$ define $T_i = \{f(\ell_i, r) \mid r \in R\}$ and $k_i = |T_i|$. Then, $\mathcal{P}:=\{(\ell_i, R, k_i) \mid i \in \{0, 1\}\}$ is the cyclic partition of f.

To see that $(\{\ell_i\}, R, k_i)$ is a cyclic minor for $i \in \{0, 1\}$, assume w.l.o.g. that $T_i = \{0, 1, \dots, k_i - 1\}$ and define $\sigma_L(\ell_i) = 0$, $\sigma_R(r) = f(\ell_i, r)$, and $\sigma_T(t) = t$. Thus, $\mathcal{P}$ is a cyclic partition of f of cost $k_0 + k_1 = |T_0| + |T_1|$.

Observe that $|T_0| = |V_{\mathsf{src}}(G_f)| + |V_{\mathsf{mid}}(G_f)|$ as every $t \in T_0$ has an outgoing edge in G_f, and $|T_1| = |V_{\mathsf{snk}}(G_f)| + |V_{\mathsf{mid}}(G_f)|$ as every $t \in T_1$ has an incoming edge in G_f. Hence,

$$\begin{aligned} \mathrm{cost}(\mathcal{P}) &= |T_0| + |T_1| \\ &= |V_{\mathsf{src}}(G_f)| + |V_{\mathsf{mid}}(G_f)| + |V_{\mathsf{snk}}(G_f)| + |V_{\mathsf{mid}}(G_f)| \\ &= |V(G_f)| + |V_{\mathsf{mid}}(G_f)| \end{aligned}$$

which finishes the proof. □

Bounding the Cost of Cyclic Partitions Now, we combine the results from Lemmas 4.10 and 4.11,. We first show how the number of edges relates to the total deficiency of a graph and the number of middle vertices.

Lemma 4.12 *For every directed graph G it holds that* $|V_{\mathsf{mid}}(G)| + \mathrm{Defi}(G) \le |E(G)|$.

Proof Let m be the number of edges of G and let $e_1, \dots, e_m \in E(G)$ be some arbitrarily fixed order of its edges. For every $i \in \{0, \dots, m\}$ let G_i be the graph with vertices $V(G)$ and edges $E(G_i) = \{e_1, \dots, e_i\}$. Hence G_0 is an independent set of $V(G)$ and $G_m = G$.

For every $i \in \{0, \dots, m\}$ let $\mathrm{LHS}(G_i):=|V_{\mathsf{mid}}(G_i)| + \mathrm{Defi}(G_i)$ be the quantity we need to bound. We show that

$$\mathrm{LHS}(G_i) - \mathrm{LHS}(G_{i-1}) \le 1 \text{ for every } i \in [m] \tag{4.1}$$

which then concludes the proof because

$$|V_{\mathsf{mid}}(G)| + \mathrm{Defi}(G) = \mathrm{LHS}(G_m) = \sum_{i=1}^{m} (\mathrm{LHS}(G_i) - \mathrm{LHS}(G_{i-1})) \le m = |E(G)|.$$

From now, we focus on the proof of Eq. 4.1. For every $v \in V(G)$ and $i \in \{0, \dots, m\}$, let $\mathrm{defi}_i(v)$ be the deficiency of vertex v in graph G_i. Next, for every $v \in V(G)$ and $i \in [m]$, we define

$$\Delta_i(v):=\mathrm{defi}_i(v) - \mathrm{defi}_{i-1}(v) + [\![v \in V_{\mathsf{mid}}(G_i) \setminus V_{\mathsf{mid}}(G_{i-1})]\!]$$

Consider a step $i \in [m]$. Let $e_i = (s, t)$ be an ith edge that starts at a vertex s and terminates at a vertex t. It holds that

$$|V_{\mathsf{mid}}(G_i)| + \mathrm{Defi}(G_i) = |V_{\mathsf{mid}}(G_{i-1})| + \mathrm{Defi}(G_{i-1}) + \Delta_i(s) + \Delta_i(t).$$

Therefore $\mathrm{LHS}(G_i) - \mathrm{LHS}(G_{i-1}) = \Delta_i(s) + \Delta_i(t)$ and to establish Eq. 4.1 it is enough to show that $\Delta_i(s) \le 1$ and $\Delta_i(t) \le 0$.

Claim 4.13 It holds that $\Delta_i(s) \le 1$.

Proof We consider two cases depending on whether u became a middle vertex. If it happened that $s \in V_{\mathsf{mid}}(G_i) \backslash V_{\mathsf{mid}}(G_{i-1})$, then $s \in V_{\mathsf{snk}}(G_{i-1})$ which means that s has more incoming than outgoing edges in G_{i-1}. Hence $\mathrm{defi}_{i-1}(s) = \mathrm{defi}_i(s) = 0$ and we conclude that $\Delta_i(s) = 1$.

Otherwise $s \notin V_{\mathsf{mid}}(G_i) \backslash V_{\mathsf{mid}}(G_{i-1})$. Because the edge e_i starts at s, the deficiency of s can increase by at most 1. Hence, by $(\mathrm{defi}_i(s) - \mathrm{defi}_{i-1}(s)) \le 1$ we conclude that $\Delta_i(s) \le 1$. □

Finally, we consider the end vertex t of the edge e_i.

Claim 4.14 It holds that $\Delta_i(t) \le 0$.

Proof We again distinguish two cases depending on whether t became a middle vertex. If $t \in V_{\mathsf{mid}}(G_i) \setminus V_{\mathsf{mid}}(G_{i-1})$, then $t \in V_{\mathsf{src}}(G_{i-1})$ and moreover, t has no incoming edges and the positive number of outgoing edges in G_{i-1}. Therefore $\mathrm{defi}_i(t) = \mathrm{defi}_{i-1}(t) - 1$ which means that $\Delta_i(t) \le 0$.

It remains to analyse the case when $t \notin V_{\mathsf{mid}}(G_i) \backslash V_{\mathsf{mid}}(G_{i-1})$. Since the edge e_i ends at t, the deficiency of t cannot increase and $\mathrm{defi}_i(v) \le \mathrm{defi}_{i-1}(v)$. This means that $\Delta_i(t) \le 0$. □

By Claims 4.13 and 4.14, it follows that $\Delta_i(s) + \Delta_i(t) \le 1$. This establishes Eq. 4.1 and concludes the proof. □

Now we are ready to combine Lemmas 4.10 and 4.11, and prove Lemma 4.2.

Proof of Lemma 4.2 As before, we denote by G_f the representation graph of f. Let V and E be the set of vertices and edges of graph G_f.

Let $\mathcal{P}_1$ be the cyclic partition of f from Lemma 4.10 with cost at most $|E| + \mathrm{Defi}(G_f)$ and let $\mathcal{P}_2$ be the cyclic partition of f from Lemma 4.11 with cost at most $|V| + |V_{\mathsf{mid}}(G_f)|$.

We define $\mathcal{P}$ as the minimum cost partition among $\mathcal{P}_1$ and $\mathcal{P}_2$. This implies that

$$\begin{aligned} \mathrm{cost}(\mathcal{P}) &\le \min\{\mathrm{cost}(\mathcal{P}_1), \mathrm{cost}(\mathcal{P}_2)\} \le \frac{\mathrm{cost}(\mathcal{P}_1) + \mathrm{cost}(\mathcal{P}_2)}{2} \\ &\le \frac{|E| + |V| + |V_{\mathsf{mid}}(G_f)| + \mathrm{Defi}(G_f)}{2}. \end{aligned}$$

Next, we use the inequality $|V_{\mathsf{mid}}(G_f)| + \mathrm{Defi}(G_f) \le |E|$ from Lemma 4.12, and get

$$\mathrm{cost}(\mathcal{P}) \le |E| + \frac{|V|}{2}.$$

Since $|E| \le |R|$ and $|V| = |T|$ this concludes the proof. □

4.2 General Case: Proof of Lemma 4.1

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 $L_1, \dots, L_\lambda$ 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

$$\operatorname{cost}(\mathcal{P}) \le \sum_{i=1}^{\lambda} \operatorname{cost}(\mathcal{P}_i) \le \lambda \cdot \left(|R| + \frac{|T|}{2}\right).$$

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

$$\operatorname{cost}(\mathcal{P}) \le \operatorname{cost}(\mathcal{P}_0) + \operatorname{cost}(\mathcal{P}') \le |R| + \left\lfloor \frac{|L|}{2} \right\rfloor \left(|R| + \frac{|T|}{2}\right).$$

□

Remark 4.15 If $|L|$ and $|R|$ are both even, one can easily achieve a cost of

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

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 $r \in R$.

4.3 Tight Example: Lower Bound on Lemma 4.2

To complement the previous results, we show that Lemma 4.2 is tight. That is, there is a function $f \colon L \times R \to T$ with $|L| = 2$ such that no cyclic partition $\mathcal{P}$ of f has smaller cost, i.e., $\operatorname{cost}(\mathcal{P}) < |R| + |T|/2$. In particular, this demonstrates that to improve the constant c:=3/4 in Theorem 1.1 new ideas are needed.

Lemma 4.16 *There exist sets L, R, and T with $|L| = 2$ and a function $f \colon L \times R \to T$ such that, every cyclic partition $\mathcal{P}$ of f has* $\operatorname{cost}(\mathcal{P}) \ge |R| + |T|/2$.

Proof Define $L = \{\ell_0, \ell_1\}$, $R = \{r_1, r_2, r_3, r_4\}$, and $T = \{a, b, c, d\}$. Let f be the function as defined in Fig. 3. Note that we need to show that every cyclic partition of f has cost at least 6.

	r_1	r_2	r_3	r_4
ℓ_0	a	b	c	a
ℓ_1	b	c	d	d

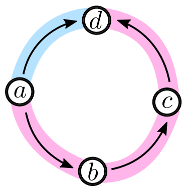


Fig. 3 The definition of the function f used by Lemma 4.16, which shows that the bound from Lemma 4.2 is tight. The representation graph of f is depicted on the right. We highlight the cyclic partition returned by Lemma 4.16. The red path contains 4 vertices and the blue path contains 2 vertices. Hence, the cost of that cyclic partition is 6. Lemma 4.16 shows that this is the best possible (Color figure online)

Let $\mathcal{P}$ be a cyclic partition of f. We first claim that the cyclic partition $\mathcal{P}$ of f contains a single cyclic minor, i.e., $\mathcal{P} = \{(L, R, k)\}$ for some integer k. For contradictions sake, we analyse every other remaining structure of $\mathcal{P}$ and argue that in each case $\text{cost}(\mathcal{P}) \geq 6 = |R| + |T|/2$.

- Every cyclic minor in $\mathcal{P}$ is of the form $(\{\ell_i\}, B, k)$ (i.e., uses only values from a single row). Then, $\text{cost}(\mathcal{P}) \geq 6$ as each row has 3 distinct values.
- There is a cyclic minor $(\{\ell_0, \ell_1\}, \{r_j\}, k)$ in $\mathcal{P}$. Since each column contains two distinct elements, It must hold that $k \geq 2$. Furthermore, the cyclic minors which cover the remainder of the graph must have a total cost of 4 (or more) as all values in T appear in the remainder of the graph. Hence $\text{cost}(\mathcal{P}) \geq 6$.
- There is a cyclic minor $(\{\ell_0, \ell_1\}, \{r_j, r_{j'}\}, k)$ in $\mathcal{P}$. Since each pair of two columns contains (at least) three values, it must hold that $k \geq 3$. There are at least 3 distinct values in the remainder of the graph, hence, the cost of the remaining minors in $\mathcal{P}$ is at least 3. Thus $\text{cost}(\mathcal{P}) \geq 6$.
- There is a cyclic minor $(\{\ell_0, \ell_1\}, R \setminus \{r_j\}, k)$ in $\mathcal{P}$. It holds that $k \geq 4$ as every three columns include all values in T. In each case, there are two different values in the remaining column. Hence, the cost of the remaining minors is at least 2. Therefore $\text{cost}(\mathcal{P}) \geq 6$.

With this, we know that $\mathcal{P}$ contains only the single cyclic minor (L, R, k). Let σ_L, σ_R and σ_T be the relabelling functions of (L, R, k). From the definition of the relabeling functions, we get that $F{:=}\{\sigma_L(\ell_i) + \sigma_R(r_j) \mod k \mid i \in \{0, 1\} \text{ and } j \in \{1, 2, 3\}\}$ contains at least four elements.

We claim that $(\sigma_L(\ell_0) + \sigma_R(r_4) \mod k) \notin F$. For the sake of contradiction assume otherwise. Then, by the definition of σ_T, it must hold that $\sigma_L(\ell_0) + \sigma_R(r_1) \equiv_k \sigma_L(\ell_0) + \sigma_R(r_4)$. As this implies $\sigma_R(r_1) = \sigma_R(r_4)$, we get

$$\begin{aligned} b = f(\ell_1, r_1) &= \sigma_T(\sigma_L(\ell_1) + \sigma_R(r_1) \mod k) \\ &= \sigma_T(\sigma_L(\ell_1) + \sigma_R(r_4) \mod k) = f(\ell_1, r_4) = d, \end{aligned}$$

which is a contradiction.

Similarly, we get that $(\sigma_L(\ell_1) + \sigma_R(r_4) \mod k) \notin F$. Again assuming otherwise, we have that $\sigma_R(r_3) = \sigma_R(r_4)$ which then implies

$$c = f(\ell_0, r_3) = \sigma_T(\sigma_L(\ell_0) + \sigma_R(r_3) \mod k)$$

$$= \sigma_T(\sigma_L(\ell_0) + \sigma_R(r_4) \mod k) = f(\ell_0, r_4) = a,$$

which is a contradiction.

Since, $F \cup \{\sigma_L(\ell_0) + \sigma_R(r_4) \mod k, \sigma_L(\ell_1) + \sigma_R(r_4) \mod k\} \subseteq \mathbb{Z}_k$, contains at least six distinct elements, we get $k \geq 6$ and therefore, $\text{cost}(\mathcal{P}) \geq 6$. $\square$

5 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 $f: D \times D \to D$. We assume D and f are fixed throughout this section. Let $g, h: D^n \to \{-M, \dots, M\}$ 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 $\mathbf{v}^{(\text{high})}, \mathbf{v}^{(\text{low})} \in D^{n/2}$ be the unique vectors such that $\mathbf{v}^{(\text{high})} \| \mathbf{v}^{(\text{low})} = \mathbf{v}$. Indeed, to achieve this assumption let n be odd, fix an arbitrary $d \in D$, and define $\widetilde{g}, \widetilde{h}: D^{n+1} \to \{-M, \dots, M\}$ 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 present the algorithm and show correctness.

Intuition We define a directed multigraph G where the vertices are partitioned into four layers $\mathrm{L}^{(\text{high})}, \mathrm{L}^{(\text{low})}, \mathrm{R}^{(\text{low})}$, and $\mathrm{R}^{(\text{high})}$. 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 $\mathrm{L}^{(\text{high})}$ to $\mathbf{x} \in D^{n/2}$ in $\mathrm{L}^{(\text{low})}$.
– One edge from $\mathbf{x} \in D^{n/2}$ in $\mathrm{L}^{(\text{low})}$ to $\mathbf{y} \in D^{n/2}$ in $\mathrm{R}^{(\text{low})}$ if and only if $\mathbf{x} \oplus_f \mathbf{y} = v^{(\text{low})}$.
– $h(\mathbf{z} \| \mathbf{y})$ parallel edges from $\mathbf{y} \in D^{n/2}$ in $\mathrm{R}^{(\text{low})}$ to $\mathbf{z} \in D^{n/2}$ in $\mathrm{R}^{(\text{high})}$.
– One edge from $\mathbf{z} \in D^{n/2}$ in $\mathrm{R}^{(\text{high})}$ to $\mathbf{w} \in D^{n/2}$ in $\mathrm{L}^{(\text{high})}$ if and only if $\mathbf{w} \oplus_f \mathbf{z} = v^{(\text{high})}$.

In the formal proof, we denote the adjacency matrix between $\mathrm{L}^{(\text{high})}$ and $\mathrm{L}^{(\text{low})}$ by W, between $\mathrm{L}^{(\text{low})}$ and $\mathrm{R}^{(\text{low})}$ by X, between $\mathrm{R}^{(\text{low})}$ and $\mathrm{R}^{(\text{high})}$ by Y, and between $\mathrm{R}^{(\text{high})}$ and $\mathrm{L}^{(\text{high})}$ by Z. See Fig. 4 for an example of this construction.

Let $\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z} \in D^{n/2}$ be vertices in $\mathrm{L}^{(\text{high})}$, $\mathrm{L}^{(\text{low})}$, $\mathrm{R}^{(\text{low})}$, and $\mathrm{R}^{(\text{high})}$. 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 $\mathrm{L}^{(\text{high})}$, have length four, and end at the same vertex $\mathbf{w}$ in $\mathrm{L}^{(\text{high})}$ again.

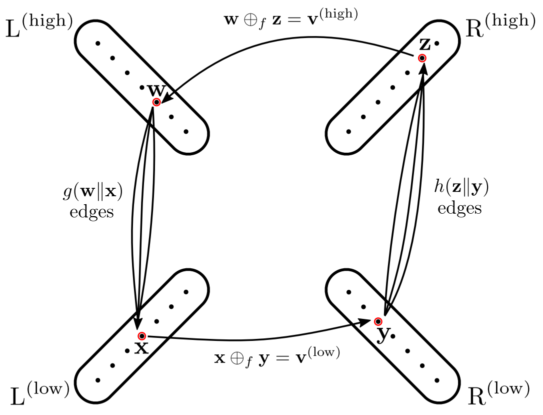

$\mathrm{L}^{(\mathrm{high})}$
$\mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\mathrm{high})}$
$\mathrm{R}^{(\mathrm{high})}$
$\mathbf{w}$
$\mathbf{z}$
$g(\mathbf{w}\|\mathbf{x})$ edges
$h(\mathbf{z}\|\mathbf{y})$ edges
$\mathbf{x}$
$\mathbf{y}$
$\mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\mathrm{low})}$
$\mathrm{L}^{(\mathrm{low})}$
$\mathrm{R}^{(\mathrm{low})}$

Fig. 4 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 $\operatorname{tr}(W \cdot X \cdot Y \cdot Z)$

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

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

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

Lemma 5.1 *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,*

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

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

$$(W \cdot X)_{\mathbf{w},\mathbf{y}} = \sum_{\mathbf{x} \in D^{n/2}} W_{\mathbf{w},\mathbf{x}} \cdot X_{\mathbf{x},\mathbf{y}} = \sum_{\mathbf{x} \in D^{n/2}} [\![\mathbf{x} \oplus_f \mathbf{y} = \mathbf{v}^{(\mathrm{low})}]\!] \cdot g(\mathbf{w}\|\mathbf{x}). \tag{5.1}$$

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

$$(Y \cdot Z)_{\mathbf{y},\mathbf{w}} = \sum_{\mathbf{z} \in D^{n/2}} Y_{\mathbf{y},\mathbf{z}} \cdot Z_{\mathbf{z},\mathbf{w}} = \sum_{\mathbf{z} \in D^{n/2}} [\![\mathbf{w} \oplus_f \mathbf{z} = \mathbf{v}^{(\mathrm{high})}]\!] \cdot h(\mathbf{z}\|\mathbf{y}). \tag{5.2}$$

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

$$(W \cdot X \cdot Y \cdot Z)_{\mathbf{w},\mathbf{w}} = \sum_{\mathbf{y} \in D^{n/2}} (W \cdot X)_{\mathbf{w},\mathbf{y}} \cdot (Y \cdot Z)_{\mathbf{y},\mathbf{w}}$$

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

where the second equality follows by (5.1) and (5.2). Thus,

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

□

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 5.1 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 $\widetilde{\mathcal{O}}(|D|^{\omega\cdot n/2} \cdot \operatorname{polylog}(M))$. □

6 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 $\widetilde{\mathcal{O}}((c|D|^2)^n \cdot \operatorname{polylog}(M))$ time algorithm that computes f-Convolution for $c{:=}3/4$ 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., [19, 23, 29, 34]) the function f has a cyclic partition with a single cyclic minor. Nevertheless, in our proof we only use cyclic minors where one domain is of size is

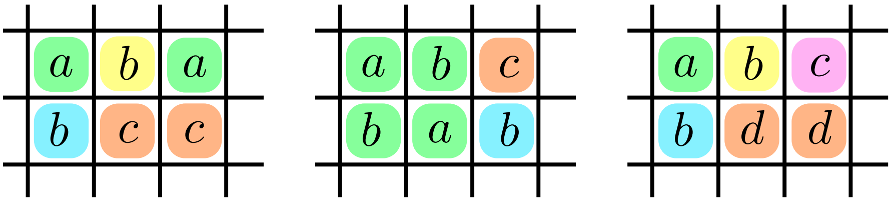


Fig. 5 Here are three concrete examples of functions f for which we expect that the running times for f-CONVOLUTION should be $\widetilde{\mathcal{O}}(3^n \cdot \text{polylog}(M))$, $\widetilde{\mathcal{O}}(3^n \cdot \text{polylog}(M))$ and $\widetilde{\mathcal{O}}(4^n \cdot \text{polylog}(M))$. However, 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 $\widetilde{\mathcal{O}}(5^n \cdot \text{polylog}(M))$ (Color figure online)

at most 2. We suspect that larger minors have to be considered to obtain better results. Indeed, the lower bound from Lemma 4.16 implies that our technique of considering two *arbitrary* rows together cannot give a faster algorithm than $\widetilde{\mathcal{O}}((3/4 \cdot |D|^2)^n \cdot \text{polylog}(M))$ in general. An improved algorithm would have to select these rows very carefully or consider three or more rows at the same time.

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 $\widetilde{\mathcal{O}}(|D|^{(2-\epsilon)n} \cdot \text{polylog}(M))$ time algorithm for f-CONVOLUTION for some $\epsilon > 0$? We are not aware of any lower bounds, so in principle even an $\widetilde{\mathcal{O}}(|D|^n \cdot \text{polylog}(M))$ time algorithm is plausible.

Ideally, we would expect that the f-CONVOLUTION problem can be solved in $\widetilde{\mathcal{O}}((|L|^n + |R|^n + |T|^n) \cdot \text{polylog}(M))$ for any function $f : L \times R \to T$. In Fig. 5 we include three examples of functions that are especially difficult for our methods.

Finally, we gave an $\widetilde{\mathcal{O}}(|D|^{\omega \cdot n/2} \cdot \text{polylog}(M))$ time algorithm for f-QUERY problem. For $\omega = 2$ this algorithm runs in almost linear-time, however for the current bound $\omega < 2.372$ our algorithm runs in time $\widetilde{\mathcal{O}}(|D|^{1.19n} \cdot \text{polylog}(M))$. Can f-QUERY be solved in $\widetilde{\mathcal{O}}(|D|^n \cdot \text{polylog}(M))$ time without assuming $\omega = 2$?

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.

Funding Open Access funding enabled and organized by Projekt DEAL. Research supported by the European Research Council (ERC) consolidator Grant No. 725978 SYSTEMATICGRAPH and the project TIPEA (Grant No. 850979).

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

A Proof of Theorem 2.6

In this section we prove Theorem 2.6. We crucially rely on the following result by van Rooij [33].

Theorem A.1 ([33, Lemma 3]) *There is an algorithm which given $k \in \mathbb{N}$, $\mathbf{r} \in \mathbb{N}^k$ a prime p, an $\mathbf{r}_j$-th primitive root of unity ω_j for every $j \in [k]$ and two functions $g, h\colon \mathbb{Z}_{\mathbf{r}_1} \times \cdots \times \mathbb{Z}_{\mathbf{r}_k} \to \mathbb{Z}$ computes the cyclic convolution of g and h modulo p (that is, return a function ϕ such that $\phi(\mathbf{q}) = (g \odot h)(\mathbf{v}) \mod p$ for every $\mathbf{v} \in \mathbb{Z}_{r_1} \times \cdots \times \mathbb{Z}_{r_k}$) in $\mathcal{O}(R \log(R))$ arithmetic operations where $R = \prod_{j=1}^{k} \mathbf{r}_j$.*

Ideally, we would like to use the algorithm from Theorem A.1 with a sufficiently large prime p such that the values of $g \odot h$ could be recovered from the values of $g \odot h$ modulo p. Finding such a prime p along with the required roots of unity is, however, a non trivial task which we do not know how to perform deterministically while retaining the running time at $\widetilde{\mathcal{O}}(R \cdot \text{polylog}(M))$. The basic idea behind our approach is to compute $g \odot h$ modulo p_i for a sufficiently large number of distinct small primes p_i using Theorem A.1. If $\prod_i p_i$ is sufficiently large, then the values of $g \odot h$ can be uniquely recovered using the Chinese Remainder Theorem.

Theorem A.2 (Chinese Remainder Theorem) *Let $p_1, \ldots, p_m$ denote a sequence of integers that are pairwise coprime and define $P := \prod_{i \in [m]} p_i$. Also let $0 \le a_i < p_i$ for all $i \in [m]$. Then there is a unique number $0 \le s < P$ such that*

$$s \equiv a_i \mod p_i$$

for all $i \in [m]$. Moreover, there is an algorithm that, given $p_1, \ldots, p_m$ and $a_1, \ldots, a_m$, computes the number s in time $\mathcal{O}((\log P)^2)$.

To find the small primes for the application of the Chinese Remainder Theorem, we additionally use density properties of primes in arithmetic progression. Given $q \in \mathbb{N}$, we say $p \in \mathbb{N}$ is a *q-prime* if p is a prime number and $p \equiv 1 \mod q$. We use $\mathsf{prime}_q(i)$ to denote the i-th q-prime. That is, $\mathsf{prime}_q(i)$ is a q-prime such that the number of q-primes smaller than $\mathsf{prime}_q(i)$ is exactly $i - 1$. Also, for any $B, q \in \mathbb{N}$, we define

$$\mathsf{prime\_bound}_q(B) := \min \left\{ m \in \mathbb{N} \;\middle|\; \prod_{i=1}^{m} \mathsf{prime}_q(i) \ge B \right\}$$

to be the minimal number m such that the product of the first m q-primes is at least B. We use the following upper bound on $\mathsf{prime\_bound}$.

Lemma A.3 *Let $B, q \in \mathbb{N}$ be integers such that $B, q \geq 3$ and $m = \mathsf{prime\_bound}_q(B)$. Then $m \leq \ln(B) + 1$ and $\mathsf{prime}_q(m) \leq \max\left\{\exp\left(8 \cdot \sqrt{q} \cdot \ln^3(q)\right),\ \exp(q),\ 2q \cdot \ln(B)\right\}$.*

In the proof of Lemma A.3 we use a known result for the density of primes in arithmetic progressions taken from [3]. For any $x, q \in \mathbb{N}$, define $\theta(x, q)$ to be the sum of $\ln(p)$ for all q-primes p such that $p \leq x$. Formally, we define

$$\theta(x,q) := \sum_{i=1}^{\infty} [\![\mathsf{prime}_q(i) \leq x]\!] \cdot \ln\left(\mathsf{prime}_q(i)\right).$$

With this definition, we can now state the result about the density of primes in arithmetic progressions.

Lemma A.4 ([3, Corollary 1.8]) *Let q and x be integers with $q > 3$ and $x \geq \exp(8 \cdot \sqrt{q} \cdot \ln^3 q)$. Then,*

$$\theta(x,q) \geq \frac{x}{\varphi(q)} - \frac{1}{160} \cdot \frac{x}{\ln x}$$

where φ is Euler's totient function.

Now we have everything ready to prove Lemma A.3.

Proof of Lemma A.3 We first prove the bound for m. By the definition of m as $m = \mathsf{prime\_bound}_q(B)$, we get $\prod_{i=1}^{m-1} \mathsf{prime}_q(i) < B$. As $\ln(\mathsf{prime}_q(i)) > 1$ for every i, we have

$$m - 1 < \sum_{i=1}^{m-1} \ln(\mathsf{prime}_q(i)) = \ln\left(\prod_{i=1}^{m-1} \mathsf{prime}_q(i)\right) < \ln(B)$$

which implies $m < \ln(B) + 1$.

Now we prove the bound for $\mathsf{prime}_q(m)$. For this we set

$$x = \max\left\{\exp\left(8 \cdot \sqrt{q} \cdot \ln^3(q)\right),\ \exp(q),\ 2q \cdot \ln(B)\right\}.$$

By Lemma A.4, we get

$$\begin{aligned}\theta(x,q) &\geq \frac{x}{\varphi(q)} - \frac{1}{160} \cdot \frac{x}{\ln x}\\ &= x\left(\frac{1}{\varphi(q)} - \frac{1}{160 \cdot \ln x}\right)\end{aligned}$$

and, using $\varphi(q) \leq q$ and $\ln(x) \geq q$, we have

$$\theta(x,q) \geq x \cdot \left(\frac{1}{q} - \frac{1}{160 \cdot q}\right)$$

$$\begin{aligned}&\geq x \cdot \frac{1}{2 \cdot q}\\&\geq \ln B.\end{aligned}\tag{A.1}$$

Let $\ell = \max\{j \mid \mathsf{prime}_q(j) \leq x\}$ be the index of the largest q-prime which is not greater than x. Then,

$$\begin{aligned}\prod_{i=1}^{\ell} \mathsf{prime}_q(i) &= \exp\left(\sum_{i=1}^{\ell} \ln(\mathsf{prime}_q(i))\right)\\&= \exp\left(\sum_{i=1}^{\infty} [\![\mathsf{prime}_q(i) \leq x]\!] \cdot \ln(\mathsf{prime}_q(i))\right)\\&= \exp\left(\theta(x, q)\right) \geq B,\end{aligned}$$

where the inequality follows from (A.1). By the definition of $\mathsf{prime\_bound}_q$, we get $m = \mathsf{prime\_bound}_q(B) \leq \ell$. Hence, $\mathsf{prime}_q(m) \leq \mathsf{prime}_q(\ell) \leq x$ which finishes the proof. □

In the remainder we give the $\widetilde{\mathcal{O}}\left((\prod_{i=1}^{k} \mathbf{r}_i) \cdot \mathrm{polylog}(M)\right)$ algorithm for the K-Cyclic Convolution Problem.

Proof of Theorem 2.6 Fix a finite set $K = \{c_1, \ldots, c_\ell\} \subseteq \mathbb{N}$ which is considered as a constant throughout this proof. Let integers $k, M \in \mathbb{N}$, integer vector $\mathbf{r} \in K^k$ and functions $g, h\colon Z \to \{-M, \ldots, M\}$ where $Z = \mathbb{Z}_{\mathbf{r}_1} \times \cdots \times \mathbb{Z}_{\mathbf{r}_k}$ be an input for the K-Cyclic Convolution Problem.

For every $t \in [\ell]$, let D_t be the prime factors of c_t. We define $R = \prod_{j=1}^{k} \mathbf{r}_j$ and observe that for any $\mathbf{v} \in Z$ it holds that $|(g \odot h)(\mathbf{v})| \leq R \cdot M^2$. Further define $B{:=}3 \cdot R \cdot M^2$ and $q = \prod_{c \in K} c = \prod_{t=1}^{\ell} c_t$. Assume without loss of generality that $q \geq 3$ and note that q depends only on the fixed finite set K and therefore, can be viewed as a constant.

With this notation we can formally state the algorithm.

1. Iterate over the numbers of the form $q \cdot a + 1$ for $a \in \{1, 2, \ldots\}$ and test for each one if it is prime. The process continues until the product of the q-primes exceeds B. Denote these numbers by $p_1, \ldots, p_m$.
2. For every $i \in [m]$ and $t \in [\ell]$, iterate over all elements $x \in \mathbb{F}_{p_i}$ and test whether $x^{c_t} \equiv 1 \mod p_i$ and $x^{c_t/d} \not\equiv 1 \mod p_i$ for every $d \in D_t$. If so, then set x as the c_t-th root of unity in $\mathbb{F}_{p_i}$.
3. For all $i \in [m]$, use Theorem A.1 with the prime p_i and appropriate roots of unity to compute the function $f^{(i)}\colon Z \to \mathbb{Z}_{p_i}$ defined by
$$f^{(i)}(\mathbf{v}){:=}(g \odot h)(\mathbf{v}) \mod p_i \qquad \forall \mathbf{v} \in Z.$$
4. Define $P = \prod_{i=1}^{m} p_i$, we define a function $f_P\colon Z \to \mathbb{Z}_P$ as follows. For each $\mathbf{v} \in Z$, use the Chinese Remainder Theorem (cf. Theorem A.2) to compute the value $0 \leq f_P(\mathbf{v}) < P$ such that $f_P(\mathbf{v}) \equiv f^{(i)}(\mathbf{v}) \mod p_i$ for all $i \in [m]$.

5. Finally, compute the function $f\colon Z \to \mathbb{Z}$ using the formula

$$f(\mathbf{v}) = \begin{cases} f_P(\mathbf{v}) & \text{if } f_P(\mathbf{v}) < \frac{P}{2} \\ f_P(\mathbf{v}) - P & \text{if } f_P(\mathbf{v}) \geq \frac{P}{2} \end{cases}$$

   for all $\mathbf{v} \in Z$ and return f.

Before we move to proving the correctness, we first argue that the algorithm is well-defined. From the definition, the first step computes the first $m = \mathsf{prime\_bound}_q(B)$ q-primes such that $p_1 = \mathsf{prime}_q(1), \dots, p_m = \mathsf{prime}_q(m)$. It remains to show that, for every $i \in [m]$ and $t \in [\ell]$, the c_t-th primitive root of unity in $\mathbb{F}_{p_i}$ exists. Indeed, since c_t divides $p_i - 1$ (which is in turn true as $p_i \equiv 1 \mod q$ and c_t divides q), such a root of unity exists. Moreover, as D_t contains all prime factors of c_t, one can easily show that it actually suffices to consider only values of the form $x^{c_t/d}$ for every $d \in D_t$ to correctly decide if x is a primitive c_t-th root of unity in $\mathbb{F}_{p_i}$. The application of Theorem A.1 in the second step is possible as $\mathbf{r}_j \in K = \{c_1, \dots, c_\ell\}$ for every $j \in [n]$ and the roots of unity are computed by the second step.

Now we argue about the correctness of the algorithm.

Claim A.5 For all $\mathbf{v} \in Z$, we have $f(\mathbf{v}) = (g \odot h)(\mathbf{v})$.

Proof As the algorithm is well defined, the third step computes, the convolution of g and h modulo p_i for every $i \in [m]$.

Now fix some $\mathbf{v} \in Z$. We define $b(\mathbf{v}) = (h \odot g)(\mathbf{v}) \mod P$ and observe $0 \leq b(\mathbf{v}) < P$. Moreover, for every $i \in [m]$ it holds that

$$b(\mathbf{v}) \mod p_i = ((h \odot g)(\mathbf{v}) \mod P) \mod p_i = (h \odot g)(\mathbf{v}) \mod p_i = f^{(i)}(\mathbf{v}).$$

Since Theorem A.2 also guarantees the resulting number to be unique, it follows that $f_P(\mathbf{v}) = b(\mathbf{v})$ which implies $f_P(\mathbf{v}) = (g \odot h)(\mathbf{v}) \mod P$.

Now we focus on the last step. By the definition of $m = \mathsf{prime\_bound}_q(B)$, it holds that $P = \prod_{i=1}^{m} p_i \geq B = 3 \cdot R \cdot M^2$. Consider the following cases.

- In case $(g \odot h)(\mathbf{v}) \geq 0$ we have

$$(g \odot h)(\mathbf{v}) \leq R \cdot M^2 < \frac{B}{2} \leq P.$$

  This implies that $f_P(\mathbf{v}) = (g \odot h)(\mathbf{v}) \mod P = (g \odot h)(\mathbf{v}) < \frac{B}{2}$. Thus, $f(\mathbf{v}) = f_P(\mathbf{v}) = (g \odot h)(\mathbf{v})$.
- In case $(g \odot h)(\mathbf{v}) < 0$ it holds that

$$(g \odot h)(\mathbf{v}) \geq -R \cdot M^2 > -P.$$

  This now implies that

$$f_P(\mathbf{v}) = (g \odot h)(\mathbf{v}) + P \geq P - R \cdot M^2 > \frac{P}{2}.$$

Hence, $f(\mathbf{v}) = f_P(\mathbf{v}) - P = (g \odot h)(\mathbf{v}) + P - P = (g \odot h)(\mathbf{v})$.

Hence, $f(\mathbf{v}) = (g \odot h)(\mathbf{v})$ for all $\mathbf{v} \in Z$, which concludes the proof. □

From Claim A.5 we know that the algorithm is correct and the function f returned by the algorithm is indeed $(g \odot h)$. It only remains to analyze the running time of the procedure.

Claim A.6 The procedure terminates in time $\widetilde{\mathcal{O}}(R \cdot \text{polylog}(M))$.

Proof We consider each step on its own.

1. Since prime testing can be done in polynomial time (in the representation size of the number), we can find the sequence $p_1, \dots, p_m$ in time $\mathcal{O}(p_m \cdot \text{polylog}\, p_m)$. By Lemma A.4, and since q is a constant, it follows that
$$p_m \le \max\left\{\exp\left(8 \cdot \sqrt{q} \cdot \ln^3(q)\right),\ \exp(q),\ 2q \cdot \ln(B)\right\}$$
$$= \mathcal{O}(\ln(B)) = \mathcal{O}(\log(R \cdot M))$$
and $m \le \ln(B) + 1 = \ln(3RM^2) + 1$. Hence, the running time of this step is $\mathcal{O}(p_m \cdot \text{polylog}\, p_m) = \mathcal{O}(\text{polylog}(R \cdot M))$.
2. For each $i \in [m]$ and $t \in [\ell]$, in Step 2. of the algorithm we iterate over p_i values and check $|D_t|$ values. Since D_t are the prime factors of c_t (and hence $|D_t|$ is a constant), this takes time $\mathcal{O}(p_i\, \text{polylog}\, p_i)$ which can be bounded by $\mathcal{O}(\text{polylog}(R \cdot M))$. Since $m \le \log(3 \cdot R \cdot M^2) + 1$ and ℓ is a constant, the overall running time of the step is $\mathcal{O}(\text{polylog}(R \cdot M))$.
3. By Theorem A.1, the number of arithmetic operations required to compute $f^{(i)}$ is $\mathcal{O}(R \cdot \log(R))$. Since each arithmetic operation is performed in $\mathbb{F}_{p_i}$, the total time spent to compute $f^{(i)}$ is
$$\mathcal{O}(R \cdot \log(R) \cdot \log^2(p_i)) = \mathcal{O}(R \cdot \log(R) \cdot \log^2(\log(R \cdot M)))$$
$$= \mathcal{O}(R \cdot \text{polylog}(R \cdot M)),$$
where the first equality holds because $p_i \le p_m = \mathcal{O}(\log(R \cdot M))$. Finally, as $m = \mathcal{O}(\log(R \cdot M))$, the overall computation time of this step is $m \cdot \mathcal{O}(R \cdot \text{polylog}(R \cdot M) = \mathcal{O}(R \cdot \text{polylog}(R \cdot M))$.
4. As we iterate over all R values from Z and by Theorem A.2, this computation can be done in time $\mathcal{O}(R \cdot (\log P)^2)$. Since $\log P \le m \cdot p_m \le \mathcal{O}(\text{polylog}(R \cdot M))$ the overall running time of this step is $\mathcal{O}(R \cdot \text{polylog}(R \cdot M))$.
5. As we again iterate over all elements from Z, the computation time of this step is $\mathcal{O}(R \cdot \text{polylog}\, P) = \mathcal{O}(R \cdot \text{polylog}(R \cdot M))$ where we use $P = \mathcal{O}(\text{polylog}(R \cdot M))$.

As the running time of each step is at most $\widetilde{\mathcal{O}}(R \cdot \text{polylog}(M))$, the overall running time of the algorithm is $\widetilde{\mathcal{O}}(R \cdot \text{polylog}(M))$. □

The proof now follows by Claim A.5 and A.6. □

References

1. Abboud, A., Williams, R.R., Yu, H.: More applications of the polynomial method to algorithm design. In: Indyk, P. (ed.) Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4–6, 2015, pp. 218–230. SIAM (2015)
2. Alman, J., Williams, V.V.: A refined laser method and faster matrix multiplication. In: Marx D (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021. SIAM, pp. 522–539 (2021)
3. Bennett, M.A., Martin, G., O'Bryant, K., Rechnitzer, A.: Explicit bounds for primes in arithmetic progressions. Ill. J. Math. **62**(1–4), 427–532 (2018)
4. Beth, T.: Verfahren der schnellen Fourier-Transformation: die allgemeine diskrete Fourier-Transformation–ihre algebraische Beschreibung, Komplexität und Implementierung, vol. 61. Teubner (1984)
5. Björklund, A., Husfeldt, T.: 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, pp. 727–735. IEEE Computer Society (2013)
6. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Fourier meets Möbius: fast subset convolution. In: Johnson, D.S., Feige, U. (eds.) Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11–13, 2007, pp. 67–74. ACM (2007)
7. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Counting paths and packings in halves. In: Fiat A, Sanders P (eds.) Algorithms—ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7–9, 2009. Proceedings, volume 5757 of Lecture Notes in Computer Science, pp. 578–586. Springer (2009)
8. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M.: Covering and packing in linear space. Inf. Process. Lett. **111**(21–22), 1033–1036 (2011)
9. Björklund, A., Husfeldt, T., Kaski, P., Koivisto, M., Nederlof, J., Parviainen, P.: Fast zeta transforms for lattices with few irreducibles. ACM Trans. Algorithms **12**(1), 4:1-4:19 (2016)
10. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion–exclusion. SIAM J. Comput. **39**(2), 546–563 (2009)
11. Brand, C.: Discriminantal subset convolution: Refining exterior-algebraic methods for parameterized algorithms. J. Comput. Syst. Sci. **129**, 62–71 (2022)
12. Bringmann, K., Fischer, N., Hermelin, D., Shabtay, D., Wellnitz, P.: Faster minimization of tardy processing time on a single machine. Algorithmica **84**(5), 1341–1356 (2022)
13. Bringmann, K., Künnemann, M., Węgrzycki, K.: 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, pp. 943–954 (2019)
14. Chan, T.M., He, Q.: Reducing 3SUM to convolution-3SUM. In: Farach-Colton, M., Gørtz, I.L. (eds.) 3rd Symposium on Simplicity in Algorithms, SOSA 2020, Salt Lake City, UT, USA, January 6–7, 2020, pp. 1–7. SIAM (2020)
15. Chan, T.M., Williams, R.R.: Deterministic APSP, Orthogonal Vectors, and more: quickly derandomizing Razborov–Smolensky. ACM Trans. Algorithms **17**(1), 2:1-2:14 (2021)
16. Clausen, M.: Fast generalized Fourier transforms. Theor. Comput. Sci. **67**(1), 55–63 (1989)
17. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**(90), 297–301 (1965)
18. Cygan, M., Mucha, M., Węgrzycki, K., Włodarczyk, M.: On problems equivalent to (min, +)-convolution. ACM Trans. Algorithms **15**(1), 14:1-14:25 (2019)
19. Cygan, M., Nederlof, J., Pilipczuk, M., Pilipczuk, M., van Rooij, J.M.M., Wojtaszczyk, J.O.: Solving connectivity problems parameterized by treewidth in single exponential time. ACM Trans. Algorithms **18**(2), 17:1-17:31 (2022)
20. Cygan, M., Pilipczuk, M.: Exact and approximate bandwidth. Theor. Comput. Sci. **411**(40–42), 3701–3713 (2010)
21. Duan, R., Wu, H., Zhou, R.: Faster Matrix Multiplication via Asymmetric Hashing. CoRR arXiv:2210.10173 (2022)
22. Hall, P.: A contribution to the theory of groups of prime-power order. Proc. Lond. Math. Soc. **2**(1), 29–95 (1934)
23. Hegerfeld, F., Kratsch, S.: Solving connectivity problems parameterized by treedepth in single-exponential time and polynomial space. In: Paul, C., Bläser, M. (eds.) 37th International Symposium

on Theoretical Aspects of Computer Science, STACS 2020, March 10–13, 2020, Montpellier, France, volume 154 of LIPIcs, pp. 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)

24. Hegerfeld, F., Kratsch, S.: Tight algorithms for connectivity problems parameterized by clique-width. In: Proceedings of ESA (2023) (to appear)
25. Künnemann, M., Paturi, R., Schneider, S.: On the fine-grained complexity of one-dimensional dynamic programming. In: Chatzigiannakis, I., Indyk, P., Kuhn, F., Muscholl, A. (eds.) 44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10–14, 2017, Warsaw, Poland, volume 80 of LIPIcs, pp. 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
26. Lincoln, A., Polak, A., Williams, V.V.: Monochromatic triangles, intermediate matrix products, and convolutions. In: Vidick, T. (ed.) 11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12–14, 2020, Seattle, Washington, USA, volume 151 of LIPIcs, pp. 53:1–53:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
27. Nederlof, J.: Personal communication (2022)
28. Nederlof, J., Pawlewicz, J., Swennenhuis, C.M.F., Węgrzycki, K.: A faster exponential time algorithm for bin packing with a constant number of bins via additive combinatorics. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021, pp. 1682–1701. SIAM (2021)
29. Nederlof, J., Pilipczuk, M., Swennenhuis, C.M.F., Węgrzycki, K.: Hamiltonian cycle parameterized by treedepth in single exponential time and polynomial space. In: Adler, I., Müller, H. (eds) 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, pp. 27–39. Springer (2020)
30. Nederlof, J., Węgrzycki, K.: Improving Schroeppel and Shamir's algorithm for subset sum via Orthogonal Vectors. In: Khuller, S., Williams, V.V. (eds.) STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21–25, 2021, pp. 1670–1683. ACM (2021)
31. Rockmore, D.N.: Recent progress and applications in group FFTs. In: Byrnes, J. (ed.) Computational noncommutative algebra and applications, pp. 227–254. Springer, Berlin (2004)
32. Umans, C.: Fast generalized DFTs for all finite groups. In: Zuckerman, D. (ed.) 60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9–12, 2019, pp. 793–805. IEEE Computer Society (2019)
33. van Rooij, J.M.M.: Fast algorithms for join operations on tree decompositions. In: Fomin, F.V., Kratsch, S., van Leeuwen, E.J. (eds.) 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, pp. 262–297. Springer (2020)
34. van Rooij, J.M.M.: A generic convolution algorithm for join operations on tree decompositions. In: Santhanam, R., Musatov, D. (eds.) 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, pp. 435–459. Springer (2021)
35. van Rooij, J.M.M., Bodlaender, H.L., Rossmanith, P.: Dynamic programming on tree decompositions using generalised fast subset convolution. In: Fiat, A., Sanders, P. (eds.) Algorithms—ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7–9, 2009. Proceedings, volume 5757 of Lecture Notes in Computer Science, pp. 566–577. Springer (2009)
36. Vassilevska-Williams, V.: On some fine-grained questions in algorithms and complexity. In: Proceedings of the International Congress of Mathematicians (ICM 2018), pp. 3447–34 (2018)
37. Weisner, L.: Abstract theory of inversion of finite series. Trans. Am. Math. Soc. **38**(3), 474–484 (1935)
38. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theor. Comput. Sci. **348**(2–3), 357–365 (2005)
39. Włodarczyk, M.: Clifford algebras meet tree decompositions. Algorithmica **81**(2), 497–518 (2019)
40. Yates, F.: The design and analysis of factorial experiments. Imperial Bureau of Soil Science. Technical Communication (1937)
41. Zamir, O.: Breaking the 2^n barrier for 5-coloring and 6-coloring. In: Bansal, N., Merelli, E., Worrell, J. (eds.) 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12–16, 2021, Glasgow, Scotland (Virtual Conference), volume 198 of LIPIcs, pp. 113:1–113:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)