Tensor Product of Functional Graphs: New Insights into Factorization and Root-Finding

Christophe Crespelle[1], Thi Ha Duong Phan[2,*], Adrien Richard[1]
[1]Université Côte d'Azur, CNRS, I3S, Sophia Antipolis, France
{christophe.crespelle, adrien.richard}@univ-cotedazur.fr
[2] ICRTM - Institute of Mathematics, Vietnam Academy of Science and Technology, Vietnam
Corresponding author. phanhaduong@math.ac.vn

March 2, 2025

Abstract

Functional graphs, where each vertex has out-degree one, model discrete dynamical systems. The series and parallel operations on these graphs correspond to the addition and tensor product of the corresponding graphs. Consequently, the decomposition of a dynamical system translates to factorization, root finding for powers, and root finding for polynomials on functional graphs.

This paper focuses on a principal class of functional graphs - the class of sums of cycles. We introduce an algebraic framework that reformulates these graph-theoretic problems as computations over the semi-ring of multi-variable polynomials with natural number coefficients. This approach leverages cycle lengths, greatest common divisors, and least common multiples to develop efficient algorithms. We construct a general algorithm for counting the divisions of two sums of cycles and present some of its applications. In particular, we propose a logarithmic-time algorithm that fully resolves the root-of-power and root-of-polynomial problems.

1 Introduction

In the study of discrete dynamical systems, the state space can be represented as a directed graph, where each state corresponds to a vertex, and each transition from one state to another is represented by a directed edge from the first state to the second. If the system is deterministic, each vertex has an out-degree of exactly one. This directed graph is termed a *functional graph*. The set of all functional digraphs is denoted by $\mathbb{D}$.

Two fundamental operations on dynamical systems are of particular interest: the *series operation* and the *parallel operation* which describe how smaller systems combine to form larger ones.

- The **series operation** of two systems A and B corresponds to the disjoint union of two systems, meaning they evolve independently without interaction. In terms of functional graphs, this operation simply merges the vertex sets while keeping their transitions separate.

- The **parallel operation**, which models the simultaneous evolution of two systems A and B, results in a new system C whose state space is the Cartesian product of the state spaces of A and B. In the corresponding graph, there is a directed edge from state (a, b) to state (a', b') in C (where a, a' are states of A and b, b' are states of B) if and only if there exist directed edges from a to a' in A and from b to b' in B.

Mathematically, the state spaces of two systems A and B can be represented as directed graphs, and these two operations correspond to the *addition* $A + B$ and the *tensor product* AB of graphs A and B, respectively. The tensor product is a well-established concept, originally introduced in the famous book *Principia Mathematica* by Alfred North Whitehead and Bertrand Russell [11], and also known as the Kronecker product [10]. The set of functional graphs, equipped with these two operations, forms a semi-ring. It was first introduced in [4] and has since been widely studied [3, 8, 5].

In the study of dynamical systems, a fundamental question is whether a system can be constructed from smaller subsystems. This leads to the following key problems:

1. **Division Problem** (or factorization problem): Given a system B, can it be constructed via the parallel operation from smaller systems? Equivalently, do there exist two functional graphs A and X such that $AX = B$? A simpler variant of this question is: Given two functional graphs A and B, does there exist a functional graph X such that $AX = B$? (see for example [5]).

2. **Root of a Power:** Can we find a smaller system whose multi-parallel composition results in a given system? In graph-theoretic terms, given a functional graph B and a natural number m, does there exist a functional graph A such that $A^m = B$? (see for example [12]).

3. **Root of a Polynomial:** More generally, given a scheme involving both series and parallel compositions represented by a polynomial P with natural number coefficients, does there exist a functional graph A such that $P(A) = B$? (see for example [4]).

These problems have been widely studied for general graphs. For example, the factorization problem for the tensor product of undirected graphs has been shown to be solvable in polynomial time [7]. In contrast, the same problem for directed graphs is still open. Since then, various studies have explored the problem for special classes of directed graphs, yielding different complexity results [5, 2].

In our case, functional graphs have a specific structure, which enables the development of tailored algorithms for these problems. A functional digraph consists of two distinct components: the *cyclic part*, which comprises a set of disjoint directed cycles, and the *transient part*, which consists of disjoint directed rooted trees obtained by removing the cycles.

The factorization of functional graphs has been explored from various perspectives. Precise results have been obtained for functional graphs containing only a transient part [12, 6], and the problem of root-finding for polynomials on the transient part was recently studied in [9]. Meanwhile, factorization results for functional graphs consisting solely of a cyclic part have also been established [1]. It is expected that by analyzing these two types of functional graphs separately, one can combine the results to derive conclusions for general functional graphs.

In this paper, we focus on the second type, which we refer to as the *sum of cycles*.

Our main contribution is the construction of algorithms that address the three aforementioned problems in the general setting of sums of cycles. Until now, this problem has been discussed in the literature, but existing results are limited to specific cases, and no efficient, implementable algorithm has yet been proposed to fully resolve them. The novelty of our approach lies in transforming the problem on graphs into problems in algebra and number theory. Specifically, the multiplication of sums of cycles depends on the lengths of the cycles, utilizing the greatest common divisor and the least common multiple. Consequently, the decomposition of natural numbers (which represent cycle lengths) into prime factors plays a crucial role. The key idea is to represent a sum of cycles as a polynomial in multiple variables, where each variable is assigned a prime number. As a result, our computations reduce to operations in the semi-ring of multi-variable polynomials over the set of natural numbers, $\mathbb{N}$.

In Section 2, we present in detail our theory on the semi-ring of polynomials and describe the translation from dynamical systems terminology to algebraic language.

In Section 3, we investigate problems related to the divisors of a graph that can be expressed as a sum of cycles. Initially, we focus on the case where the cycle lengths are powers of the same prime number. We then generalize the problem using a recurrence method based on the number of distinct prime factors appearing in the cycles.

Section 4 is devoted to the study of the root of powers and the root of polynomials. We propose an efficient algorithm with logarithmic complexity and provide a complete solution to the problem.

2 Functional graphs and Semi-ring of sum of cycles

2.1 Sum of cycles

Let G be a directed graph in which every vertex has out-degree exactly one. It is evident that G consists of a union of directed trees and directed cycles. The only possible attachments between them occur when a cycle is connected to the root of a tree.

In this paper, we focus on a particular subclass of such graphs, namely those that are disjoint unions of directed cycles. We refer to these graphs as *sums of cycles* and represent a sum X in the form

$$X = x_1C_1 + x_2C_2 + \cdots + x_mC_m,$$

where each C_i is a directed cycle of i vertices, and the coefficients x_i are nonnegative integers for $i = 1, 2, \dots, m$, indicating the multiplicity of each cycle in the decomposition.

We define the *types* of X, denoted as $L(X)$, to be the set length of cycles in X. The *cardinal* of X, denoted as $\mathrm{card}(X)$, is the number of distinct length of cycles in X, that is, the number of nonzero coefficients x_i. Equivalently, $\mathrm{card}(X) = |L(X)|$.

Additionally, we define the *size* of X, denoted as $N(X)$, to be the total number of vertices in X. This is given by the formula:

$$N(X) = x_1 \cdot 1 + x_2 \cdot 2 + \cdots + x_m \cdot m.$$

We denote $\mathbf{D}$ the set of functional graphs, and $\mathbf{\Lambda}$ the set of sums of cycles.

2.2 Operations over the set of sums of cycles

Recall that the *series operation* of two dynamical systems consists of running them separately, meaning that the state space of the resulting system is simply the disjoint union of the state spaces of the two component systems. When these systems correspond to sums of cycles, the series operation translates into the following operation on sums of cycles.

Let X and Y be two sums of cycles

$$X = x_1C_1 + x_2C_2 + \cdots + x_kC_k,$$

$$Y = y_1C_1 + y_2C_2 + \cdots + y_kC_l.$$

Addition: The sum $Z = X + Y$ is defined as

$$Z = \sum_{i=1}^{\max(k,l)} z_iC_i,$$

where $z_i = x_i + y_i$ for each coefficient i. If an index i is not present in X or Y, the corresponding coefficient is taken as zero by convention.

On the other hand, if T is obtained by the parallel operation of two systems X and Y, then the state space of T is the Cartesian product of the state spaces of X and Y. In the corresponding graph, there is a directed edge from state (x, y) to state (x', y') in T (where x, x' are states of X and y, y' are states of Y) if and only if there exists a directed edge from x to x' in X and from y to y' in Y.

It is easy to check that if $X = C_k$ and $Y = C_l$ are simple cycles of lengths k and l, then T consists of d disjoint cycles of length m, where

$$d = \gcd(k, l), \quad m = \operatorname{lcm}(k, l),$$

with $\gcd(k, l)$ and $\operatorname{lcm}(k, l)$ denoting the greatest common divisor and least common multiple of k and l, respectively.

Thus, we define the *multiplication* of sums of cycles as follows.

Tensor product: The product $T = X \cdot Y$ (or, by sort $T = XY$) is defined by

$$T = \sum_{i,j} x_iy_jC_iC_j,$$

$$T = \sum_{i,j} x_iy_j \gcd(i, j)C_{\operatorname{lcm}(i,j)}.$$

Figure 1 and Figure 2 present examples of tensor product.

The set of sums of cycles $\mathbf{\Lambda}$, equipped with the addition and multiplication, forms a semi-ring with the zero element 0 and the unit element C_1.

2.3 Properties of the semi-ring $\mathbf{\Lambda}$

Because multiplication in the semi-ring $\mathbf{\Lambda}$ is based on the greatest common divisor and least common multiple operations, $\mathbf{\Lambda}$ exhibits some properties that differ from those of the usual semi-ring of natural numbers or the semi-ring of polynomials. We present here some preliminary properties.

Let $X \in \mathbf{\Lambda}$ be a sums of cycle, $X = x_1C_1 + x_2C_2 + \cdots + x_kC_k$.

For an integer n, we denote by $X_{D(n)}$ the sum of all cycles whose lengths are divisors of n:

$$X_{D(n)} = \sum_{i|n} x_iC_i.$$

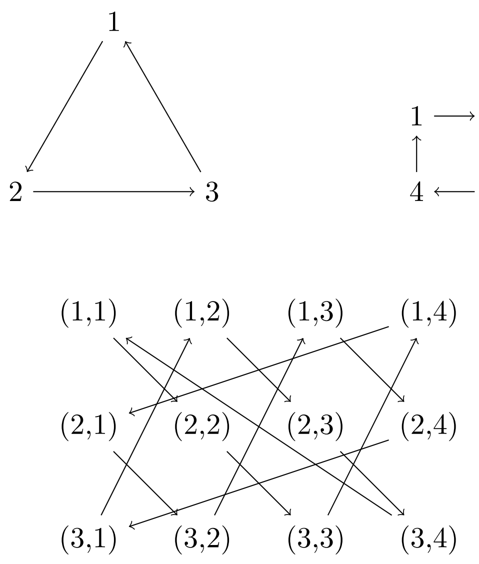

Figure 1: The tensor product $C_3 \cdot C_4$ consists of one cycle C_{12}.

Lemma 2.1 *Let X and Y be two sums of cycles, and let Z and T denote their sum and product, respectively. Then, for any integer n, we have*

$$Z_{D(n)} = X_{D(n)} + Y_{D(n)}, \text{ and}$$

$$T_{D(n)} = X_{D(n)} \cdot Y_{D(n)}.$$

The statement for the sum follows directly from the definition. The case of the product is less immediate, and we leave its proof as an exercise for the reader.

Corollary 2.2 *Let X and Y be two sums of cycles, and let Z and T denote their sum and product, respectively. Then, for any integer n, we have*

$$N(Z_{D(n)}) = N(X_{D(n)}) + N(Y_{D(n)}), \text{ and}$$

$$N(T_{D(n)}) = N(X_{D(n)})N(Y_{D(n)}).$$

Note that prime numbers play a crucial role in the study of the multiplication operation. In the following subsections, we will focus on this aspect in detail.

2.4 Semi-ring of Polynomials

2.4.1 Polynomials over a Single Prime Number

Consider the case where the lengths of all cycles are powers of the same prime number p.

Such a sum of cycles X can be expressed as

$$X = x_1C_1 + x_pC_p + x_{p^2}C_{p^2} + \cdots + x_{p^m}C_{p^m},$$

where the coefficients $x_1, x_p, \ldots, x_{p^m}$ are natural numbers.

This representation allows us to view X as a polynomial in p.

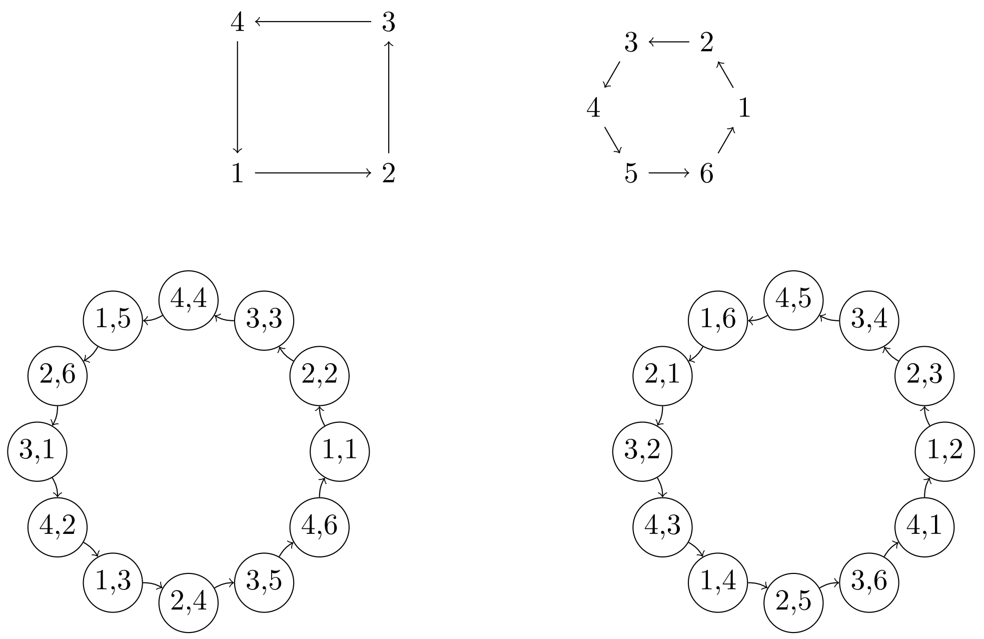

Figure 2: The tensor product $C_4 \cdot C_6$ consists of two disjoint cycles C_{12}.

For two such sums of cycles,

$$X = x_1 C_1 + x_p C_p + x_{p^2} C_{p^2} + \cdots + x_{p^m} C_{p^m},$$

$$Y = y_1 C_1 + y_p C_p + y_{p^2} C_{p^2} + \cdots + y_{p^k} C_{p^k},$$

their product $X \cdot Y$ follows the rule

$$X \cdot Y = \sum_{0 \le i \le m,\, 0 \le j \le k} x_{p^i} y_{p^j} p^{\min(i,j)} C_{p^{\max(i,j)}}.$$

We define the set of all such sums of cycles as the **semi-ring of polynomials in** p, denoted by $\mathbb{N}[p]$.

To extend this structure, we introduce $\mathbb{Q}[p]$, where coefficients are allowed to be rational numbers. The set $\mathbb{Q}[p]$, equipped addition and multiplication, forms a **ring over** $\mathbb{Q}$.

2.4.2 Polynomial over multi prime numbers

Let $p_1, p_2, \ldots, p_k$ be k different prime numbers. We define $\mathbb{N}[p_1, p_2, \ldots, p_k]$ (resp. $\mathbb{Q}[p_1, p_2, \ldots, p_k]$) as the set of sums of cycles X of the form

$$X = \sum_{0 \le i_1, i_2, \ldots, i_k} x_{i_1, i_2, \ldots, i_k} C_{p_1^{i_1}} C_{p_2^{i_2}} \ldots C_{p_k^{i_k}},$$

where the coefficients $x_{i_1, i_2, \ldots, i_k}$ belong to $\mathbb{N}$ (resp. $\mathbb{Q}$). The operations of addition and multiplication are well-defined over these sets.

It is clear that the semi-ring $\mathbb{N}[p_1, p_2, \ldots, p_k, p_{k+1}]$ can be viewed as $\mathbb{N}[p_1, p_2, \ldots, p_k][p_{k+1}]$.

Note that $\mathbb{N}[p_1]$ is a sub-semi-ring of $\mathbb{N}[p_1, p_2, \ldots, p_k]$, which in turn is a sub-semi-ring of $\mathbf{\Lambda}$. Moreover, any element of $\mathbf{\Lambda}$ belongs to some $\mathbb{N}[p_1, p_2, \ldots, p_k]$ for suitable primes $p_1, p_2, \ldots, p_k$.

Thus, instead of studying $\mathbf{\Lambda}$ directly, one can restrict the analysis to $\mathbb{N}[p_1, p_2, \ldots, p_k]$ and, in particular cases, to $\mathbb{N}[p]$.

3 Algorithm for Division Problem

We are interested in the following problem: given a sum of cycles B, what are its divisors? A specific case of this problem is the following: given two sums of cycles A and B, does A divide B? If so, what is (are) the quotient(s) of $\frac{B}{A}$? In other words, we seek the sum of cycles X such that $A \cdot X = B$.

We first provide a precise algorithm with time complexity $O(\log(N(B)))$ for the case $B \in \mathbb{N}[p]$, then present an algorithm for the general case.

3.1 Algorithm for Division with Single prime

Let B be a sum of cycles having length that is a power of the same prime p. For sort, we consider only the non-zero terms of B and write B as follows.

$$B = b_1 C_1 + b_{p^{\alpha_1}} C_{p^{\alpha_1}} + b_{p^{\alpha_2}} C_{p^{\alpha_2}} + \ldots + b_{p^{\alpha_k}} C_{p^{\alpha_k}},$$

with $0 = \alpha_0 < \alpha_1 < \alpha_2 < \ldots < \alpha_k$, and $b_{p^{\alpha_1}}, b_{p^{\alpha_2}}, \ldots, b_{p^{\alpha_k}}$ being positive integers and b_1 non negative integer.

First of all, we state the form of a divisor A of B.

Lemma 3.1 *If $B = A \cdot X$ then A and X must be of the following form:*

$$A = A_* + a_{p^{\alpha_1}} C_{p^{\alpha_1}} + a_{p^{\alpha_2}} C_{p^{\alpha_2}} + \ldots + a_{p^{\alpha_k}} C_{p^{\alpha_k}},$$

$$X = X_* + x_{p^{\alpha_1}} C_{p^{\alpha_1}} + x_{p^{\alpha_2}} C_{p^{\alpha_2}} + \ldots + x_{p^{\alpha_k}} C_{p^{\alpha_k}},$$

where $A_* = a_1 C_1 + a_p C_p + a_{p^2} C_{p^2} + \ldots + a_{p^{\alpha_1 - 1}} C_{p^{\alpha_1 - 1}}$ with $a_1, a_p, a_{p^2}, \ldots, a_{p^{\alpha_1 - 1}}$ being non-negative integers; and

$X_* = x_1 C_1 + x_p C_p + x_{p^2} C_{p^2} + \ldots + x_{p^{\alpha_1 - 1}} C_{p^{\alpha_1 - 1}}$ with $x_1, x_p, x_{p^2}, \ldots, x_{p^{\alpha_1 - 1}}$ being non-negative integers.

Moreover, if $b_1 \neq 0$ then $A_* = a_1 C_1$ and $X_* = x_1 C_1$ with $a_1 \neq 0, x_1 = \frac{b_1}{a_1}$. Otherwise $b_1 = 0$ then $A_* = 0$ or $X_* = 0$.

We call X a quotient of $\frac{B}{A}$, and there may exist different quotients X.

We keep the above notation for the remainder of this section.

We now consider the size of the above sums of cycles. Denote by a_* and x_* the size of A_* and X_* respectively,

$$a_* = a_1 + p a_p + p^2 a_{p^2} + \ldots + p^{\alpha_1 - 1} a_{p^{\alpha_1 - 1}},$$

$$x_* = x_1 + p x_p + p^2 x_{p^2} + \ldots + p^{\alpha_1 - 1} x_{p^{\alpha_1 - 1}}.$$

By using the Lemma 2.1 and the Corollary 2.2, we have:

Lemma 3.2 *If $B = A \cdot X$ then*

$$\begin{aligned}
&A_* X_* &&= b_1 C_1 \\
&(a_* + a_{p^{\alpha_1}} C_{p^{\alpha_1}})(x_* + x_{p^{\alpha_1}} C_{p^{\alpha_1}}) &&= b_1 + b_{p^{\alpha_1}} C_{p^{\alpha_1}} \\
&(a_* + a_{p^{\alpha_1}} C_{p^{\alpha_1}} + a_{p^{\alpha_2}} C_{p^{\alpha_2}})(x_* + x_{p^{\alpha_1}} C_{p^{\alpha_1}} + x_{p^{\alpha_2}} C_{p^{\alpha_2}}) &&= b_1 + b_{p^{\alpha_1}} C_{p^{\alpha_1}} + b_{p^{\alpha_2}} C_{p^{\alpha_2}} \\
&\dots \\
&(a_* + a_{p^{\alpha_1}} C_{p^{\alpha_1}} + \ldots + a_{p^{\alpha_k}} C_{p^{\alpha_k}})(x_* + x_{p^{\alpha_1}} C_{p^{\alpha_1}} + \ldots + x_{p^{\alpha_k}} C_{p^{\alpha_k}}) &&= b_1 + b_{p^{\alpha_1}} C_{p^{\alpha_1}} + \ldots + b_{p^{\alpha_k}} C_{p^{\alpha_k}}.
\end{aligned}$$

If $b_1 \neq 0$, then it follows that $a_* = a_1$ and $x_* = x_1$.
In all cases:

$$\begin{aligned}
&a_* x_* &&= b_1 \\
&(a_* + p^{\alpha_1} a_{p^{\alpha_1}})(x_* + p^{\alpha_1} x_{p^{\alpha_1}}) &&= b_1 + p^{\alpha_1} b_{p^{\alpha_1}} \\
&(a_* + p^{\alpha_1} a_{p^{\alpha_1}} + p^{\alpha_2} a_{p^{\alpha_2}})(x_* + p^{\alpha_1} x_{p^{\alpha_1}} + p^{\alpha_2} x_{p^{\alpha_2}}) &&= b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + p^{\alpha_2} b_{p^{\alpha_2}} \\
&\dots \\
&(a_* + p^{\alpha_1} a_{p^{\alpha_1}} + \ldots + p^{\alpha_k} a_{p^{\alpha_k}})(x_* + p^{\alpha_1} x_{p^{\alpha_1}} + \ldots + p^{\alpha_k} x_{p^{\alpha_k}}) &&= b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + \ldots + p^{\alpha_k} b_{p^{\alpha_k}}.
\end{aligned}$$

Or equivalently,

$$\begin{aligned}
a_* x_* &= b_1 \\
x_{p^{\alpha_1}} &= \frac{1}{p^{\alpha_1}} \left(\frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}}} - x_* \right) \\
x_{p^{\alpha_2}} &= \frac{1}{p^{\alpha_2}} \left(\frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + p^{\alpha_2} b_{p^{\alpha_2}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}} + p^{\alpha_2} a_{p^{\alpha_2}}} - \frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}}} \right) \\
&\dots \\
x_{p^{\alpha_i}} &= \frac{1}{p^{\alpha_i}} \left(\frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + \ldots + p^{\alpha_i} b_{p^{\alpha_i}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}} + \ldots + p^{\alpha_i} a_{p^{\alpha_i}}} - \frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + \ldots + p^{\alpha_{i-1}} b_{p^{\alpha_{i-1}}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}} + \ldots + p^{\alpha_{i-1}} a_{p^{\alpha_{i-1}}}} \right) \\
&\dots \\
x_{p^{\alpha_k}} &= \frac{1}{p^{\alpha_k}} \left(\frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + \ldots + p^{\alpha_k} b_{p^{\alpha_k}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}} + \ldots + p^{\alpha_k} a_{p^{\alpha_k}}} - \frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + \ldots + p^{\alpha_{k-1}} b_{p^{\alpha_{k-1}}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}} + \ldots + p^{\alpha_{k-1}} a_{p^{\alpha_{k-1}}}} \right).
\end{aligned}$$

Let us define Q_i for $i = 1, \ldots, k$,

$$Q_i = \frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + \ldots + p^{\alpha_i} b_{p^{\alpha_i}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}} + \ldots + p^{\alpha_i} a_{p^{\alpha_i}}}.$$

Then for $i = 2, \ldots, k$,

$$x_{p^{\alpha_i}} = \frac{Q_i - Q_{i-1}}{p^{\alpha_i}}.$$

Note that $Q_k = \frac{n(B)}{n(A)}$.

It is clear that for $i = 2, \dots, k$, $x_{p^{\alpha_i}}$ is uniquely determined. Moreover, it is non-negative if $Q_i \geq Q_{i-1}$, and it is an integer if $Q_i \equiv Q_{i-1} \mod p^{\alpha_i}$. Furthermore, $x_{p^{\alpha_1}}$ is an integer if $x_* \equiv Q_1 \mod p^{\alpha_1}$.

We can constate that these conditions is equivalent to the following condition, called Condition (*):

$$\forall i = 1, \dots, k: \quad Q_i \equiv Q_k \mod p^{\alpha_i+1}, \text{ and } Q_i \geq Q_{i-1} \text{ (except for } i = 1).$$

The quotient Q_i is called *suitable* if it satisfies condition (*) for i. The list $(Q_1, Q_2, \dots, Q_k)$ is called *suitable* if for all $i = 1, 2, \dots, k$, Q_i is suitable.

On the other hand, the variables x_{p^j}, for $0 \leq j \leq \alpha_1$, depend on $a_*, a_{p^{\alpha_1}}, b_1, b_{p^{\alpha_1}}$. Determining whether there is no solution, a unique solution, or multiple solutions can be done in $O(1)$ time.

Thus, to determine whether A divides B, it is essential to verify Condition (*), for each $i = 1, \dots, k$.

Algorithm for Division with Single prime
Input:

- The coefficients related to the first equation: a_*, b_1;
- The other coefficients$\{a_{p^{\alpha_i}}, b_{p^{\alpha_i}}\}$ and p^{α_i} for $i = 1, 2, \dots, k$.

Output:

- Solution X, if it exists, or a message stating that no solution exists.
- If solutions exist, return the number of solutions and list them.

1. **Step 0: Handle the condition of size**
   - Compute $Q_k = \frac{n(B)}{n(A)}$;
   - If Q_k is not integer then no solution.
   - Else Continue.
2. **Step 1: Compute all x_{p^j} for $0 \leq j \leq \alpha_1$**
   - If $b_1 \neq 0$: *(the solution is UNIQUE, if it exists.)*
     - If $a_* \neq a_1$ or $a_1 = 0$ then no solution.
     - If $a_* = a_1 \neq 0$ then Compute

$$x_* = x_1 = \frac{b_1}{a_1};$$

$$Q_1 = \frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}}}{a_1 + p^{\alpha_1} a_{p^{\alpha_1}}};$$

$$x_{p^{\alpha_1}} = \frac{1}{p^{\alpha_1}} (Q_1 - x_1);$$

     - If not ($x_1 \in \mathbb{N}$ and $x_{p^{\alpha_1}} \in \mathbb{N}$ and $Q_1 \equiv Q_k \mod p^{\alpha_2}$) then no solution.
       Else Proceed to Step 2.

- If $b_1 = 0$:
  - If $a_* \neq 0$ then $x_* = 0$, and Proceed to Step 2.
  - If $a_* = 0$ and $a_{p^{\alpha_1}} = 0$ then no solution.
  - If $a_* = 0$ and $a_{p^{\alpha_1}} \neq 0$ then compute $Q_1 = \frac{b_{p^{\alpha_1}}}{a_{p^{\alpha_1}}}$;
  - If $Q_1 \not\equiv Q_k \mod p^{\alpha_2}$ then no solution.
    Else Continue.
    - For the first variables x_{p^j} for $0 \leq j \leq \alpha_1$, solve the equation:

$$x_1 + px_p + p^2 x_{p^2} + \ldots + p^{\alpha_1 - 1} x_{p^{\alpha_1 - 1}} + x_{p^{\alpha_1}} p^{\alpha_1} = Q_1.$$

    - There are $s(Q_1, p, \alpha_1)$ solutions, where $s(n, p, k)$ is the number of solutions of the equation: $x_1 + px_p + p^2 x_{p^2} + \ldots + p^{k-1} x_{p^{k-1}} + x_{p^k} p^k = n$.
    - For other variables, Proceed to Step 2.

3. **Step 2: Compute all x_{p^i} for $i \geq 2$**

- For each $i = 2, \ldots, k-1$:
  - Compute

$$Q_i = \frac{b_1 + p^{\alpha_1} b_{p^{\alpha_1}} + \ldots + p^{\alpha_i} b_{p^{\alpha_i}}}{a_* + p^{\alpha_1} a_{p^{\alpha_1}} + \ldots + p^{\alpha_i} a_{p^{\alpha_i}}}.$$

  - Check condition (*): $Q_i \equiv Q_k \mod p^{\alpha_{i+1}}$, and $Q_i \geq Q_{i-1}$.
  - If the condition is (*) not satisfied then no solution. Else compute for each $i = 2, \ldots, k$:

$$x_{p^{\alpha_i}} = \frac{Q_i - Q_{i-1}}{p^{\alpha_i}}.$$

Theorem 3.3 *Let B be a sum of cycles with a single prime number and let A be a sum of cycles. There exists an algorithm with complexity of $O(card(B))$ (or equivalently, $O(log(N(B)))$ to determine if A divides B. This algorithm also describes all quotients $X = \frac{B}{A}$.*

Proof Let us consider Algorithm 3.1 with

$$B = b_1 C_1 + b_{p^{\alpha_1}} C_{p^{\alpha_1}} + b_{p^{\alpha_2}} C_{p^{\alpha_2}} + \ldots + b_{p^{\alpha_k}} C_{p^{\alpha_k}}.$$

- Step 1 take $O(1)$ time.
- Step 2: there are k iterations. At each iteration i, r_i and s_i can be computed from r_{i-1} and s_{i-1} by one addition, then Q_i can be computed in $O(1)$ time.
- Step 3: to describe all solutions, it take $O(1)$ time.

In total, the algorithm takes $O(k)$ time, where $k = card(B)$, and bounded by $log_p N(B)$.

Now, if we are interested to lists all solutions. There are $s\left(\frac{b_{p^{\alpha_1}}}{a_{p^{\alpha_1}}}, p, \alpha_1\right)$ solutions, where $s(n, p, k)$ is the number of solutions of the equation: $x_1 + px_p + p^2 x_{p^2} + \ldots + p^{k-1} x_{p^{k-1}} + x_{p^k} p^k = n$. The time to lists all solutions are $ks(n, p, k)$ time. □

Let us consider some examples to illustrate the algorithm.

Example 3.4 *Find solutions X for the following equation.*

$$AX = B$$

$$A = 2C_3 + C_9, \quad B = 80C_3 + 40C_9.$$

By Lemma 3.2, X is of the form $X = x_1C_1 + x_3C_3 + x_9C_9$, and satisfies the following system of equations:

$$\begin{aligned} x_1 + 3x_3 &= \frac{80}{2} = 40, \\ x_9 &= \frac{1}{9}\left(\frac{3 \times 80 + 9 \times 40}{3 \times 2 + 9} - \frac{800}{2}\right) = 0. \end{aligned}$$

So there exist 14 solutions (for x_3 be one integer from 0 to 13) satisfying $x_1 + 3x_3 = 40$, and $X = x_1C_1 + x_3C_3$.

Example 3.5 *Find solutions Y for the following equation.*

$$AY = B$$

$$A = 32 + 8C_3 + 4C_9, \quad B = 576 + 704C_3 + 192C_9.$$

By Lemma 3.2, Y is of the form $Y = y_1C_1 + y_3C_3 + y_9C_9$, and satisfies the following system of equations:

$$\begin{aligned} y_1 &= \frac{576}{32} = 18, \\ y_1 + 3y_3 &= \frac{576 + 3 \times 704}{32 + 8 \times 3} = 48, \\ y_9 &= \frac{1}{9}\left(\frac{576 + 3 \times 704 + 9 \times 192}{32 + 8 \times 3 + 4 \times 9} - 48\right) = 0. \end{aligned}$$

So there exists a unique solution satisfying $y_1 = 180$ and $y_3 = 10$, and $Y = 18C_1 + 10C_3$.

3.2 Algorithm for Division in general case

We now consider the general case where B is an arbitrary sum of cycles. This means that, in the lengths of the cycles of B, several prime numbers may appear. The main idea is to extend the results from the previous section by considering a ring with $k + 1$ variables, and then, by recursion, treat it as a ring with one variable, where the coefficients are elements of a ring with k variables.

We state the main result of this section.

Theorem 3.6 *Let B be a general sum of cycles. Let A be a sum of cycles. There exists an algorithm to determine whether A is a divisor of B. In the affirmative case, the algorithm also describes all quotients $X\frac{B}{A}$.*

Proof We prove this theorem by induction on the number m of prime numbers appearing in the cycles of B.

- If $m = 0$, then $B = b_1C_1$, and the only possible solution is $A = a_1C_1$ and $X = x_1C_1$, where $a_1x_1 = b_1$.
- If $m = 1$, then the problem reduces to the case analyzed in Subsection 3.1, which has already been proven correct.
- Assume that the theorem holds for $m = k \geq 1$. We will prove its validity for $m = k + 1$. Let $p_1, p_2, \ldots, p_k, p$ be these prime numbers.

  We can write any $C_{p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k} p^{\alpha}}$ as $C_{p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}} C_{p^{\alpha}}$.

  And we can write $B = b_0C_1 + b_{p^{\alpha_1}}C_{p^{\alpha_1}} + b_{p^{\alpha_2}}C_{p^{\alpha_2}} + \ldots + b_{p^{\alpha_k}}C_{p^{\alpha_k}}$, with $b_i \in \mathbb{N}[p_1, \ldots, p_k]$.

  We return to Lemma 3.1: A and X must be of the following form:

$$A = A_* + a_{p^{\alpha_1}}C_{p^{\alpha_1}} + a_{p^{\alpha_2}}C_{p^{\alpha_2}} + \ldots + a_{p^{\alpha_k}}C_{p^{\alpha_k}},$$

$$X = X_* + x_{p^{\alpha_1}}C_{p^{\alpha_1}} + x_{p^{\alpha_2}}C_{p^{\alpha_2}} + \ldots + x_{p^{\alpha_k}}C_{p^{\alpha_k}},$$

  where

$$A_* = a_1C_1 + a_pC_p + a_{p^2}C_{p^2} + \ldots + a_{p^{\alpha_1-1}}C_{p^{\alpha_1-1}},$$
$$X_* = x_1C_1 + x_pC_p + x_{p^2}C_{p^2} + \ldots + x_{p^{\alpha_1-1}}C_{p^{\alpha_1-1}}$$

  Here all $b_i, a_i, x_i \in \mathbb{N}[p_1, p_2, \ldots, p_k]$ for all indices i.

  By Lemma 3.2, we have the following system:

$$\begin{aligned}
a_*x_* &= b_1\\
x_{p^{\alpha_1}} &= \frac{1}{p^{\alpha_1}}\left(\frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}}} - x_*\right)\\
x_{p^{\alpha_2}} &= \frac{1}{p^{\alpha_2}}\left(\frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}} + p^{\alpha_2}b_{p^{\alpha_2}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}} + p^{\alpha_2}a_{p^{\alpha_2}}} - \frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}}}\right)\\
&\ldots\\
x_{p^{\alpha_i}} &= \frac{1}{p^{\alpha_i}}\left(\frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}} + \ldots + p^{\alpha_i}b_{p^{\alpha_i}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}} + \ldots + p^{\alpha_i}a_{p^{\alpha_i}}} - \frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}} + \ldots + p^{\alpha_{i-1}}b_{p^{\alpha_{i-1}}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}} + \ldots + p^{\alpha_{i-1}}a_{p^{\alpha_{i-1}}}}\right)\\
&\ldots\\
x_{p^{\alpha_k}} &= \frac{1}{p^{\alpha_k}}\left(\frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}} + \ldots + p^{\alpha_k}b_{p^{\alpha_k}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}} + \ldots + p^{\alpha_k}a_{p^{\alpha_k}}} - \frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}} + \ldots + p^{\alpha_{k-1}}b_{p^{\alpha_{k-1}}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}} + \ldots + p^{\alpha_{k-1}}a_{p^{\alpha_{k-1}}}}\right).
\end{aligned}$$

  The challenge of the problem lies in the fact that each equation involves a division in $\mathbb{N}[p_1, p_2, \ldots, p_k]$, which can yield different solutions.

  Our objective is to find all suitable lists Q_i in $\mathbb{N}[p_1, p_2, \ldots, p_k]$.

  To verify suitable lists, we use the condition described in Equation 1, which states:

$$Q_{i-1} = \frac{b_1 + p^{\alpha_1}b_{\alpha_1} + \ldots + p^{\alpha_{i-1}}b_{\alpha_{i-1}}}{a_* + p^{\alpha_1}a_{\alpha_1} + \ldots + p^{\alpha_{i-1}}a_{\alpha_{i-1}}} \equiv \frac{n(B)}{n(A)} \mod p^{\alpha_i}, \text{ for all } i = 2, \ldots, k.$$

The first step is to use an algorithm for k primes to compute $Q_k = \frac{n(B)}{n(A)}$, with $Q_k \in \mathbb{N}[p_1, p_2, \ldots, p_k]$. If such a Q_k does not exist, then the equation has no solution, and we stop.

Otherwise, we denote by S_k the set of all solutions Q_k.

Next, for each i from $k-1$ down to 1, we compute

$$Q_i = \frac{b_1 + p^{\alpha_1} b_{\alpha_1} + \ldots + p^{\alpha_i} b_{\alpha_i}}{a_* + p^{\alpha_1} a_{\alpha_1} + \ldots + p^{\alpha_i} a_{\alpha_i}}.$$

If such a Q_i does not exist, the equation has no solution, and we stop.

If multiple Q_i exist, for each Q_i, we check the suitability condition. This means that there exists a Q_k in S_k such that $Q_i \equiv Q_k \mod p^{\alpha_i+1}$ and there exists Q_{i+1} marked by Q_k such that $Q_i \leq Q_{i+1}$. In this case, we mark Q_i by Q_k and retain Q_k in S_k. If Q_k is not compatible with any Q_i, we remove Q_k from S_k.

By doing this, each step for i provides us with a suitable solution for Q_i marked by Q_k.

After iterating through all steps for i down to 1, if S_k remains non-empty, it means that a suitable list $(Q_1, Q_2, \ldots, Q_k)$ exists, and we have a solution.

Furthermore, in this manner, we obtain all Q_i marked by Q_k as solutions to the system. Therefore, we find all possible solutions to the equation $A \times X = B$.

□

Example 3.7 *Find all X such that $A \cdot X =$ with*
$B = 72C_8 + 80C_{12} + 48C_{24} + 40C_{36} + 4C_{72}$
and $A = 4C_8 + 2C_{12} + C_{36}$.
We consider $p = 3, q = 2$, *and* $B \in \mathbb{N}[3][2]$.
So we can write B and A as:

$$B = (80C_3 + 40C_9)C_{2^2} + (72 + 48C_3 + 4C_9)C_{2^3},$$
$$A = (2C_3 + C_9)C_{2^2} + 4C_{2^3}.$$

Then X must be of the form:

$$X = y_1C_1 + y_2C_2 + y_4C_{2^2} + y_8C_{2^3},$$

with $y_1, y_2, y_4, y_8 \in \mathbb{N}[3]$.
Apply the Algorithm in Theorem 3.3, we have:

$$\begin{aligned}
y_1 + 2y_2 + 4y_4 &= \frac{80C_3 + 40C_9}{2C_3 + C_9}, \\
y_8 &= \frac{1}{8}\left(\frac{(80C_3 + 40C_9) * 4 + (72 + 48C_3 + 4C_9) * 8}{(2C_3 + C_9) * 4 + 4 * 8} - \frac{80C_3 + 4C_9}{2C_3 + C_9}\right) \\
&= \frac{1}{8}\left(\frac{576 + 704C_3 + 192C_9}{32 + 8C_3 + 4C_9} - \frac{80C_3 + 40C_9}{2C_3 + C_9}\right)
\end{aligned}$$

We first compute $C_k = \frac{n(B)}{n(A)}$. By using Example 2.6, we have:

$$\frac{576 + 704C_3 + 192C_9}{32 + 8C_3 + 4C_9} = 18C_1 + 10C_3,$$

Then, we treat the first equation. By using Example 2.5, we have:

$$y_1 + 2y_2 + 4y_4 = \frac{80C_3 + 40C_9}{2C_3 + C_9} = r_1C_1 + r_3C_3 \quad with \quad r_1 + 3r_3 = 40.$$

Now, compute y_8 by

$$y_8 = \frac{1}{8}(18C_1 + 10C_3 - (r_1 + r_3C_3)) = \frac{18 - r_1}{8} + \frac{10 - r_3}{8}C_3 \quad with \quad r_1 + 3r_3 = 40.$$

Since all coefficients of y_8 are non-negative integers, r_3 can only take the value 10. Hence, $r_1 = 10$ and $y_8 = 1$.

Now, $y_1 + 2y_2 + 4y_4 = 10C_1 + 10C_3$.

There are then $f(10, 2, 2)$ solutions for the coefficient of C_1, and the same for the coefficient of C_3. The number of solutions for X is equal to $f^2(10, 2, 2)$. It is easy to check that $f(10, 2, 2) = 12$, so there are 144 solutions.

For example, let us consider $y_4 = C_1 + C_3$, $y_2 = C_1$, then $y_1 = 4C_1 + 6C_3$. In this case:

$$X = 4C_1 + 6C_3 + C_2 + C_4 + C_{12} + C_8.$$

A second solution: if $y_4 = 2C_1 + C_3$, $y_2 = 2C_3$, then $y_1 = 2C_1 + 2C_3$. In this case:

$$X = 4C_1 + 2C_3 + 2C_6 + 2C_4 + C_8.$$

A third solution: if $y_4 = 2C_1$, $y_2 = 5C_3$, then $y_1 = 2C_1$. In this case:

$$X = 2C_1 + 5C_6 + 2C_4 + C_8.$$

and so on...

Algorithm for Division in the general case

Input: - A and B: Two general sums of cycles.

Output: - Solution X, if it exists, or a message stating that no solution exists. - If solutions exist, return the number of solutions and list them.

Algorithm for m primes

1. Determine the ring of polynomials.
   - Let $p_1, \dots, p_m$ be the primes appearing in the length of cycles of B.
   - Write B and A in $\mathbb{N}[p_1, p_2, \dots, p_m]$.
   - If $m = 0$ or $m = 1$, apply the algorithm for the single case.
   - Else, continue to Step 2.
2. If $m \geq 2$ and Algorithm for $m - 1$ primes has already been solved.
   - Let $k = m - 1$, and express B and A in $\mathbb{N}[p_1, p_2, \dots, p_k][p]$ with $p = p_m$.

- Object: Find the solution X such that: $B = A \times X$, with

$$B = b_0C_1 + b_{p^{\alpha_1}}C_{p^{\alpha_1}} + b_{p^{\alpha_2}}C_{p^{\alpha_2}} + \cdots + b_{p^{\alpha_k}}C_{p^{\alpha_k}},$$
$$A = A_* + a_{p^{\alpha_1}}C_{p^{\alpha_1}} + a_{p^{\alpha_2}}C_{p^{\alpha_2}} + \cdots + a_{p^{\alpha_k}}C_{p^{\alpha_k}},$$
$$X = X_* + x_{p^{\alpha_1}}C_{p^{\alpha_1}} + x_{p^{\alpha_2}}C_{p^{\alpha_2}} + \cdots + x_{p^{\alpha_k}}C_{p^{\alpha_k}},$$

where all $b_i, a_i, x_i \in \mathbb{N}[p_1, p_2, \ldots, p_k]$ for all i.

(a) **Step 2.1: Handle the total polynomials.**
  - Compute the quotient $Q_k = \frac{n(B)}{n(A)}$ in $\mathbb{N}[p_1, p_2, \ldots, p_k]$, using Algorithm for $m-1$ primes. Let S_k be the set of all quotients Q_k.

$$Q_k = \frac{n(B)}{n(A)} = \frac{b_1 + p^{\alpha_1}b_{\alpha_1} + \cdots + p^{\alpha_k}b_{\alpha_k}}{a_* + p^{\alpha_1}a_{\alpha_1} + \cdots + p^{\alpha_k}a_{\alpha_k}}.$$

  - If $S_k = \emptyset$ then No solution. Else Continue

(b) **Step 2.2:**
  - For each $i = k-1, k-2, \ldots, 2, 1$, let S_i be the set of all quotients Q_i.

$$Q_i = \frac{b_1 + p^{\alpha_1}b_{p^{\alpha_1}} + \cdots + p^{\alpha_i}b_{p^{\alpha_i}}}{a_* + p^{\alpha_1}a_{p^{\alpha_1}} + \cdots + p^{\alpha_i}a_{p^{\alpha_i}}}.$$

  - Check the suitable conditions (*) for each $Q_i \in S_i$:
    - There exists Q_k in S_k such that $Q_i \equiv Q_k \mod p^{\alpha_{i+1}}$, and
    - There exists Q_{i+1} marked by Q_k such that $Q_i \leq Q_{i+1}$.
  - If Q_i satisfies these two conditions (*) then mark Q_i by Q_k; otherwise delete Q_i from S_i.
  - Delete all elements in S_k that are not used to make any Q_i in this step.
  - If $S_k = \emptyset$ then No solution. Else Continue

(c) **Step 2.3:** After Step 2.2, we obtain S_1, which contains all Q_1 that have a suitable list $(Q_1, Q_2, Q_3, \ldots, Q_{k-1}, Q_k)$.
  - For each suitable list $(Q_1, Q_2, Q_3, \ldots, Q_{k-1}, Q_k)$, we have a solution:

$$\forall i \in \{2, \ldots, k\}, \quad x_{p^{\alpha_i}} = \frac{Q_i - Q_{i-1}}{p^{\alpha_i}},$$

    where these values belong to $\mathbb{N}[p_1, p_2, \ldots, p_k]$.
  - **Handle the first equation.**
    - Check the conditions for a_1, a_*, b_1 as in the case of a single prime.
    - In the case where $b_1 = 0$ and $a_* = 0$, there may be multiple solutions for the first equation.
      The solution set S_* of possible tuples $(x_1, x_p, x_{p^2}, \ldots, x_{p^{\alpha_1}})$ contains $s(Q_1, p, \alpha_1)$ solutions of the equation:

$$x_1 + px_p + p^2x_{p^2} + \ldots + p^{k-1}x_{p^{k-1}} + x_{p^k}p^{\alpha_1} = Q_1.$$

Thus, using the above theorem, we have an algorithm to determine if a A divides a B for two given general sums of cycles A and B.

3.3 Discussion on the Complexity of Algorithms

In the case where $B \in \mathbb{N}[p]$, Theorem 3.3 shows that the time complexity for determining whether A divides B is $O(\log(N(B))$. Moreover, the algorithm also provides a description of all possible quotients $X = \frac{B}{A}$. However, if one wishes to explicitly list all quotients X, the process requires linear time with respect to the number of solutions. This number is given by

$$s\left(\frac{b_{p^{\alpha_1}}}{a_{p^{\alpha_1}}}, p, \alpha_1\right),$$

where $s(n, p, k)$ denotes the number of solutions to the equation

$$x_1 + px_p + p^2 x_{p^2} + \cdots + p^{k-1} x_{p^{k-1}} + x_{p^k} p^k = n.$$

In the general case where $B \in \mathbb{N}[p_1, \ldots, p_m]$, the situation becomes more complex. To determine whether A divides B, one must compute (not merely describe) the quotients resulting from multiple divisions of sums of cycles. Therefore, before delving into the number of divisions required, we first analyze the function $s(n, p, k)$.

3.3.1 Evaluation of the function $s(n, p, k)$

Let $s(n, p, k)$ be the number of solutions to the equation:

$$x_1 + px_p + p^2 x_{p^2} + \ldots + p^{k-1} x_{p^{k-1}} + x_{p^k} p^k = n.$$

Fixing p, we estimate $s(n, p, k)$ by recurrence on k.

First, it is straightforward to observe that this function is monotonically increasing as n increases. Thus, we can estimate $s(n, p, k)$ for values of n that are multiples of p^k and then derive bounds for other values of n between two consecutive multiples of p^k.

Now, consider $n = m \times p^k$, and define $g(m, k) = s(m \cdot p^k, p, k)$.

Then $g(m, k)$ is the number of solutions to the equation:

$$x_1 + px_p + p^2 x_{p^2} + \ldots + p^{k-1} x_{p^{k-1}} + x_{p^k} p^k = m \cdot p^k.$$

To find $g(m, k)$, we assign integer values to x_{p^k} from m down to 0. If $x_{p^k} = i$, then the equation becomes:

$$x_1 + px_p + p^2 x_{p^2} + \ldots + p^{k-1} x_{p^{k-1}} = n - i \cdot p^k = (m - i) \cdot p^k,$$

which has $g((m - i)p, k - 1)$ solutions.

Thus, we obtain the following recurrence formula:

$$g(m, k) = g(0, k - 1) + g(p, k - 1) + g(2p, k - 1) + \ldots + g(mp, k - 1).$$

By performing an induction on k for evaluating g and s, we can prove the following

Theorem 3.8 *The value $s(n, p, k)$, representing the number of solutions to the equation*

$$x_1 + px_p + p^2 x_{p^2} + \ldots + p^{k-1} x_{p^{k-1}} + x_{p^k} p^k = n,$$

is bounded by

$$s(n, p, k) \leq \frac{n^k}{p^{\frac{k^2}{2}} k!}.$$

3.3.2 Discussion on the Complexity of the Algorithm in the General Case

The most computationally demanding step in Algorithm 3.2 is Step 2.2, where we identify all suitable tuples $(Q_1, Q_2, \ldots, Q_i)$. The primary cost of the algorithm arises from the evaluation of all possible solutions for each division step in the process.

In certain special cases, where the function $s(n, p, k)$ appears frequently, the number of possible solutions can increase significantly, leading to a high computational cost. However, this is an inherent characteristic of the problem, as the solution space itself can be extremely large. For example, if $B = bC_n$ and $A = C_n$, where $n = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$ and b are very large, the number of solutions corresponds to the number of ways to partition b into powers of p_i under certain conditions, an inherently large number.

However, such special cases are relatively rare. In most practical scenarios, the algorithm runs efficiently, often with linear or even logarithmic complexity in $N(B)$. This efficiency is particularly evident when B consists of cycles of varying lengths or when these cycle lengths are associated with different prime factors.

In the next subsection, we present examples to illustrate how the algorithm performs in typical cases. More importantly, in the following section on finding roots of polynomials, we apply a similar idea to solve the problem efficiently.

3.4 Application

Consider the case where A is composed of cycles of prime sizes. For example:

Example 3.9

$$A = C_2 + C_3, \quad B = 160C_2 + 20C_4 + 40C_3 + 560C_6 + 10C_{12} + 30C_9 + 10C_{18}.$$

We apply the algorithm 3.2 by writing:

$$X = X_1 + X_3C_3 + X_9C_9, \quad \textit{where } X_i \in \mathbb{N}[2].$$

From the coefficients of C_2:

$$X_1 = \frac{160C_2 + 20C_4}{C_2} = C_4 + aC_2 + (160 - 2a)C_1, \quad \textit{where } 0 \le a \le 80.$$

From the coefficients of C_3:

$$X_1 + 3X_3 = \frac{160C_2 + 20C_4 + 120C_1 + 1680C_2 + 30C_4}{C_3 + C_2} = 10C_4 + 360C_2 + 40C_1.$$

From the coefficients of C_9:

$$X_1 + 3X_3 + 9X_9 = \frac{120C_1 + 1840C_2 + 50C_4 + 270C_1 + 90C_2}{C_9 + C_3 + C_2} = 10C_4 + 360C_2 + 130C_1.$$

From these equations, we deduce:

$$X_9 = 10C_1,$$

$$X_1 = 10C_4 + aC_2 + (160 - 2a)C_1, \quad \textit{where } 60 \le a \le 80 \textit{ and } a \textit{ is a multiple of 3},$$

$$X_3 = \frac{360 - a}{3}C_2 + \frac{2a - 120}{3}C_1.$$

The possible values of a are:

$$a = 60, 63, 66, 69, 72, 75, 78.$$

Examples of solutions include:

- *For $a = 60$: $X = 40C_1 + 60C_2 + 10C_4 + 100C_6 + 10C_9$.*
- *For $a = 63$: $X = 34C_1 + 63C_2 + 10C_4 + 2C_3 + 99C_6 + 10C_9$.*
- *For $a = 78$: $X = 4C_1 + 78C_2 + 10C_4 + 12C_3 + 94C_6 + 10C_9$.*

We observe that for $A, B \in \mathbb{N}[p_1, p_2, \dots, p_k][p]$ and $X_1, X_p, \dots, X_{p^\alpha} \in \mathbb{N}[p_1, \dots, p_k]$, all coefficients X_i with $i \geq 2$ are uniquely determined. Only X_1 and X_p remain undetermined initially. However, X_p is uniquely determined once X_1 is fixed. Hence, the solution depends entirely on determining X_1.

By recursion, $X_1 \in \mathbb{N}[p_1, \dots, p_{k-1}][p_k]$ is uniquely determined by the first coefficient of the decomposition.

Moreover, if

$$(x_1 C_1 + p x_p) a_p C_p = b_p C_p,$$

then

$$x_1 = \frac{b_p}{a_p} - p x_p.$$

Thus, we have two constraints on x_1:

$$X_1 \leq \min \left(\frac{b_{p_i}}{a_{p_i}} \right),$$

since X_1 must be non-negative for all p_i.

The modular properties also determine $x_1 \mod (p_1 p_2 \dots p_m)$, as:

$$x_1 \equiv \frac{b_{p_i}}{a_{p_i}} \mod p_i \quad \text{for each}\, p_i.$$

Combining these constraints, the modular structure determines $x_1 \mod (p_1 p_2 \dots p_m)$. The total number of possible solutions for x_1 (and for X by consequence) is bounded by:

$$\frac{\min\left(\frac{b_{p_i}}{a_{p_i}}\right)}{p_1 p_2 \dots p_m}.$$

And the time for calculation is linear on the number of solution, so smaller than

$$\frac{\min\left(\frac{b_{p_i}}{a_{p_i}}\right)}{p_1 p_2 \dots p_m} + card(B).$$

4 Algorithm for Finding the roots of Polynomial

4.1 Finding the m-root of a sum of cycles

A fundamental problem in the study of the semi-ring of functional digraphs is determining the roots of its elements. Specifically, given a sum of cycles B and an integer m, we seek to determine whether there exists a sum of cycles A such that $A^m = B$.

It has been proven in [12] that if such an A exists, it is unique. In this section, we present an efficient algorithm for computing A. The proposed algorithm achieves optimal time complexity, as its runtime scales linearly with the number of terms that need to be computed.

We apply the idea of the general algorithm 3.2 to handle it.

First, let us consider the case of elements in $\mathbb{N}[p]$. By applying Lemma 2.1 and Corollary 2.2, we have:

Lemma 4.1 *Let p be a prime number. In $\mathbb{N}[p]$, let*

$$A = a_1C_1 + a_{p^{\alpha_1}}C_{p^{\alpha_1}} + a_{p^{\alpha_2}}C_{p^{\alpha_2}} + \ldots + a_{p^{\alpha_k}}C_{p^{\alpha_k}}.$$

For a positive integer m, let $B = A^m$. Then

$$B = b_1C_1 + b_{p^{\alpha_1}}C_{p^{\alpha_1}} + b_{p^{\alpha_2}}C_{p^{\alpha_2}} + \ldots + b_{p^{\alpha_k}}C_{p^{\alpha_k}},$$

where b_i, for $i = 0, \ldots, k$, are uniquely determined by

$$\begin{aligned}
&b_1 &&= a_1^m, \\
&b_1 + b_{p^{\alpha_1}}p^{\alpha_1} &&= (a_1 + a_{p^{\alpha_1}}p^{\alpha_1})^m, \\
&\vdots \\
&b_1 + b_{p^{\alpha_1}}p^{\alpha_1} + b_{p^{\alpha_2}}p^{\alpha_2} + \ldots + b_{p^{\alpha_k}}p^{\alpha_k} &&= (a_1 + a_{p^{\alpha_1}}p^{\alpha_1} + a_{p^{\alpha_2}}p^{\alpha_2} + \ldots + a_{p^{\alpha_k}}p^{\alpha_k})^m.
\end{aligned}$$

Next, for $i = 0, 1, \ldots, k$, we denote r_i and s_i the size prefix sum of A and B respectively:

$$r_i = a_1 + a_{p^{\alpha_1}}p^{\alpha_1} + a_{p^{\alpha_2}}p^{\alpha_2} + \cdots + a_{p^{\alpha_i}}p^{\alpha_i}.$$

$$s_i = b_1 + b_{p^{\alpha_1}}p^{\alpha_1} + b_{p^{\alpha_2}}p^{\alpha_2} + \cdots + b_{p^{\alpha_i}}p^{\alpha_i}.$$

Then, the above lemma can be rewritten as follows:

Lemma 4.2 *Keeping the above notation, if $B = A^m$, then*

$$s_i = r_i^{\cdot} \quad \textit{for } i = 0, 1, \ldots, k.$$

From this lemma, we can construct an algorithm to compute the m-root of B, if it exists.
Algorithm: Finding the m-root of a sum of cycles with a single prime
Input: An element $B \in \mathbb{N}[p]$ and a positive integer m.
Output: An element $A \in \mathbb{R}[p]$ such that $A^m = B$. If $A \in \mathbb{N}[p]$, then A is a sum of cycles m-root of B.

1. Express B as:
$$B = b_1C_1 + b_{p^{\alpha_1}}C_{p^{\alpha_1}} + b_{p^{\alpha_2}}C_{p^{\alpha_2}} + \cdots + b_{p^{\alpha_k}}C_{p^{\alpha_k}}.$$

2. Initialize $\alpha_0 = 0; r_{-1} = 0$ and $s_{-1} = 0$.
3. For $i = 0$ to k, do:
   (a) Update:
   $$s_i = s_{i-1} + b_{p^{\alpha_i}} p^{\alpha_i};$$
   (b) Compute:
   $$r_i = s_i^{\frac{1}{m}};$$
   (c) Compute:
   $$a_{p^{\alpha_i}} = \frac{r_i - r_{i-1}}{p^{\alpha_i}}.$$
4. If all $a_i \in \mathbb{N}$, then A is a sum of cycles m-root of B.

Example 4.3 *Let $B = 2C_2 + 48C_4 + 540C_{16}$ and $m = 2$. We apply the above algorithm to find $A = B^{\frac{1}{2}}$.*

1. *Initialize: $r_{-1} = 0$, $s_{-1} = 0$.*
2. *Compute:*
$$s_0 = s_{-1} + b_0 = 0, \quad r_0 = 0, \quad a_{2^0} = 0 \quad (a_1 = 0).$$
3. *Compute:*
$$s_1 = s_0 + b_{2^1} \cdot 2^1 = 0 + 2 \cdot 2 = 4;$$
$$r_1 = \sqrt{4} = 2;$$
$$a_2 = a_{2^1} = \frac{r_1 - r_0}{2^1} = \frac{2-0}{2} = 1.$$
4. *Compute:*
$$s_2 = s_1 + b_{2^2} \cdot 2^2 = 4 + 48 \cdot 4 = 196;$$
$$r_2 = \sqrt{196} = 14;$$
$$a_4 = a_{2^2} = \frac{r_2 - r_1}{2^2} = \frac{14-2}{4} = 3.$$
5. *Compute:*
$$s_3 = s_2 + b_{2^4} \cdot 2^4 = 196 + 540 \cdot 16 = 8836;$$
$$r_3 = \sqrt{8836} = 94;$$
$$a_{16} = a_{2^4} = \frac{r_3 - r_2}{2^4} = \frac{94-14}{16} = 5.$$
6. *Conclusion:*
$$A = C_2 + 3C_4 + 5C_{16} \in \mathbb{N}[2],$$
which is a 2-root of B.

Let us now consider the general case where there are several primes in B. We can apply the idea from the previous section and combine it with the single-prime case to construct the general algorithm.

Let $p_1, p_2, \ldots, p_k$ be k different prime numbers. The semiring $\mathbb{N}[p_1, p_2, \ldots, p_k, p_{k+1}]$ can be seen as $\mathbb{N}[p_1, p_2, \ldots, p_k][p]$ with $p = p_{k+1}$.

We consider the above algorithm to find the root of a sum of cycles, where the coefficients are elements of $\mathbb{N}[p_1, p_2, \ldots, p_k]$. All computations of s_i and $a_{p^{\alpha_i}}$ are simple on real numbers; only the computation $r_i = s_i^{\frac{1}{m}}$ remains to be discussed. However, this is nothing but the problem of finding the m-root of s_i in the semiring $\mathbb{N}[p_1, p_2, \ldots, p_k]$.

Thus, the algorithm for $k+1$ primes can be constructed recursively from the algorithm for k primes.

Example 4.4 *Let $B = 2C_2 + 48C_4 + 170C_6 + 60C_{12}$. Find the 2-root of B if it exists.*

There are two primes, 2 and 3 in B, so we write:

$$B = (2C_1 + 170C_3)C_2 + (48C_1 + 60C_3)C_4.$$

We seek A such that $A^2 = B$.

Firstly, write A as:

$$A = y_2C_2 + y_4C_4, \quad \text{with } y_2, y_4 \in \mathbb{N}[3].$$

By applying the above algorithm, we obtain:

$$2y_2 = \sqrt{2(2C_1 + 170C_3)};$$

$$2y_2 + 4y_4 = \sqrt{2(2C_1 + 170C_3) + 4(48C_1 + 60C_3)}.$$

To find y_2, we apply the algorithm for $\mathbb{N}[3]$. We write:

$$2y_2 = x_1C_1 + x_3C_3.$$

Then:

$$x_1C_1 + x_3C_3 = \sqrt{4C_1 + 340C_3}.$$

This implies:

$$x_1 = \sqrt{4} = 2;$$

$$x_1 + 3x_3 = \sqrt{4 + 340 \cdot 3} \Rightarrow x_3 = 10.$$

Similarly, we write:

$$2y_2 + 4y_4 = z_1C_1 + z_3C_3 = \sqrt{2(2C_1 + 170C_3) + 4(48C_1 + 60C_3)} = \sqrt{196C_1 + 580C_3}.$$

This implies:

$$z_1 = \sqrt{196} = 14;$$

$$z_1 + 3z_3 = \sqrt{196 + 3 \cdot 580} = 44 \Rightarrow z_3 = 10.$$

At the end, we find:

$$y_2 = C_1 + 5C_3, \quad y_4 = 3C_1.$$

Therefore:

$$A = y_2C_2 + y_4C_4 = (C_1 + 5C_3)C_2 + (3C_1)C_4 = C_2 + 5C_6 + 3C_4,$$

which is a 2-root of B.

4.2 Algorithm for Finding the Root of a Polynomial

Now let us consider P a polynomial with coefficients over $\mathbb{N}$:

$$P(X) = c_0 + c_1X + c_2X^2 + \cdots + c_mX^m.$$

We consider the following problem: given a sum of cycles B, determine whether there exists a sum of cycles A such that $P(A) = B$.

One can observe that this is a generalization of the problem in Subsection 4.1, by taking $P(X) = X^m$. The algorithm for this problem is systematically similar to that in the previous section.

Let us first recall the root of P in $\mathbb{N}$.

Lemma 4.5 *Let $b \in \mathbb{N}$ be a natural number. If $b > c_0$, then the equation $P(x) = b$ has a unique solution in $\mathbb{R}^+$. Moreover, if there exists an integer solution $x \in \mathbb{N}$, it can be found in time complexity $O\left(\frac{\log b}{m}\right)$.*

Proof The polynomial $P(x)$ is an increasing function over $\mathbb{R}^+$ because all its coefficients are non-negative. Moreover, since $P(0) = c_0$ and $P(x) \to \infty$ as $x \to \infty$, the equation $P(x) = b \geq c_0$ has a unique solution in $\mathbb{R}^+$.

Now, to find a non-negative integer a such that $P(a) = b$, we can apply a binary search approach: 1. Choose two values x_1 and x_2 such that $P(x_1) < b < P(x_2)$. 2. Compute $x_3 = \left\lfloor \frac{x_1+x_2}{2} \right\rfloor$, then compare $P(x_3)$ with b and continue iterating accordingly. 3. This process continues until we either find an integer a such that $P(a) = b$ or conclude that such an integer a does not exist.

Since $P(x)$ is a polynomial of degree m and the algorithm is based on binary search, the complexity is $O(\log(b^{\frac{1}{m}})) = O\left(\frac{\log b}{m}\right)$. □

We can now formally state the result for the prefix sum of a polynomial.

Lemma 4.6 *Let P be a polynomial with coefficients in $\mathbb{N}$. In $\mathbb{N}[p]$, if $B = P(A)$ then $s_i = P(r_i)$, where r_i and s_i the prefix sums of A and B, respectively.*

From this lemma, we can construct an algorithm for finding the Root of a Polynomial.
Algorithm: Finding the Root of a Polynomial with a Single Prime
Input: An element $B \in \mathbb{N}[p]$ and a polynomial $P \in \mathbb{N}[X]$.
Output: An element $A \in \mathbb{N}[p]$ such that $P(A) = B$, if it exists.

1. Express B as:
$$B = b_1C_1 + b_{p^{\alpha_1}}C_{p^{\alpha_1}} + b_{p^{\alpha_2}}C_{p^{\alpha_2}} + \cdots + b_{p^{\alpha_k}}C_{p^{\alpha_k}}.$$
2. Initialize $\alpha_0 = 0$, $r_{-1} = 0$, and $s_{-1} = 0$.
3. For $i = 0$ to k, do:
   (a) Update:
   $$s_i = s_{i-1} + b_{p^{\alpha_i}}p^{\alpha_i}.$$
   (b) Find r_i, the natural root of the equation $P(x) = s_i$. If no such r_i exists: Stop. No solution. Otherwise, continue.

(c) Compute:

$$a_{p^{\alpha_i}} = \frac{r_i - r_{i-1}}{p^{\alpha_i}}.$$

If $a_{p^{\alpha_i}} \notin \mathbb{N}$: Stop. No solution. Otherwise, continue.

4. If all $a_{p^{\alpha_i}} \in \mathbb{N}$, then A is a sum of cycles and a root of $P(X) = B$.

Next, for the general case of a sum of cycles with multiple primes, we apply the algorithm recursively. This approach is efficient because, unlike the case of divisors, the root of each equation is unique if it exists.

Example 4.7 *Let $P(x) = x^2 + 7x$, and consider the element*

$$B = 9C_2 + 69C_4 + 205C_6 + 60C_{12}.$$

We seek a solution to the equation $P(A) = B$.

Since B involves two primes, 2 and 3, we rewrite it as:

$$B = (9C_1 + 205C_3)C_2 + (69C_1 + 60C_3)C_4.$$

We aim to find A such that $P(A) = B$.

First, express A as:

$$A = y_2C_2 + y_4C_4, \quad \text{where } y_2, y_4 \in \mathbb{N}[3].$$

Applying the algorithm, we obtain:

$$P(2y_2) = 2(9C_1 + 205C_3),$$

$$P(2y_2 + 4y_4) = 2(9C_1 + 205C_3) + 4(69C_1 + 60C_3),$$

To determine y_2, y_4, we apply the algorithm for $\mathbb{N}[3]$. We write:

$$2y_2 = x_1C_1 + x_3C_3.$$

$$2y_2 + 4y_4 = z_1C_1 + z_3C_3,$$

Then:

$$P(x_1C_1 + x_3C_3) = 18C_1 + 410C_3.$$

This implies:

$$P(x_1) = 7x_1 + x_1^2 = 18.$$

$$P(x_1 + 3x_3) = 18 + 410 \times 3 = 1248.$$

Solving this equation, we find $x_1 = 2; x_1 + 3x_3 = 32$, so $x_3 = 10$.

Similarly,

$$P(z_1C_1 + z_3C_3) = 294C_1 + 650C_3.$$

This implies:

$$P(z_1) = 7z_1 + z_1^2 = 294.$$

$$P(z_1 + 3z_3) = 294 + 650 \times 3 = 2244.$$

Solving this equation, we find $z_1 = 14; z_1 + 3z_3 = 44$, so $z_3 = 10$.

Finally, we determine:

$$y_2 = C_1 + 5C_3, \quad y_4 = 3C_1.$$

Therefore,

$$A = y_2C_2 + y_4C_4 = (C_1 + 5C_3)C_2 + (3C_1)C_4 = C_2 + 5C_6 + 3C_4,$$

which is a root of $P(A) = B$.

4.3 Complexity Analysis of Root-Finding Algorithms

Theorem 4.8 *The compexity for finding polynomial roots runs in $O(\text{card}(B) \log N(B))$ time.*

Proof In algorithms for computing the m-th root or solving polynomial root equations, a fundamental approach is to express A as a sum of cycles, ensuring that the set of cycle lengths $L(A)$ is a subset of $L(B)$. The primary computational challenge involves finding a non-negative integer solution for $x^m = y$ in the first case or $P(x) = y$ in the second case.

A detailed analysis reveals that the number of equations to be solved corresponds to the number of terms in B, denoted as $\text{card}(B)$.

- Given that computing $x = y^{\frac{1}{m}}$ requires constant time $O(1)$, the algorithm to calculate the m -th root runs in $O(\text{card}(B))$ time.
- For solving $P(x) = y$ in $\mathbb{N}$, the complexity is $O(\log y)$, as justified by the following observations:
  - The search space for x is approximately $[1, y^{1/n}]$, resulting in at most $O(\log y^{1/n}) = O\left(\frac{\log y}{n}\right)$ iterations.
  - Each iteration involves evaluating $P(x)$, which requires $O(n)$ time.

Consequently, the overall complexity is:

$$O\left(n \cdot \frac{\log y}{n}\right) = O(\log y).$$

Since y is bounded by the size $N(B)$ of B, the algorithm for finding polynomial roots runs in $O(\text{card}(B) \log N(B))$ time. □

5 Conclusion and Perspectives

In this paper, we investigate the algorithm for the factorization of functional graphs, a key question in the study of discrete dynamical systems. This problem is closely related to a fundamental topic in graph theory: the factorization of tensor product of graphs.

Our main contribution is to place the problem in the context of studying polynomials in multiple variables, where the variables take values as cycles of prime lengths. We introduce the idea of recursively analyzing the factorization of these polynomials based on the number of primes involved. Our algorithm is both precise and implementable. In some special cases, if the space of possible solutions is large, the running time of our algorithm can become significant. However, in many cases, the running time can be effectively bounded. In particular, when applied to the problem of finding the root of a power or a polynomial, our approach proves to be highly efficient, achieving logarithmic complexity.

We can consider our algorithm as a general framework for solving various factorization problems in functional graphs. In the future, we aim to analyze this problem for other important classes of graphs and extend our study to related problems, such as the irreducibility problem and the primality testing of functional graphs.

References

[1] F. Bridoux, C. Crespelle, T.H.D. Phan, and A. Richard. Dividing permutations in the semiring of functional digraphs. In M. Gadouleau and A. Castillo-Ramirez, editors, *Cellular Automata and Discrete Complex Systems. AUTOMATA 2024*, volume 14782 of *Lecture Notes in Computer Science*. Springer, Cham, 2024.

[2] Sheng Chen and Xiaomei Chen. Weak connectedness of tensor product of digraphs. *Discrete Applied Mathematics*, 185:52–58, 2015.

[3] Oscar Defrain, Antonio E. Porreca, and Ekaterina Timofeeva. Polynomial-delay generation of functional digraphs up to isomorphism. *Discrete Applied Mathematics*, 357:24–33, 2024.

[4] Alberto Dennunzio, Valentina Dorigatti, Enrico Formenti, Luca Manzoni, and Antonio E Porreca. Polynomial equations over finite, discrete-time dynamical systems. In *Cellular Automata: 13th International Conference on Cellular Automata for Research and Industry, ACRI 2018, Como, Italy, September 17–21, 2018, Proceedings 13*, pages 298–306. Springer, 2018.

[5] Alberto Dennunzio, Enrico Formenti, Luciano Margara, and Sara Riva. A note on solving basic equations over the semiring of functional digraphs. *arXiv preprint arXiv:2402.16923*, 2024.

[6] François Doré, Enrico Formenti, Antonio E. Porreca, and Sara Riva. Decomposition and factorisation of transients in functional graphs. *Theoretical Computer Science*, 999:114514, June 2024.

[7] Wilfried Imrich. Factoring cardinal product graphs in polynomial time. *Discrete Mathematics*, 192:119–144, 1998.

[8] AS Jarrah and R Laubenbacher. Finite dynamical systems: A mathematical framework for computer simulation. In *Mathematical Modeling, Simulation, Visualization and e-Learning*, pages 343–358. Springer, 2007.

[9] Antonio E. Porreca and Marius Rolland. Injectivity of polynomials over finite discrete dynamical systems. *arXiv preprint*, 2025.

[10] Paul M. Weichsel. The kronecker product of graphs. *Proceedings of the American Mathematical Society*, 13:47–52, 1962.

[11] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica.* Cambridge University Press, 1910–1913. 3 volumes.

[12] Émile Naquin and Maximilien Gadouleau. Factorisation in the semiring of finite dynamical systems. *Theoretical Computer Science*, 998:114509, 2024.