Discrete Mathematics and Theoretical Computer Science .... **vol. 27:3 #6 (2025)**

On solving basic equations over the semiring of functional digraphs

Alberto Dennunzio[1] Enrico Formenti[2] Luciano Margara[3] Sara Riva[4]

[1] *Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano-Bicocca, Milano, Italy*
[2] *Université Côte d'Azur, CNRS, I3S, Nice, France*
[3] *Department of Computer Science and Engineering, University of Bologna, Cesena Campus, Cesena, Italy*
[4] *Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, Lille, France*

revisions 17th Oct. 2024, 26th May 2025, 16th July 2025; accepted 23rd July 2025.

Endowing the set of functional digraphs with the sum (disjoint union of digraphs) and product (standard direct product on digraphs) operations induces on FDs a structure of a commutative semiring R. The operations on R can be naturally extended to the set of univariate polynomials $R[X]$ over R. This paper provides a polynomial time algorithm for deciding if equations of the type $AX = B$ have solutions when A is just a single cycle and B a set of cycles of identical size. We also prove a similar complexity result for some variants of the previous equation.

Keywords: functional digraphs, direct product, digraphs factorization, equations on digraphs

1 Introduction

A **functional digraph** (FD) is the digraph of a function with finite domain *i.e.,* a digraph with outgoing degree 1. Endowing the set of functional digraphs FGs with the sum (disjoint union of digraphs) and product (standard direct product on digraphs) operations provides FGs with the structure of a commutative semiring R in which the empty digraph (resp., the single loop) is the neutral element of addition (resp., of product) Dennunzio et al. (2018). This semiring can be naturally extended to the semiring of multivariate polynomials $R[X_1, X_2, \ldots, X_k]$.

A polynomial in $R[X_1, X_2, \ldots, X_k]$ represents a (infinite) set of functions which have a common substructure provided by the coefficients (modulo isomorphism). The factorization is an interesting inverse problem in this context. Indeed, assume to have a functional digraph G. Can G be factorized *i.e.,* decomposed into the product or the sum (or a combination of the two) of functional digraphs with smaller vertex set[(i)]? If this question has been largely investigated for general graphs (see, for instance, Weichsel (1962); Abay-Asmerom et al. (2010); Hammack et al. (2011)), little is known for functional digraphs.

Let us consider another version of the factorization problem. Assume that we have partial information (or partial assumptions) on the factorization of G. This partial information is represented by a sequence

[(i)] We stress that factorization here is meant in the context of abstract algebra which has nothing to do with the notion of factor graph often used in graph theory.

ISSN 1365–8050

of functional digraphs $A_1, A_2, \ldots, A_k$ and the question of factorization can be reformulated as "does the following equation

$$A_1 \cdot X_1 + A_2 \cdot X_2 + \ldots + A_k \cdot X_k = G$$

admit a solution?" And more generally, one can ask for solutions of the following

$$A_1 \cdot X_1^{w_1} + A_2 \cdot X_2^{w_2} + \ldots + A_k \cdot X_k^{w_k} = G \tag{1}$$

where X^w, as usual, is the multiplication of X with itself w times. Figure 1 provides an example of equation between two multivariate polynomials and one of its solutions.

In Dennunzio et al. (2018), it is shown that the problem of deciding if there exist solutions to Equation (1) is in NP. However, we do not know if the problem is NP-complete. Our current conjecture is that it might be in P. In Dennunzio et al. (2023), an algorithmic pipeline is provided for finding the solutions of (1) when all the digraphs involved are the digraph of a permutation (*i.e.,* they are unions of loops and cycles). The software pipeline essentially relies on the solution of a finite (potentially exponential) number of **basic equations** of the type

$$AX = B \tag{2}$$

where A is a single cycle of size p and B is a union of cycles of size q (of course, it might be $p \neq q$). In this paper, we prove that deciding if equations of type (2) have solutions or not is in P.

The interest in this result is manyfold. Without a doubt, it will significantly improve the performance of the software pipeline in Dennunzio et al. (2023) for the case in which one is interested in the existence of solutions and not in their enumeration. It will shed some light on some connected important questions concerning the cancellation problem for functional digraphs *i.e.,* the problem of establishing if $G \cdot F \approx G \cdot H$ implies $F \approx H$ where F, G and H are FDs and $\approx$ is the relation of graph isomorphism Doré et al. (2024a); Émile Naquin and Gadouleau (2024). We refer the reader to the classical book of Hammack et al. (2011) for the factorization and cancelation problems for more general graphs.

2 Background and basic facts

In Dennunzio et al. (2018), an abstract algebraic setting for studying finite discrete dynamical systems was introduced. This setting, can be adapted in a straightforward way to FDs. In this section, we recall just the minimal concepts to understand the problem we want to solve.

Definition 1 (Sum of FDs)**.** *For any pair of FGs $F = (V_F, E_F)$ and $G = (V_G, E_G)$, the **sum** $H = (V_H, E_H)$ of F and G is defined as*

$$V_H = V_F \sqcup V_G$$
$$E_H = \{((a,0),(b,0)) \text{ s.t. } (a,b) \in E_F\} \cup \{((a,1),(b,1)) \text{ s.t. } (a,b) \in E_G\}$$

where $A \sqcup B$ is the disjoint union of A and B defined as $A \sqcup B = (A \times \{0\}) \cup (B \times \{1\})$.

The product of FDs is the standard direct product of graphs that we recall here for completeness sake.

Definition 2 (Product of FDs)**.** *For any pair of FGs $F = (V_F, E_F)$ and $G = (V_G, E_G)$, the **product** $H = (V_H, E_H)$ of F and G is defined as*

$$V_H = V_F \times V_G$$
$$E_H = \{((a,c),(b,d)) \in V_H \times V_H \text{ s.t } (a,b) \in E_F \text{ and } (c,d) \in E_G\}$$

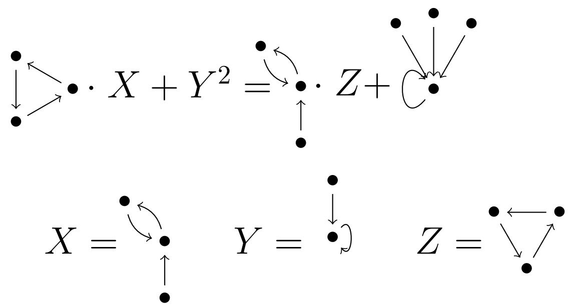

Fig. 1: An example of polynomial equation over FDs (above) and a solution (below).

When no misunderstanding is possible, we will denote with $+$ (resp., $\cdot$) the sum (resp., the product) of FDs.

As already informally stated in the introduction, we have the following algebraic characterisation of the class of FDs.

Theorem 1 (Dennunzio et al. (2018))**.** *The class of FDs (modulo isomorphisms) equipped with the operations of $+$ and $\cdot$ is a commutative semiring.*

We also recall the following important result on the complexity of solving polynomial equations.

Theorem 2 (Dennunzio et al. (2018))**.** *The problem of deciding if there are solutions to*

- *polynomial equations over $R[X_1, X_2, \ldots]$ is undecidable;*
- *polynomial equations over $R[X_1, X_2, \ldots]$ with constant right-hand side is in* NP.

We remark that the problem of finding solutions for polynomial equations with a constant right-hand term has some similarities with other well-known NP-complete problems (knapsack, change-making, for instance). However, we do not know if it is complete or not. This further motivates the quest for simpler equations which admit solution algorithms with lower complexity. These algorithms will be very convenient for use in practical applications. Indeed, in biology, when analyzing the behavior of a gene regulation network,for example, one can ask if the observed behavior is produced by the network at hand or if it is the result of the cooperation of simpler systems. These questions can be easily translated in terms of equations over FDs. However, gene regulation networks may contain hundreds of genes and hence effective methods are necessary to solve those equations.

3 Solving basic equations on permutations

From now on, we will focus on a subclass of functional digraphs, namely, the digraphs of permutations. For this reason, we introduce a convenient notation called **C-notation** inspired from Dennunzio et al. (2023). We note C_p the graph made by a single cycle of size p. Similarly, $n \cdot C_p$ (or simply nC_p) denotes a graph that is the sum of n graphs C_p. According to this notation, it is clear that $aC_p + bC_p = (a+b)C_p$ and $a \cdot bC_p = abC_p$ for any natural a, b. With this notation, the classical result that the graph G of a

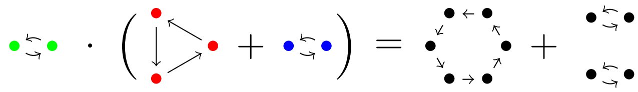

Fig. 2: Example of operations on FDs of permutations. Using the C-notation, we have $C_2 \cdot (C_3 + C_2) = C_2 \cdot C_3 + C_2 \cdot C_2 = C_6 + 2C_2$.

permutation is a union of disjoint cycles translates into

$$G = \sum_{i=1}^{l} n_i C_{p_i}$$

for suitable positive integers $p_1, \ldots, p_l$ and $n_1, \ldots, n_l$. Figure 2 provides an example of operations on permutation digraphs and their expression through the C-notation.

The following proposition provides an explicit expression for the product of unions of cycles.

Proposition 1. *For any natural $l > 1$ and any positive naturals $n_1, \ldots, n_l$, $p_1, \ldots, p_l$, it holds*

$$\prod_{i=1}^{l} n_i C_{p_i} = \frac{\tilde{p}_l \tilde{n}_l}{\lambda_l} C_{\lambda_l} \quad .$$

where $\lambda_l = \operatorname{lcm}(p_1, \ldots, p_l)$, $\tilde{p}_l = \prod_{i=1}^{l} p_i$ and $\tilde{n}_l = \prod_{i=1}^{l} n_i$.

Proof: We proceed by induction over l. First of all, we prove that the statement is true for $l = 2$, *i.e.,*

$$n_1 C_{p_1} \cdot n_2 C_{p_2} = \frac{(p_1 p_2)(n_1 n_2)}{\lambda_2} C_{\lambda_2} \quad . \tag{3}$$

Consider the case $n_1 = n_2 = 1$. Since C_{p_1} and C_{p_2} can be viewed as finite cyclic groups of order p_1 and p_2, respectively, each element of the product of such cyclic groups has order $\lambda_2 = \operatorname{lcm}(p_1, p_2)$ or, in other words, each element of $C_{p_1} \cdot C_{p_2}$ belongs to some cycle of length λ_2. So, $C_{p_1} \cdot C_{p_2}$ consists only of $(p_1 \cdot p_2)/\lambda_2$ cycles, all of length λ_2, and therefore

$$C_{p_1} \cdot C_{p_2} = \frac{p_1 p_2}{\lambda_2} C_{\lambda_2} \quad .$$

In the case $n_1 \neq 1$ or $n_2 \neq 1$, since the product is distributive over the sum, we get

$$n_1 C_{p_1} \cdot n_2 C_{p_2} = \sum_{i=1}^{n_1} C_{p_1} \cdot \sum_{j=1}^{n_2} C_{p_2} = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} (C_{p_1} \cdot \ C_{p_2}) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \frac{p_1 p_2}{\lambda_2} C_{\lambda_2} = \frac{(p_1 p_2)(n_1 n_2)}{\lambda_2} C_{\lambda_2} \quad .$$

Assume now that the equality holds for any $l > 2$. Then, we get

$$\sum_{i=1}^{l+1} C_{p_i}^{n_i} = \frac{\tilde{p}_l \tilde{n}_l}{\lambda_l} C_{\lambda_l} \cdot \ n_{l+1} C_{p_{l+1}} = \frac{\tilde{p}_l p_{l+1} \tilde{n}_l n_{l+1}}{\operatorname{lcm}(\lambda_l, p_{l+1})} C_{\operatorname{lcm}(\lambda_l, p_{l+1})} = \frac{\tilde{p}_{l+1} \tilde{n}_{l+1}}{\lambda_{l+1}} C_{\lambda_{l+1}} \quad ,$$

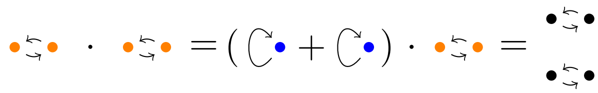

Fig. 3: Example of functional digraph having multiple factorizations. Using the C-notation, we have $C_2 \cdot C_2 = (C_1 + C_1) \cdot C_2 = 2C_2$.

where $\lambda_{l+1} = \text{lcm}(\lambda_l, p_{l+1})$, $\tilde{p}_{l+1} = \tilde{p}_l p_{l+1}$ and $\tilde{n}_{l+1} = \tilde{n}_l n_{l+1}$. □

In the sequel, we will also make use of the following notation. Let n be any integer strictly greater than 1. We note $\text{factors}\,(n)$ the set of prime factors of n *i.e.,* if $n = n_1^{m_1} \cdot \ldots \cdot n_t^{m_t}$, then $\text{factors}\,(n) = \{n_1, \ldots, n_t\}$ where $n_1, \ldots, n_t$ are distinct primes.

According to the C-notation, **basic equations** will have the following form

$$C_p \cdot X = nC_q \quad , \tag{4}$$

where X is the unknown and p, n and q are positive integers. We stress that the numbers p, q and n are just positive integers. They should not be thought of as prime integers unless explicitly mentioned. One of the issues of basic equations is that they might admit several distinct solutions. This is because some digraphs admit several distinct factorizations. Figure 3 provides an easy example of this fact.

As we already stressed in the introduction, in many cases, we are not interested in knowing the exact solutions of a basic equation over permutations but we are interested in deciding if an equation admits a solution or not. More formally, in this paper, we are interested in the computational complexity of the following decision problem.

Problem (DEEP, DEciding basic Equations on Permutations)**.**
Instance: $p, q, n \in \mathbb{N} \setminus \{0\}$.
Question: *Does the equation $C_p \cdot X = nC_q$ admit any solution?*

DEEP is in NP. In fact, the number of nodes in the solution X must be a divisor of nq. Hence, one can simply compute the products with C_p of the disjoint cycles that make up X (using the previous proposition, for example) and verify that n copies of C_q are obtained. All this can be clearly done in polynomial time in the size of n, p and q.

However, we are going to show that the problem can be solved in polynomial time in Theorem 4. Remark that this results is stronger than expected. Indeed, as we already said in the introduction, the generic case $AX = B$ is in NP when the input is given in unary (*i.e.,* the size of a cycle C_p is p). Even if this problem would have been proven in P, this does not imply that DEEP in P, because the input of DEEP are integers and hence it has logarithmic size (*i.e.,* a cycle C_p is represented using $\log_2 p$ bits).

The proof of Theorem 4 is based on the following characterization of the solutions of basic equations on the digraphs of permutations. In the sequel, we denote $\text{factors}\,(p) = \{p_1, \ldots, p_r\}$ the set of primes appearing in the prime factor decomposition $p_1^{h_1} \cdot \ldots \cdot p_r^{h_r}$ of the integer p.

Theorem 3. *Let p, q, n be positive integers and let $p_1^{h_1} \cdot \ldots \cdot p_r^{h_r}$ (resp., $q_1^{k_1} \cdot \ldots \cdot q_t^{k_t}$) be the prime factor decomposition of p (resp., q). The equation $C_p \cdot \sum_{i=1}^{l} C_{z_i} = nC_q$ has solutions (the values z_i are the unknowns and need not to be distinct) if and only if the two following conditions hold:*

1. *p divides q;*
2. *e divides n,*

where $e = \prod_{v \in E} v$ with $E = \left\{ q_j^{k_j} \mid \exists i \text{ s.t. } p_i = q_j \text{ and } h_i < k_j \right\}$.

Proof: The left part of equation can be rewritten as $\sum_{i=1}^{l} C_p \cdot C_{z_i}$. Hence, by Proposition 1, p must divide q. This means that $\text{factors}(p) \subseteq \text{factors}(q)$ and that for any $p_i^{h_i}$ there exists $j \in \{1, \ldots, t\}$ such that $q_j \in \text{factors}(q)$ and $h_i \leq k_j$.
($\Leftarrow$) Given the factorization of q and Proposition 1, we can rewrite nC_q as follows.

$$nC_q = C_q \cdot nC_1 = C_{q_1^{k_1} \cdot \ldots \cdot q_s^{k_t}} \cdot nC_1 = C_{q_1^{k_1}} \cdot \ldots \cdot C_{q_t^{k_t}} \cdot nC_1$$

Suppose now that, for some $i \in \{1, \ldots, r\}$, $p_i^{h_i}$ is not contained in $\left\{ q_1^{k_1}, \ldots, q_t^{k_t} \right\}$. Then, there exists $j \in \{1, \ldots, s\}$ such that $p_i = q_j$ and $h_i < k_j$. Now, if $p_i^{h_i}$ divides n, we have

$$C_{q_1^{k_1}} \cdot \ldots \cdot C_{q_s^{k_t}} \cdot nC_1 = C_{q_1^{k_1}} \cdot \ldots \cdot C_{q_s^{k_s}} \cdot C_{\mathsf{p}_i^{h_i}} \cdot \frac{n}{\left(p_i^{h_i}\right)} C_1.$$

Repeating the same operation for all $p_i^{h_i}$ not included in $\left\{ q_1^{k_1}, \ldots, q_t^{k_t} \right\}$, we find a solution since $\sum_{i=1}^{l} C_{z_i}$ will contain a cycle C_{z_i} (with z_i equal to $q_j^{k_j}$) for any $p_i^{h_i}$ not included, and the set of self-loops.
($\Rightarrow$) Let us suppose that the equation $C_p \cdot \sum_{i=1}^{l} C_{z_i} = nC_q$ holds for some $z_1, z_2, \ldots, z_l$. Let us assume that there exists i such that $p_i^{h_i} \notin \left\{ q_1^{k_1}, \ldots, q_t^{k_t} \right\}$. Then, $p_i^{h_i}$ will be equal to a certain q_j^y with $y < k_j$. This implies that all $z_1, z_2, \ldots, z_l$ must contain exactly $q_j^{k_j}$ as a factor, since the lcm with p must be q. Thus we can write:

$$\sum_{i=1}^{l} C_{z_i} = C_{q_j^{k_j}} \cdot \sum_{i=1}^{l} C_{\frac{z_i}{\left(q_j^{k_j}\right)}}$$

which leads us to the following

$$\begin{aligned} C_p \cdot \sum_{i=1}^{l} C_{z_i} &= C_p \cdot C_{q_j^{k_j}} \cdot \sum_{i=1}^{l} C_{\frac{z_i}{\left(q_j^{k_j}\right)}} \\ &= C_{\frac{p}{\left(q_j^y\right)}} \cdot C_{q_j^y} \cdot C_{q_j^{k_j}} \cdot \sum_{i=1}^{l} C_{\frac{z_i}{\left(q_j^{k_j}\right)}} \\ &= C_{\frac{p}{\left(q_j^y\right)}} \cdot q_j^y C_{q_j^{k_j}} \cdot \sum_{i=1}^{l} C_{\frac{z_i}{\left(q_j^{k_j}\right)}} \end{aligned}$$

Then, according to Proposition 1, $p_i^{h_i}$ (*i.e.*, q_j^y) must divide n. □

Theorem 3 provides a condition to decide whether a solution to a basic equation can exist. However, we are not going to exploit it directly, since it requires the prime factorization of the integers p and q which would imply unnecessarily high complexity. In fact, the algorithms we are going to conceive will essentially use simple arithmetical operations or basic functions on the integers. In this regard, we consider that the worst-case time complexity of the addition (resp., multiplication) between two integers is $O(n)$ (resp., $O(n^2)$), where n is the input size in bits. The division between two integers is assumed to have the same complexity as the multiplication. Concerning $\gcd$, we consider the classical Euclidean GCD algorithm which has worst-case time complexity $O(n^2)$. In the conclusions, we succinctly discuss how the choices in the implementations impacts the final complexity of our algorithm.

To determine the complexity of DEEP, we need two important lemmas. We emphasize that they will be applied to the peculiar case that we are studying but they are valid for any pair of positive integers.

Lemma 1. *Let p, q be two positive integers. Let $p_1^{h_1} \cdot \ldots \cdot p_r^{h_r}$ and $q_1^{k_1} \cdot \ldots \cdot q_t^{k_t}$ be the prime factor decompositions of p and q, respectively. Finally, let $\mathbf{F}$ be the set of all $q_j^{k_j}$ such that $q_j \in \text{factors}(q) \setminus \text{factors}(p)$. Algorithm 1 computes $\Pi_{\mathbf{F}} = \prod_{\mathbf{f} \in \mathbf{F}} \mathbf{f}$ without using the factorizations of p and q. Moreover, the worst-case time complexity of Algorithm 1 is $O(s^3)$, where s is the size of the input in bits.*

Proof: Denote $P = \text{factors}(p)$ and $Q = \text{factors}(q)$. If $g = \gcd(p, q) = 1$ (Line 2), then $P \cap Q = \emptyset$ and $\Pi_{\mathbf{F}} = q$. Hence, the algorithm returns q (Line 4). If $g \neq 1$, then taking q/g keeps unchanged the $q_j^{k_j}$ belonging to $Q \setminus P$ and at the same time decreases of at least 1 the exponents of the q_j belonging to $P \cap Q$. Therefore, calling recursively (Line 6) $\Pi_{\mathbf{F}}$ with $(q = q/g, g)$ keeps decreasing the exponents of q_j belonging to $P \cap Q$ until they become 0. At that point, $\gcd(q, g) = 1$ and the algorithm exits returning $\Pi_{\mathbf{F}}$. Remark that at each call of Π_F, we compute a $\gcd$ (Line 2) plus a division (Line 6) which cost $O(s^2)$ in total. Since the depth of the recursion is $O(s)$, the worst-case time complexity for the algorithm is $O(s^3)$. □

Algorithm 1:

1 **Function** `Compute-`$\mathbf{\Pi_F}$ (p, q)
  Input : p and q positive integers
  Output: $\Pi_{\mathbf{F}}$
2   $g \leftarrow \gcd(p, q)$;
3   **if** $g == 1$ **then**
4   **return** q;
5   **else**
6   **return** $\Pi_{\mathbf{F}}(g, \frac{q}{g})$;
7   **end**

Lemma 2. *Let p and q be two positive integers such that $p|q$. Let $p_1^{h_1} \cdot \ldots \cdot p_r^{h_r}$ and $q_1^{k_1} \cdot \ldots \cdot q_t^{k_t}$ be the prime factor decompositions of p and q, respectively. Finally, let $\mathbf{E}$ be the set of all $p_i = q_j \in \text{factors}(p) \cap \text{factors}(q)$ such that $h_i < k_j$. Then, Algorithm 2 computes $\Pi_{\mathbf{E}} = \prod_{\mathbf{e} \in \mathbf{E}} \mathbf{e}$ without using* *the factorizations of* p *and* q*. The worst-case time complexity of Algorithm 2 is* $O(s^2 \log s)$*, where* s *is the size of the input in bits.*

Proof: By executing Lines 2 and 3 of Algorithm 2, we save in g the product of the factors $q_j^{k_j}$ such that $q_j \in \text{factors}(p) \cap \text{factors}(q)$. However, the exponents of the factors q_j in g are those of q while to compute $\mathbf{\Pi_E}$ we need those of p. This is the purpose of the while loop (Lines 5-9). Indeed, by recursively squaring d, it will make the exponents of the factors of q selected in g grow bigger than those in p and hence the $\gcd$ of line 8 will select the factors and the corresponding exponents that we were looking for. Concerning the complexity, it is enough to remark that initialisation part of the algorithm (Lines 2 to 4) costs $O(s^2)$ and that the number of iterations of the while loop is $O(\log s)$ in the worst-case with a cost of $O(s^2)$ per each iteration. □

Algorithm 2:

```
1  Function Compute-Π_E (p, q)
       Input  : p and q integers s.t. p|q
       Output: ∏_{e∈E} e
2      d ← q/p;
3      g ← gcd(d, p);
4      g′ ← gcd(d ∗ d, p);
5      while g ≠ g′ do
6          g ← g′;
7          d ← d ∗ d;
8          g′ ← gcd(d, p);
9      end
10     return g;
```

Theorem 4. *For any positive integers* p, q *and* n*,* DEEP *has worst-case time complexity* $O(s^3)$*, where* s *is the size of the input in bits.*

Proof: First of all, let us prove that Algorithm 3 solves DEEP. Indeed, choose three positive integers p, q and n and assume that $p_1^{h_1} \cdot \ldots \cdot p_r^{h_r}$ and $q_1^{k_1} \cdot \ldots \cdot q_t^{k_t}$ are the prime number factorizations of p and q, respectively. Line 2 checks if p divides q which is the first condition of Theorem 3. Let us focus on Line 5. Computing $\Pi_{\mathbf{F}}(p, q)$, we get the product of all those factors $q_j^{k_j}$ such that $q_j \in \text{factors}(q) \setminus \text{factors}(p)$. Hence, dividing q by $\Pi_{\mathbf{F}}(p, q)$ we get the product of all the prime factors of q such that $q_j \in \text{factors}(p) \cap \text{factors}(q)$. However, in order to check the second condition of Theorem 3 we need this quantity but the exponents of q_j must be the corresponding ones in the prime factorization of p. This is computed by calling $\Pi_{\mathbf{E}}$ at Line 5. Then, Line 6 checks if the product of the factors $p_i^{h_i} \in \text{factors}(p) \cap \text{factors}(q)$ for which $q_j^{k_j}$ is such that $h_i < k_j$ divides n. We stress that if the product divides n, then each single factor of the product divides n. Hence, Algorithm 3 answers 'yes' if and only if the conditions of Theorem 3 are verified.

Concerning the complexity, we have that the divisibility test in Line 2 can be computed in $O(s^2)$. By Lemmata 1 and 2, we know that computing e takes $O(\max(s^3, s^2 \log s, s^2))$ that is $O(s^3)$. Remark that the size in bits of the quantities involved in the calculation of e is bounded by s. Indeed, both Π_F and Π_E return a divisor of q.

Finally, the checks in Lines 2 and 6 have complexity $O(s^2)$ since e is a divisor of p and, in its turn, p is a divisor of q. We conclude that the time complexity is $O(s^3)$. □

Algorithm 3:

```
1  Function DEEP (p, q, n)
       Input  : p, q and n positive integers
       Output: true if C_p · X = nC_q has solutions, false otherwise
2      if p does not divide q then
3          return false;
4      else    // p divides q
5          e ← Π_E(p, q / Π_F(p,q));
6          if e divides n then
7              return true;
8          else
9              return false;
10         end
11     end
```

We show how the previous algorithms interact to produce the expected results by the following numerical example.

Example 1. *Consider the following equation*

$$C_{8400} \cdot X = 6000C_{8316000}$$

where $p = 8400$*,* $q = 8316000$ *and* $n = 6000$*. Let us first see how by means of the presented algorithms that we can verify the conditions of Theorem 3 without knowing the factorizations of the numbers involved. Later, we will see how the method acts at the level of the factorizations.*

Since p *divides* q*, we want to calculate* $\Pi_{\mathbf{E}}(\frac{8316000}{\Pi_{\mathbf{F}}(8316000,8400)}, 8400)$*. Let us begin by considering* $\Pi_{\mathbf{F}}(8316000\ , 8400)$*. Since* 8316000 *and* 8400 *are not coprime, the method is iterated,* i.e.,

$$\Pi_{\mathbf{F}}(\frac{8316000}{\gcd(8316000, 8400)}, \gcd(8316000, 8400)) = \Pi_{\mathbf{F}}(990, 8400).$$

Again, 990 *and* 8400 *are not coprime, we call recursively the function* $\Pi_{\mathbf{F}}(\frac{990}{\gcd(990,8400)}, \gcd(990, 8400)) = \Pi_{\mathbf{F}}(33, 30)$ *which brings us to* $\Pi_{\mathbf{F}}(\frac{33}{\gcd(33,30)}, \gcd(33, 30)) = \Pi_{\mathbf{F}}(11, 3)$*. Since* 11 *and* 3 *are coprime, the method returns* 11*.*

Let us therefore study $\Pi_{\mathbf{E}}(\frac{8316000}{11}, 8400) = \Pi_{\mathbf{E}}(756000, 8400)$*. With* $i = 1$*,* $d = \frac{756000}{8400} = 90$*,* $g = \gcd(90, 8400) = 30$ *and* $g' = \gcd(90^2, 8400) = \gcd(8100, 8400) = 300$*. Since* $g \neq g'$*,* i *becomes*

2, g takes value 300 *and* g' *becomes* $\gcd(90^3, 8400) = \gcd(729000, 8400) = 600$. *Hence, since* g *and* g' *are still not equal, the method continues as follows.*

$$i = 3, g = 600, g' = \gcd(90^4, 8400) = \gcd(65610000, 8400) = 1200$$

$$i = 4, g = 1200, g' = \gcd(90^5, 8400) = \gcd(5904900000, 8400) = 1200$$

At this point, since $g = g'$, *the method returns* 1200. *Finally, since* 1200 *divides* $n = 6000$ *we know that the equation admits a solution.*

Let us now see what happens, from the point of view of the factorizations, by applying the previous method. Considering the values in input of this example, the factorizations are:

$$p = 2^4 \cdot 3 \cdot 5^2 \cdot 7, \quad q = 2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, \quad n = 2^4 \cdot 3 \cdot 5^3$$

so we have $\text{factors}\,(p) = \{2, 3, 5, 7\}$ *and* $\text{factors}\,(q) = \{2, 3, 5, 7, 11\}$. *The goal of* $\Pi_{\mathbf{F}}$ *is to compute the product of all* $q_j^{k_j}$ *such that* $q_j \in \text{factors}\,(q) \setminus \text{factors}\,(p)$, *and, in fact, the method calculates the following.*

$$\begin{aligned}
&\Pi_{\mathbf{F}}(2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = \\
&\quad = \Pi_{\mathbf{F}}\Big(\frac{2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11}{\gcd(2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)}, \gcd(2^5 \cdot 3^3 \cdot 5^3 \cdot 7 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)\Big) \\
&\quad = \Pi_{\mathbf{F}}(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)
\end{aligned}$$

$$\begin{aligned}
&\Pi_{\mathbf{F}}(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = \\
&\quad = \Pi_{\mathbf{F}}\Big(\frac{2 \cdot 3^2 \cdot 5 \cdot 11}{\gcd(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)}, \gcd(2 \cdot 3^2 \cdot 5 \cdot 11, 2^4 \cdot 3 \cdot 5^2 \cdot 7)\Big) \\
&\quad = \Pi_{\mathbf{F}}(3 \cdot 11, 2 \cdot 3 \cdot 5)
\end{aligned}$$

$$\begin{aligned}
&\Pi_{\mathbf{F}}(3 \cdot 11, 2 \cdot 3 \cdot 5) = \\
&\quad = \Pi_{\mathbf{F}}\Big(\frac{3 \cdot 11}{\gcd(3 \cdot 11, 2 \cdot 3 \cdot 5)}, \gcd(3 \cdot 11, 2 \cdot 3 \cdot 5)\Big) \\
&\quad = \Pi_{\mathbf{F}}(11, 3) = 11.
\end{aligned}$$

Once $\Pi_{\mathbf{F}}$ *has been calculated, through* $\frac{q}{\Pi_{\mathbf{F}}(q,p)}$ *we obtain the product of all* $q_j^{k_j}$ *such that* $q_j \in \text{factors}\,(q) \cap \text{factors}\,(p)$, i.e., $2^5 \cdot 3^3 \cdot 5^3 \cdot 7$. *Then, the goal of* $\Pi_{\mathbf{E}}$ *is to calculate the product of all* $q_j^{k_j}$ *such that* $q_j = p_i \in \text{factors}\,(p) \cap \text{factors}\,(q)$ *such that* $h_i < k_j$. *Note that* $90 = 2 \cdot 3^2 \cdot 5$.

$$\begin{aligned}
&i = 1, g = \gcd(2 \cdot 3^2 \cdot 5, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2 \cdot 3 \cdot 5, g' = \gcd(2^2 \cdot 3^4 \cdot 5^2, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^2 \cdot 3 \cdot 5^2, \\
&i = 2, g = 2^2 \cdot 3 \cdot 5^2, g' = \gcd(2^3 \cdot 3^6 \cdot 5^3, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^3 \cdot 3 \cdot 5^2, \\
&i = 3, g = 2^3 \cdot 3 \cdot 5^2, g' = \gcd(2^4 \cdot 3^8 \cdot 5^4, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^4 \cdot 3 \cdot 5^2, \\
&i = 4, g = 2^4 \cdot 3 \cdot 5^2, g' = \gcd(2^5 \cdot 3^{10} \cdot 5^5, 2^4 \cdot 3 \cdot 5^2 \cdot 7) = 2^4 \cdot 3 \cdot 5^2.
\end{aligned}$$

3.1 Some special cases

In this section we provide some special cases of equations with constant right-hand side which are more general than basic equations and have quadratic complexity but they provide either they only sufficient criteria for the existence of of solutions or necessary and sufficient criteria but have quite limited application scope.

The proof of the following proposition is trivial since it boils down to remark the the number of the vertices of the digraphs involved in the left part of the equation must equal the number of vertices of digraphs in the right part. However, we shall provide an alternative proof to explicit some relations that are discussed right after the end of the proof.

Proposition 2. *Given $r+1$ positive integers $p_1,\ldots,p_r,q$ and $r+1$ integers $m_1,\ldots,m_r$ and n. If the following equation*

$$\sum_{i=1}^{r} m_i C_{p_i} \cdot X = nC_q \tag{5}$$

has solutions then $\sum_{i=1}^{r} m_i p_i$ must divide nq.

Proof: Denote $d(q)$ the set of divisors of q. A generic solution to Equation (5) (if it exists) has the form $\hat{X} = \sum_{i=1}^{|d(q)|} s_i C_{t_i}$, where t_i is the i-th divisor of q (divisors are considered ordered by the standard order on integers) and s_i is an integer (possibly 0). We have

$$nC_q = \sum_{k=1}^{r} m_k C_{p_k} \cdot \sum_{i=1}^{|d(q)|} s_i C_{t_i} = \sum_{k=1}^{r}\sum_{i=1}^{|d(q)|} m_k C_{p_k} \cdot s_i C_{t_i} = \sum_{k=1}^{r}\sum_{i=1}^{|d(q)|} m_k s_i \gcd(p_k,t_i) C_{\mathrm{lcm}(p_k,t_i)} \tag{6}$$

Since $\hat{X}$ is a solution it must be either $q = \mathrm{lcm}(p_k,t_i)$ or $s_i = 0$. Hence, from $p_k t_i = \gcd(p_k,t_i)\,\mathrm{lcm}(p_k,t_i)$ and the previous consideration, we get $\gcd(p_k,t_i) = p_k t_i/q$. If we replace this last quantity in Equation (6) we get

$$nC_q = \sum_{k=1}^{r}\sum_{i=1}^{|d(q)|} \frac{m_k s_i p_k t_i}{q} C_q \tag{7}$$

which holds iff

$$\sum_{k=1}^{r}\sum_{i=1}^{|d(q)|} m_k p_k s_i t_i/q = n$$

that is to say iff

$$\left(\sum_{k=1}^{r} m_k p_k\right)\left(\sum_{i=1}^{|d(q)|} s_i t_i\right) = nq \tag{8}$$

We deduce that if Equation (8) has solutions then $\sum_{k=1}^{r} m_k p_k$ divides nq. □

Let us make some remarks concerning the proof of the previous result. Assume that $H = \frac{nq}{\sum_{k=1}^{r} m_k p_k}$ is an integer. Then, the Diophantine equation

$$\sum_{i=1}^{|d(q)|} s_i t_i = H \tag{9}$$

has solutions iff $\gcd(t_1, \ldots, t_{|d(q)|})$ divides H, which is always true if $\sum_{k=1}^{r} m_k p_k$ divides n. With standard techniques (see for instance (Niven et al., 1991, Theorem 5.1, page 213)), one can recursively find the solutions in linear time in the size of q in bits. We point out that it might happen that all the solutions are non admissible *i.e.,* every solution contains at least one $s_i < 0$ for $i \in \{1, \ldots, r\}$. This issue can also be settled in linear time in the size of q in bits (see the considerations following Theorem 5.1 in (Niven et al., 1991, pages 213-214)). However, the big problem that we have is that we need to compute the divisors of q and, as far as we know, the best algorithms for computing them have worst-case time complexity which is sub-exponential in the size of q (see Buhler et al. (1993) for details).

In a similar manner as in Proposition 2, one can derive another necessary condition for Equation (5) as provided in the following proposition.

Proposition 3. *Given* $r + 1$ *positive integers* $p_1, \ldots, p_r, q$ *and* $r + 1$ *integers* $m_1, \ldots, m_r$ *and* n. *If Equation* (5) *has solutions then* $\gcd(m_1, \ldots, m_k)$ *divides* n.

Proof: Denote $d(q)$ the set of divisors of q. From Equation (6) in the proof of Proposition 2, it is easy to check that $\hat{X} = \sum_{i=1}^{|d(q)|} s_i C_{t_i}$ is a solution if and only if

$$\sum_{k=1}^{r} \sum_{i=1}^{|d(q)|} m_k s_i \gcd(p_k, t_i) = n \ . \tag{10}$$

Now, set $S_k = \sum_{i=1}^{|d(q)|} s_i \gcd(p_k, t_i)$ we have

$$\sum_{k=1}^{r} m_k S_k = n \ , \tag{11}$$

where the S_k are the unknowns. This last equation has solutions iff $\gcd(m_1, \ldots, m_r)$ divides n. However, this last condition is only necessary since solutions might be non-admissible (*i.e.,* one or more of the S_k might be negative). □ Again, we would like to stress that solutions to Equation (11) can be found in quadratic time in the size of the input giving raise to r equations

$$S_k = \sum_{i=1}^{|d(q)|} s_i \gcd(p_k, t_i)$$

which can be also solved in quadratic time provided that the divisors t_i of q are known. Hence, once again the problem of finding solutions to Equation (5) can be solved in polynomial time in the size of q in bits (of course, always knowing the divisors of q). However, for some particular values of q, Equation (5) can be solved in a more effective way. An example is shown by the following proposition.

Proposition 4. *Given $r+1$ positive integers $p_1,\dots,p_r,q$ and $r+2$ integers $m_1,\dots,m_r,t$ and n. Let $R=\{1,\dots,r\}$. Assume q is prime and that for every $i\in R$ we have $p_i\neq q^t$. Then, the following equation*

$$\sum_{i=1}^{r} m_i C_{p_i}\cdot X = nC_{q^t} \tag{12}$$

has solutions iff $\sum_{i=1}^{r} m_i p_i$ divides n and for every $i\in R$, $p_i=q^{t_i}$ for some integer $t_i<t$. Moreover, if a solution exists, then it is unique.

Proof:
($\Rightarrow$) Assume that $\sum_{k=1}^{l} C_{v_k}^{u_k}$ is a solution for the equation. Then, we have

$$\sum_{i=1}^{r} m_i C_{p_i}\cdot\sum_{k=1}^{l} u_k C_{v_k} = \sum_{i=1}^{r}\sum_{k=1}^{l} m_i C_{p_i}\cdot u_k C_{v_k} = \sum_{i=1}^{r}\sum_{k=1}^{l} m_i u_k \gcd(p_i,v_k) C_{\mathrm{lcm}(p_i,v_k)} = nC_{q^t}$$

which implies $\mathrm{lcm}(p_i,v_k)=q^t$. Since q is prime, we deduce that either $p_i=q^t$ or $v_k=q^t$. Using the hypothesis we have that $v_k=q^t$ for all $k\in\{1,\dots,l\}$. Hence, the solution can be written as uC_{q^t}. Now, for every $i\in R$, we have $\gcd(p_i,q^t)=p_i$. Hence, uC_{q^t} is a solution iff the following holds

$$\sum_{i=1}^{r} m_i C_{p_i}\cdot uC_{q^t} = \sum_{i=1}^{r} m_i u p_i C_{q^t} = \left(\sum_{i=1}^{r} m_i u p_i\right) C_{q^t} = nC_{q^t}$$

which holds iff $\sum_{i=1}^{r} m_i u p_i = u\cdot\sum_{i=1}^{r} m_i p_i = n$.
($\Leftarrow$) It is not difficult to see that if $\sum_{i=1}^{r} m_i p_i$ divides n, then $\frac{n}{\sum_{i=1}^{r} m_i p_i} C_{q^t}$ is a solution provided that p_i divides q^t which implies that $p_i=q^{t_i}$ for some $t_i\leq t$.

Concerning the uniqueness of the solution, it is enough to remark that the question is equivalent to asking for solutions of a Diophantine equation on a single variable. □

Clearly, the complexity of the algorithm verifying the conditions of Propositions 2 and 3 is quadratic in the size of the input but they are only necessary conditions. On the other hand, Proposition 4 is both a necessary and sufficient condition, still having a quadratic complexity in the size of the input, but it has a limited applicability because it requires quite specific relations between the coefficients.

4 Conclusions

This paper is concerned with `DEEP` *i.e.,* the problem of deciding if a basic equation on permutations has a solution or not. We show that `DEEP` has cubic complexity. However, it is clear that with more care in the choice of the implementation of some of the components of the decision algorithm, we could obtain a time complexity located between $O(s^{2+\epsilon}\log^k s)$ with optimised arithmetic operations and $O(s^3)$ with naive implementation of arithmetic operations. Our purpose was essentially to prove that the problem could be solved in polynomial time with a polynomial of *reasonable* degree, but it is clear that when the instances have large sizes, like in some practical applications, then the shift towards more complex implementations of the components should be taken into account.

For some other variants of `DEEP`, we were able to prove only some necessary conditions which are testable in quadratic time. However, they have some interest of their own as they might also be of help in pruning the search space for software that aims to solve general polynomial equations on functional digraphs such as the software pipeline proposed in Dennunzio et al. (2023).

Several extensions to our results are possible. The most natural one consists of moving from functional digraphs of permutations to larger classes of functional digraphs taking into account the new recent results of Doré et al. (2024a,b) and Émile Naquin and Gadouleau (2024).

Another quite interesting extension would consider general digraphs and not just functional digraphs. From the work of Calderoni et al. (2021), we know that the compositeness testing problem (*i.e.,* answering 'yes' if the graph in input is the direct product of two other graphs) for general graphs is GI-hard. Since a basic equation on general graphs can be seen as a variant of the compositeness problem, the computational complexity of solving basic equations for general graphs is expected to be comparable to GI or even harder. This motivates the search for further digraphs/graphs classes for which there exist efficient algorithms for solving basic equations. One starting point could be the class of digraphs with out-degree 2.

Acknowledgements

This work was partially supported by:

- the HORIZON-MSCA-2022-SE-01 project 101131549 "Application-driven Challenges for Automata Networks and Complex Systems (ACANCOS)";
- the PRIN 2022 PNRR project "Cellular Automata Synthesis for Cryptography Applications (CASCA)" (P2022 MPFRT) funded by the European Union – Next Generation EU;
- ANR "Ordinal Time Computations" (OTC), ANR-24-CE48-0335-01.

References

G. Abay-Asmerom, R. H. Hammack, C. E. Larson, and D. T. Taylor. Direct product factorization of bipartite graphs with bipartition-reversing involutions. *SIAM Journal on Discrete Mathematics*, 23(4): 2042–2052, 2010.

J. P. Buhler, H. W. Lenstra, and C. Pomerance. Factoring integers with the number field sieve. In A. K. Lenstra and H. W. Lenstra, editors, *The development of the number field sieve*, pages 50–94, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

L. Calderoni, L. Margara, and M. Marzolla. Direct product primality testing of graphs is GI-hard. *Theoretical Computer Science*, 860:72–83, 2021. ISSN 0304-3975.

A. Dennunzio, V. Dorigatti, E. Formenti, L. Manzoni, and A. E. Porreca. Polynomial equations over finite, discrete-time dynamical systems. In *Proc. of ACRI'18*, pages 298–306, 2018.

A. Dennunzio, E. Formenti, L. Margara, and S. Riva. An algorithmic pipeline for solving equations over discrete dynamical systems modelling hypothesis on real phenomena. *Journal of Computational Science*, 66:101932, 2023.

F. Doré, E. Formenti, A. E. Porreca, and S. Riva. Decomposition and factorisation of transients in functional graphs. *Theor. Comput. Sci.*, 999:114514, 2024a.

F. Doré, K. Perrot, A. E. Porreca, S. Riva, and M. Rolland. Roots in the semiring of finite deterministic dynamical systems. In M. Gadouleau and A. Castillo-Ramirez, editors, *Cellular Automata and Discrete Complex Systems - 30th IFIP WG 1.5 International Workshop, AUTOMATA 2024, Durham, UK, July 22-24, 2024, Proceedings*, volume 14782 of *Lecture Notes in Computer Science*, pages 120–132. Springer, 2024b.

R. H. Hammack, W. Imrich, S. Klavžar, W. Imrich, and S. Klavžar. *Handbook of product graphs*, volume 2. CRC press Boca Raton, 2011.

I. Niven, H. Zuckerman, and H. Montgomery. *An Introduction to the Theory of Numbers*. Wiley, 1991.

P. M. Weichsel. The kronecker product of graphs. In *Proc. Amer. Math. Soc.*, volume 13, pages 47–52, 1962.

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