A Graph Joining Greedy Approach to Binary de Bruijn Sequences

Zuling Chang, Martianus Frederic Ezerman, Adamas Aqsa Fahreza, and Qiang Wang

Abstract

Using greedy algorithms to generate de Bruijn sequences is a classical approach. It has produced numerous interesting results theoretically. This paper proposes an algorithm, named GPO algorithm, which includes all prior greedy algorithms as specific instances, excluding the application of the Fleury Algorithm on the de Bruijn graph. Our general algorithm can produce all binary periodic sequences with nonlinear complexity greater than one, by choosing suitable feedback functions and initial states. In particular, a sufficient and necessary condition for the GPO Algorithm to generate binary de Bruijn sequences is established. This requires the use of feedback functions with a unique cycle or loop in their state graphs. Moreover, we discuss modifications to the GPO Algorithm to handle more families of feedback functions whose state graphs have multiple cycles or loops. These culminate in a graph joining method. Several large classes of feedback functions are subsequently used to illustrate how the GPO Algorithm and its modification into the Graph Joining GPO Algorithm work in practice.

Index Terms

Binary periodic sequence, de Bruijn sequence, feedback function, greedy algorithm, LFSR, state graph.

I. Introduction

In a binary *de Bruijn sequence* of order n, if we look at any string of length 2^n, each n-tuple occurs exactly once per period. For any n, there are $2^{2^{n-1}-n}$ cyclically inequivalent de Bruijn sequences [1]. While many properties of such sequences are well-known, still a lot more remains to be discovered as applications in diverse areas such as cryptography, bioinfomatics, and robotics continue to be discussed. Instead of attempting a comprehensive survey of the application landscape, we highlight a number of implementations in the literature.

It is a major interest in cryptography to identify a large number of highly nonlinear de Bruijn sequences and, if possible, to quickly generate them, either on hardware or software. Works on the design and evaluation of de Bruijn sequences for cryptographic implementations are too numerous to list. We mention a few as starting points for interested readers to consult. Some de Bruijn sequences with certain linear complexity profiles, see the discussion in [2], can be used as keystream generators in stream chipers [3]. An efficient hardware implementation, capable of handling large order n, is proposed in [4].

An influential work of Pevzner *et al.* showed deep connections between Eulerian paths in de Bruijn graphs and the fragment assembly problem in DNA sequencing [5]. More recent works, *e.g.*, that of Aguirre *et al.* in [6] described how to use de Bruijn sequences in neuroscientific studies on the neural response to stimuli. The sequences form a rich source of hard-to-predict orderings of the stimuli in the experimental designs.

Position locators, which are extremely useful in robotics, often contain de Bruijn sequences as ingredients. Scheinerman determined the absolute two dimensional location of a robot in a workspace by tiling the workspace with black and white squares, based on a variant of a chosen de Bruijn sequence, in [7]. Further discussion on the roles of de Bruijn sequences in robust positioning patterns, with some discussion on their error-control capabilities, can be found in [8]. A nice application of de Bruijn sequences in image acquisition was proposed in [9]. They form a design of coloured stripe patterns to locate both the intensity peaks and the edges without loss of accuracy, while reducing the number of required hue levels in the pattern. A variant of the 32-card magic trick, from a de Bruijn sequence of order 5, inspired Gagie to come up with two new lower bounds for data compression and a new lower bound on the entropy of a Markov source in [10].

Extensive studies on a topic of long history such as de Bruijn sequences must have resulted in numerous generating methods. An excellent survey for approaches up to the end of 1970s can be found in [11]. Numerous construction routes continue to be proposed. One can arguably put them into several clusters.

In a graph-theoretic construction, one may start with the n-dimensional de Bruijn graph over the binary symbols. A de Bruijn sequence of order n is a Hamiltonian path in the graph. Equivalently, the sequence is an Eulerian cycle in the $n-1$-dimensional

Z. Chang is with the School of Mathematics and Statistics, Zhengzhou University, Zhengzhou 450001, China, e-mail: zuling_chang@zzu.edu.cn.

M. F. Ezerman and A. A. Fahreza are with the Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, 21 Nanyang Link, Singapore 637371, e-mails: {fredezerman,adamas}@ntu.edu.sg.

Q. Wang is with the School of Mathematics and Statistics, Carleton University, 1125 Colonel By Drive, Ottawa, ON K1S 5B6, Canada, e-mail: wang@math.carleton.ca.

The work of Z. Chang is supported by the National Natural Science Foundation of China under Grant 61772476. Nanyang Technological University Grant Number M4080456 supports the research carried out by M. F. Ezerman and A. A. Fahreza. Q. Wang is supported by an individual grant from the Natural Sciences and Engineering Research Council of Canada.

de Bruijn graph. A complete enumeration can be done, *e.g.*, by Fleury algorithm, but is painfully slow. Scores of ideas on how to identify specific paths or cycles have also been put forward, typically with additonal properties imposed on the sequences.

Some approaches are algebraic. One can take a primitive polynomial of order n in the ring of polynomials $\mathbb{F}_2[x]$, turn it into a linear Feedback Shift Register (FSR), and output a maximal length sequence, also known as m-sequence. Appending another 0 to the string of 0s of length $n-1$ in the m-sequence results in a de Bruijn sequence. The *cycle joining method* (CJM) is another well-known generic construction approach. As discussed in, *e.g.*, [11] and [12], the main idea is to join all cycles produced by a given FSR into a single cycle by identifying the so-called conjugate pairs.

Numerous FSRs have been shown to yield a large number of output sequences. We mention two of the many works available in the literature. Etzion and Lempel chose the *pure cycling register* (PCR) and the *pure summing register* (PSR) in [13] to generate a remarkable number, exponential in n, of de Bruijn sequences. When the characteristic polynomials of the FSRs are product of pairwise distinct irreducible polynomials, Chang *et al.* devised some algorithms to determine the exact number of de Bruijn sequences that the CJM can produce in [14]. Interested readers may want to consult [14, Table 4] for a summary of the input parameters and the performance complexity of prior works. There have also been a lot of constructions based on *ad hoc* rules. They are cheap to implement but yield only a few de Bruijn sequences. An example is the joining, in lexicographic order, of all Lyndon words whose length divides n from [15]. Jansen *et al.* established a requirement to determine some conjugate pairs in [16], leading to an efficient generating algorithm. A special case of the requirement was highlighted in [17] and a generalization was subsequently given by Gabric *et al.* in [18].

In the *cross-join pairing* method, one starts with a known de Bruijn sequence and proceeds to identify cross-join pairs that allow the sequence to be cut and reglued into inequivalent de Bruijn sequences. Further details and examples were supplied, *e.g.*, by Helleseth and Kløve in [19] and by Mykkeltveit and Szmidt in [20].

Our present work focuses on the greedy algorithms, of which `Prefer-One` [21] is perhaps the most famous, followed by `Prefer-Same` [11] and `Prefer-Opposite` [22]. Although using greedy algorithms to generate de Bruijn sequences tends to be impractical due to the usually exponential storage demand, it remains theoretically interesting. Aside from being among the oldest methods of generating de Bruijns sequences, greedy algorithms often shed light on various properties of de Bruijn sequences and their connections to other combinatorial and discrete structures. New greedy algorithms continue to be discussed, *e.g.*, a recent one in [23]. The current first three authors recently proposed the `Generalized Prefer-Opposite` (GPO) Algorithm to generalize known greedy algorithms and provided several new families of de Bruijn sequences in [24]. A good number of greedy algorithms make use of *preference functions* as important construction tools. A discussion on this approach was given in [12]. Many results on preference functions have been established by Alhakim in [25].

Here we prove further properties of the GPO Algorithm and show that this new approach covers *all* previously-known ad hoc greedy algorithms as well as those based on preference functions. More specifically, our investigation yields the following contributions.

1) We confirm that *any* periodic sequence with nonlinear complexity $n \geq 2$ can be produced by the GPO Algorithm by showing how to explicitly determine a corresponding input pair. The pair consists of a suitable initial state and a feedback function $f(x_0, x_1, \ldots, x_{n-1})$, in which the coefficient of x_0 is zero. Along the way, we correct a minor mistake [12, Lemma 3 on p. 132]. We then characterize the feedback function f and the initial state(s) that the GPO Algorithm can use to generate de Bruijn sequences of any order n in Theorem 10.
2) If a particular function f fails to directly generate any de Bruijn sequence via the algorithm but meets a simple condition, then we modify the algorithm by adding assignment rules, to ensure that the output sequences become de Bruijn. This novel *graph joining* (GJ) approach results in our `Graph Joining Prefer-Opposite` (GJPO) Algorithm.

We note that the results extend naturally to nonbinary setups. A more comprehensive treatment, however, lies beyond the scope of our present construction work. So is a deeper analysis on the linear complexity profiles of the resulting sequences. After this introduction, Section II gathers useful preliminary notions and known results. Section III establishes the conditions for the GPO Algorithm to generate periodic sequences. Section IV characterizes the conditions for which the GPO Algorithm produces de Bruijn sequences. Section V shows how to modify the GPO Algorithm to still produce de Bruijn sequences even when the required conditions are not satisfied. This allows us to include larger classes of feedback functions in our new graph-joining construction method. We study the graph theoretic properties of a class of feedback functions whose structures can be easily determined and succinctly stored in Section VI. This shows that mitigation is possible to reduce the steep storage cost as n grows, if choices are made judiciously. The last section contains a summary and a few directions to consider.

II. Preliminaries

An *n-stage shift register* is a circuit. It has n consecutive storage units and is clock-regulated. Each unit holds a bit. As the clock pulses, the bit shifts to the next stage in line. The register outputs a new bit s_n based on the n-bit *initial state* $\mathbf{s}_0 := s_0, \ldots, s_{n-1}$. The corresponding *feedback function* $f(x_0, \ldots, x_{n-1})$ is the Boolean function over $\mathbb{F}_2^n$ that, on input $\mathbf{s}_0$, outputs s_n.

The output of a feedback shift register (FSR) is a binary sequence $\mathbf{s} := s_0, s_1, \ldots, s_n, \ldots$ satisfying $s_{n+\ell} = f(s_\ell, s_{\ell+1}, \ldots, s_{\ell+n-1})$ for $\ell = 0, 1, 2, \ldots$. The *smallest integer N* that satisfies $s_{i+N} = s_i$ for all $i \geq 0$ is the *period* of $\mathbf{s}$. The *N-periodic* sequence $\mathbf{s}$

can then be written as $\mathbf{s} := (s_0, s_1, s_2, \ldots, s_{N-1})$. We call $\mathbf{s}_i := s_i, s_{i+1}, \ldots, s_{i+n-1}$ *the i-th state*[1] of $\mathbf{s}$. The states $\mathbf{s}_{i-1}$ and $\mathbf{s}_{i+1}$, analogously defined, are the *predecessor* and *successor* of $\mathbf{s}_i$, respectively. In tabular form, an n-string $c_0, c_1, \ldots, c_{n-1}$ is often written concisely as $c_0 c_1 \ldots c_{n-1}$. The *complement* $\overline{c}$ of $c \in \mathbb{F}_2$ is $1+c$. The complement of a string is produced by taking the complement of each element. The respective strings of zeroes and of ones, each of length ℓ, are denoted by $\mathbf{0}^\ell$ and $\mathbf{1}^\ell$. Given an n-stage state $\mathbf{a} := a_0, a_1, \ldots, a_{n-1}$, its *conjugate state* is $\widehat{\mathbf{a}} := \overline{a_0}, a_1, \ldots, a_{n-1}$ while $\widetilde{\mathbf{a}} := a_0, a_1, \ldots, \overline{a_{n-1}}$ is its *companion state*.

A feedback function $f(x_0, x_1, \ldots, x_{n-1}) = x_0 + g(x_1, \ldots, x_{n-1})$, where g is a Boolean function over $\mathbb{F}_2^{n-1}$, is said to be *nonsingular*. Otherwise f is said to be *singular*. An FSR with nonsingular feedback function generates periodic sequences [12, pp. 115–116].

The FSR that generates a binary (ultimately) periodic sequence $\mathbf{s}$ is, in general, not unique. In his doctoral thesis [26], Jansen introduced the notion of *nonlinear complexity*. The nonlinear complexity of $\mathbf{s}$, denoted by $\mathrm{nlc}(\mathbf{s})$, is the smallest k such that there is a k-stage FSR, with feedback function $f(x_0, x_1, \ldots, x_{k-1})$, that generates $\mathbf{s}$. When this holds, then we can make three assertions. First, each k-stage state of $\mathbf{s}$ appears at most once. Second, there exists at least one $(k-1)$-stage state of $\mathbf{s}$ that appears twice. Third, the period of $\mathbf{s}$ is at most 2^k. Seen in this light, binary de Bruijn sequences of order n are periodic sequences with nonlinear complexity n having the maximum period. Furthermore, there are only two periodic sequences with nonlinear complexity 0, namely, the all zero sequence $(0, 0, \ldots, 0)$ with feedback function $f = 0$ and the all one sequence $(1, 1, \ldots, 1)$ with $f = 1$. The unique sequence with nonlinear complexity 1 is $(0, 1)$ with $f = 1 + x_0$. Henceforth, we concentrate on a binary periodic sequence $\mathbf{s}$ with $\mathrm{nlc}(\mathbf{s}) \geq 2$.

The *state graph* of FSR with feedback function $f(x_0, x_1, \ldots, x_{n-1})$ is a directed graph $\mathcal{G}_f$. Its vertex set $V_{\mathcal{G}_f}$ consists of all 2^n n-stage states. There is an edge directed from a state $\mathbf{u} := u_0, u_1, \ldots, u_{n-1}$ to a state $\mathbf{v} := u_1, u_2, \ldots, f(u_0, u_1, \ldots, u_{n-1})$. We call $\mathbf{u}$ a *child* of $\mathbf{v}$ and $\mathbf{v}$ the *parent* of $\mathbf{u}$. We allow $\mathbf{u} = \mathbf{v}$, in which case $\mathcal{G}_f$ contains a loop, *i.e.*, a cycle with only one vertex. Notice that if f is singular, then some states have two distinct children in $\mathcal{G}_f$. A *leaf* in $\mathcal{G}_f$ is a vertex with no child. A vertex $\mathbf{w}$ is a *descendent* of $\mathbf{u}$ if there is a directed path that starts at $\mathbf{w}$ and ends at $\mathbf{u}$. In turn, $\mathbf{u}$ is called an *ancestor* of $\mathbf{w}$. A *rooted tree* $T_{f,\mathbf{b}}$ in $\mathcal{G}_f$ is the largest tree in $\mathcal{G}_f$ in which the vertex $\mathbf{b}$ has been designated the *root* and the edge that emanates out of $\mathbf{b}$ has been removed from $\mathcal{G}_f$. In this work the orientation is **towards** the root $\mathbf{b}$, *i.e.*, $T_{f,\mathbf{b}}$ is an *in-tree* or an *anti-arborescence*.

Preference function is an important tool in constructing t-ary de Bruijn sequences by greedy algorithm [12, Chapter VI]. We first recall the next two definitions.

Definition 1. *Given a positive integer $t > 1$ and an alphabet $A = \{0, 1, \ldots, t-1\}$ of size t, a preference function P of $n-1$ variables is a t-dimensional vector valued function of $(n-1)$-stage states such that $P_0(\mathbf{a}), \ldots, P_{t-1}(\mathbf{a})$ is a rearrangement of $0, 1, \ldots, t-1$, for each choice of $\mathbf{a} = a_0, a_1, \ldots, a_{n-2} \in A^{n-1}$.*

Definition 2. *For any initial state $\mathbf{u} = u_0, u_1, \ldots, u_{n-1}$ and preference function P, the following inductive definition determines a unique finite sequence $\mathbf{s}$.*

1) $s_0 = u_0$, $s_1 = u_1$, $\ldots$, $s_{n-1} = u_{n-1}$.
2) *If $s_{N+1}, \ldots, s_{N+n-1}$, for $N \geq 0$, have been defined, then $s_{N+n} := P_i(s_{N+1}, \ldots, s_{N+n-1})$, where i is the smallest integer such that the state $s_{N+1}, \ldots, s_{N+n-1}, P_i(s_{N+1}, \ldots, s_{N+n-1})$ has not previously occurred in $s_0, s_1, \ldots, s_{N+n-1}$.*
3) *Let $L(s_i)$ be the first value of N such that no i can be found to satisfy Item 2. Then the last digit of the sequence is s_{L+n-2} and $L(s_i)$ is called the cycle period.*

The sequence generated by the algorithm described in Definition 2 is known to be $L(s_i)$-periodic. Golomb presented various ways to use preference function to generate de Bruijn sequences in [12]. In the next section we modify the procedure in Definition 2 to design a new algorithm that generates periodic sequences.

III. GPO Algorithm and Periodic Sequences

When $t = 2$, the preference function is equivalent to a Boolean function $f(x_0, x_1, \ldots, x_{n-1})$ over $\mathbb{F}_2^n$ that satisfies

$$f(0, \mathbf{a}) = f(1, \mathbf{a}) = P_1(\mathbf{a}). \tag{1}$$

The procedure in Definition 2 can be carried out by Algorithm 1. It was called the `Generalized Prefer-Opposite` (GPO) Algorithm in [24]. The `Prefer-One` de Bruijn sequence, for example, is the output on input $f = 0$ and $\mathbf{u} = \mathbf{0}^n$.

Remark 1. *We make some remarks on the GPO Algorithm in relation to preference functions and the procedure in Definition 2.*

1) *If the current state is $c_0, c_1, \ldots, c_{n-1}$, then the GPO Algorithms prefers $\mathbf{c} := c_1, c_2, \ldots, c_{n-1}, \overline{f(c_0, \ldots, c_{n-1})}$ as the next state, unless $\mathbf{c}$ had appeared before. The algorithm prefers the opposite of $f(c_0, \ldots, c_{n-1})$. Since the name* `Prefer-Opposite` *Algorithm had already been used in [22], we call Algorithm 1 the* `Generalized Prefer-Opposite` *(GPO) Algorithm.*

[1] In most references, a state is written in between parentheses, *e.g.*, $\mathbf{s}_i := (s_i, s_{i+1}, \ldots, s_{i+n-1})$. We remove the parentheses, throughout, for brevity and to avoid the cumbersome $f((s_i, s_{i+1}, \ldots, s_{i+n-1}))$ notation for state evaluation.

Algorithm 1 `Generalized Prefer-Opposite` (GPO)

Input: A feedback function $f(x_0,x_1,\ldots,x_{n-1})$ and an initial state $\mathbf{u}$.
Output: A binary sequence.

$\mathbf{c} := c_0,c_1,\ldots,c_{n-1} \leftarrow \mathbf{u}$
do
  Print(c_0)
  $y \leftarrow f(c_0,c_1,\ldots,c_{n-1})$
  if $c_1,c_2,\ldots,c_{n-1},\overline{y}$ has not appeared before **then**
    $\mathbf{c} \leftarrow c_1,c_2,\ldots,c_{n-1},\overline{y}$
  else
    $\mathbf{c} \leftarrow c_1,c_2,\ldots,c_{n-1},y$
while $\mathbf{c} \neq \mathbf{u}$

2) *The GPO Algorithm and the procedure in Definition 2 differ slightly. If $\mathbf{u} = u_0,u_1,\ldots,u_{n-1}$ is the initial state, then $P_0(u_0,\ldots,u_{n-2}) := u_{n-1}$ in Definition 2. To guarantee that the GPO Algorithm terminates, we let $f(x,u_0,\ldots,u_{n-2}) := u_{n-1}$, instead of $P_1(u_0,\ldots,u_{n-2}) := \overline{u_{n-1}}$.*
3) *We restrict our attention to the binary case because of its wider interest, even though the GPO Algorithm extends naturally to the nonbinary case.*

To successfully use the GPO Algorithm to generate binary de Bruijn sequences, suitable feedback functions and initial states must be identified. For this purpose, we begin by studying the properties of this algorithm to establish the condition that ensures the output is de Bruijn.

When does the GPO Algorithm generate a periodic sequence? If the output is periodic, is it necessarily de Bruijn? Do distinct input pairs always yield inequivalent outputs? The answer to all three questions is no. For some input pairs $(f,\mathbf{u})$, the algorithm never revisits the initial state $\mathbf{u}$ and, hence, does not terminate. In such cases, the output sequence is *ultimately periodic* but not periodic. On occasions, the algorithm generates cyclically equivalent sequences on distinct input pairs $\{(f_i,\mathbf{u}_i) : i \in I\}$ for some index set I. Take $f_1(x_0,\ldots,x_{n-1}) := 0$ and $f_2(x_0,\ldots,x_{n-1}) := \prod_{j=0}^{n-1} x_j$, with $\mathbf{u} = \mathbf{0}^n$, for example. Both input pairs $(f_1,\mathbf{u})$ and $(f_2,\mathbf{u})$ yield an identical de Bruijn sequence of order n.

Next we show that any binary periodic sequence with $\mathrm{nlc}(\mathbf{s}) > 1$ can be generated by the GPO algorithm, with well-chosen feedback function and initial state.

Theorem 1. *Let $\mathbf{s}$ be any binary periodic sequence with $\mathrm{nlc}(\mathbf{s}) = n \geq 2$. Then $\mathbf{s}$ can be generated by the GPO Algorithm with input pair $(f,\mathbf{u})$ such that*

1) *The feedback function f has the form*
$$f(x_0,x_1,\ldots,x_{n-1}) := g(x_1,\ldots,x_{n-1}) \tag{2}$$
for some Boolean function $g(x_0,\ldots,x_{n-2})$ over $\mathbb{F}_2^{n-1}$.
2) *The initial state $\mathbf{u} = u_0,\ldots,u_{n-2},u_{n-1}$ satisfies the requirement that $u_0,\ldots,u_{n-2}$ appears twice in each period of $\mathbf{s}$.*

Proof: Let an N-periodic sequence $\mathbf{s} = (s_0,s_1,\ldots,s_{N-1})$ be given. In one period of $\mathbf{s}$, since $\mathrm{nlc}(\mathbf{s}) = n$, there must exist at least one $(n-1)$-stage state $a_0,a_1,\ldots,a_{n-2}$ that appears twice. Hence, the n-stage state $\mathbf{a} = a_0,\ldots,a_{n-2},a$ and its companion state $\widetilde{\mathbf{a}} = a_0,\ldots,a_{n-2},\overline{a}$ with $a \in \mathbb{F}_2$ appear once each. Any of these two can be the initial state of the GPO Algorithm. Without loss of generality, let $\mathbf{u} := \mathbf{s}_0 = s_0,s_1,\ldots,s_{n-1} = \mathbf{a}$.

We construct a feedback function $f(x_0,\ldots,x_{n-1})$ based on $\mathbf{s}$. Let
$$f(x_0,a_0,\ldots,a_{n-2}) := g(a_0,\ldots,a_{n-2}) = a \text{ for } x_0 \in \mathbb{F}_2.$$
Our starting point is the initial state $\mathbf{u}$. For any state $\mathbf{s}_i = s_i,s_{i+1},\ldots,s_{i+n-1}$ with $0 \leq i \leq N-2$, if the $(n-1)$-stage state $s_{i+1},\ldots,s_{i+n-1} \neq a_0,\ldots,a_{n-2}$ appears for the first time in $\mathbf{s}$, define
$$f(s_i,s_{i+1},\ldots,s_{i+n-1}) := f(\overline{s_i},s_{i+1},\ldots,s_{i+n-1}) = g(s_{i+1},\ldots,s_{i+n-1}) = \overline{s_{i+n}}.$$
If $s_{i+1},\ldots,s_{i+n-1}$ appears for the second time in $\mathbf{s}$, then we define
$$f(s_i,s_{i+1},\ldots,s_{i+n-1}) := f(\overline{s_i},s_{i+1},\ldots,s_{i+n-1}) = g(s_{i+1},\ldots,s_{i+n-1}) = s_{i+n}.$$
The above two definitions of f coincide if the $(n-1)$-stage state $s_{i+1},\ldots,s_{i+n-1}$ appears twice in $\mathbf{s}$. If the $(n-1)$-stage state $b_1,\ldots,b_{n-1}$ has not appeared in $\mathbf{s}$, we define
$$f(x_0,b_1,\ldots,b_{n-1}) := g(b_1,\ldots,b_{n-1}) = b \text{ for any } x_0 \text{ and } b \in \mathbb{F}_2,$$
i.e., $g(b_1,\ldots,b_{n-1})$ can take any arbitrary binary value.

Now we prove that the output of the GPO Algorithm on input $(f,\mathbf{u})$ defined above is indeed $\mathbf{s}$. As the run of the algorithm begins, $\mathbf{c} = c_0,\ldots,c_{n-1} = \mathbf{u} = s_0,\ldots,s_{n-1}$. Inductively, suppose that $c_i,\ldots,c_{i+n-1} = s_i,\ldots,s_{i+n-1}$ for some i with $0 \le i \le N-2$ and in sequence $\mathbf{s}$ the bit after the state $s_i,\ldots,s_{i+n-1}$ is s_{i+n}. If $s_{i+1},\ldots,s_{i+n-1} = a_0,\ldots,a_{n-2}$, then this $(n-1)$-stage state appears in $\mathbf{s}$ for the second time and $s_{i+n} = \overline{a}$. Because the state

$$\mathbf{v} = c_{i+1},\ldots,c_{i+n-1},\overline{f(c_i,c_{i+1},\ldots,c_{i+n-1})} = a_0,\ldots,a_{n-2},\overline{f(c_i,a_0,\ldots,,a_{n-2})} = a_0,\ldots,a_{n-2},\overline{a}$$

has not appeared, the algorithm dictates the next state to be $s_{i+1},\ldots,s_{i+n-1},\overline{a}$ and $c_{i+n} = s_{i+n} = \overline{a}$.

Let $s_{i+1},\ldots,s_{i+n-1} \neq a_0,\ldots,a_{n-2}$. We consider the state

$$\mathbf{v} = c_{i+1},\ldots,c_{i+n-1},\overline{f(c_i,c_{i+1},\ldots,c_{i+n-1})} = s_{i+1},\ldots,s_{i+n-1},\overline{f(s_i,s_{i+1},\ldots,s_{i+n-1})}.$$

If $s_{i+1},\ldots,s_{i+n-1}$ appears for the first time, then, by the definition of f, we have $f(s_i,s_{i+1},\ldots,s_{i+n-1}) = \overline{s_{i+n}}$ and

$$\mathbf{v} = s_{i+1},\ldots,s_{i+n-1},\overline{\overline{s_{i+n}}} = s_{i+1},\ldots,s_{i+n-1},s_{i+n}$$

has not appeared. Hence, the algorithm dictates the next state to be $s_{i+1},\ldots,s_{i+n}$ and $c_{i+n} = s_{i+n}$. If $s_{i+1},\ldots,s_{i+n-1}$ appears for the second time, then, by the definition of f, we know that $f(s_i,s_{i+1},\ldots,s_{i+n-1}) = s_{i+n}$ and

$$\mathbf{v} = s_{i+1},\ldots,s_{i+n-1},\overline{s_{i+n}}$$

must have appeared earlier. Hence, the next state must be

$$s_{i+1},\ldots,s_{i+n-1},f(s_i,s_{i+1},\ldots,s_{i+n-1}) = s_{i+1},\ldots,s_{i+n-1},s_{i+n} \text{ and } c_{i+n} = s_{i+n}.$$

Finally, for $i = N-1$, the state after

$$c_{N-1},c_N,\ldots,c_{N+n-2} = s_{N-1},s_0,\ldots,s_{n-2} = s_{N-1},a_0,\ldots,a_{n-2}$$

must be $a_0,\ldots,a_{n-2},a = \mathbf{u}$, since $a_0,\ldots,a_{n-2},\overline{a}$ has appeared earlier. The algorithm terminates and outputs $\mathbf{s}$. ■

Theorem 1 states that any periodic sequence with nonlinear complexity ≥ 2 can be GPO-generated with feedback function having 0 as the coefficient of x_0 and some initial state. Henceforth, we just consider such feedback functions and we say that a feedback function f is *standard* or *in the standard form* if it has the form specified in Equation (2). For such an f, each non-leaf vertex in $\mathscr{G}_f$ has two children and, if there is a loop, one of the vertex's two children is the vertex itself.

The following lemma puts a necessary and sufficient condition on the initial state for the GPO Algorithm to generate a periodic sequence.

Lemma 2. *Given a standard $f(x_0,x_1,\ldots,x_{n-1})$ and a state* $\mathbf{u}$. *The GPO Algorithm on $(f,\mathbf{u})$ generates a periodic sequence if and only if* $\mathbf{u}$ *is not a leaf in $\mathscr{G}_f$.*

Proof: Since f is in the standard form, $\mathscr{G}_f$ contains leaves. Suppose that the initial state $\mathbf{u}$ is a leaf. To have

$$\mathbf{a} = a_0,a_1,\ldots,a_{n-1} \text{ and } \widehat{\mathbf{a}} = \overline{a_0},a_1,\ldots,a_{n-1}$$

as its two possible predecessors, $\mathbf{u}$ must be

$$a_1,\ldots,a_{n-1},\overline{f(a_0,a_1,\ldots,a_{n-1})} = a_1,\ldots,a_{n-1},\overline{g(a_1,\ldots,a_{n-1})}.$$

As the algorithm visits either $\mathbf{a}$ or $\widehat{\mathbf{a}}$, we keep in mind that $a_1,\ldots,a_{n-1},\overline{f(a_0,a_1,\ldots,a_{n-1})} = \mathbf{u}$ had appeared before, as it is the initial state. Hence, the next state must be

$$a_1,\ldots,a_{n-1},f(a_0,a_1,\ldots,a_{n-1}) = a_1,\ldots,a_{n-1},g(a_1,\ldots,a_{n-1}) = \widetilde{\mathbf{u}}.$$

This implies that the initial state $\mathbf{u}$ will never be revisited. Thus, the output is an infinite and ultimately periodic sequence, *i.e.*, not a periodic one.

Now, let $\mathbf{u}$ be a non-leaf state in $\mathscr{G}_f$. Since the number of states in $\mathscr{G}_f$ is finite, the algorithm must eventually revisit some state. We show that it is impossible for the algorithm to visit any state twice before it visits the initial state $\mathbf{u}$ for the second time. For a contradiction, suppose that $\mathbf{a} \neq \mathbf{u}$ is the first state to be visited twice. The above analysis confirms that $\mathbf{a}$ is not a leaf and, hence, has two children in $\mathscr{G}_f$, say $\mathbf{v}$ and $\widehat{\mathbf{v}}$. When $\mathbf{a}$ is visited by the algorithm for the first time, one of its two children, say $\mathbf{v}$, is the actual predecessor of $\mathbf{a}$ as the algorithm runs. This implies that $\widehat{\mathbf{v}}$ has been visited before $\mathbf{v}$. Otherwise, the successor of $\mathbf{v}$ must be $\widetilde{\mathbf{a}}$, by the rules of the GPO Algorithm. So we have deduced that both children of $\mathbf{a}$ must have been visited when $\mathbf{a}$ is first visited. The second time $\mathbf{a}$ is visited, one of its two children must have also been visited twice. This contradicts the assumption that $\mathbf{a}$ is the first vertex to have been visited twice.

We have thus shown that the initial state $\mathbf{u}$ must be the first state to be visited twice. The algorithm stops and outputs a periodic sequence when it reaches $\mathbf{u}$ for the second time. ■

The next corollary follows immediately from Theorem 1.

Corollary 3. *Let a given binary periodic sequence* $\mathbf{s}$ *with* $\mathrm{nlc}(\mathbf{s}) = n \geq 2$ *be generated by the GPO Algorithm on input* $(f, \mathbf{u})$*, where* $f(x_0, x_1, \ldots, x_{n-1})$ *is in the standard form and* $\mathbf{u} = u_0, u_1, \ldots, u_{n-1}$*. Then the* $(n-1)$*-stage state* $u_0, \ldots, u_{n-2}$ *appears twice in each period of* $\mathbf{s}$*.*

We note that if a periodic sequence $\mathbf{s}$ is generated by the GPO Algorithm with standard feedback function $f(x_0, \ldots, x_{n-1})$, then any n-stage state appears in $\mathbf{s}$ at most once.

Corollary 4. *On input the standard feedback function* $f(x_0, \ldots, x_{n-1})$*, the nonlinear complexity of the periodic sequence generated by the GPO Algorithm is* n*.*

Proof: Since all of the n-stage states of the generated sequence $\mathbf{s}$ are distinct, the nonlinear complexity satisfies $\mathrm{nlc}(\mathbf{s}) \leq n$. On the other hand, by Corollary 3, there is at least one $(n-1)$-stage state in $\mathbf{s}$ that appears twice, making $\mathrm{nlc}(\mathbf{s}) \geq n$. ■

When the GPO Algorithm generates periodic sequence, a known result from [25], also stated with an alternative proof as [24, Lemma 2], follows from the proof of Lemma 2.

Lemma 5. *Let* $\mathscr{G}_f$ *be the state graph of the FSR with standard feedback function* f*. Let* $\mathbf{v}$ *be any vertex with two children. By the time the GPO Algorithm, on input* $(f, \mathbf{u})$ *where* $\mathbf{u}$ *is not a leaf, visits* $\mathbf{v} \neq \mathbf{u}$*, it must have visited both children of* $\mathbf{v}$*.*

Lemma 5 does not hold if the initial state is a leaf. Let $f(x_0, x_1, x_2) = x_1 + 1$, for example. If we use 010, which is a leaf in $\mathscr{G}_f$, as the initial state, then the states appear in the order

$$010 \to 101 \to 011 \to 111 \to \ldots.$$

When 011 is visited, one of its children, 001, has not been visited.

We discover that [12, Lemma 3 on p. 132] is in fact incorrect. The original statement is reproduced here for convenience.

> "For any cyclic recursive sequence $\{a_i\}$ of degree n and for any n-digit word in it, there exists a preference function which generates the sequence, using the given word as the initial word."

The quoted statement holds for t-ary de Bruijn sequences, but fails in general. In the binary case, Corollary 3 tells us that the initial state $\mathbf{u}$ must have the property that the $(n-1)$-stage state $u_0, \ldots, u_{n-2}$ appears twice in $\mathbf{s}$. For a general periodic sequence $\mathbf{s}$, such a condition may not be met. Consider $\mathbf{s} = (0011101)$, for instance, and start with 001 as the initial state. Then we have either $(P_0(00) = 0 \wedge P_1(00) = 1)$ or $(P_0(00) = 1 \wedge P_1(00) = 0)$. If the former, then, upon the second visit to 00, the smallest i is therefore 0 and the string 000 is formed. If the latter, then, upon the second visit to 00, the smallest i is now 1 and, again, the string 000 is formed. But $\mathbf{s}$ does not contain the string 000.

Furthermore, in the t-ary case, the initial state must satisfy that $u_0, \ldots, u_{n-2}$ appears t times in $\mathbf{s}$. Thus, there exist periodic sequences of degree n that can never be generated by the algorithm described in Definition 2. We state the conclusion as a lemma, without a proof.

Lemma 6. *For a cyclic recursive sequence* $\{a_i\}$ *of degree n satisfying that there is a* $(n-1)$*-digit* $u_0, \ldots, u_{n-2}$ *appearing t times in a period of it, there exists a preference function which generates the sequence, using the initial word with the first* $n-1$ *digits being* $u_0, \ldots, u_{n-2}$*.*

IV. A Characterization for the GPO Algorithm to Produce de Bruijn Sequences

Among all of the binary periodic sequences that the GPO Algorithm can generate, the most interesting sequences are the de Bruijn ones. We now investigate the conditions for the algorithm to generate **only** de Bruijn sequences. For any order n, this is equivalent to ensuring that all n-stage states are visited as the algorithm runs its course. It is clear that $\mathscr{G}_f$ contains at least one cycle or loop. To guarantee that the GPO Algorithm visits the states in a cycle or loop, the initial state must satisfy some conditions.

Lemma 7. *Let* $\mathscr{C}$ *be a cycle or a loop in a state graph* $\mathscr{G}_f$ *for a given standard* $f(x_0, x_1, \ldots, x_{n-1})$*. If the GPO Algorithm with input* $(f, \mathbf{u})$ *is to visit all states in* $\mathscr{C}$*, then* $\mathbf{u}$ *must also be in* $\mathscr{C}$*.*

Proof: We prove the contrapositive of the statement. Suppose that $\mathbf{u}$ is not in $\mathscr{C}$. If $\mathscr{C}$ is a loop, then the only state in $\mathscr{C}$ is either $\mathbf{0}^n$ or $\mathbf{1}^n$. We start with the loop $\mathscr{C}$ having $\mathbf{0}^n$ as the only state. Before reaching $\mathbf{0}^n$, the algorithm must have visited its predecessor $1\mathbf{0}^{n-1}$ first. We note that the other possible successor of $1\mathbf{0}^{n-1}$ is $\mathbf{0}^{n-1}1$, which has not appeared earlier because its two possible predecessors are $1\mathbf{0}^{n-1}$ and $\mathbf{0}^n$. Thus, the actual successor of $1\mathbf{0}^{n-1}$ must have been $\mathbf{0}^{n-1}1$, instead of $\mathbf{0}^n$, leaving the latter out. So $\mathbf{0}^n$ would never be visited by the GPO Algorithm and, hence, the generated sequence is not de Bruijn. The case of the loop $\mathscr{C}$ having $\mathbf{u} = \mathbf{1}^n$ as the only state can be similarly argued.

If $\mathscr{C}$ is a cycle, then let $\mathbf{a}$ be the first state that the algorithm visits from among all of the states in $\mathscr{C}$. Lemma 5 says that both of $\mathbf{a}$'s children must have been visited before. One of the two, however, must also belong to $\mathscr{C}$, contradicting the assumption that $\mathbf{a}$ is the first. ■

Using the same method in the proof of Lemma 7, we can directly infer the next result, whose proof is omitted here for brevity.

Lemma 8. *Let $f(x_0,x_1,\dots,x_{n-1})$ be in the standard form. If there are $t>1$ cycles or loops in $\mathscr{G}_f$, then, taking an n-stage state in any one of the cycles or loops as the initial state, the GPO Algorithm will not visit any of the states belonging to the other cycles or loops.*

The following lemma characterizes the input on which the GPO Algorithm outputs a de Bruijn sequence.

Lemma 9. *On input a standard $f(x_0,x_1,\dots,x_{n-1})$ and an n-stage state $\mathbf{u}$, the GPO Algorithm generates a binary de Bruijn sequence of order n if $\mathscr{G}_f$ satisfies one of the following equivalent conditions.*

1) *There is a unique directed path from any state $\mathbf{v} \neq \mathbf{u}$ to $\mathbf{u}$.*
2) *There is a unique cycle or loop in $\mathscr{G}_f$. This cycle or loop contains $\mathbf{u}$.*

Proof: Suppose that there is a unique directed path from any state $\mathbf{v} \neq \mathbf{u}$ to $\mathbf{u}$ in $\mathscr{G}_f$. Then any $\mathbf{v}$ can be viewed as a descendent of $\mathbf{u}$. Lemma 5 implies that by the time the algorithm revisits $\mathbf{u}$, it must have visited $\mathbf{u}$'s two children, if $\mathbf{u}$ is in a cycle. If $\mathbf{u}$ is in a loop instead, then it must have visited $\mathbf{u}$'s only child. Applying the lemma recursively, the grandchildren of $\mathbf{u}$ must have all been visited prior to that. Continuing the process, we confirm that all of $\mathbf{u}$'s descendants must have been covered in the running of the algorithm. Thus, the generated sequence is de Bruijn of order n. ■

We combine Lemmas 7, 8 and 9 to obtain the following result.

Theorem 10. *Let a state $\mathbf{u}=(u_0,u_1,\dots,u_{n-1})$ and a standard $f(x_0,x_1,\dots,x_{n-1})$ be given as the input. Then the GPO Algorithm generates a binary de Bruijn sequence of order n if and only if there is a unique directed path from any state $\mathbf{v} \neq \mathbf{u}$ to $\mathbf{u}$ in $\mathscr{G}_f$. The condition is equivalent to the existence of a unique cycle or loop in $\mathscr{G}_f$ with the property that $\mathbf{u}$ is contained in this unique cycle or loop in $\mathscr{G}_f$.*

It is by now clear that, to generate a de Bruijn sequence by the GPO Algorithm, it suffices to find a pair $(f,\mathbf{u})$ that satisfies the requirement of Theorem 10. This general task is both technically important and practically interesting.

Example 1. *Let $f(x_0,\dots,x_{n-1})=0$. The state graph $\mathscr{G}_f$ contains only one loop (0), from $\mathbf{0}^n$ to itself. All other states are descendants of $\mathbf{0}^n$. Taking $\mathbf{u}=\mathbf{0}^n$, the GPO algorithm produces the* `Prefer-One` *de Bruijn sequence [11]. When $n=4$, the sequence is* (0000 1111 0110 0101)*. Similarly, let $f(x_0,\dots,x_{n-1})=1$ and $\mathbf{u}=\mathbf{1}^n$. Then the algorithm produces the* `Prefer-Zero` *de Bruijn sequence [21], which is the complement of the* `Prefer-One` *sequence.* □

Theorem 10 leads to the next two corollaries.

Corollary 11. *[24, Theorem 4] Let $n>m\geq 2$ and $h(x_0,x_1,\dots,x_{m-1})$ be a feedback function whose FSR generates a de Bruijn sequence $\mathbf{s}_m$ of order m. Let*

$$f(x_0,x_1,\dots,x_{n-1}) := h(x_{n-m},x_{n-m+1},\dots,x_{n-1})$$

and $\mathbf{u}$ be any n-stage state of $\mathbf{s}_m$. Then the GPO Algorithm, on input $(f,\mathbf{u})$, generates a de Bruijn sequence of order n.

Proof: The unique cycle in $\mathscr{G}_f$ is $\mathbf{s}_m$ and the initial state $\mathbf{u}$ is in this cycle. ■

Example 2. *An order 4 de Bruijn sequence* (0000 1001 1010 1111) *is produced by the FSR with*

$$g(x_0,x_1,x_2,x_3) := 1+x_0+x_2+x_3+x_1\cdot x_2+x_1\cdot x_3+x_2\cdot x_3+x_1\cdot x_2\cdot x_3.$$

Letting $n=6$ implies $f(x_0,x_1,x_2,x_3,x_4,x_5)=1+x_2+x_4+x_5+x_3\cdot x_4+x_3\cdot x_5+x_4\cdot x_5+x_3\cdot x_4\cdot x_5$. Adding $\mathbf{b}=000010$ to the input, the GPO Algorithm yields the de Bruijn sequence

(00001010 00111011 00101101 11001111 11010010 00000110 00100110 10101111).

□

Corollary 12. *Suppose that a standard $f_1(x_0,x_1,\dots,x_{m-1})$ can be used in the GPO Algorithm to generate de Bruijn sequences of order m. Let $n>m$ and*

$$f_2(x_0,x_1,\dots,x_{n-1}) := f_1(x_{n-m},x_{n-m+1},\dots,x_{n-1}).$$

Then f_2 can be used in the GPO Algorithm to generate de Bruijn sequences of order n.

Proof: Theorem 10 implies the existence of a unique cycle or loop in $\mathscr{G}_{f_1}$. This, in turn, implies the existence of a unique cycle or loop in $\mathscr{G}_{f_2}$. Let $\mathbf{u}$ be any n-stage state in the cycle or loop in $\mathscr{G}_{f_2}$. The conclusion follows by applying Theorem 10 to the input pair $(f_2,\mathbf{u})$. ■

Remark 2. *If f_1 is not standard, then Corollary 12 fails to hold in general. For a counterexample, let $f_1(x_0,\dots,x_{m-1})=\prod_{i=0}^{m-1} x_i$. There are two loops, namely (0) and (1) in $\mathscr{G}_{f_1}$. The GPO Algorithm with $(f_1,\mathbf{0}^n)$ produces the* `prefer-one` *de Bruijn sequence of order m. Let $n>m$ and $f_2(x_0,x_1,\dots,x_{n-1}) := f_1(x_{n-m},x_{n-m+1},\dots,x_{n-1})=\prod_{i=n-m}^{n-1} x_i$. Since the GPO Algorithm, on input $(f_2,\mathbf{0}^n)$, cannot visit $\mathbf{1}^n$, the resulting sequence is not de Bruijn.*

We have now achieved our first objective of providing a thorough treatment on when the GPO Algorithm generates de Bruijn sequences. As a consequence, we see that all prior greedy algorithms, including those from preference functions, and their respective generalizations are special cases of the GPO Algorithm, once the respective feedback function and initial state pairs are suitably chosen. The three families of sequences treated in [24, Section 3] and those produced from the input pairs listed in [24, Table 2] are examples of de Bruijn sequences whose construction fall into our general framework of `Prefer-Opposite`. Here we offer another class of examples consisting of feedback functions that satisfy a simple number theoretic constraint on the indices of their variables.

We look into the feedback function of the form

$$f(x_0,x_1,\ldots,x_{n-1}) = \prod_{j=1}^{n-1} x_j + x_k \cdot x_\ell \text{ for } 0<k<\ell<n \tag{3}$$

and characterize pairs k and ℓ such that the connected state graph $\mathscr{G}_f$ contains a unique loop (0).

Proposition 13. *There is a unique cycle, which is the loop* (0)*, in the state graph* $\mathscr{G}_f$ *of the feedback function given in Equation (3) if and only if* $\gcd(n-k,\ell-k)=1$.

Proof: We define the following three feedback functions.

$$\begin{aligned}
f_1(x_0,x_1,\ldots,x_{n-2}) &:= x_0 \cdot x_1 \cdots x_{n-2} + x_{k-1}\cdot x_{\ell-1},\\
f_2(x_0,x_1,\ldots,x_{n-2}) &:= x_{k-1}\cdot x_{\ell-1}, \text{ and}\\
f_3(x_0,x_1,\ldots,x_{n-k-1}) &:= x_0 \cdot x_{\ell-k}.
\end{aligned}$$

Vertices in $\mathscr{G}_{f_1}$ and $\mathscr{G}_{f_2}$ are $(n-1)$-stage states, while those in $\mathscr{G}_{f_3}$ are $(n-k)$-stage states. It is clear that the following equivalences hold. The only cycle in $\mathscr{G}_f$ is the loop (0) if and only if $\mathscr{G}_{f_1}$ has a unique cycle (0) if and only if $\mathscr{G}_{f_2}$ has exactly two cycles (0) and (1) if and only if $\mathscr{G}_{f_3}$ has exactly two cycles (0) and (1). Hence, it suffices to establish that $\mathscr{G}_{f_3}$ has exactly the two cycles (0) and (1) if and only if $\gcd(n-k,\ell-k)=1$.

It is immediate to verify that the two loops (0) and (1) are in $\mathscr{G}_{f_3}$. We show that there exists another cycle in it if and only if $\gcd(n-k,\ell-k)>1$.

The output sequence $\mathbf{s}=s_0,s_1,s_2,\ldots,$ of the $(n-k)$-stage FSR with feedback function f_3 has $s_{i+n-k}=1$ if and only if $s_i=s_{i+\ell-k}=1$, for any $i\geq 0$. Suppose that there is another cycle $C=(a_0,a_1,\ldots,a_{N-1})$ in $\mathscr{G}_{f_3}$ and $N>1$ is its least period. If $a_i=1$ for some integer i, then we have

$$1=a_i=a_{i-(n-\ell)}=a_{i-(n-k)}=a_{i-2(n-\ell)}=a_{i-2(n-k)}=\cdots.$$

Inductively we can deduce that if $a_i=1$ then

$$1=a_i=a_{i-t(n-\ell)}=a_{i-t(n-k)} \text{ for all } t=0,1,2,\ldots. \tag{4}$$

It is immediate to confirm that if $a_i=0$, then $a_{i-t(n-\ell)}=a_{i-t(n-k)}=0$, since $a_{i-t(n-\ell)}=1$ or $a_{i-t(n-k)}=1$ implies $a_i=1$ by Equation (4). So the cycle C must satisfy

$$a_i=a_{i-t(n-\ell)}=a_{i-t(n-k)} \text{ for all } t=0,1,2,\ldots,$$

which means that the period N divides $n-k$ and $n-\ell$ simultaneously. Thus

$$\gcd(n-k,n-\ell)=\gcd(n-k,\ell-k)>1.$$

Conversely, if $\gcd(n-k,\ell-k)=t>1$, then $\mathscr{G}_{f_3}$ contains the cycle

$$(\underbrace{10\cdots0}_{t}\ldots\underbrace{10\cdots0}_{t}),$$

establishing the existence of at least another cycle when $\gcd(n-k,\ell-k)>1$. ■

Whenever $\gcd(n-k,\ell-k)=1$, the GPO Algorithm can use the feedback function f in Equation (3) as an input. Otherwise, the function can be used as an input in the GJPO Algorithm, to be introduced later. There is a combinatorial identity that connects $\gcd(n-k,\ell-k)$ with the number of cycles in $\mathscr{G}_f$. The number of cycles is the number of cyclic decomposition of $\gcd(n-k,\ell-k)$. For example, when $n=11$ with $(k,\ell)=(1,6)$, we have $\gcd(10,5)=5$, so there are 7 cycles. The details can be found, *e.g.*, in Sequence A008965 in OEIS [27].

The next result generalizes Proposition 13. The proof follows similarly and, thus, is omitted.

Theorem 14. *The state graph of the FSR with feedback function*

$$f(x_0,x_1,\ldots,x_{n-1}) = \prod_{j=1}^{n-1} x_j + x_{k_1}\cdot x_{k_2}\cdots x_{k_t} \textit{ for } 0<k_1<k_2<\ldots<k_t<n, \tag{5}$$

has a unique loop (0) *if and only if* $\gcd(n-k_1,k_2-k_1,\ldots,k_t-k_1)=1$.

V. Graph Joining and the GPO Algorithm

A typical standard feedback function $f(x_0,\ldots,x_{n-1})$ often fails to satisfy the condition in Theorem 10, *i.e.*, its $\mathscr{G}_f$ tends to decompose into $t \geq 2$ *components*, each with a unique cycle or loop. Recall that the components of any non-null graph in its decomposition are non-null detached subgraphs, no two of them share any edge or vertex in common. The graph is then said to be a *union* of its components. When $\mathscr{G}_f$ has $t \geq 2$ components, then the sequence produced by the GPO Algorithm is not de Bruijn. This section introduces a new method of joining the components to turn the outputs into de Bruijn sequences.

Suppose that, for a given standard f, its $\mathscr{G}_f$ has components $G_1,G_2,\ldots,G_t$. Let C_i be the unique cycle or loop in G_i for each $1 \leq i \leq t$. Let us take any n-stage state $\mathbf{u}$ in C_1 as the initial state to run the GPO Algorithm. Lemmas 7 and 8 tell us that, as the sequence is being generated, the algorithm visits all of the states in G_1 but none of the states in the cycles or loops $C_2,\ldots,C_t$. We now modify the algorithm so that it can continue to cover all of the remaining states in $\mathscr{G}_f$ before revisiting $\mathbf{u}$.

Lemma 15. *For a given standard f, let $G_1,G_2,\ldots,G_t$ be the components in $\mathscr{G}_f$. Let C_i be the unique cycle or loop in G_i, for each i. Let $(f,\mathbf{u})$ be the input of the GPO Algorithm where $\mathbf{u}$ is a state in C_1. Let $\mathbf{w}$ in C_j be an n-stage state such that its companion state $\widetilde{\mathbf{w}}$ is a leaf in G_1, for some $j \in \{2,3,\ldots,t\}$. Let $\mathbf{v}$ be the child of $\mathbf{w}$ that does not belong to C_j. Then the GPO Algorithm can be modified to visit all of the states in G_1 and in G_j if we assign $\mathbf{w}$ to be the successor of $\mathbf{v}$.*

Proof: Prior to the additional assignment rule, starting from $\mathbf{u}$ in C_1, the GPO Algorithm visits all states in G_1 but cannot cover any of the states in $C_2 \cup C_3 \cup \ldots \cup C_t$.

Suppose that C_j is a cycle. Note that $\mathbf{w}$ has two children, namely $\mathbf{v}$, which does not belong to C_j, and $\widehat{\mathbf{v}}$, which is in C_j. As the algorithm runs, let $\widetilde{\mathbf{w}}$ be its currently visited state. Since $\widehat{\mathbf{v}} \in C_j$, the algorithm cannot reach it by Lemma 8. Hence, the actual predecessor of $\widetilde{\mathbf{w}}$ in the algorithm's output must be $\mathbf{v}$. The new assignment rule, however, forces $\mathbf{w}$ to be the successor of $\mathbf{v}$. It is easy to check that after adding this assignment, the algorithm also terminates when it revisits $\mathbf{u}$ and any other n-stage state is visited at most once. To guarantee that the modified algorithm revisits $\mathbf{u}$, all the states in G_1 must be visited, including $\widetilde{\mathbf{w}}$. Its unique possible predecessor has to be $\widehat{\mathbf{v}}$, because the new assignment rule forces $\mathbf{v}$ to be the actual predecessor of $\mathbf{w}$. So we have deduced that $\widehat{\mathbf{v}} \in C_j$ must be visited by the algorithm. Applying Lemma 5, repeatedly if necessary, all states in G_j can be visited by the algorithm. All states in G_1 and G_j can thus be visited by the modified algorithm.

If C_j is a loop, then $\mathbf{w}$ and $\mathbf{v}$ are conjugate states. Adding the rule that assigns $\mathbf{w}$ to be the actual successor of $\mathbf{v}$ makes $\widetilde{\mathbf{w}}$ the successor of $\mathbf{w}$ in the output sequence. Once $\mathbf{w}$ is visited, $\mathbf{v}$ must have been visited, and all states in G_j can be reached by the modified algorithm. ■

The selection of states, including $\mathbf{w}$, $\widetilde{\mathbf{w}}$, $\mathbf{v}$ and $\widehat{\mathbf{v}}$, in the above proof will be illustrated in Figure 1 of Example 3. The first state in C_j that the modified algorithm visits must be $\mathbf{w}$. By Lemma 5, the order of appearance of the states in C_j in the output sequence follows the direction of their edges. The next state visited after $\mathbf{w}$ is the parent of $\mathbf{w}$ in C_j, and so on. The last to be visited is the child $\widehat{\mathbf{v}}$ of $\mathbf{w}$. This fact implies that distinct valid options for $\mathbf{w}$ in C_j lead to shift-inequivalent output sequences when the initial state $\mathbf{u}$ is fixed. Keeping everything fixed but changing the initial state to any other state in the same cycle C_1, however, may produce shift-equivalent output sequences. Such instances of collisions, albeit being rare, should be taken into account in the enumeration of inequivalent output sequences.

The process detailed in the proof of Lemma 15 explains the thinking behind our modification of the GPO Algorithm. Suppose that the modified algorithm can now cover all of the states in two subgraphs, say G_1 and G_2. We identify a state $\mathbf{x}$ in C_3 such that $\widetilde{\mathbf{x}}$ is a leaf in G_1 or G_2. If $\mathbf{y}$ is the child of $\mathbf{x}$ that does not belong to C_3, then we add the rule that assign $\mathbf{x}$ to be the successor of $\mathbf{y}$ to ensure that all states in G_3 can be visited. If such an assignment can be done to "join" $G_1,G_2,\ldots,G_t$ together, then the resulting sequence of the modified algorithm is de Bruijn of order n.

We outline the process of graph joining, from a given state graph of a standard feedback function, in the following steps.

Step 1 Choose an arbitrary component, say G_1. Set $\Omega \leftarrow \{G_1\}$.

Step 2 Find a component, say G_2, such that $\mathbf{w}_2$ is a state in C_2, $\widetilde{\mathbf{w}_2}$ is a leaf of G_1, and $\mathbf{v}_2$ is a child of $\mathbf{w}_2$ which is not in C_2. Set $\Omega \leftarrow \Omega \cup \{G_2\}$.

Step 3 Continue inductively by finding a G_i such that $\mathbf{w}_i$ is a state in C_i, $\widetilde{\mathbf{w}}_i$ is a leaf of some subgraph in Ω, and $\mathbf{v}_i$ is a child of $\mathbf{w}_i$ which is not in C_i. Set $\Omega \leftarrow \Omega \cup \{G_i\}$.

Step 4 End if Ω contains all of the component graphs. Otherwise declare failure since the output will not be de Bruijn.

Generate Run the GPO Algorithm on an arbitrary state $\mathbf{u}$ in C_1 as the initial state. Upon reaching $\mathbf{v}_i$ for any $i = 2,\ldots,t$ as the current state, the usual procedure in the algorithm is modified such that the next state is assigned to be $\mathbf{w}_i$. In all other occasions, comply with the rules of the algorithm to generate the next state. Output a de Bruijn sequence.

Example 3. *Let $f(x_0,x_1,x_2,x_3) = x_1 + x_2 \cdot x_3$. Then $\mathscr{G}_f$ is divided into three components, namely, G_1 with $C_1 = (1110)$, G_2 with $C_2 = (010)$, and G_3 with $C_3 = (0)$. Figure 1 shows $\mathscr{G}_f$ with a configuration that allows for the merging of the three components using the added assignments.*

Taking as $\mathbf{u}$ the respective four states in C_1, namely $1110, 1101,$ $1011,$ 0111*, we obtain three inequivalent de Bruijn sequences. On* $\mathbf{u} = 1110$*, the states are visited in the following order. We use $\Rightarrow$ to indicate that the state transition is governed*

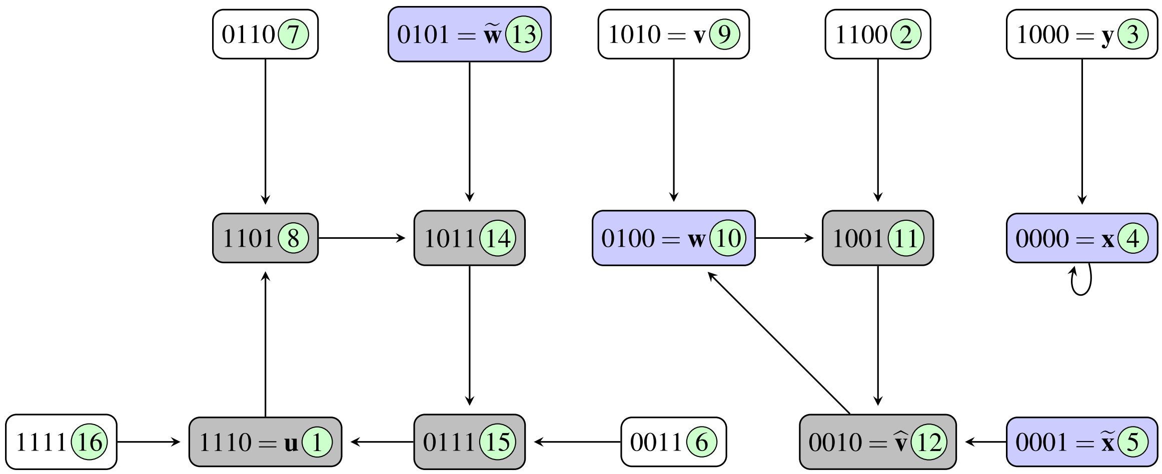


Fig. 1: The state graph $\mathscr{G}_f$ for $f(x_0,x_1,\ldots,x_3) = x_1 + x_2 \cdot x_3$. The components G_1, G_2, G_3 are from left to right. The encircled label indicates the state's order of appearance in the de Bruijn sequence generated on the initial state $\mathbf{u}$.

by an additional assignment rule.

$$\begin{aligned} \mathbf{u} = & 1110 \rightarrow 1100 \rightarrow 1000 \Rightarrow 0000 \rightarrow 0001 \rightarrow 0011 \rightarrow 0110 \rightarrow 1101 \rightarrow \\ & 1010 \Rightarrow 0100 \rightarrow 1001 \rightarrow 0010 \rightarrow 0101 \rightarrow 1011 \rightarrow 0111 \rightarrow 1111 \rightarrow \mathbf{u}. \end{aligned}$$

The output sequence is (1110 0001 1010 0101)*. Choosing* $\mathbf{u} = 0111$*, the output sequence* (0111 1000 0110 1001) *is shift-equivalent to the previous one,* i.e.*, here there is a collision. The other two inequivalent output sequences are, respectively,* (1101 0000 1100 1011) *on* $\mathbf{u} = 1101$ *and* (1011 0000 1111 0100) *on* $\mathbf{u} = 1011$*.*

Suppose that we change $\mathbf{w}$ *from* 0100 *to* 0010*, making* $\widetilde{\mathbf{w}} = 0011$*. The output sequences* (1110 0001 0100 1101) *and* (0111 1000 0101 0011)*, on respective initial states* 1110 *and* 0111*, are again shift-equivalent. The other two outputs are* (1011 0000 1010 0111) *on* $\mathbf{u} = 1011$ *and* (1101 0110 0001 0011) *on* $\mathbf{u} = 1101$*. In total, we have generated six shift-inequivalent de Bruijn sequences of order* 4 *in this example.* □

The underlying idea of our *graph joining method* is similar with that of the *cycle joining method* [12], which is the main tool in numerous constructions of de Bruijn sequences. While there are too many references to mention, a list of representative works where the tool plays a crucial role can be found in [14, Table IV]. We briefly recall how the cycle joining method works.

The state graph of an FSR with a **nonsingular** feedback function $h(x_0,x_1,\ldots,x_{n-1})$ is divided into disjoint cycles. If an n-stage state $\mathbf{w}$ and its companion state $\widetilde{\mathbf{w}}$ belong to two distinct cycles, then the cycles can be joined by exchanging the predecessors of $\mathbf{w}$ and $\widetilde{\mathbf{w}}$. Such pair of cycles are said to be *adjacent* and the pair $(\mathbf{w}, \widetilde{\mathbf{w}})$ is called a *companion pair*. The process is repeated to join the rest of the cycles by identifying suitable companion pair(s). If all of the initially disjoint cycles can be joined together, then the final cycle must be a de Bruijn sequence of order n.

For an FSR with nonsingular feedback function h, its *adjacency graph* is an *undirected* multigraph whose vertices correspond to the cycles in the corresponding state graph. There exists an edge between two vertices if and only if they share a companion pair. The number of shared companion pairs labels the edge. The number of resulting inequivalent de Bruijn sequences is the number of spanning trees in the adjacency graph.

We use similar concepts to formally define our graph joining method. Given a standard f, suppose that $\mathscr{G}_f$ consists of t components $G_1,\ldots,G_t$. Let $(\mathbf{w}, \widetilde{\mathbf{w}})$ be a companion pair with $\mathbf{w}$ a state in the cycle (or loop) C_i of G_i and $\widetilde{\mathbf{w}}$ a leaf in G_j for $1 \leq i \neq j \leq t$. Companion pairs play an important role in our new method because they can modify the GPO Algorithm to visit all states in G_i and G_j when the initial state is one of the state(s) in C_j. We call such $(\mathbf{w}, \widetilde{\mathbf{w}})_{i,j}$ pair a *preference companion pair* (PCP) **from** G_i **to** G_j. Each state in the pair is the other's *preference companion state*. Two components G_i and G_j are *adjacent* if they share a PCP $(\mathbf{w}, \widetilde{\mathbf{w}})_{i,j}$. The pair joins adjacent components into one.

Definition 3. *The preference adjacency graph (PAG) for an FSR with a standard feedback function f is a* directed multigraph *$\mathbb{G}_f$ whose t vertices correspond to the t components $G_1,\ldots,G_t$ in $\mathscr{G}_f$. There exists a directed edge from G_i to G_j if and only if there exists a PCP $(\mathbf{w}, \widetilde{\mathbf{w}})_{i,j}$. The number of directed edges, each identified by a PCP, from G_i to G_j is the number of their PCPs.*

By definition, $\mathbb{G}_f$ contains no loops. A *rooted spanning tree* in $\mathbb{G}_f$ is defined to be a rooted tree that contains all of $\mathbb{G}_f$'s vertices. We use $\Upsilon_{\mathbb{G}}$, or simply Υ if $\mathbb{G}_f$ is clear from the context, to denote a rooted spanning tree in $\mathbb{G}_f$. By Lemma 15, once we obtain a rooted spanning tree in $\mathbb{G}_f$, with the root being in some cycle C_k, then we can run the GPO Algorithm with

Algorithm 2 `Graph Joining Prefer-Opposite` (GJPO)

Input: A standard feedback function $f(x_0, x_1, \ldots, x_{n-1})$.
Output: de Bruijn sequences.

1: Construct the state graph $\mathscr{G}_f$
2: $V_{\mathbb{G}} \leftarrow \{G_1, G_2, \ldots, G_t\}$ and $\mathscr{C} \leftarrow \{\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_t\}$ ▷ $V_{\mathbb{G}}$ is vertex set of $\mathbb{G}$
3: **for** i from 1 to t **do** ▷ Populate the edges in $\mathbb{G}$
4:     **for** every state $\mathbf{w} \in \mathscr{C}_i$ **do**
5:         **if** $\widetilde{\mathbf{w}}$ is a leaf in $\mathscr{C}_j$ for $j \neq i$ **then**
6:             add edge in $\mathbb{G}$ from G_i to G_j labelled by $\mathbf{w}$
7:     $K_{i,j} \leftarrow$ the list of PCPs from G_i to G_j for $i \neq j$
8: Derive the simplified undirected graph $\mathbb{H}$ from $\mathbb{G}$: The vertex set of $\mathbb{H}$ is the vertex set of $\mathbb{G}$. If $K_{i,j}$ is nonempty for $1 \leq i \neq j \leq t$, then add an undirected edge $e_{i,j}$ in $\mathbb{H}$.
9: Generate the set $\Gamma_{\mathbb{H}}$ of all spanning trees in $\mathbb{H}$ ▷ We use [14, Algorithm 5]
10: **for** each $\Upsilon \in \Gamma_{\mathbb{H}}$ **do**
11:     $E_{\Upsilon} \leftarrow$ the set $\{e_1, e_2, \ldots, e_{t-1}\}$ of edges in Υ
12:     **for** k from 1 to $t-1$ **do**
13:         **for** each of the two possible directions for e_k **do** ▷ from G_i to G_j or G_j to G_i
14:             **if** the corresponding $K_{i,j}$ (respectively $K_{j,i}$) is nonempty **then**
15:                 choose each of its element, in sequence, as an edge in the directed tree
16:     $\Sigma_{\Upsilon} \leftarrow$ all directed trees in $\mathbb{G}$ that corresponds to Υ
17:     **for** each $\Lambda \in \Sigma_{\Upsilon}$ **do**
18:         **if** Λ is not a rooted tree **then**
19:             $\Sigma_{\Upsilon} \leftarrow \Sigma_{\Upsilon} \setminus \Lambda$
20: $\Omega \leftarrow \bigcup_{\Upsilon \in \Gamma_{\mathbb{H}}} \Sigma_{\Upsilon}$ ▷ The set of all rooted spanning trees in $\mathbb{G}$
21: **for** each $\Delta \in \Omega$ **do**
22:     $E_{\Delta} \leftarrow$ set of states that label the directed edges in Δ
23:     **for** each state $\mathbf{u}$ in $\mathscr{C}$ in the root component G of Δ **do**
24:         $\mathbf{c} := c_0, c_1, \ldots, c_{n-1} \leftarrow \mathbf{u}$
25:         **do**
26:             Print(c_0)
27:             **if** $c_1, c_2, \ldots, c_{n-1}, f(c_0, c_1, \ldots, c_{n-1}) \notin E_{\Delta}$ **then**
28:                 **if** $c_1, c_2, \ldots, c_{n-1}, \overline{f(c_0, c_1, \ldots, c_{n-1})}$ has not appeared before **then**
29:                     $\mathbf{c} \leftarrow c_1, c_2, \ldots, c_{n-1}, \overline{f(c_0, c_1, \ldots, c_{n-1})}$
30:                 **else**
31:                     $\mathbf{c} \leftarrow c_1, c_2, \ldots, c_{n-1}, f(c_0, c_1, \ldots, c_{n-1})$
32:             **else**
33:                 $\mathbf{c} \leftarrow c_1, c_2, \ldots, c_{n-1}, f(c_0, c_1, \ldots, c_{n-1})$ ▷ The added assignment
34:                 $E_{\Delta} \leftarrow E_{\Delta} \setminus \{\mathbf{c}\}$
35:         **while** $\mathbf{c} \neq \mathbf{u}$

an arbitrary state in C_k as the initial state. Suppose that three conditions are satisfied. First, the current state is $\mathbf{v}$, which is the predecessor of $\mathbf{w}$. Second, $\mathbf{v}$ is not in the cycle that contains $\mathbf{w}$. Third, $(\mathbf{w}, \widetilde{\mathbf{w}})_{i,j}$ is an edge in the spanning tree. Then we assign $\mathbf{w}$ to be the successor of $\mathbf{v}$. This ensures that the resulting sequence is de Bruijn.

We modify the GPO Algorithm by adding the graph joining method in the assignment rule, guided by the relevant PCPs. We call the modified algorithm the `Graph Joining Prefer-Opposite` (GJPO) Algorithm, presented here as Algorithm 2. The algorithm sets out to find all preference companion pairs, before determining all possible rooted spanning trees. Once this is done and if there exists at least one rooted spanning tree, then we can choose one such tree, either deterministically or randomly, and generate the corresponding Bruijn sequence.

To analyze its complexity, we break the GJPO Algorithm down into five stages.

Stage 1 Described in Line 1, the state graph $\mathscr{G}_f$ is built in $O(n)$ time. In its most basic implementation, storing this graph is not efficient, requiring $O(2^n)$ space. One can reduce this memory requirement by storing, per graph components, only its cycle and a list of leaves. The saving varies, depending on f, and is not so easy to quantify. Checking whether a vertex state $c_1, c_2, \ldots, c_{n-1}, y$, for some $y \in \mathbb{F}_2$, is a non-leaf is very efficient since one only needs to determine if evaluating the algebraic normal form on inputs $x, c_1, c_2, \ldots, c_{n-1}$, for both $x \in \mathbb{F}_2$, yields the specified state.

Stage 2 The procedure in Lines 2 to 7 builds the PAG $\mathbb{G}$. For each component graph, the number of operations to perform is roughly $(t-1)$ times the period of its cycle, giving the total time estimate to be $(t-1)$ times the sum of the periods of the t cycles. Storing $K_{i,j}$ for $1 \leq i \neq j \leq t$ needs $O(t^3)$ space since there are $t^2 - t$ entries, each having at most $\max\{|K_{i,j}|\} \leq t-1$ elements.

Stage 3 Once $\mathbb{G}$ has been determined, the procedure in Line 8 derives a simpler graph $\mathbb{H}$, taking roughly $O(t^2)$ in both time and storage, to use as an input in an intermediate step to eventually identify all rooted spanning trees in $\mathbb{G}$. In the simpler graph, the edge direction is removed and multiple edges are collapsed into a single edge. This is done since generating all spanning trees in a graph is quite costly. The time requirement, using [14, Algorithm 5], is in the order of the number of spanning trees in the input graph. The simpler $\mathbb{H}$ is faster to work on. Line 9 continues the process by generating all spanning trees in $\mathbb{H}$.

Stage 4 The steps given in Lines 10 to 20 combine $\Gamma_{\mathbb{H}}$ and the lists of PCPs stored in $K_{i,j}$, for all applicable i and j, to generate a set Ω of all rooted spanning trees in the PAG graph $\mathbb{G}$. The time complexity to generate Ω is $O(|\Gamma_{\mathbb{H}}|) \cdot 2^{t-1} \cdot t^3$. Storing it takes $O(|\Gamma_{\mathbb{H}}|) \cdot t$ space.

Stage 5 The routine described in Line 21 onward generates an actual de Bruijn sequence per identified rooted spanning tree. This process is negligible in resource requirements.

Generating the entire set Ω is often neither necessary nor desirable. Our presentation and analysis above are geared toward completeness, not practicality. In actual deployment, computational routines to identify only a required number of rooted spanning trees can be carefully designed for speed and space savings.

Theorem 16. *Given a standard feedback function $f(x_0, \ldots, x_{n-1})$, Algorithm 2 outputs de Bruijn sequences of order n if and only if the set $\Gamma_{\mathbb{H}}$ in Line 9 is nonempty.*

Proof: The correctness of Algorithm 2 follows from Lemma 15. ■

Suppose that, for a chosen PCP $(\mathbf{w}, \widetilde{\mathbf{w}})_{i,j}$ that connects G_i to G_j, $\mathbf{w}$ is the first state in C_i that Algorithm 2 visits. Then the following result is a direct consequence of Theorem 16.

Corollary 17. *The number of distinct rooted trees is a lower bound for the number of inequivalent de Bruijn sequences produced.*

Writing a basic implementation in `python 2.7` and feeding it many standard feedback functions lead us to an interesting computational observation. For the majority of the input functions, distinct choices of initial states lead to inequivalent de Bruijn sequences, for a fixed rooted spanning tree. We harvest many more valid sequences than the number of distinct rooted spanning trees in $\mathbb{G}_f$. This suggests the following problem.

Open Problem 1. *Let a standard feedback function f, or a class $\mathscr{F}$ of standard feedback functions, be given. Provide a closed formula, or a good estimate, of the number of inequivalent outputs.*

We require the spanning trees to use in Algorithm 2 to be the rooted ones only. Otherwise, there will be a cycle C_j, for some $1 \leq j \leq t$, whose states are never visited. This clearly implies that the resulting sequence fails to be de Bruijn.

Example 4. *We use the setup in Example 3 and refer to Figure 1 for the state graph $\mathscr{G}_f$. Note that the tree in Figure 2 is a non-rooted spanning tree in $\mathbb{G}_f$. We use the PCPs $(0100, 0101)_{2,1}$ and $(1001, 1000)_{2,3}$ as the directed edges in the tree.*

$$\boxed{G_1} \longleftarrow \boxed{G_2} \longrightarrow \boxed{G_3}$$

Fig. 2: A Non-rooted Spanning Tree in $\mathbb{G}_f$ for $f(x_0, x_1, \ldots, x_3) = x_1 + x_2 \cdot x_3$.

We choose $\mathbf{u} = 1011 \in C_1$ as the initial state and run the modified version of the GPO Algorithm, with the added assignment rules that the successor of 1100 *must be* 1001*, and the successor of* 1010 *must be* 0100*. The algorithm will never visit $C_3 = (0)$, producing the output* $(1011\ 0011\ 1101\ 000)$*. Now, if we replicate the process with $\mathbf{u} = 0000 \in C_3$, then the output sequence is* $(0000\ 1100\ 1010)$*, leaving the four states in $C_1 = (1011)$ out.* □

VI. The Preference Adjacency Graph of a Class of Feedback Functions

For an arbitrary standard feedback function f, determining $\mathscr{G}_f$, *e.g.*, identifying its cycle(s) or loop(s), is generally not an easy task. Some feedback functions cannot be used by Algorithm 2 to generate de Bruijn sequence, as illustrated by the next example.

Example 5. *Let $n \geq 4$ and $f(x_0, x_1, \ldots, x_{n-1}) := x_{n-3} \cdot x_{n-2} + x_{n-3} \cdot x_{n-1} + x_{n-2} \cdot x_{n-1}$. The state graph $\mathscr{G}_f$ has two loops, one in each of the two components, namely $C_1 = (0) \in G_1$ and $C_2 = (1) \in G_2$. The companion state $\mathbf{0}^{n-1}1$ of $\mathbf{0}^n \in C_1$ is in G_1 and, similarly, the companion state $\mathbf{1}^{n-1}0$ of $\mathbf{1}^n \in C_2$ is in G_2. Hence, we cannot join the two components.* □

In this section we examine the PAG of a class of feedback functions. Our main motivation is to identify connections between structures of different orders that help in building the PAG. Let $1 \leq m < n$ be integers. Let $h(x_0,\ldots,x_{m-1})$ be any given nonsingular feedback function. The preference adjacency graph $\mathbb{G}_{F_h}$ of F_h, which is given by

$$F_h(x_0,x_1,\ldots,x_{n-1}) := x_{n-m} + g(x_{n-m+1},\ldots,x_{n-1}) = h(x_{n-m},\ldots,x_{n-1}), \tag{6}$$

becomes easier to determine. Algorithm 2 can then use F_h to generate de Bruijn sequences.

Since h is nonsingular, the t components of $\mathscr{G}_h$ are cycles, say $C_{h,1},C_{h,2},\ldots,C_{h,t}$. The states in $\mathscr{G}_h$ are m-stage. These t cycles can be joined into a de Bruijn sequence of order m by the cycle joining method if there exists enough companion pairs $(\mathbf{w},\widetilde{\mathbf{w}})$ with $\mathbf{w} \in C_{h,k}$ and $\widetilde{\mathbf{w}} \in C_{h,\ell}$ between appropriate pairs $1 \leq k \neq \ell \leq t$.

Let us now consider $\mathscr{G}_{F_h}$, whose states are n-stage. It also has t components $G_1,G_2,\ldots,G_t$. For $1 \leq i \leq t$, let C_i be the unique cycle or loop in G_i, labelled based on the one-to-one correspondence between $C_{h,i}$ and C_i induced by how F_h is defined. The component subgraph G_i can be divided further into distinct rooted trees when the edges that connect the states in C_i are deleted. The respective roots are the states in C_i and each tree contains 2^{n-m} states. Furthermore, in each such tree, there are 2^{n-m-1} leaves. The *last m* consecutive bits in *each* of these leaves is the *first m* consecutive bits of the corresponding root, by Equation (6). In other words, for a tree with a state $a_0,a_1,\ldots,a_n$ in some C_i as the root, all of its 2^{n-m-1} leaves have the form

$$y_0,\ldots,y_{n-m-2},\overline{g(a_0,\ldots,a_{m-2})+a_{m-1}},a_0,a_1,\ldots,a_{m-1} \in \mathbb{F}_2^n,$$

where the choices for $y_0,\ldots,y_{n-m-2}$ range over all vectors in $\mathbb{F}_2^{n-m-1}$.

Suppose that $(\mathbf{w},\widetilde{\mathbf{w}})_{i,j}$ is a PCP, *i.e.*, there is a state $\mathbf{w} = w_0,\ldots,w_{n-1}$ in C_i such that $\widetilde{\mathbf{w}}$ is a leaf in G_j with $i \neq j$. By the above analysis, the m-stage state $w_{n-m},\ldots,\overline{w_{n-1}}$ is in $C_{h,j}$ while the consecutive bits $w_{n-m},\ldots,w_{n-1}$ is an m-stage state in $C_{h,i}$, and there is a one-to-one correspondence between this latter state and $\mathbf{w}$. Thus, a PCP from G_i to G_j in $\mathscr{G}_{F_h}$ uniquely determines an m-stage companion pair shared by $C_{h,i}$ and $C_{h,j}$ in $\mathscr{G}_h$.

On the other hand, for $1 \leq i \neq j \leq t$, any m-stage companion pair $\mathbf{v} = v_0,\ldots,v_{m-1} \in C_{h,i}$ and $\widetilde{\mathbf{v}} \in C_{h,j}$, corresponds to two PCPs in $\mathscr{G}_{F_h}$. The first PCP is $(\mathbf{w}_1,\widetilde{\mathbf{w}}_1)_{i,j}$. Here, $\mathbf{w}_1$ is an n-stage state in C_i with $v_0,\ldots,v_{m-1}$ as its last m consecutive bits and $\widetilde{\mathbf{w}}_1$ is a leaf in G_j. The second PCP is $(\mathbf{w}_2,\widetilde{\mathbf{w}}_2)_{j,i}$ such that $\mathbf{w}_2$ is an n-stage state in C_j with $v_0,\ldots,\overline{v_{m-1}}$ as its last m consecutive bits and $\widetilde{\mathbf{w}}_2$ is a leaf in G_i.

We have thus proved the following result.

Proposition 18. *Let $1 \leq m < n$ be integers. If we view each m-stage companion pair $(\mathbf{v},\widetilde{\mathbf{v}})$ between distinct cycles $C_{h,i}$ and $C_{h,j}$ in $\mathscr{G}_h$ as two distinct pairs $(\mathbf{v},\widetilde{\mathbf{v}})_{i,j}$ and $(\widetilde{\mathbf{v}},\mathbf{v})_{j,i}$, then there is a one-to-two correspondence between the set of all m-stage companion pairs in $\mathscr{G}_h$ and the set of all n-stage PCPs in $\mathscr{G}_{F_h}$.*

Let $\mathscr{A}_h$ be the adjacency graph of $h(x_0,\ldots,x_{m-1})$ as defined in the cycle joining method. Recall that $\mathscr{A}_h$ is, therefore, a simple undirected multigraph. Its vertices are the cycles $C_{h,1},\ldots,C_{h,t}$ and the edges represent companion pairs. Proposition 18 tells us that, if we replace each edge in $\mathscr{A}_h$ by bidirectional edges and each vertex $C_{h,i}$ by the corresponding G_i, then we obtain the PAG $\mathbb{G}_{F_h}$. The number M of distinct spanning trees in $\mathscr{A}_h$ can be computed by the well-known BEST Theorem, *e.g.*, as stated in [14, Theorem 1]. Because the spanning trees in $\mathbb{G}_{F_h}$ are directional, by taking distinct roots, the graph has $t \times M$ distinct rooted spanning trees in total.

Example 6. *Given $n = 5$ and $m = 4$, we have*

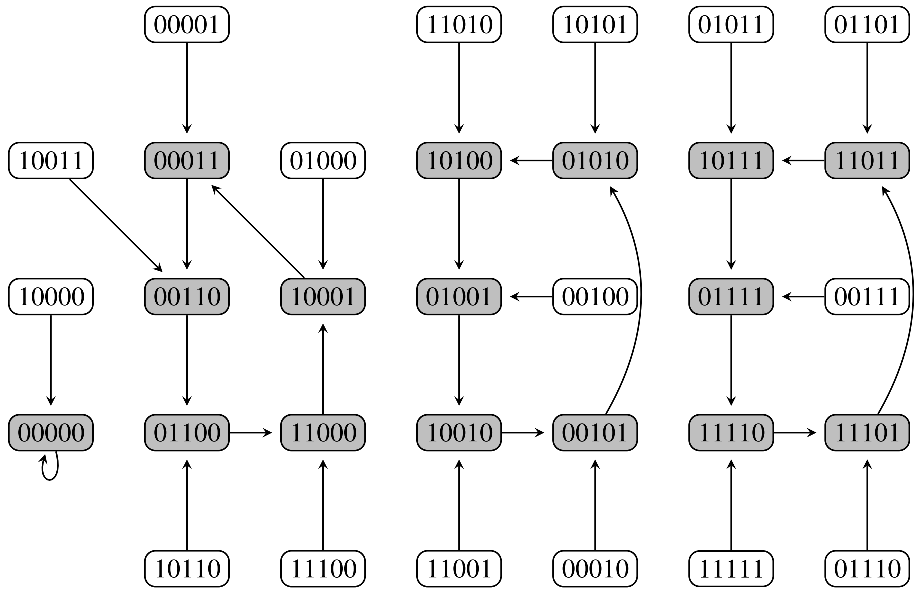
$$F_h(x_0,x_1,x_2,x_3,x_4) = x_1+x_2+x_3+x_4 \text{ when } h(x_0,x_1,x_2,x_3) = x_0+x_1+x_2+x_3.$$

Figure 3a presents $\mathscr{G}_{F_h}$. The cycles in $\mathscr{G}_h$, whose states are 4-state, are $C_{h,1} = (0)$, $C_{h,2} = (00011)$, $C_{h,3} = (00101)$, and $C_{h,4} = (01111)$. Figure 3b gives the compressed (undirected) adjacency graph $\mathscr{A}_h$. By computing the cofactor of any entry in its derived matrix

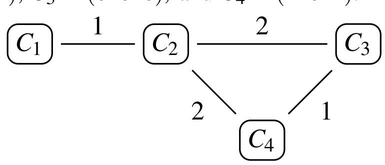
$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 5 & -2 & -2 \\ 0 & -2 & 3 & -1 \\ 0 & -2 & -1 & 3 \end{pmatrix},$$

we know that $\mathscr{A}_h$ has 8 distinct spanning trees. The compressed PAG $\mathbb{G}_{F_h}$, where multiple directed edges (if any) are compressed into one, is in Figure 4. It has 12 distinct types of rooted spanning trees, grouped based on the respective roots. The PCPs, for any (i,j) with $1 \leq i \neq j \leq 4$, can be easily determined from $\mathscr{G}_{F_h}$. Table I provides the list for ease of reference.

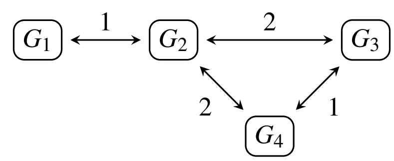
The number of de Bruijn sequences that Algorithm 2 outputs can now be easily determined. From Figure 4 (a) and since C_1 is a loop (0), the algorithm produces 8 de Bruijn sequences. There are five states each in C_k for $k \in \{2,3,4\}$. Hence, reading from Figures 4 (b), (c), and (d), the algorithm yields 120 more sequences. Alternatively, note that the cofactor of any entry in the derived matrix of $\mathbb{G}_{F_h}$ is 8. The product of this cofactor and the number of 5-stage states in the union of cycles, which is 16, is 128. These are the 128 choices for the input $(f,\mathbf{u})$ in the GJPO Algorithm. Since the number of distinct rooted spanning trees is $8 \times 4 = 32$, there are at least 32 inequivalent de Bruijn sequences among the 128. In fact, there are 70 sequences that appear once each. There are 23 sequences that appear twice each. There is one sequence that appears three, four, and five



(a) The state graph $\mathscr{G}_{F_h}$ of $f(x_0,x_1,x_2,x_3,x_4) = x_1 + x_2 + x_3 + x_4$ with the components G_1, G_2, G_3, and G_4 ordered left to right, *i.e.*, $C_1 = (0)$, $C_2 = (10001)$, $C_3 = (01010)$, and $C_4 = (11011)$.



(b) The compressed adjacency graph $\mathscr{A}_h$, where all edges between adjacent vertices are merged into one. The label gives the number of edges, *i.e.*, the number of shared companion pairs.



(c) The compressed preference adjacency graph $\mathbb{G}_{F_h}$, where all directed edges between adjacent vertices are merged into one. The label gives the number of PCP in *each* direction.

Fig. 3: Relevant graphs in Example 6.

TABLE I: List of Preference Companion Pairs for F_h in Example 6

(i,j)	PCPs $\{(\mathbf{w},\widetilde{\mathbf{w}})_{i,j}\}$	(i,j)	PCPs $\{(\mathbf{w},\widetilde{\mathbf{w}})_{i,j}\}$
$(1,2)$	$\{(00000,00001)\}$	$(2,1)$	$\{(10001,10000)\}$
$(2,3)$	$\{(00011,00010),(11000,11001)\}$	$(3,2)$	$\{(01001,01000),(10010,10011)\}$
$(2,4)$	$\{(00110,00111),(01100,01101)\}$	$(4,2)$	$\{(10111,10110),(11101,11100)\}$
$(3,4)$	$\{(01010,01011)\}$	$(4,3)$	$\{(11011,11010)\}$

times, respectively. These are (00000100 10111110 10100011 01100111)*,* (00000100 01110101 00110110 01011111)*, and* (00000101 11000111 11010100 11011001)*. Thus, Algorithm 2 produces* 96 *inequivalent de Bruijn sequences.* □

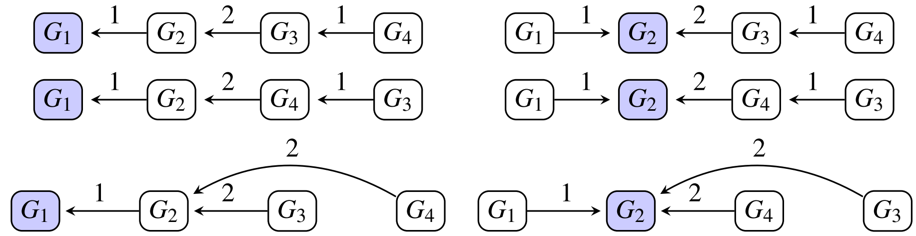
Computational evidences point to the following conjecture.

Conjecture 1. *Given the feedback function F_h, defined based on a nonsingular h in Equation (6) with $n \geq m+2$, distinct input pairs of rooted spanning tree and initial state generate inequivalent de Bruijn sequences of order n. This holds in both the GPO and the GJPO Algorithms.*

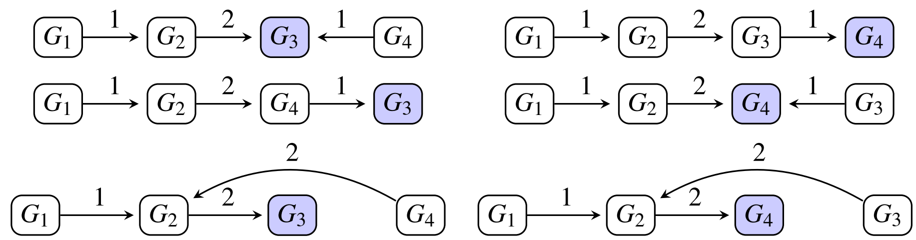
VII. Conclusion and Future Directions

We recap what this work has accomplished and point out several directions to explore. In terms of the GPO Algorithm, we have characterized a set of conditions for which the algorithm is certified to produce de Bruijn sequences. Carefully constructed classes of feedback functions, as demonstrated in Section VI, can be easily shown to meet the conditions.

The insights learned from studying the GPO Algorithm led us to a modification, which we call the Graph Joining Prefer Opposite (GJPO) Algorithm, that greatly enlarges the classes of feedback functions that can be used to greedily generate de Bruijn sequences. We adapt several key steps from the cycle joining method to join graph components in the state graph of any suitable function and initial state pair. The use of greedy algorithms to generate de Bruijn sequences remains of deep theoretical attraction, despite their practical drawbacks. Once some special state is visited, then we can be sure that all other



(a) Spanning Trees with Root G_1 (b) Spanning Trees with Root G_2



(c) Spanning Trees with Root G_3 (d) Spanning Trees with Root G_4

Fig. 4: List of 12 distinct types of rooted spanning trees in $\mathbb{G}_{F_h}$ with multiple edges compressed into one. The label above each directed edge is the number of PCPs to choose from.

states must have been visited before. This may be of independent interest and could be useful in other domains, *e.g.*, in the designs and verification of experiments.

There are several important challenges to overcome if we are to turn this novel idea of graph joining into a more practical tool. First, the task of identifying general classes of feedback functions whose respective state graphs are easy to characterize and efficient to store and manipulate may be the most important open direction. Accomplishing this can greatly reduces the complexity of the two proposed algorithms. We have discussed some examples of such classes, yet we believe that many better ones must exist and are waiting to be discovered. Second, instead of determining all preference companion pairs (PCPs), one can opt to seek for tools to quickly identify some of them to build a fast algorithm with low memory requirement. Third, studying the suitability of the resulting sequences for specific application domain(s) may appeal more to practitioners.

References

[1] N. G. de Bruijn, "A combinatorial problem," *Koninklijke Nederlandse Akademie v. Wetenschappen*, vol. 49, pp. 758–764, 1946.

[2] M. Stamp and C. Martin, "An algorithm for the k-error linear complexity of binary sequences with period 2^n," *IEEE Transactions on Information Theory*, vol. 39, no. 4, pp. 1398–1401, Jul 1993.

[3] M. Robshaw and O. Billet, Eds., *New Stream Cipher Designs*. Springer Berlin Heidelberg, 2008.

[4] B. Yang, K. Mandal, M. D. Aagaard, and G. Gong, "Efficient composited de Bruijn sequence generators," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1354–1368, Aug 2017.

[5] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, pp. 9748–9753, 2001.

[6] G. K. Aguirre, M. G. Mattar, and L. Magis-Weinberg, "De Bruijn cycles for neural decoding," *NeuroImage*, vol. 56, no. 3, pp. 1293–1300, Jun 2011.

[7] E. Scheinerman, "Determining planar location via complement-free de Brujin sequences using discrete optical sensors," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 883–889, 2001.

[8] A. M. Bruckstein, T. Etzion, R. Giryes, N. Gordon, R. J. Holt, and D. Shuldiner, "Simple and robust binary self-location patterns," *IEEE Transactions on Information Theory*, vol. 58, no. 7, pp. 4884–4889, Jul 2012.

[9] J. Pagès, J. Salvi, C. Collewet, and J. Forest, "Optimised de Bruijn patterns for one-shot shape acquisition," *Image and Vision Computing*, vol. 23, no. 8, pp. 707–720, Aug 2005.

[10] T. Gagie, "Bounds from a card trick," *Journal of Discrete Algorithms*, vol. 10, pp. 2–4, Jan 2012.

[11] H. Fredricksen, "A survey of full length nonlinear shift register cycle algorithms," *SIAM Review*, vol. 24, no. 2, pp. 195–221, 1982.

[12] S. W. Golomb, *Shift Register Sequences*, 3rd ed. World Scientific, 2017.

[13] T. Etzion and A. Lempel, "Algorithms for the generation of full-length shift-register sequences," *IEEE Transactions on Information Theory*, vol. 30, no. 3, pp. 480–484, May 1984.

[14] Z. Chang, M. F. Ezerman, S. Ling, and H. Wang, "On binary de Bruijn sequences from LFSRs with arbitrary characteristic polynomials," *Designs, Codes and Cryptography*, vol. 87, no. 5, pp. 1137–1160, May 2019.

[15] H. Fredricksen and J. Maiorana, "Necklaces of beads in k colors and k-ary de Bruijn sequences," *Discrete Mathematics*, vol. 23, no. 3, pp. 207 – 210, 1978.

[16] C. J. A. Jansen, W. G. Franx, and D. E. Boekee, "An efficient algorithm for the generation of de Bruijn cycles," *IEEE Transactions on Information Theory*, vol. 37, no. 5, pp. 1475–1478, Sep 1991.

[17] J. Sawada, A. Williams, and D. Wong, "A surprisingly simple de Bruijn sequence construction," *Discrete Mathematics*, vol. 339, no. 1, pp. 127–131, Jan 2016.

[18] D. Gabric, J. Sawada, A. Williams, and D. Wong, “A framework for constructing de Bruijn sequences via simple successor rules,” *Discrete Mathematics*, vol. 341, no. 11, pp. 2977–2987, Nov 2018.
[19] T. Helleseth and T. Klove, “The number of cross-join pairs in maximum length linear sequences,” *IEEE Transactions on Information Theory*, vol. 37, no. 6, pp. 1731–1733, Nov 1991.
[20] J. Mykkeltveit and J. Szmidt, “On cross joining de Bruijn sequences,” *Contemporary Mathematics*, vol. 632, pp. 333–344, 2015.
[21] M. H. Martin, “A problem in arrangements,” *Bulletin of the American Mathematical Society*, vol. 40, no. 12, pp. 859–865, Dec 1934.
[22] A. M. Alhakim, “A simple combinatorial algorithm for de Bruijn sequences,” *The American Mathematical Monthly*, vol. 117, no. 8, pp. 728–732, 2010.
[23] X. Wang, D. Wong, and W. G. Zhang, “A simple greedy de Bruijn sequence construction,” presented at Sequences and Their Applications (SETA) 2018.
[24] Z. Chang, M. Frederic Ezerman, and A. Aqsa Fahreza, “On greedy algorithms for binary de Bruijn sequences,” *arXiv e-prints*, arXiv:1902.08744, Feb 2019.
[25] A. Alhakim, “Spans of preference functions for de Bruijn sequences,” *Discrete Applied Mathematics*, vol. 160, no. 7, pp. 992 – 998, 2012.
[26] C. J. A. Jansen, “Investigations on nonlinear stream cipher systems: Construction and evaluation methods,” Ph.D. dissertation, Technical University of Delft, 1989.
[27] N. J. A. Sloane. The Online Encyclopedia of Integer Sequences: Number of necklaces of sets of beads containing a total of n beads. [Online]. Available: https://oeis.org/A008965