MATHEMATICAL STRUCTURES
IN COMPUTER SCIENCE

CAMBRIDGE
UNIVERSITY PRESS

A Gray Code for cross-bifix-free sets

Journal:	*Mathematical Structures in Computer Science*
Manuscript ID:	MSCS-2014-015
Manuscript Type:	XIV ICTCS Special Issue
Date Submitted by the Author:	19-Jan-2014
Complete List of Authors:	Bernini, Antonio; Universita degli Studi, Dipartimento di Matematica e Informatica Bilotta, Stefano; Universita degli Studi, Dipartimento di Matematica e Informatica Pinzani, Renzo; Universita degli Studi, Dipartimento di Matematica e Informatica Vajnovszki, Vincent; Le2i, Cs

Under consideration for publication in Math. Struct. in Comp. Science

A Gray Code for cross-bifix-free sets

A N T O N I O B E R N I N I[1], S T E F A N O B I L O T T A[1],
R E N Z O P I N Z A N I[1] and V I N C E N T V A J N O V S Z K I[2]

[1] *Dipartimento di Matematica e Informatica "U. Dini", Università degli Studi di Firenze, Viale G.B. Morgagni 65, 50134 Firenze, Italy.*
Email: antonio.bernini@unifi.it, stefano.bilotta@unifi.it, renzo.pinzani@unifi.it
[2] *LE2I, Université de Bourgogne, BP 47 870, 21078 Dijon Cedex, France*
Email: vvajnov@u-bourgogne.fr

Received 19 January 2014

A cross-bifix-free set of words is a set in which no prefix of any length of any word is the suffix of any other word in the set. A construction of cross-bifix-free sets has recently been proposed in (Chee *et al.* 2013) within a constant factor of optimality. We propose a *trace partitioned* Gray code for these cross-bifix-free sets and a CAT algorithm generating it.

1. Introduction

A cross-bifix-free set of words is a set where, given any two words over an alphabet, possibly the same, any prefix of the first one is not a suffix of the second one and *vice versa*. Cross-bifix-free sets are involved in the study of distributed sequences for frame synchronization (de Lind van Wijngaarden and Willink 2000). The problem of determining such sets is also related to several other scientific applications, for instance in pattern matching (Crochemore *et al.* 2007) and automata theory (Berstel *et al.* 2009).

Fixed the cardinality q of the alphabet and the length n of the words, a matter is the construction of a cross-bifix-free set with the cardinality as large as possible. An interesting method has been proposed in Bajic (2007) for words over a binary alphabet. In a recent paper (Chee *et al.* 2013) the authors revisit the construction of Bajic (2007) and generalize it obtaining cross-bifix-free sets of words with greater cardinality over an alphabet of arbitrary size. They also show that their cross-bifix-free sets have a cardinality close to the maximum possible; and to our knowledge this is the best result in literature about the size of cross-bifix-free sets.

It is worth to mention that an intermediate step between the original method (Bajic 2007) and its generalization (Chee *et al.* 2013) has been proposed by Bilotta *et al.* (2012): it is constituted by a different construction of binary cross-bifix-free sets based on lattice paths which allows to obtain greater cardinality if compared to the ones in Bajic (2007).

Once a class of objects is defined, in our case words, often it could be useful to list or generate them according to a particular criterion. A special way to do this is their

generation in a way such that any two consecutive words differ as little as possible, i.e., in Gray code order (Gray 1953). In the case the objects are words, as in our, we can specialize the concept of Gray code saying that it is *an infinite set of word-lists with unbounded word-length such that the Hamming distance between any two adjacent words is bounded independently of the word-length* (Walsh 2003) (the Hamming distance is the number of positions in which the two successive words differ (Hamming 1950)). Gray codes find useful applications in circuit testing, signal encoding, data compression, telegraphy, error correction in digital communication and others. They are also widely studied in the context of combinatorial objects as: permutations (Johnson 1963), Motzkin and Schröder words (Vajnovszki(2) 2001), derangements (Baril and Vajnovszki 2004), involutions (Walsh 2001), compositions, combinations, set-partitions (Ruskey 1993; Sagan 2010), and so on.

In this work we propose a Gray code for the cross-bifix-free set $S_{n,q}^{(k)}$ (Chee *et al.* 2013). It is formed by length n words over the q-ary alphabet $A = \{0, 1, \ldots, q-1\}$ containing a particular sub-word avoiding k consecutive 0's (for more details see the next section). First we propose a Gray code for $S_{n,2}^{(k)}$ over the binary alphabet $\{0, 1\}$, then we expand each binary word to the alphabet A. The expansion of a binary word α is obtained replacing all the 1's with the symbols of A different from 0 producing a set of words with the same *trace* α. The Gray code we get is *trace partitioned* in the sense that all the words with the same trace are consecutive.

2. Definitions and tools

Let $n \geq 3$, $q \geq 2$ and $1 \leq k \leq n-2$. The cross-bifix-free set $S_{n,q}^{(k)}$ is the set of all length n words $s_1 s_2 \cdots s_n$ over the alphabet $\{0, \ldots, q-1\}$ satisfying:

— $s_1 = \ldots = s_k = 0$;
— $s_{k+1} \neq 0$;
— $s_n \neq 0$;
— the subword $s_{k+2} \ldots s_{n-1}$ does not contain k consecutive 0's.

Throughout this paper we are going to use several standard notations which are typical in the framework of sets and lists of words. For the sake of clearness we summarize the ones used here.

For a set of words L over an alphabet A we denote by $\mathcal{L}$ an ordered list for L, and

— $\overline{\mathcal{L}}$ denotes the list obtained by covering $\mathcal{L}$ in reverse order;
— if $\mathcal{L}'$ is another list, then $\mathcal{L} \circ \mathcal{L}'$ is the concatenation of the two lists, obtained by appending the words of $\mathcal{L}'$ after those of $\mathcal{L}$;
— $\mathsf{first}(\mathcal{L})$ and $\mathsf{last}(\mathcal{L})$ are the first and the last word of $\mathcal{L}$, respectively;
— if u is a word in A^*, then $u \cdot \mathcal{L}$ (resp. $\mathcal{L} \cdot u$) is a new list where each word has the form $u\omega$ (resp. ωu) where ω is any word of $\mathcal{L}$;
— if u is a word in A^*, then $|u|$ is its length, and $u^n = \underbrace{uuu \ldots u}_{n}$.

For our purpose we need a Gray code list for the set of words of a certain length over the $(q-1)$-ary alphabet $\{1, 2, \ldots, q-1\}$, $q \geq 3$. An obvious generalization of the Binary Reflected Gray Code (Gray 1953) to the alphabet $\{1, 2, \ldots, q-1\}$ is the list $\mathcal{G}_{n,q}$ for the set of words $\{1, 2, \ldots, q-1\}^n$ defined by Er and Williamson (Er 1984; Williamson 1985) where is also shown that it is a Gray code with Hamming distance 1. The authors defined this list as:

$$\mathcal{G}_{n,q} = \begin{cases} \lambda & \text{if } n = 0, \\ 1 \cdot \mathcal{G}_{n-1,q} \circ 2 \cdot \overline{\mathcal{G}_{n-1,q}} \circ \cdots \circ (q-1) \cdot \mathcal{G}'_{n-1,q} & \text{if } n > 0, \end{cases} \tag{1}$$

where $\mathcal{G}'_{n-1,q}$ is $\mathcal{G}_{n-1,q}$ or $\overline{\mathcal{G}_{n-1,q}}$ according on whether q is even or odd. The reader can easily verify the following proposition.

Proposition 2.1. For $q \geq 3$,

— $\mathsf{first}(\mathcal{G}_{n,q}) = 1^n$;
— $\mathsf{last}(\mathcal{G}_{n,q}) = (q-1)1^{n-1}$ if q is odd, and $(q-1)^n$ if q is even.

Now we are going to present another tool we need in the paper. If β is a binary word of length n such that $|\beta|_1 = t$ (the number of 1's in β), we define the *expansion* of β, denoted by $\epsilon(\beta)$, as the list of $(q-1)^t$ words, where the i-th word is obtained by replacing the t 1's of β by the t symbols (read from left to right) of the i-th word in $\mathcal{G}_{t,q}$. For example, if $q = 3$ and $\beta = 01011$ (the trace), then $\mathcal{G}_{3,3} = (111, 112, 122, 121, 221, 222, 212, 211)$ and $\epsilon(\beta) = (01011, 01012, 01022, 01021, 02021, 02022, 02012, 02011)$. Notice that in particular $\mathsf{first}(\epsilon(\beta)) = \beta$ and all the words of $\epsilon(\beta)$ have the same trace.

We observe that $\epsilon(\beta)$ is the list obtained from $\mathcal{G}_{t,q}$ inserting some 0's, each time in the same positions. Since $\mathcal{G}_{t,q}$ is a Gray code and the insertions of the 0's does not change the Hamming distance between two successive word of $\epsilon(\beta)$ (which is 1), the following proposition holds.

Proposition 2.2. For any $q \geq 3$ and binary word β, the list $\epsilon(\beta)$ is a Gray code.

3. Trace partitioned Gray code for $S^{(k)}_{n,q}$

Our construction of a Gray code for the set $S^{(k)}_{n,q}$ of cross-bifix-free words is based on two other lists:

— $\mathcal{F}^{(k)}_n$, a Gray code for the set of binary words of length n avoiding k consecutive 0's, and
— $\mathcal{H}^{(k)}_{n,q}$, a Gray code for the set of q-ary words of length n which begin and end by a non zero value and avoiding k consecutive 0's. In particular, $\mathcal{H}^{(k)}_{n,2} = 1 \cdot \mathcal{F}^{(k)}_{n-2} \cdot 1$.

Finally, we will define the Gray code list $\mathcal{S}^{(k)}_{n,q}$ for the set $S^{(k)}_{n,q}$ as $0^k \cdot \mathcal{H}^{(k)}_{n-k,q}$.

3.1. *The list* $\mathcal{F}_n^{(k)}$

Let $\mathcal{C}_n$ be the list of binary words defined as:

$$\mathcal{C}_n = \begin{cases} \lambda & \text{if} \quad n = 0, \\ 1 \cdot \overline{\mathcal{C}_{n-1}} \circ 0 \cdot \mathcal{C}_{n-1} & \text{if} \quad n \geq 1, \end{cases} \tag{2}$$

with λ the empty word. The list $\mathcal{C}_n$ is a Gray code for the set $\{0,1\}^n$ and it is a slight modification of the original Binary Reflected Gray Code list defined in Gray (1953).

By the definition of $\mathcal{C}_n$ given in relation (2), we have for $n \geq 1$,

— $\mathsf{last}(\mathcal{C}_n) = 0 \cdot \mathsf{last}(\mathcal{C}_{n-1}) = 0^n$;
— $\mathsf{first}(\mathcal{C}_n) = 1 \cdot \mathsf{first}(\overline{\mathcal{C}_{n-1}}) = 1 \cdot \mathsf{last}(\mathcal{C}_{n-1}) = 10^{n-1}$.

Let now define the list $\mathcal{F}_n^{(k)}$ of length n binary words as:

$$\mathcal{F}_n^{(k)} = \begin{cases} \mathcal{C}_n & \text{if } 0 \leq n < k, \\ 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}} \circ 01 \cdot \overline{\mathcal{F}_{n-2}^{(k)}} \circ 001 \cdot \overline{\mathcal{F}_{n-3}^{(k)}} \circ \cdots \circ 0^{k-1}1 \cdot \overline{\mathcal{F}_{n-k}^{(k)}} & \text{if } n \geq k. \end{cases} \tag{3}$$

For $k \geq 2$ and $n \geq 0$, $\mathcal{F}_n^{(k)}$ is a list for the set of length n binary words with no k consecutive 0's, and Proposition 3.2 says that it is a Gray code (actually, $\mathcal{F}_n^{(k)}$ is a adaptation of a similar list defined earlier (Vajnovszki(1) 2001)).

It is easy to see that the number of binary words in $\mathcal{F}_n^{(k)}$ is given by $f_n^{(k)}$, the well known k-Fibonacci integer sequence defined by:

$$f_n^{(k)} = \begin{cases} 2^n & \text{if } 0 \leq n < k, \\ f_{n-1}^{(k)} + f_{n-2}^{(k)} + \cdots + f_{n-k}^{(k)}, & \text{if } n \geq k, \end{cases}$$

and the words in $\mathcal{F}_n^{(k)}$ are said *k-generalized Fibonacci words*. For example, the list $\mathcal{F}_3^{(3)}$ for the length 3 binary words avoiding 3 consecutive 0's is

$$\mathcal{F}_3^{(3)} = (100, 101, 111, 110, 010, 011, 001).$$

Proposition 3.1.
— $\mathsf{first}(\mathcal{F}_n^{(k)})$ is the length n prefix of the infinite periodic word $(10^{k-1}1)(10^{k-1}1)\ldots$;
— $\mathsf{last}(\mathcal{F}_n^{(k)})$ is the length n prefix of the infinite periodic word $(0^{k-1}11)(0^{k-1}11)\ldots$.

Proof. For the first point, if $1 \leq n < k$, then $\mathsf{first}(\mathcal{F}_n^{(k)}) = \mathsf{first}(\mathcal{C}_n) = 10^{n-1}$; and if $n = k$, then $\mathsf{first}(\mathcal{F}_n^{(k)}) = 1 \cdot \mathsf{first}(\overline{\mathcal{F}_{n-1}^{(k)}}) = 1 \cdot \mathsf{last}(\mathcal{C}_{n-1}) = 10^{k-1}$, and the statement holds in both cases.

Now, if $n > k$, by the definition of $\mathcal{F}_n^{(k)}$ we have

$$\begin{aligned} \mathsf{first}(\mathcal{F}_n^{(k)}) &= 1 \cdot \mathsf{first}(\overline{\mathcal{F}_{n-1}^{(k)}}) \\ &= 1 \cdot \mathsf{last}(\mathcal{F}_{n-1}^{(k)}) \\ &= 10^{k-1}1 \cdot \mathsf{last}(\overline{\mathcal{F}_{n-k-1}^{(k)}}) \end{aligned}$$

$$= \quad 10^{k-1}1 \cdot \mathsf{first}(\mathcal{F}^{(k)}_{n-k-1}),$$

and recursion on n completes the proof.

For the second point, if $1 \leq n < k$, then $\mathsf{last}(\mathcal{F}^{(k)}_n) = \mathsf{last}(\mathcal{C}_n) = 0^n$; and if $n = k$, then $\mathsf{last}(\mathcal{F}^{(k)}_n) = 0^{k-1}1$, and the statement holds in both cases.

Now, if $n > k$, we have

$$\begin{aligned} \mathsf{last}(\mathcal{F}^{(k)}_n) &= 0^{k-1}1 \cdot \mathsf{last}(\overline{\mathcal{F}^{(k)}_{n-k}}) \\ &= 0^{k-1}1 \cdot \mathsf{first}(\mathcal{F}^{(k)}_{n-k}), \end{aligned}$$

and by the first point of the present proposition, recursion on n completes the proof. □

Proposition 3.2. The list $\mathcal{F}^{(k)}_n$ is a Gray code where two consecutive strings differ in a single position.

Proof. It is enough to prove that there is a 'smooth' transition between any two consecutive lists in the definition of $\mathcal{F}^{(k)}_n$ given in relation (3), that is, for any ℓ, $1 \leq \ell \leq k-1$, the words

$$\alpha = 0^{\ell-1}1 \cdot \mathsf{last}(\overline{\mathcal{F}^{(k)}_{n-\ell}}) = 0^{\ell-1}1 \cdot \mathsf{first}(\mathcal{F}^{(k)}_{n-\ell})$$

and

$$\beta = 0^{\ell}1 \cdot \mathsf{first}(\overline{\mathcal{F}^{(k)}_{n-\ell-1}}) = 0^{\ell}1 \cdot \mathsf{last}(\mathcal{F}^{(k)}_{n-\ell-1})$$

differ in a single position. By Proposition 3.1,

$$\alpha = 0^{\ell-1}1\alpha'$$

and

$$\beta = 0^{\ell}1\beta'$$

with α' and β' appropriate length prefixes of $(10^{k-1}1)(10^{k-1}1)\ldots$ and $(0^{k-1}11)(0^{k-1}11)\ldots$, and so α and β differ only in position ℓ. □

As a by-product of the proof of the previous proposition we have the following remark which is critical in algorithm `process` used for the generating algorithm in Section 4.2.

Remark 1. If $\alpha = a_1a_2\ldots a_n$ and $\beta = b_1b_2\ldots b_n$ are two successive words in $\mathcal{F}^{(k)}_n$ which differ in position ℓ, then either $\ell = n$ or $a_{\ell+1} = b_{\ell+1} = 1$.

3.2. *The list* $\mathcal{H}^{(k)}_{n,q}$

Let $\mathcal{H}^{(k)}_{n,q}$ be the list defined by:

$$\mathcal{H}^{(k)}_{n,q} = \epsilon(\alpha_1) \circ \overline{\epsilon(\alpha_2)} \circ \epsilon(\alpha_3) \circ \overline{\epsilon(\alpha_4)} \circ \cdots \circ \epsilon'(\alpha_{f^{(k)}_{n-2}}) \tag{4}$$

with $\alpha_i = 1\phi_i1$ and ϕ_i is the i-th binary word in the list $\mathcal{F}^{(k)}_{n-2}$, and $\epsilon'(\alpha_{f^{(k)}_{n-2}})$ is $\epsilon(\alpha_{f^{(k)}_{n-2}})$ or $\overline{\epsilon(\alpha_{f^{(k)}_{n-2}})}$ according on whether $f^{(k)}_{n-2}$ is odd or even.

Clearly, $\mathcal{H}_{n,q}^{(k)}$ is a list for the set of q-ary words of length n which begin and end by a non zero value, and with no k consecutive 0's. In particular, $\mathcal{H}_{n,2}^{(k)} = 1 \cdot \mathcal{F}_{n-2}^{(k)} \cdot 1$.

Proposition 3.3. The list $\mathcal{H}_{n,q}^{(k)}$ is a Gray code.

Proof. From Proposition 2.2 it follows that consecutive words in each list $\epsilon(\alpha_i)$ and $\overline{\epsilon(\alpha_i)}$ differ in a single position (and by $+1$ or -1 in this position). To prove the statement it is enough to show that, for two consecutive binary words ϕ_i and ϕ_{i+1} in $\mathcal{F}_{n-2}^{(k)}$, both pair of words

— $\mathsf{last}(\epsilon(1\phi_i 1))$ and $\mathsf{first}(\overline{\epsilon(1\phi_{i+1}1)}) = \mathsf{last}(\epsilon(1\phi_{i+1}1))$, and
— $\mathsf{last}(\overline{\epsilon(1\phi_i 1)}) = \mathsf{first}(\epsilon(1\phi_i 1))$ and $\mathrm{first}(\epsilon(1\phi_{i+1}1))$

differ in a single position.

In the first case, by Proposition 2.1, the first symbols of $\mathsf{last}(\epsilon(1\phi_i 1))$ and of $\mathsf{last}(\epsilon(1\phi_{i+1}1))$ are both $(q-1)$, and the other symbols are either 1 if q is odd, or $(q-1)$ if q is even; and since ϕ_i and ϕ_{i+1} differ in a single position, the result holds.

In the second case, $\mathsf{first}(\epsilon(1\phi_i 1)) = 1\phi_i 1$ and $\mathsf{first}(\epsilon(1\phi_{i+1}1)) = 1\phi_{i+1}1$, and again the result holds. □

3.3. *The list* $\mathcal{S}_{n,q}^{(k)}$

Now we define the list $\mathcal{S}_{n,q}^{(k)}$ as

$$\mathcal{S}_{n,q}^{(k)} = 0^k \cdot \mathcal{H}_{n-k,q}^{(k)},$$

and clearly, $\mathcal{S}_{n,q}^{(k)}$ is a list for the set of cross-bifix-free words $S_{n,q}^{(k)}$. In particular,

$$\mathcal{S}_{n,2}^{(k)} = 0^k 1 \cdot \mathcal{F}_{n-k-2}^{(k)} \cdot 1,$$

for example, the set $\mathcal{S}_{8,2}^{(3)}$ of length 8 binary cross-bifix-free words which begin by 000 is

$$\mathcal{S}_{8,2}^{(3)} = 0001 \cdot \mathcal{F}_{3}^{(3)} \cdot 1 =$$

$$= (00011001, 00011011, 00011111, 00011101, 00010101, 00010111, 00010011).$$

A consequence of Proposition 3.3 is the next proposition.

Proposition 3.4. The list $\mathcal{S}_{n,q}^{(k)}$ is a Gray code.

For the sake of clearness, we illustrate the previous construction for the Gray code list $\mathcal{S}_{8,3}^{(3)}$ on the alphabet $A = \{0, 1, 2\}$. We have:

$$\begin{aligned}
\mathcal{G}_{3,3} &= (111, 112, 122, 121, 221, 222, 212, 211);\\
\mathcal{G}_{4,3} &= (1111, 1112, 1122, 1121, 1221, 1222, 1212, 1211, 2211, 2212, 2222,\\
&\quad 2221, 2121, 2122, 2112, 2111);\\
\mathcal{G}_{5,3} &= (11111, \ldots, 12111, 22111, \ldots, 21111);
\end{aligned}$$

and

$$\begin{aligned}\mathcal{S}_{8,3}^{(3)} = (&00011001, 00011002, 00012002, 00012001, 00022001, 00022002,\\ &00021002, 00021001, 00021011, \ldots, 00011011, 00011111, \ldots\\ &\ldots, 00021111, 00021101, \ldots, 00011101, 00010101, 00010102,\\ &00010202, 00010201, 00020201, 00020202, 00020102, 00020101,\\ &00020111, \ldots, 00010111, 00010011, 00010012, 00010022,\\ &00010021, 00020021, 00020022, 00020012, 00020011).\end{aligned}$$

4. Algorithmic considerations

In this section we give a generating algorithm for binary words in the list $\mathcal{F}_n^{(k)}$ and an algorithm expanding binary words; then, combining them, we obtain a generating algorithm for the list $\mathcal{H}_{n,q}^{(k)}$, and finally prepending 0^k to each word in $\mathcal{H}_{n-k,q}^{(k)}$ the list $\mathcal{S}_{n,q}^{(k)}$ is obtained. The given algorithms are shown to be efficient.

The list $\mathcal{F}_n^{(k)}$ defined in (3) has not a straightforward algorithmic implementation, and now we explain how $\mathcal{F}_n^{(k)}$ can be defined recursively as the concatenation of at most two lists, then we will give a generating algorithm for it. Let $\mathcal{F}_n^{(k)}(u)$, $0 \le u \le k-1$, be the sublist of $\mathcal{F}_n^{(k)}$ formed by strings beginning by at most u 0's. By the definition of $\mathcal{F}_n^{(k)}$, it follows that $\mathcal{F}_n^{(k)} = \mathcal{F}_n^{(k)}(k-1)$, and

$$\begin{aligned}\mathcal{F}_n^{(k)}(0) &= 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}}\\ &= 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}(k-1)},\end{aligned}$$

and for $u > 0$,

$$\begin{aligned}\mathcal{F}_n^{(k)}(u) &= 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}} \circ 01 \cdot \overline{\mathcal{F}_{n-2}^{(k)}} \circ \cdots \circ 0^u 1 \cdot \overline{\mathcal{F}_{n-u-1}^{(k)}}\\ &= 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}} \circ 0 \cdot (1 \cdot \overline{\mathcal{F}_{n-2}^{(k)}} \circ \cdots \circ 0^{u-1} 1 \cdot \overline{\mathcal{F}_{n-u-1}^{(k)}})\\ &= 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}} \circ 0 \cdot \mathcal{F}_{n-1}^{(k)}(u-1).\end{aligned}$$

By the above considerations we have the following proposition.

Proposition 4.1. Let $k \ge 2$, $0 \le u \le k-1$, and $\mathcal{F}_n^{(k)}(u)$ be the list defined as:

$$\mathcal{F}_n^{(k)}(u) = \begin{cases} \lambda & \text{if} \quad n = 0, \\ 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}(k-1)} & \text{if} \quad n > 0 \text{ and } u = 0, \\ 1 \cdot \overline{\mathcal{F}_{n-1}^{(k)}(k-1)} \circ 0 \cdot \mathcal{F}_{n-1}^{(k)}(u-1) & \text{if} \quad n, u > 0. \end{cases} \tag{5}$$

Then $\mathcal{F}_n^{(k)}(k-1)$ is the list $\mathcal{F}_n^{(k)}$ defined by relation (3).

Now we explain how the relation (5) defining the list $\mathcal{F}_n^{(k)}(u)$ can be implemented in a generating algorithm. It is easy to check that $\mathcal{F}_n^{(k)} = \mathcal{F}_n^{(k)}(k-1)$ has the following

properties: for $\alpha = a_1 a_2 \dots a_n$ and $\beta = b_1 b_2 \dots b_n$ two consecutive binary words in $\mathcal{F}_n^{(k)}$, there is a p such that

— $a_i = b_i$ for all i, $1 \le i \le n$, except $b_p = 1 - a_p$,
— 0^{k-1} can not be a suffix of $a_1 a_2 \dots a_{p-1} = b_1 b_2 \dots b_{p-1}$,
— the sublist of $\mathcal{F}_n^{(k)}$ formed by the strings with the prefix $b_1 b_2 \dots b_p$ is $b_1 b_2 \dots b_p \cdot \mathcal{L}$, where $\mathcal{L}$ is $\mathcal{F}_{n-p}^{(k)}(u-1)$ or $\overline{\mathcal{F}_{n-p}^{(k)}(u-1)}$ according to the prefix $b_1 b_2 \dots b_p$ has an even or odd number of 1's, and u is equal to k minus the length of the maximal 0 suffix of $b_1 b_2 \dots b_p$.

Let us consider procedure `gen_fib` in Figure 1 where `process` switches the value of $b[pos]$ (that is, $b[pos] := 1 - b[pos]$), and prints the obtained binary string b. By the above remarks and relation (5) in Proposition 4.1 it follows that after the initialization of b by the first string in $\mathcal{F}_m^{(k)}$ (given in Proposition 3.1) and printing it out, the call of `gen_fib(1,`$k-1$`,0)` produces the list $\mathcal{F}_m^{(k)}$. Moreover, as we will show below, for $m = n-1$ and after the appropriate initialization of $b = b_1 b_2 \dots b_n$ the call of `gen_fib(`$k+2$`,`$k-1$`,0)` produces the list $0^k 1 \cdot \mathcal{F}_{n-k-2}^{(k)} \cdot 1 = \mathcal{S}_{n,2}^{(k)}$.

Procedure `gen_fib` is an efficient generating procedure. Indeed, each recursive call induced by `gen_fib` is either

— a terminal call (which does not produce other calls), or
— a call producing two recursive calls, or
— a call producing one recursive call, which in turn is in one of the previous two cases.

By *'CAT'* principle in (Ruskey book) it follows that procedure `gen_fib` runs in constant amortised time.

```
procedure gen_fib(pos,u,dir)
global b,k,m;
if pos ≤ m
then  if u = 0
      then gen_fib(pos + 1,k − 1,1 − dir);
      else if dir = 0
           then  gen_fib(pos + 1,k − 1,1);
                 process(pos);
                 gen_fib(pos + 1,u − 1,0);
           else  gen_fib(pos + 1,u − 1,1);
                 process(pos);
                 gen_fib(pos + 1,k − 1,0);
           end if
      end if
end if
end procedure.
```

Fig. 1. Algorithm producing the list $\mathcal{F}_n^{(k)}$ or $\mathcal{S}_{n,q}^{(k)}$, according to the initial values of m, b and the definition of `process` procedure.

4.1. *Generating* $\mathcal{S}_{n,2}^{(k)}$

After the initialization of $b_1b_2\dots b_n$ by $0^k1\cdot\mathsf{first}(\mathcal{F}_{n-k-2}^{(k)})\cdot 1$, with $\mathsf{first}(\mathcal{F}_{n-k-2}^{(k)})$ given in Proposition 3.1, and printing it out, the call of `gen_fib`$(k+2,k-1,0)$ where

— $m=n-1$, and
— procedure `process` called by `gen_fib` switches the value of $b[pos]$ (that is, $b[pos]:=1-b[pos]$) and prints b

produces, in constant amortized time, the list $0^k1\cdot\mathcal{F}_{n-k-2}^{(k)}\cdot 1=0^k\cdot\mathcal{H}_{n-k,2}^{(k)}$ which is, as mentioned before, the list $\mathcal{S}_{n,2}^{(k)}$.

4.2. *Generating* $\mathcal{S}_{n,q}^{(k)}$, $q>2$

Before discussing the expansion algorithm `expand` needed to produce the list $\mathcal{S}_{n,q}^{(k)}$ when $q>2$ we show that `gen_tuple` procedure in Figure 2, on which `expand` is based, is an efficient generating algorithm for the list $\mathcal{G}_{n,q}$ defined in relation (1). Procedure `gen_tuple` is a 'naive' odometer principle based algorithm, see again (Ruskey book), and we have the next proposition.

Proposition 4.2. After the initialization of v by $11\cdots1$, the first word in $\mathcal{G}_{n,q}$, and d_i by 1, for $1\le i\le n$, procedure `gen_tuple` produces the list $\mathcal{G}_{n,q}$ in constant amortized time.

Proof. The total amount of computation of `gen_tuple` is proportional to the number of times the statement $i:=i-1$ is performed in the inner `while` loop; and for a given q and n let denote by c_n this number. So, the average complexity (per generated word) of `gen_tuple` is $\frac{c_n}{q^n}$. Clearly, $c_1=q-1$ and $c_n=(q-1)\cdot n+q\cdot c_{n-1}$, and a simple recursion shows that $c_n=q\cdot\frac{q^n-1}{q-1}-n$ and finally the average complexity of `gen_tuple` is $\frac{c_n}{q^n}\le\frac{q}{q-1}$. □

```
procedure gen_tuple
global v,d,n;
output v;
do  i := n;
    while  i ≥ 1 and
           (v[i] = q − 1 and d[i] = 1 or v[i] = 1 and d[i] = −1)
           d[i] := −d[i];
           i := i − 1;
    end while
    if i ≥ 1 then v[i] := v[i] + d[i]; output v; end if
while i ≥ 1
end procedure.
```

Fig. 2. Odometer algorithm producing the list $\mathcal{G}_{n,q}$.

Now we adapt algorithm `gen_tuple` in order to obtain procedure `expand` producing the expansion of a words; and like `gen_tuple`, procedure `expand` has a constant average time complexity. More precisely, for a words $b = b_1 b_2 \dots b_n$ in $\{0, 1, \dots, q\}^n$, with $b_{\ell+1}, b_n \neq 0$ let b' denote the *trace* of $b_{\ell+1} b_{\ell+2} \dots b_n$, that is, the word obtained from $b_{\ell+1} b_{\ell+2} \dots b_n$ by replacing each non-zero value by 1, and b'' that obtained by erasing each 0 letter in $b_{\ell+1} b_{\ell+2} \dots b_n$. Procedure `expand` produces the list:

— $b_1 b_2 \dots b_\ell \cdot \epsilon(b')$ if the initial value of b is such that b'' is the first word in $\mathcal{G}_{|b''|,q}$, or
— $b_1 b_2 \dots b_\ell \cdot \overline{\epsilon(b')}$ if the initial value of b is such that b'' is the last word in $\mathcal{G}_{|b''|,q}$.

The initial value of $d_{\ell+1}, d_{\ell+2}, \dots, d_n$ are given by: if $b_i = 1$, then $d_i = 1$; and if $b_i = q-1$, then $d_i = -1$; otherwise d_i is not defined. In order to access in constant time from a position i in the current word b, with $b_i \neq 0$, to the previous one, additional data structures are used. The array *prec* is defined by: if $b_i \neq 0$, then $prec_i = j$, where j is the rightmost position in b, at the left of i and with $b_j \neq 0$; and for convenience $prec_i = 0$ if i is the leftmost non-zero position in b.

```
procedure expand
global b,d,ℓ,n,prec;
output v;
do  i := n;
    while  i ≥ ℓ + 1 and
           (b[i] = q − 1 and d[i] = 1 or b[i] = 1 and d[i] = −1)
           d[i] := −d[i];
           i := prec[i];
    end while
    if i ≥ ℓ + 1 then b[i] := b[i] + d[i]; output b; end if
while i ≥ ℓ + 1
end procedure.
```

Fig. 3. Algorithm expanding a word b and mimicking procedure `gen_tuple`.

Now we explain procedure `process`; it calls `expand` and we will show that when `gen_fib` in turn calls procedure `process` in Figure 4, then it produces the list $\mathcal{S}^{(k)}_{n,q}$, with $q > 2$. The parameter *pos* of `process` is given by the corresponding call of `gen_fib`, and it gives the position in the current word $b_1 b_2 \dots b_n$ in $\mathcal{S}^{(k)}_{n,q}$ where b_{pos} changes from a non-zero value to 0, or *vice versa*. By Remark 1 and the definition of the list $\mathcal{S}^{(k)}_{n,q}$ from $\mathcal{H}^{(k)}_{n-k,q}$, and so from $\mathcal{F}^{(k)}_{n-k-2,q}$, it follows that $b_{pos+1} \neq 0$. Procedure `process`, sets b_{pos} to 0 if previously $b_{pos} \neq 0$; and sets b_{pos} to b_{pos+1} if previously $b_{pos} = 0$, which according to Proposition 2.1, Remark 1 and the definition of the expansion operation is the new value of b_{pos}. In order to access in constant time from a non-zero position in the array b to the previous one, `process` uses array *prec* of procedure `expand` and array *succ*, defined as: $succ_i = j$, with j the leftmost position in b, at the right of i and with $b_j \neq 0$, and $succ_i$ is not defined if i is the rightmost non-zero position. In addition, procedure `process` updates both arrays *prec* and *succ*.

```
procedure process(pos)
global b,d,succ,prec;
if b[pos] = 0
then a := prec[pos + 1]; succ[a] := pos; succ[pos] := pos + 1;
     prec[pos] := a; prec[pos + 1] := pos;
     b[pos] := b[pos + 1];
     d[pos] := d[pos + 1];
else a := prec[pos]; z := succ[pos];
     prec[z] := a; succ[a] := z;
     b[pos] := 0;
expand;
end procedure.
```

Fig. 4. Procedure `process` called by `gen_fib` in order to generate the list $\mathcal{S}_{n,q}^{(k)}$.

For given $q > 2$, $k \geq 2$ and $n \geq k + 2$, after the initialization of $b_1 b_2 \dots b_n$ by $0^k 1 \cdot \mathsf{first}(\mathcal{F}_{n-k-2}^{(k)}) \cdot 1$, as for generating $\mathcal{S}_{n,2}^{(k)}$, the call of `gen_fib(`$k+2$`,`$k-1$`,0)` where
— $m = n - 1$, and
— procedure `process` is that in Figure 4, and
— procedure `expand` that in Figure 3, with $\ell = k + 1$
produces, in constant amortized time, the list $\mathcal{S}_{n,q}^{(k)}$.

5. Conclusion and further works

The cross-bifix-free sets $S_{n,q}^{(k)}$ (Chee *et al.* 2013) have the cardinality close to the optimum. They are constituted by particular words $s_1 s_2 \dots s_n$ of length n over a q-ary alphabet. Each word has the form $0^k s_{k+1} s_{k+2} \dots s_n$ where s_{k+1} and s_n are different from 0 and $s_{k+1} s_{k+2} \dots s_{n-1}$ does not contain k consecutive 0's. We have provided a Gray code for $S_{n,q}^{(k)}$ by defining a Gray code for the words $s_{k+1} s_{k+2} \dots s_n$ and then prepending the prefix 0^k to them. Moreover, an efficient generating algorithm for the obtained Gray code is given. We note that this Gray code is *trace partitioned* in the sense that all the words with the same trace are consecutive. To this aim we used a Gray code for restricted binary strings (Vajnovszki(1) 2001), opportunely replacing the bits 1 with the symbols of the alphabet different from 0.

A future investigation could be the definition of a Gray code which is *prefix partitioned*, where all the words with the same prefix are consecutive. Actually, the definition of the sets $S_{n,q}^{(k)}$ shows that it is sufficient to define a prefix partitioned Gray code for the subwords $s_{k+1} s_{k+2} \dots s_n$.

An interesting question arising when one deals with a Gray code $\mathcal{L}$ on a set is the possibility to define it in such a way that the Hamming distance between $\mathsf{last}(\mathcal{L})$ and $\mathsf{first}(\mathcal{L})$ is 1 (circular Gray code). Usually it is not so easy to have a circular Gray code, unless the elements of the set are not subject to constraints; in our case it is worth to study if the ground-set we are dealing with (which is a cross-bifix free set) allows to find a circular Gray code.

References

Bajic, D. (2007) On Construction of Cross-Bifix-Free Kernel Sets. 2nd MCM COST 2100, TD(07)237, Lisbon, Portugal.

Baril, J. and Vajnovszki, V. (2004) Gray code for derangements. *Discrete Applied Mathematics* **140** 207–221

Berstel, J., Perrin, D. and Reutenauer, C. (2009) Codes and Automata (Encyclopedia of Mathematics and its Applications). Cambridge University Press.

Bilotta, S., Pergola, E. and Pinzani, R. (2012) A new approach to cross-bifix-free sets. *IEEE Transactions on Information Theory* **58** 4058–4063.

Chee, Y. M., Kiah, H. M., Purkayastha, P. and Wang, C. (2013) Cross-bifix-free codes within a constant factor of optimality. *IEEE Transactions on Information Theory* **59** 4668–4674.

Crochemore, M., Hancart, C. and Lecroq, T. (2007) Algorithms on strings. Cambridge University Press, Cambridge.

Er, M. C. (1984) On generating the N-ary reflected Gray code. *IEEE Transaction on Computer* **33** 739–741.

Gray, F. (1953) Pulse Code Communication. U.S. Patent 2 632 058.

Hamming, R. W. (1950) Error detecting and error correcting codes. *Bell System Technical Journal* **29** 147–160.

Johnson, S. M. (1963) Generation of permutations by adjacent transpositions. *Mathematics of Computation* **17** 282–285.

de Lind van Wijngaarden, A. J. and Willink, T. J. (2000) Frame synchronization using distributed sequences. *IEEE Transactions on Commununications* **48** 2127-2138.

Ruskey, F. (1993) Simple combinatorial Gray codes constructed by reversing sublist. *Lecture Notes in Computer Science* **762** 201–208.

Ruskey F. Combinatorial generation, Book in preparation.

Sagan, B. E. (2010) Pattern avoidance in set partitions. *Ars Combinatoria* **94** (2010) 79–96.

Vajnovszki, V. (2001) A loopless generation of bitstrings without p consecutive ones. *Discrete Mathematics and Theoretical Computer Science* Springer (2001), 227–240.

Vajnovszki, V. (2001) Gray visiting Motzkin. *Acta Informatica* **38** 793–811.

Walsh, T. (2001) Gray codes for involutions. *Journal of Combinatorial Mathematics and Combinatorial Computing* **36** 95–118.

Walsh, T. (2003) Generating Gray Codes in O(1) worst-case time per word. *Lecture Notes in Computer Science* **2731** 73–88.

Williamson, S. G. (1985) Combinatorics for computer science. Computer Science Press, Rockville, Maryland.