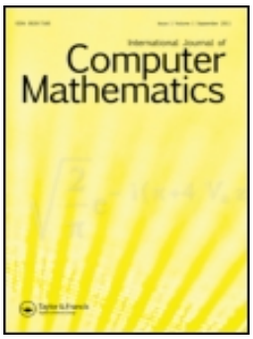

International Journal of Computer Mathematics

ISSN: 0020-7160 (Print) 1029-0265 (Online) Journal homepage: www.tandfonline.com/journals/gcom20

Generating fixed density bracelets of arbitrary base

S. Karim, Z. Alamgir & S. M. Husnine

To cite this article: S. Karim, Z. Alamgir & S. M. Husnine (2014) Generating fixed density bracelets of arbitrary base, International Journal of Computer Mathematics, 91:3, 434-446, DOI: 10.1080/00207160.2013.805753

To link to this article: https://doi.org/10.1080/00207160.2013.805753

Published online: 18 Jun 2013.

International Journal of Computer Mathematics, 2014
Vol. 91, No. 3, 434–446, http://dx.doi.org/10.1080/00207160.2013.805753

Taylor & Francis
Taylor & Francis Group

Generating fixed density bracelets of arbitrary base

S. Karim[a]*, Z. Alamgir[b] and S.M. Husnine[b]

[a]*Department of Computer Science, University of Management and Technology C-II, Johar Town, Lahore, Pakistan;* [b]*Department of Computer Science, National University of Computer and Emerging Sciences Block B, Faisal Town, Lahore, Pakistan*

(*Received 29 October 2012; revised version received 6 February 2013; accepted 29 April 2013*)

Bracelets are lexicographically minimal k-ary strings symmetric under rotation and reversal. In this paper, we present an algorithm for lexicographic listing of bracelets with fixed density. Our algorithm works for arbitrarily large alphabet size and each successive bracelet is generated in constant amortized time.

Keywords: bracelet with fixed density; necklace; CAT algorithm; combinatorial generation

2010 AMS Subject Classifications: 05-04; 68R05; 68R15

1. Introduction

Generation of discrete combinatorial objects is of immense importance in mathematics and computer science. Knuth [7] and Ruskey [9] have recently compiled their books on the subject. Moreover, the algorithms for constructing exhaustive lists of discrete objects are widely used in areas such as computational biology, combinatorial chemistry, operations research and data mining [2,5,14].

Several schemes have been developed for the generation of combinatorial structures such as necklaces, Lyndon words and their variants. Cattell *et al.* [1] give a recursive framework to generate prenecklaces, necklaces and Lyndon words. This framework is used to list restricted classes of this family of objects. Ruskey and Sawada [10] develop the algorithms to generate necklaces with fixed density and Sawada [13] gives the algorithm for necklaces with fixed content using the same framework. Recently, Vajnovszki answers the problem of generating unrestricted binary necklaces in Gray code order [15]. Later, Vajnovszki and Weston give a generalized and optimized Gray code algorithm for arbitrarily large alphabet size [16,17]. However, no significant work is done to list restricted classes of bracelets. Lisonek [8] proposes a linear algorithm for generating a list of all bracelets by modifying the necklace generation algorithm of Ruskey *et al*. [11]. Sawada [12] gives a constant amortized time (CAT) algorithm for bracelet generation. Bracelets are of great practical significance; the exhaustive list of bracelets is used in the calibration of colour printers [3].

In this paper, we present a recursive scheme to list restricted class of bracelets, namely those with fixed density. We have designed our algorithm such that the total number of basic operations performed is proportional to the number of bracelets generated by our algorithm. Hence, our

*Corresponding author. Email: sairakarim@gmail.com

algorithm takes constant time on average to list a bracelet. It is extremely desirable in generation algorithms that the number of steps taken in listing successive objects remains constant.

In the following section, we give basic definitions and brief background on the generation of necklaces and related objects. Section 3 presents an algorithm for the generation of bracelets with fixed density, and Section 4 analyses the time complexity of our algorithm. Finally, Section 5 contains concluding remarks and future works.

2. Preliminaries

A *necklace* is a lexicographically minimal k-ary string equivalent under string rotation, that is, $a_1a_2\cdots a_i\cdots a_n$ is equivalent to $a_ia_{i+1}\cdots a_na_1\cdots a_{i-1}$ for $1 < i \leq n$. The set of all necklaces of length n on k alphabets is represented as $\mathbf{N}_k(n)$ and the cardinality of $\mathbf{N}_k(n)$ is denoted by $N_k(n)$. A *prenecklace* of length n is a prefix of some necklace of length m, where $m \geq n$. The set of all prenecklaces of length n is denoted by $\mathbf{P}_k(n)$. The cardinality of $\mathbf{P}_k(n)$ is $P_k(n)$. Another restricted class of necklaces, aperiodic necklaces, is called *Lyndon words*. The set of Lyndon words of length n is denoted by $\mathbf{L}_k(n)$. A *bracelet* is a lexicographically minimal k-ary string that is symmetric under rotation and reversal. $\mathbf{B}_k(n)$ represents a set of length n bracelets and its cardinality is denoted by $B_k(n)$.

A k-ary string is said to be of fixed density, if the number of occurrences of symbol 0 is fixed. Necklaces, prenecklaces and bracelets with fixed density are denoted in the similar manner. We add an additional parameter d, number of non-zero symbols, to denote the density of the string. We use the following notations to denote these objects.

- $\mathbf{N}_k(n,d)$: the set of k-ary necklace having length n and density d,
- $\mathbf{P}_k(n,d)$: the set of k-ary prenecklace having length n and density d,
- $\mathbf{B}_k(n,d)$: the set of k-ary bracelets having length n and density d.

Similarly, $N_k(n,d)$, $P_k(n,d)$, and $B_k(n,d)$ denote the cardinality of $\mathbf{N}_k(n,d)$, $\mathbf{P}_k(n,d)$ and $\mathbf{B}_k(n,d)$, respectively.

Let $N_k(n_0,n_1,\ldots,n_{k-1})$ be the number of necklaces with n_i occurrences of symbol i, where $0 \leq i < k$. It is shown in [4] that following relation holds:

$$N_k(n_0,n_1,\ldots,n_{k-1}) = \frac{1}{n}\sum_{j\backslash \gcd(n_0,n_1,\ldots,n_{k-1})} \phi(j)\frac{(n/j)!}{(n_0/j)!\cdots(n_{k-1}/j)!}. \tag{1}$$

Here, $\phi(j)$ is Euler's totient function, and it gives a number of positive integers less than or equal to j that are relatively prime to j. Using Equation (1), we can count $N_k(n,d)$ as follows:

$$N_k(n,d) = \sum_{n_1+n_2+\cdots+n_{k-1}=d} N_k(n-d,n_1,\ldots,n_{k-1}). \tag{2}$$

There are at most two necklaces in each equivalence class of a bracelet. Therefore, following relations hold:

$$N_k(n) \leq 2B_k(n), \tag{3}$$

$$N_k(n,d) \leq 2B_k(n,d). \tag{4}$$

Our recursive algorithm for generating bracelets with fixed density is based on the following theorem given in [1].

THEOREM 2.1 (Fundamental theorem of necklaces) *Let* $\alpha = a_1 a_2 \cdots a_{n-1} \in \boldsymbol{P}_k(n-1)$ *and* $p = \mathrm{lyn}(\alpha)$. *The string* $\alpha b \in \boldsymbol{P}_k(n)$ *iff* $a_{n-p} \leq b \leq k-1$. *Furthermore,*

$$\mathrm{lyn}(\alpha b) = \begin{cases} p & \text{if } a_{n-p} = b, \\ n & \text{if } a_{n-p} < b. \end{cases}$$

Here, $\mathrm{lyn}(\alpha)$ is the length of the longest Lyndon prefix of α, that is, $\mathrm{lyn}(\alpha) = \max\{1 \leq p \leq n-1 \mid (a_1 \cdots a_p) \in \mathbf{L}_k(p)\}$. Also, αb is a necklace iff $n \bmod \mathrm{lyn}(\alpha b) = 0$. In [1], $\mathbf{N}_k(n)$ is generated by recursive application of fundamental theorem of necklaces.

3. Generation algorithm

In this section, we present the basic idea of our recursive scheme to list bracelets with fixed density. First, we present a naive algorithm which is a simple modification of the necklace generation algorithm given in [1]. However, this simple modification does not yield a CAT algorithm. Next, we present a CAT algorithm which uses optimizations from a recursive algorithm for the generation of fixed density necklaces [10] and the generation of an algorithm for bracelets [12].

3.1 *Naive algorithm*

The *fundamental theorem of necklaces* specifies the necessary and sufficient condition to append a character to a prenecklace such that the resulting string is also a prenecklace, provided the length of the prenecklace and the length of the longest Lyndon prefix are known. Using this theorem, it is straightforward to produce a recursive algorithm to exhaustively list all prenecklaces of length n in lexicographic order. A pseudocode is provided in Figure 1(a), where the parameter p represents the length of the longest Lyndon prefix of the current prenecklace. The function Print(p) is used to output each prenecklace, and it can easily be modified to output necklaces or Lyndon words. A prenecklace is a necklace if $n \bmod p = 0$; it is a Lyndon word if $n = p$. Each object can be generated in CAT [1]. The initial call is Necklace(1,1) with a_0 initialized to 0.

(a)
```
procedure Necklace(t, p: int)
    j, p': int

    if t > n then Print(p)
    else
        for j := a_{t-p} to k - 1 do

            a_t := j
            p' := p
            if j ≠ a_{t-p} then p' := t

            Necklace(t + 1, p')

end.
```

(b)
```
procedure SimpleBFD(t, p, r: int)
    c, j, p': int

    if t > n then
        if a_{r+1} ··· a_n ≤ a_n ··· a_{r+1} then Print(p)
    else
        for j := a_{t-p} to k - 1 do
            if j ≠ 0 then d := d - 1
            a_t := j
            p' := p
            if j ≠ a_{t-p} then p' := t
            c := CheckRev(t)
            if c = 0 and d ≥ 0 then
                SimpleBFD(t + 1, p', t)
            if c = 1 and d ≥ 0 then
                SimpleBFD(t + 1, p', r)
            if j ≠ 0 then d := d + 1
end.
```

Figure 1. (a) A simple recursive algorithm Necklace(t, p) to list all necklaces, Lyndon words or prenecklaces depending on the restrictions given by the function Print(p). (b) A simple algorithm SimpleBFD(t, p, r) to list all bracelets with fixed density.

Since, the number of symbol 0 is fixed in bracelets with density and d represents the number of non-zero symbols, a straightforward modification of the necklace generation algorithm is given in Figure 1(b). In this simple algorithm, we ensure two things. First, all prenecklaces must have density equal to d. So, we keep a check on the number of non-zero characters in the current prenecklace. Next, we only want to list bracelets. An $O(n)$ test can be used to check whether the generated necklace is a bracelet or not by first computing the necklace of the reversed string and then comparing it with the generated necklace. However, by applying this test, we cannot get a CAT algorithm. Instead, we apply the following result which follows directly from Theorem 3.1 of [12].

LEMMA 3.1 *If $\alpha = a_1a_2\cdots a_n$ is a necklace where r denotes the length of its longest prefix such that $a_1\cdots a_r = a_r\cdots a_1$, then α is a bracelet iff $a_{r+1}\cdots a_n \leq a_n\cdots a_{r+1}$ and there is no index t such that $a_1\cdots a_t > a_t\cdots a_1$.*

Using this lemma, we compare the current prenecklace with its reversal. If prenecklace is greater than its reversal, then we do not generate it any further; if they are equal, then we update the value for a new parameter r. When the prenecklace has length n, we compare $a_{r+1}\cdots a_n$ with its reversal to test if it is a bracelet.

The initial call is SimpleBFD(1,1,0) with a_0 initialized to 0. The function CheckRev(t) compares the current prenecklace with its reverse string. The values returned by CheckRev(t) are given below:

$$\text{CheckRev}(t) = \begin{cases} 1 & \text{if } a_1a_2\cdots a_t < a_ta_{t-1}\cdots a_1, \\ 0 & \text{if } a_1a_2\cdots a_t = a_ta_{t-1}\cdots a_1, \\ -1 & \text{if } a_1a_2\cdots a_t > a_ta_{t-1}\cdots a_1. \end{cases}$$

3.2 *An efficient algorithm*

The algorithm mentioned above is a simple extension of the necklace generation algorithm. However, it is not an efficient algorithm. In this section, we present a CAT algorithm using few optimizations from necklaces with the fixed density algorithm presented in [10] and the bracelet generation algorithm [12]. Merging these optimizations we develop an efficient algorithm for bracelets with fixed density.

3.2.1 *Fixed density optimizations*

The CAT algorithm given in [10] is also an extension of the necklace generation algorithm. As first optimization, it does not append one single character in each recursive call. Instead, it increases the density of current prenecklace by one. This is achieved by finding the valid value and position for the next non-zero symbol in the current prenecklace. To make this change, array a is used to store the positions of non-zero symbols in the current prenecklace, as a_i represents the position of the ith non-zero symbol in the prenecklace. The array b is used to store values of the current prenecklace. All the elements in b are initialized to zero so that during execution of the algorithm only those positions of b are updated which are stored in a. Also, t represents the density of the current prenecklace and a_t represents the length of current prenecklace. Since the Lyndon prefix must not end with the symbol zero, a_p represents the length of the longest Lyndon prefix and p represents the density of the longest Lyndon prefix in the current prenecklace. To maintain the lexicographic order, the algorithm computes the maximum position and the minimum value of the next non-zero symbol such that the resulting string remains a prenecklace. All values larger than the minimum value of next non-zero symbol are also valid for the string to remain a prenecklace. Similarly, for all positions larger than the position of the last non-zero symbol and less than the

maximum position can have values from 1 to $k-1$. The maximum position and minimum value for the next non-zero symbol is computed using the following relations:

$$a_{t+1} = a_{t+1-p} + a_p,$$
$$b_{a_{t+1}} = b_{a_{t+1-p}}.$$

Since the necklace must have density d, the non-zero symbols must be placed in some restricted positions. The first non-zero symbol must be placed between $n-d+1$ and $(n-1)/d+1$. Similarly, the ith non-zero symbol must be placed at or before $(n-d+i)$th position. Also, no necklace with positive density d can end with symbol zero. So, the last non-zero symbol must be placed at the nth position. The second optimization stops further generation when the length of the current prenecklace is less than n and its density is $d-1$. The last non-zero symbol is placed at the nth position in print function. An additional constant time test is performed to compute the valid values of the last non-zero symbol. This test is similar to finding the minimum value for the next non-zero symbol.

3.2.2 *Bracelet optimizations*

We use a couple of optimizations proposed in [12] to develop a CAT algorithm. The first and simpler optimization is that instead of checking all reverse rotations, if the necklace α is of the form $a^i a_{i+1} \cdots a_n$, where $a \neq a_{i+1}$ and a^i means i occurrences of the symbol a, then only those reverse rotations that also begin with a^i are required to be checked. For example, we only need to check two reverse rotations for the necklace 00121101211003, that is, 00300112101121 and 00112101121003. Also, we do not need to wait for the generation of entire necklace. The reverse checking can be performed as soon as the prenecklace having the above-mentioned form is generated. However, it still requires $O(t)$ amount of work whenever this test is performed.

The second optimization focuses on the last test performed for bracelet testing which compares $a_{r+1} \cdots a_n$ to its reversal. In this final test, we wait till the prenecklace of length n is generated and then compare, which may take linear time. However, once the middle point is reached we can start this comparison. Specifically, each symbol generated after position $\lfloor (n-r)/2 \rfloor + r$ can be compared with the corresponding character in the reversed string. An additional parameter RS is updated depending on the outcome of this comparison. The value RS is used to store the intermediate results of the last test. The initial value of RS is false. Whenever the last character a_{t-1} is greater than its corresponding character $a_{n-t+2+r}$, RS is updated to *false*, if it is less then RS is *true*. Otherwise, it retains its previous value. In this way, we compare at most one character in each recursive call and make it a constant time test.

3.2.3 *Merging optimizations*

We use all four optimizations mentioned above to develop an efficient algorithm for bracelets with fixed density. We use a recursive algorithm to generate prenecklaces using both the optimizations used for the generation of necklaces with fixed density. We use array a to store the positions of all non-zero symbols and array b to store the contents of the current prenecklace. Here, we assume that $d < n$, so all prenecklaces β must start with symbol 0 and will be of the form $\beta = 0^i b_{i+1} b_{i+2} \cdots b_{a_t}$, where $1 \leq i \leq n-d$. Also note that the fixed density algorithm does not generate a prenecklace that ends with 0. We perform the reverse checking only if β is of the form $0^i b_{i+1} b_{i+2} \cdots b_{a_t-i-1} 0^i b_{a_t}$ using the first optimization of bracelets. If $0^i b_{i+1} \cdots b_{a_t-i-1} 0^i = 0^i b_{a_t-i-1} \cdots b_{i+1} 0^i$, then we update the value of r and if the reversal is less, then we terminate further computation of β. The pseudocode of CheckRev(t) is given in Figure 2.

```
procedure CheckRev(t: int)
    j: int

    for j := a_1 to (t + 1)/2 do
        if b_j < b_{t-j+1} then return 1
        if b_j > b_{t-j+1} then return -1
    end;
    return 0
end.
```

Figure 2. Algorithm to compare string with its reversal.

All these optimizations cannot be merged directly. Note that the fixed density algorithm may increase the length of the prenecklace by more than one. So, the last test which incrementally updates RS cannot be performed in constant time.

The default value of RS is *false* and the new value of RS is computed only when $a_t > (n-r)/2 + r$. If we compare all the characters between $a_{t-1}+1$ and a_t, then it requires $a_t - a_{t-1}$ character comparisons. Since the first optimization of the fixed density algorithm may append a positive length string of zeros to generate the next prenecklace, the final test may take a non-constant amount of time whenever $a_t - a_{t-1} > 1$.

Note that there is only one non-zero symbol in the substring $0^{a_t - a_{t-1}-1}b_{a_t}$. The value of RS can be computed in unit time if $b_{a_t} \neq b_{n-a_t+r+1}$. Otherwise, we determine the length of the substring of zeros starting at position $n - a_t + r + 2$.

In order to compute the length of substring of zeros starting at position $n - a_t + r + 2$, we define the following variables:

s_i : the density of prenecklace $b_1 b_2 \cdots b_i$,

l_i : the length of substring of zeros starting at position i.

Let $e = n - a_t + r + 1$. We can compute $l_{e+1} = a_{s_e+1} - a_{s_e} - 1$. Hence, we compute RS as follows:

$$\text{RS} = \begin{cases} \text{false} & \text{if } b_{a_t} > b_e, \\ \text{true} & \text{if } b_{a_t} < b_e \text{ or } (a_t - a_{t-1} - 1 > l_{e+1} \text{ and } b_{a_t} = b_e). \end{cases}$$

We update the value of s_i only when a non-zero symbol is appended at position i to the current prenecklace. Note that s_i is not assigned a valid value whenever $b_i = 0$. However, this information is not necessary, since we only use s_i when $b_i \neq 0$.

3.2.4 *Algorithm*

Now, we briefly describe the steps performed by our algorithm. Here, we assume that $0 < d < n$, so each bracelet should begin with symbol 0. We design a recursive scheme BraceFD(t,p,r,RS) to list the set $\mathbf{B}_k(n,d)$. The pseudocode of this algorithm is given in Figure 3.

We carry out few steps to initialize the variables in our scheme. Note that the first non-zero symbol can be placed at positions $n-d+1$ to $(n-1)/d+1$. So, we repeatedly execute BraceFD(t,p,r,RS) after placing the first non-zero symbol at these positions (Figure 4). A sample output of the algorithm is shown in Table 1.

Since $\mathbf{B}_k(n,d) \subset \mathbf{N}_k(n,d)$, we generate strings that belong to the set $\mathbf{N}_k(n,d)$ in lexicographic order and eliminate all those necklaces whose reversed rotations are less than the necklace itself. Hence, BraceFD(t,p,r,RS) lists all bracelets with fixed density exactly once and this proves the following theorem.

THEOREM 3.1 *The function* InitFixed *() lists all elements of* $\boldsymbol{B}_k(n,d)$ *exactly once in lexicographic order.*

```
procedure BraceFD(t, p, r: int; RS: boolean)
    i, j, max, tail, rev, r2: int
    RS2: boolean
    if a_t > ⌊(n − r)/2⌋ + r then
        if b_{a_t} > b_{n−a_t+1+r} then RS := FALSE
        else if b_{a_t} < b_{n−a_t+1+r} then RS := TRUE
        else if b_{a_t} = b_{n−a_t+1+r} and a_t − a_{t−1} − 1 > a_{s_{(n−a_t+1+r)}+1} − a_{s_{(n−a_t+1+r)}} − 1
            then RS := TRUE
    if t ≥ d − 1 then Print(p, r, RS)

    else
        tail := n − (d − t) + 1
        max := a_{t+1−p} + a_p
        if max ≤ tail then
            r2 := r
            RS2 := RS
            a_{t+1} := max
            b_{a_{t+1}} := b_{a_{t+1−p}}
            s_{a_{t+1}} := t + 1
            if a_1 = a_{t+1} − a_t then
                rev :=CheckRev(a_{t+1} − 1)
                if rev = 0 then
                    r2 := a_{t+1} − 1
                    RS2 := FALSE
                if rev ≠ −1 then
                    BraceFD(t + 1, p, r2, RS2)
                    for i := b_{a_{t+1}} + 1 to k − 1 do
                                        b_{a_{t+1}} := i
                                        BraceFD(t + 1, t + 1, r2, RS2)
            else
                BraceFD(t + 1, p, r, RS)
                for i := b_{a_{t+1}} + 1 to k − 1 do
                    b_{a_{t+1}} := i
                    BraceFD(t + 1, t + 1, r2, RS2)
            b_{a_{t+1}} := 0
            s_{a_{t+1}} := 0
            tail := max − 1
        for j := tail to a_t + 1 do
            a_{t+1} := j
            s_{a_{t+1}} := t + 1
            for i := 1 to k − 1 do
                b_{a_{t+1}} := i
                BraceFD(t + 1, t + 1, r, RS)
                b_{a_{t+1}} := 0
            s_{a_{t+1}} := 0
end.
```

Figure 3. An optimized algorithm BraceFD(t, p, r,RS) to list all bracelets with fixed density.

4. Analysis

In this section, we show that our algorithm takes constant time on average to generate a bracelet. The computation tree is a way to represent the computational steps of a recursive algorithm. In our algorithm, each node of the computation tree represents a recursive call to BraceFD(t, p, r,RS) and each edge corresponds to transition from one recursive call to the other. The size of the computation tree of our algorithm is smaller than the computation tree of necklaces with the fixed density generation algorithm. Using the relation (4) and the fact that a necklace with the fixed density algorithm works in CAT, we claim that the size of the computation tree in our algorithm is proportional to $B_k(n, d)$.

```
procedure InitFixed()
    j, i: int

    for j := 1 to n do
        b_j := 0
        s_j := 0
    for j := 0 to d do
        a_j := 0

    a_d := n
    s_n := d
    for j := d − n + 1 to (n − 1)/d + 1 do
        a_1 := j
        a_j := 1
        for i := i to k − 1 do
            b_j := i
            BraceFD(1, 1, a_1 − 1, 0)
            b_j := 0
        s_j := 0

    end;
```

Figure 4. Algorithm to perform initialization for bracelets with fixed density.

For each recursive call, the amount of work done to append the next non-zero symbol in the current prenecklaces is constant. However, there are some nodes in our computation tree where CheckRev(t) is executed. Thus, they perform more than constant amount of work. Figure 5 shows the computation tree of $\mathbf{B}_2(8,4)$ and boxed nodes are the ones for which CheckRev(t) is called.

We prove that our algorithm works in CAT by showing that total symbol comparisons performed by CheckRev(t) during the entire scheme is proportional to $B_k(n,d)$. The total prenecklaces generated by our scheme is proportional to $B_k(n,d)$. Therefore, we map each comparison in CheckRev(t) to a unique prenecklace listed by our algorithm. For binary bracelets, the problems of listing bracelets with fixed density and bracelets with fixed content are same. Note that in case of binary bracelets we assume $d \leq n/2$. Fixed density bracelets with $d > n/2$ are listed by the fixed content algorithm presented in [6] in CAT.

Each recursive call in the necklaces with the fixed density algorithm increases the density of the current prenecklace. This means that the algorithm does not generate any prenecklace that ends with 0. Furthermore, CheckRev(t) is computed at most once for each prenecklace

Table 1. List of bracelets in $\mathbf{B}_3(6,4)$.

0 0 1 1 1 1	0 1 0 2 2 2
0 0 1 1 1 2	0 1 1 0 1 1
0 0 1 1 2 1	0 1 1 0 1 2
0 0 1 1 2 2	0 1 1 0 2 2
0 0 1 2 1 2	0 1 1 1 0 2
0 0 1 2 2 1	0 1 1 2 0 2
0 0 1 2 2 2	0 1 2 0 1 2
0 0 2 1 1 2	0 1 2 0 2 1
0 0 2 1 2 2	0 1 2 0 2 2
0 0 2 2 2 2	0 1 2 1 0 2
0 1 0 1 1 1	0 1 2 2 0 2
0 1 0 1 1 2	0 2 0 2 1 2
0 1 0 1 2 1	0 2 0 2 2 2
0 1 0 1 2 2	0 2 2 0 2 2
0 1 0 2 1 2	

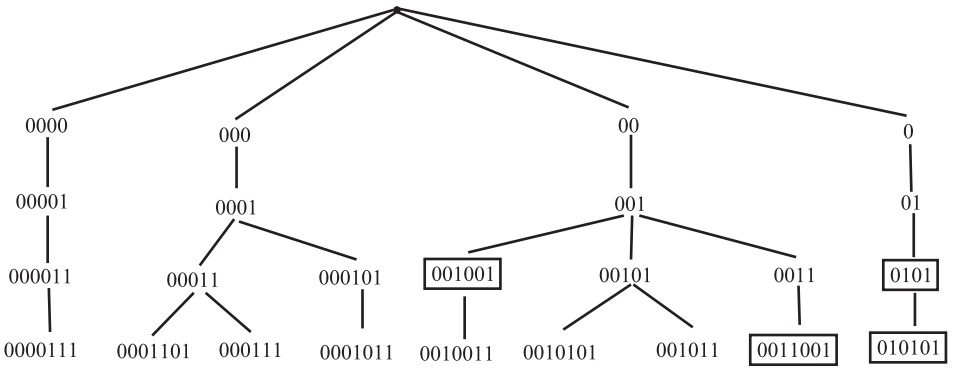

Figure 5. Computation tree for $\mathbf{B}_2(8,4)$ from BraceFD(t,p,r,RS).

$\beta = 0^i b_{i+1} b_{i+2} \cdots b_{a_t-i-1} 0^i b_{a_t}$ of length a_t, where $b_{i+1} = b_{a_t-i-1} \neq 0$ and $b_{a_t} \geq b_{i+1}$. CheckRev(t) compares symbol b_j with b_{a_t-j} for $i+1 \leq j \leq a_t/2$. We stop doing comparisons when either $j > a_t/2$ or $b_j \neq b_{a_t-j}$. Since there is at most one unequal comparison for each prenecklace, we count the cost of such comparison as constant.

Let us assume that β and γ are two different prenecklaces for which CheckRev(t) is computed:

$$\beta = 0^i b_{i+1} b_{i+2} \cdots b_{a_t-i-1} 0^i b_{a_t},$$
$$\gamma = 0^{i'} c_{i'+1} c_{i'+2} \cdots c_{a_t-i'-1} 0^{i'} c_{a_t}.$$

Let $n_k(\beta,j)$ denote the number of occurrences of symbol k in substring $b_1 b_2 \cdots b_j$. For example, $n_0(001111001011,7) = 3$ and $n_1(001111001011,7) = 4$. Also, let $R(\beta,j)$ denote a subsequence of first j characters of β in which all occurrences of symbol 0 are omitted. For example, $R(0012201022100,7) = 1221$. First, we show that symbol comparisons made by the entire algorithm is proportional to the number of prenecklaces generated for the binary case and then generalize our result. Consider the following mapping f:

$$f(\beta,j) = \begin{cases} 0^{n_0(\beta,j)} 1^{n_1(\beta,j)} b_{j+1} \cdots b_{a_t-i-1} 0^i b_{a_t} & \text{if } b_j = 0, \\ 0^{n_0(\beta,j)} b_{j+1} \cdots b_{a_t-i-1} 0^i b_{a_t} 1^{n_1(\beta,j)} & \text{if } b_j = 1. \end{cases}$$

For example, $f(001110011100111001,3) = 001100111001110011$ and $f(001110011100111001 10 01,6) = 000111011100111001$.

Here, β and j are valid only if there exists a corresponding equal comparison made by CheckRev(t). It is evident that the mapping preserves length and content. Furthermore, we claim that $f(\beta,j)$ is a valid prenecklace generated by our algorithm.

LEMMA 4.1 *For all* $\beta = 0^i b_{i+1} \cdots b_{a_t-i-1} 0^i b_{a_t}$ *and for all* j *such that* $0^i b_{i+1} \cdots b_j = 0^i b_{a_t-i-1} \cdots b_{a_t-j}$, $f(\beta,j)$ *is a prenecklace.*

Proof $f(\beta,j)$ has a maximum number of zeros in the beginning for the cases when $b_j = 0$, $n_0(\beta,j) > i$ or $b_{j+1} = 0$, hence a valid prenecklace.

Now to prove that $f(\beta,j)$ is a prenecklace when $b_j = b_{j+1} = 1$ and $n_0(\beta,j) = i$, let $\text{lyn}(\beta) = p$. Note that β is a binary prenecklace so $p < a_t$ must hold. We can rewrite $\beta = (0^i b_{i+1} b_{i+2} \cdots b_p)^d 0^i b_{i+1} b_{i+2} \cdots b_m$ and $f(\beta,j) = 0^i b_{j+1} b_{j+2} \cdots b_p (0^i b_{i+1} b_{i+2} \cdots b_p)^{d-1} 0^i b_{i+1} b_{i+2} \cdots b_m 1^{n_1(\beta,j)}$, where d is a positive integer and $i < m < p$. Since $b_{i+1} b_{i+2} \cdots b_j = 1^{j-i}$ and

$b_{j+1} = 1$, this means $j < p$ and $0^i b_{j+1} b_{j+2} \cdots b_p$ is a Lyndon word. Also, the following relation must hold:

$$0^i b_{j+1} b_{j+2} \cdots b_p < 0^i b_{i+1} b_{i+2} \cdots b_p. \tag{5}$$

Using relation (5) and repeated application of Lemma 2.2 in [10], we can conclude that $0^i b_{j+1} b_{j+2} \cdots b_p (0^i b_{i+1} b_{i+2} \cdots b_p)^{d-1}$ is also a Lyndon word. Again, using relation (5), we claim that $b_{j+1} b_{j+2} \cdots b_{j+m-i} \leq b_{i+1} b_{i+2} \cdots b_m$. This implies

$$0^i b_{j+1} b_{j+2} \cdots b_p (0^i b_{i+1} b_{i+2} \cdots b_p)^{d-1} 0^i b_{i+1} b_{i+2} \cdots b_m$$

is a prenecklace. Since 1 is the largest symbol, by Theorem 2.1,

$$0^i b_{j+1} b_{j+2} \cdots b_p (0^i b_{i+1} b_{i+2} \cdots b_p)^{d-1} 0^i b_{i+1} b_{i+2} \cdots b_m 1^{n_1(\beta,j)}$$

is also a prenecklace. ■

Now, we prove that total comparisons made by all CheckRev(t) calls is proportional to the total prenecklaces generated. We claim that $f(\beta, j)$ is one to one for all valid β and j.

LEMMA 4.2 *For all* $\beta = 0^i b_{i+1} \cdots b_{a_t - i - 1} 0^i b_{a_t}$ *and for all* j *such that* $0^i b_{i+1} \cdots b_j = 0^i b_{a_t - i - 1} \cdots b_{a_t - j}$, *the mapping* $f(\beta, j)$ *is one to one.*

Proof It is clear from the mapping that $f(\beta, j) \neq f(\beta, j')$ whenever $j \neq j'$. Now, let β and γ be two prenecklaces of same length. Suppose $f(\beta, j) = f(\gamma, j')$, where $\gamma \neq \beta$. Observe that $f(\beta, j)$ ends with 01 when $b_j = 0$ and ends with 11 when $b_j = 1$. Furthermore, the substring $b_{j+1} b_{j+2} \cdots b_{a_t - i - 1} 0^i b_{a_t}$ is not affected by the mapping f. Since, $f(\beta, j) = f(\gamma, j')$ for $j, j' \leq t/2$, it follows that $b_j = c_{j'}$. Moreover, the second half of β and γ must be equal, that is,

$$b_{a_t/2+1} b_{a_t/2+2} \cdots b_{a_t - i - 1} 0^i b_{a_t} = c_{a_t/2+1} c_{a_t/2+2} \cdots c_{a_t - i' - 1} 0^{i'} c_{a_t}.$$

Similarly, the trailing number of zeros in both strings must be equal, that is, $i = i'$. Now assume that q is the minimum index such that $b_q \neq c_q$, where $q \leq a_t/2$. For all $i + 1 \leq j, j' < q$ clearly $f(\beta, j) \neq f(\gamma, j')$. When $j = j' = q$, either $b_j \neq b_{a_t - j}$ or $c_j \neq c_{a_t - j}$ because $b_{a_t - j}$ lies in the second half of β and $b_{a_t - j} = c_{a_t - j}$. So either j or j' is invalid, a contradiction. ■

The mapping shown above works only for binary bracelets. In some cases, $f(\beta, j)$ may not be a prenecklace when $k > 2$. Consider the case when $\beta = 00122010221001$. Here, $f(\beta, 3) = 00220102210011$ is not a valid prenecklace. Now we consider another mapping g:

$$g(\beta, j) = \begin{cases} 0^{n_0(\beta,j)} R(\beta, j) b_{j+1} \cdots b_{a_t - i - 1} 0^i b_{a_t} & \text{if } b_j = 0, \\ 0^{n_0(\beta,j)} b_{j+1} \cdots b_{a_t - i - 1} 0^i b_{a_t} R(\beta, j) & \text{if } b_j \neq 0 \text{ and } n_0(\beta, j) > i, \\ 0^i b_{a_t - i - 1} \cdots b_{a_t - j} 0^i b_{i+1} \cdots b_{a_t - j - 1} b_{a_t} & \text{if } b_j \neq 0, n_0(\beta, j) = i \text{ and } n_0(\beta, a_t) = 2i, \\ 0^i b_{a_t - i - 1} \cdots b_{a_t - j} 0^i b_{i+1} \cdots b_{l-1} b_{a_t} b_l \cdots b_{a_t - j - 1} & \text{if } b_j \neq 0, n_0(\beta, j) = i, \\ & n_0(\beta, a_t) > 2i \text{ and } b_{a_t - j - 1} > 0, \end{cases}$$

where l is the largest index such that $l < a_t - j$ and $a_l = 0$. We assume that β and j are valid only if there exists a corresponding equal comparison made by CheckRev(t).

For example, $g(001220110221001, 6) = 000122110221001$, $g(001220110221001, 7) = 0001$ 02210011221 and $g(001220110221001, 4) = 001200122011102$. It is evident that the mapping preserves both length and content.

Note that the mapped string in first three cases does not end with zero. The only case left is when $n_0(\beta, j) = i, n_0(\beta, a_t) > 2i$ and $b_{a_t-j-1} = 0$. We claim that such a case can occur at most once for each prenecklace. This is due to the fact that if the next comparison between b_{j+1} and b_{a_t-j-1} is an unequal comparison, then no more comparisons are performed for β. Otherwise, $b_{j+1} = 0$ and $n_0(\beta, j+1) > i$, which ensures that this case would never arise in later comparisons. Thus, we consider constant cost for such comparisons.

LEMMA 4.3 *For all prenecklaces β of the form $0^i b_{i+1} \cdots b_{a_t-i-1} 0^i b_{a_t}$ and for all j such that $0^i b_{i+1} \cdots b_j = 0^i b_{a_t-i-1} \cdots b_{a_t-j}$, $g(\beta, j)$ is a prenecklace.*

Proof It is evident from the mapping that $g(\beta, j)$ preserves both length and content. In order to show that $g(\beta, j)$ is a valid prenecklace, we consider it case by case. In the first two cases, $g(\beta, j)$ has a maximum number of zeros in the beginning, hence it is a valid prenecklace.

In the third case, we remove the j characters before b_{a_t} and put their reversed string in the beginning. Since $0^i b_{i+1} b_{i+2} \cdots b_j = 0^i b_{a_t-i-1} b_{a_t-i-2} \cdots b_{a_t-j}$ and there are no strings of zero's other than first and last $2i$ zeros, the mapped string

$$0^i b_{i+1} b_{i+2} \cdots b_j 0^i b_{i+1} b_{i+2} \cdots b_j b_{j+1} \cdots b_{a_t-j-1} b_{a_t}$$

is a valid prenecklace.

Now to prove the last case, consider the following observations:

(1) $0^i b_{i+1} b_{i+2} \cdots b_j 0^i b_{i+1} b_{i+2} \cdots b_j b_{j+1} \cdots b_{a_t-j-1}$ is a prenecklace.
(2) Furthermore, it follows directly from Lemma 2.2 in [1] that $0^i b_{i+1} b_{i+2} \cdots b_{a_t-j-l} \leq b_l b_{l+1} \cdots b_{a_t-j-1}$ in β, where l is the largest index in β such that $b_l = 0$ and $l < a_t - j$.

In the last case of mapping g, we move j characters before b_{a_t} to the beginning and place b_{a_t} before b_l. Hence, the substring $0^i b_{a_t-i-1} b_{a_t-i-2} \cdots b_{a_t-j} 0^i b_{i+1} b_{i+2} \cdots b_{l-1} b_l$ is a prenecklace by observation 1. Since $b_l = 0$ and $b_{a_t} > 0$, it implies that $0^i b_{a_t-i-1} b_{a_t-i-2} \cdots b_{a_t-j} 0^i b_{i+1} b_{i+2} \cdots b_{l-1}$ b_{a_t} is a Lyndon word. Using observation 2 and fundamental theorem of necklace, we conclude that

$$0^i b_{a_t-i-1} b_{a_t-i-2} \cdots b_{a_t-j} 0^i b_{i+1} b_{i+2} \cdots b_{l-1} b_{a_t} b_l b_{l+1} \cdots b_{a_t-j-1}$$

is a valid prenecklace. ■

LEMMA 4.4 *For all prenecklaces β of the form $0^i b_{i+1} \cdots b_{a_t-i-1} 0^i b_{a_t}$ and for all j such that $0^i b_{i+1} \cdots b_j = 0^i b_{a_t-i-1} \cdots b_{a_t-j}$, the mapping $g(\beta, j)$ is one to one.*

Proof Now to prove that g is *one to one*, we assume that $g(\beta, j) = g(\gamma, j')$ but $\beta \neq \gamma$. Note the following observations about the strings mapped by g:

(1) The first two cases have more than i zeros in the beginning, whereas the last two cases do not contain any substring of consecutive zeros greater than i.
(2) The ending string in the first two cases is different.
(3) The contents of prenecklaces are different in the last two cases of mapping g.

This implies that same case must be applied to both β, j and γ, j'. Otherwise, $g(\beta, j)$ would not be the same as $g(\gamma, j')$. Similarly, we can conclude that $i = i'$.

In the first two cases, the substring $b_{j+1} b_{j+2} \cdots b_{a_t-i-1} 0^i b_{a_t}$ is not affected by the mapping g, since $g(\beta, j) = g(\gamma, j')$ for $j, j' \leq a_t/2$. This implies that the second half of β and γ must be equal,

that is,

$$b_{a_t/2+1}b_{a_t/2+2}\cdots b_{a_t-i-1}0^i b_{a_t} = c_{a_t/2+1}c_{a_t/2+2}\cdots c_{a_t-i-1}0^i c_{a_t}.$$

Now assume that q is the minimum index such that $b_q \neq c_q$, where $q \leq a_t/2$. For all $1 \leq j, j' < q$ clearly $g(\beta, j) \neq g(\gamma, j')$. When $j = j' = q$, either $b_j \neq b_{a_t-j}$ or $c_j \neq c_{a_t-j}$ because b_{a_t-j} lies in the second half of β and $b_{a_t-j} = c_{a_t-j}$. Therefore, either j or j' is invalid, a contradiction.

In the last two cases, j characters of β before b_{a_t} are moved to the beginning, such that $b_{i+1}b_{i+2}\cdots b_j$ does not contain any zero. Since $g(\beta, j) = g(\gamma, j')$, j must be same as j' which contradicts that $\beta \neq \gamma$.

Since it is clear from the mapping that $g(\beta, j) \neq g(\beta, j')$ whenever $j \neq j'$, it is proved that g is one to one. ■

Let $P'_k(n, d)$ be the number of prenecklaces generated by a necklace with the fixed density algorithm, and $C_k(n)$ be the number of equal comparisons made by CheckRev(t) function on all prenecklaces of length n. The total number of equal comparisons made by our algorithm are $\sum_{i=1}^{n} C_k(i)$. The following theorem proves that our algorithm works in CAT.

THEOREM 4.1 $\sum_{i=1}^{n} C_k(i) \leq cB_k(n, d)$, *where c is a constant.*

Proof In our algorithm, we stop recursion when a prenecklace of density $d - 1$ has been generated and the last non-zero symbol is appended at position n such that the generated string is a necklace. This means that the algorithm does not generate prenecklaces of length n. Using Lemma 4.4, the following bound holds:

$$\sum_{i=1}^{n-1} C_k(i) \leq P'_k(n, d).$$

From Lemma 4.1 given in [10], one can show that the number of prenecklaces of length n and density d that does not end with a zero is less than $2N_k(n, d)$. This implies that $C_k(n) \leq 2N_k(n, d)$. Also, $P'_k(n, d) \leq c'N_k(n, d)$. Hence, we obtain the following relation:

$$\begin{aligned}\sum_{i=1}^{n} C_k(i) &\leq (c' + 2)N_k(n, d)\\ &\leq 2(c' + 2)B_k(n, d)\\ &= cB_k(n, d).\end{aligned}$$

■

5. Conclusion

In this paper, we develop an efficient scheme to generate bracelets with fixed density. Using sophisticated combinatorial techniques, we prove that our algorithm works in CAT and takes asymptotic linear space. As a future work, we aim to explore the applications of our fixed density bracelet generation algorithm. Furthermore, we also want to develop schemes for listing other restricted classes of bracelets.

The scheme has been implemented in C and can be obtained from the authors upon request.

Acknowledgements

This work was partially supported by FAST-NU under the grant 11L-270/NU-R/11. The authors thank Joe Sawada for all the useful discussions and the valuable comments.

References

[1] K. Cattell, F. Ruskey, J. Sawada, M. Serra, and C.R. Miers, *Fast algorithms to generate necklaces, unlabeled necklaces, and irreducible polynomials over GF(2)*, J. Algorithms 37 (2000), pp. 267–282.

[2] V.M.F. Dias, C.M.H. Figueiredo, and J.L. Szwarcfiter, *On the generation of bicliques of a graph*, Discrete Appl. Math. 155 (2007), pp. 1826–1832.

[3] P. Emmel and R. Hersch, *Exploring ink spreading*, Proceedings of the 8th IS and T/SID Color Imaging Conference: Color Science and Engineering, Scottsdale, AZ, 2000, pp. 335–341.

[4] E.N. Gilbert and J. Riordan, *Symmetry types of periodic sequences*, Illinois J. Math. 5 (1961), pp. 657–665.

[5] Y. Ishida, Y. Kato, L. Zhao, H. Nagamochi, and T. Akutsu, *Branch-and-bound algorithms for enumerating treelike chemical graphs with given path frequency using detachment-cut*, J. Chem. Inf. Model. 50 (2010), pp. 934–946.

[6] S. Karim, J. Sawada, Z. Alamgir, and S.M. Husnine, *Generating bracelets with fixed content*, Theor. Comput. Sci. 475 (2013), pp. 103–112, doi:10.1016/j.tcs.2012.11.024.

[7] D.E. Knuth, *The Art of Computer Programming Volume 4A, Combinatorial Algorithms*, Addison-Wesley, Reading, MA, 2011.

[8] P. Lisonek, *Computer assisted studies in algebraic combinatorics*, Dissertation, Johannes Kepler University, Linz, Austria, 1994.

[9] F. Ruskey, *Combinatorial Generation*, Working Version, University of Victoria, Victoria, BC, Canada, 2003. Available at http://www.1stworks.com/ref/RuskeyCombGen.pdf.

[10] F. Ruskey and J. Sawada, *An efficient algorithm for generating necklaces of fixed density*, SIAM J. Comput. 29 (1999), pp. 671–684.

[11] F. Ruskey, C. Savage, and T. Wang, *Generating necklaces*, J. Algorithms 13 (1992), pp. 3247–3259.

[12] J. Sawada, *Generating bracelets in constant amortized time*, SIAM J. Comput. 31 (2001), pp. 259–268.

[13] J. Sawada, *A fast algorithm to generate necklaces with fixed content*, Theoret. Comput. Sci. 301 (2003), pp. 447–489.

[14] T. Uno, *An efficient algorithm for solving pseudo clique enumeration problem*, Algorithmica 56 (2010), pp. 3–16.

[15] V. Vajnovszki, *Gray code order for Lyndon words*, Discrete Math. Theor. Comput. Sci. 9 (2007), pp. 145–152.

[16] V. Vajnovszki, *More restrictive Gray codes for necklaces and Lyndon words*, Inform. Process. Lett. 106 (2008), pp. 96–99.

[17] M. Weston and V. Vajnovszki, *Gray codes for necklaces and Lyndon words of arbitrary base*, Pure Math. Appl. 17 (2006), pp. 175–182.