F.S.A.I., Vol. 11, No. 1-3, pp. 5-20, 2005

On Combinatorial Algorithms for Indexing *m*-ary Word Weight Classes

Raffaele Mascella, Danilo Pelusi and Luca G. Tallini*

Dipartimento di Scienze della Comunicazione
Università degli Studi di Teramo
Coste Sant'Agostino - 64100 Teramo, Italy
E-mail: rmascella@tin.it, pelusid@virgilio.it, ltallini@unite.it

Abstract. One problem in coding theory is to find methods to index and associate check digits to the classes of suitable partitions on the whole set of source sequences $Z_m^k, m, k \in \mathrm{N}$. Given a word $x \in Z_m^k$, let $s(x) = (s_0(x), s_1(x), ..., s_{m-1}(x))$ be the weight of *x*, where *si*(*x*) indicates the number of occurrences of the symbol *i* in the word *x*. The source sequences set Z_m^k remains partitioned in $\binom{k+m-1}{m-1}$ different classes, each of which contains words with the same symbol weights. In such context we give some combinatorial formulas and related algorithms to rank (i.e. to index preserving an ordering) and unrank in lexicographic, inverse lexicographic and co-lexicographic order the set of all possible word weight classes. In the first approach, investigated for the lexicographic case, it is required to store some binomial coefficients which globally require $S = O\left(m^2 k \log_m (m+k)\right)$ *m*-ary digits storing elements and the whole algorithm requires $T = O\left(m^2 \log_m (m+k)\right)$ *m*-ary digit operations for the ranking process, and $T = O\left(mk \log_m (m+k)\right)$ for the unranking one. In the second approach, investigated for all the orderings under consideration, the computation of the binomial coefficients is optimized and the whole algorithms require $T = O\left(mk \log_m^2 (m+k)\right)$ *m*-ary digit operations by storing $S = O\left(m \log_m (m+k)\right)$ *m*-ary digits, for both ranking and unranking functions.

Key words: combinatorial algorithms, *m*-ary codes, compositions, integer partitions, lexicographic order, inverse lexicographic order, colexicographic order.

1 Introduction

Consider the *m*-ary alphabet $Z_m = \{0,1,...,m-1\}, m \geq 2$, and given $k \in N$, let Z_m^k be the set of k-sequences, or words, over the set Z_m. Let $x = (x_1,...,x_k) \in Z_m^k$ be such a word. Let $s(x) = (s_0(x), s_1(x), ..., s_{m-1}(x))$ be the weight function of the

* This work is supported by the Italian MIUR Grant COFIN-2003013538.

word x, in which $s_i(x) = \{ j \mid j = 1,...,k,\ x_j = i \}$ is the weight of the symbol *i* in the word *x*. For example, with *m* = 3, *k* = 6 the weight of the word *x* = 012001 is *s*(x) = (3, 2, 1). By applying the weight function

$$s : Z_m^k \to S_k^m = \{(s_0, s_1,...,s_{m-1}) \mid s_0 + s_1 + ... + s_{m-1} = k\} \subset Z_{k+1}^m$$

on the set Z_m^k, it remains defined a quotient set Z_m^k / s, in which each equivalence class is of the type $[x] = \{ y \mid s_j(x) = s_j(y),\ j = 0,1,...,m-1 \}$. We can observe that considering these classes over the *m*-ary alphabet, that is S_k^m , is equivalent to consider the compositions of the integer *k* in *m* parts, that is the ordered partitions of the integer *k* with *m* blocks.

Often, in coding theory, it is necessary to design algorithms that generate the desired objects with respect to a given order, for example in lexicographic order, or inversely, that assign to any object an index corresponding to the position occupied in such order.

As in much part of the literature on the subject (see, for example [4], [6]), in this paper we are interested in the so-called ranking and unranking algorithms. A ranking algorithm determines the position of a combinatorial object among all the objects (with respect to a given order). An unranking algorithm creates (or finds) the object having a specified rank. Thus, ranking and unranking can be considered as inverse operations. Indeed, if we consider a bijection $P : S_k^m \to \{0,1,...,| S_k^m | -1\}$, where $| S_k^m | = \binom{k+m-1}{m-1}$, *P* is a ranking function if it defines a total order on the elements of S_k^m, by the obvious rule $s <_p t \Leftrightarrow P(s) < P(t)$.

Conversely, there is a unique rank function associated with any total order defined on S. In this case such rank function can be expressed, in general, through the relation $P(s) = | \{ t \in S_k^m : t <_p s \} |$. Finally, given the rank function *P*, it remains defined a unique unranking function associated with *P*, say *I*. This function is the bijection

$$I : \{0,1,...,| S_k^m | -1\} \to S_k^m,\ I(i) = s \Leftrightarrow P(s) = i$$

for all $s \in S_k^m$ and for all $i \in \{0,1,...,| S_k^m | -1\}$.

In this paper we give efficient ranking and unranking algorithms for lexicographic, inverse lexicographic and co-lexicographic orderings. The algorithms apply to a large range of A.I. problems, including linguistic descriptions – and related, fuzzy logic based-descriptions, genomics, etc.

2 Lexicographic Ranking

Let it given on S_k^m the lexicographic order (often called lex order), i.e. if $s = (s_0, s_1,...,s_{m-1})$, $t = (t_0, t_1,...,t_{m-1})$ are elements in S_k^m then $s <_{lex} t$ if, and only

if, $s \neq t$ and it exists $w \in \{0,1,...,m-1\}$ such that $s_i = t_i$ for $i = 0, 1,...,w\text{-}1$ and $s_w < t_w$. It remains so defined the ranking function associated with the lexicographic order, say P_{lex}, and a unique unranking function associated with P_{lex}, say I_{lex}.

For example, given $m = 3$ and $k = 5$, the lex order of the $\binom{5+3-1}{3-1} = 21$ weight symbol sequences is listed in Table 1.

Table 1

Lexicographic order for $m = 3$ and $k = 5$

s	$P_{lex}(s)$	s	$P_{lex}(s)$	s	$P_{lex}(s)$
(0,0,5)	0	(1,1,3)	7	(2,3,0)	14
(0,1,4)	1	(1,2,2)	8	(3,0,2)	15
(0,2,3)	2	(1,3,1)	9	(3,1,1)	16
(0,3,2)	3	(1,4,0)	10	(3,2,0)	17
(0,4,1)	4	(2,0,3)	11	(4,0,1)	18
(0,5,0)	5	(2,1,2)	12	(4,1,0)	19
(1,0,4)	6	(2,2,1)	13	(5,0,0)	20

From [5] we know that, if we assume the order to be the lexicographic one, the indexing P_{lex} of the word weight classes can be given through the formula

$$P_{lex}(s_0,...,s_{m-1}) = \sum_{h=1}^{m-1}\left[\binom{h+\sum_{l=1}^{h+1} s_{m-l}}{h} - \binom{h+\sum_{l=1}^{h} s_{m-l}}{h}\right] \quad (1)$$

Example 1. Suppose $m = 5$ and $k = 8$. Let it given the word $x = 30224402$. Then $s(x) = s = (2, 0, 3, 1, 2)$ and

$$P_{lex}(s) = \binom{4}{1} - \binom{3}{1} + \binom{8}{2} - \binom{5}{2} + \binom{9}{3} - \binom{9}{3} + \binom{12}{4} - \binom{10}{4} = 4\text{-}3\text{+}28\text{-}10\text{+}495\text{-}210 = 304.$$

The aim of this section is to analyze and project efficient algorithms for its application; also, we want to compare this algorithms with the ones projected and investigated in the following sections, i.e. the algorithms obtained by considering the inverse lexicographic order and the co-lexicographic order.

The formula (1) for lexicographic ordering immediately yields a ranking algorithm, as proposed in [5] which we present here as the Basic algorithm.

Algorithm 1 (Basic Ranking Lex Algorithm)

Input: $\vec{s}[0:m-1] \in Z_m^k$.

Output: $r \in \{0,1,...,\binom{k+m-1}{m-1} - 1\}$

Execute the following steps.

```
r ← 0
c2 ← s[m - 1]
for h ← 1 to m - 1 do {
c1 ← c2
c2 ← c2 + s[m - h - 1]
r ← r + (h+c2 choose h) − (h+c1 choose h)
}
return(r)
```

In this case we intend to store the binomial coefficients $\binom{h+c_i}{h}$ used during the ranking process. In other words, the main idea of this approach is to save in time complexity, by avoiding to compute, each time, similar coefficients. Necessarily this leads us to store all such data, requiring more memory. We advice the lector that in this paper all the integer multiplication and division are considered as they were computed in the classical way, the well-known "paper and pencil" method, so their time complexities are in both cases $O(D_1D_2)$, where D_1; D_2 indicate the number of digits of factors and divisors. Clearly, using the improvemements known in the literature for the basic operations, such as the ones based on Discrete Fourier Transform, the bounds we give here can be subsequently improved.

We have to store all the binomial coefficients of the kind $\binom{a+b}{a}$, with a = 0, 1,..,m - 1 and b = 0,1,...,k; so we have $(m - 1)k$ data to store, in which

$$size\binom{a+b}{a} \le size\binom{m+k}{m} = \log_m \binom{m+k}{m}$$

Then, by Stirling's approximation, we have

$$\log_m \binom{m+k}{m} = \log_m \frac{(m+k)!}{m!k!} \approx \log_m \frac{\sqrt{2\pi(m+k)}\; e^{-m-k}(m+k)^{m+k}}{m!\sqrt{2\pi\, k}\; e^{-k}\, k^k} \qquad (2)$$

$$= \log_m \left(\sqrt{\frac{m+k}{k}}\; e^{-m} \frac{(m+k)^{m+k}}{k^k} \right) - \log_m m!$$

From Lagrange's Theorem, $f(x + x_0) - f(x) = f'(\xi)\, x_0$, with $\xi \in (x, x + x_0)$, consider $f(x) = x \log_m x$.

Being $f'(x) = \log_m x + \log_m e$ and observing that f' is an increasing function, that implies $f'(\xi) \le f'(x + x_0)$, we have

$$\begin{aligned}(\mathrm{m}+\mathrm{k})\log_{\mathrm{m}}(m+k)-k\log_m k = f'(\xi)m = (\log_m \xi+\log_m e)m \le \\ \le (\log_m(m+k)+\log_m e)m\end{aligned} \tag{3}$$

Finally, from (2) and (3), it holds

$$\log_m\binom{m+k}{m} \le \log_m\sqrt{\frac{m+k}{k}} - m\log_m e + m\log_m(m+k) + m\log_m e - \log_m m!$$

$$= m\log_m(m+k) + \frac{1}{2}\log_m\frac{m+k}{k} - \log_m m! = O(m\log_m(m+k))$$

Then, if the binomial coefficients $\binom{h+c_i}{h}$ are computed and stored in a table, the whole algorithm has a space complexity of $O(m^2 k \log_m(m+k))$ m-ary digits.

In order to compute the time complexity, note that it follows, first of all, from the statement

$$r \leftarrow r + \binom{h+c_2}{h} - \binom{h+c_1}{h}$$

This statement, in the worst case, is executed $\Theta(m)$ times and any time it sums $\Theta(1)$ numbers of size $O(m\log_m(m+k))$. Then the complexity is $O(m^2 \log_m(m+k))$. In fact, in the first sum it may require $cm\log_m(m+k)+1$ digit operations, in the second sum it may require $cm\log_m(m+k)+2$ digit operations, in the m-th sum it may require $cm\log_m(m+k)+m$ digit operations. So the final time complexity is $O(cm^2\log_m(m+k)+\frac{m^2+m}{2}) = O(m^2\log_m(m+k))$ m-ary digit operations.

In the Basic Ranking Lex Algorithm, some operations in the loop of the variable h are not necessary since they give no contribution. Therefore, we can consider a useful embellishment.

```
r ← 0
c2 ← s[m - 1]
for h ← 1 to m - 1 do {
      c1 ← c2
      if s[m - h - 1] > 0 then  {
            c2 ← c2 + s[m - h - 1]
            r ← r + C(h + c2, h) - C(h + c1, h)
      }
}
return(r)
```

Example 2. Suppose, as in Example 1, that m = 5, k = 8 and is given the word x = 30224402 $\in S_8^5$. Then s = (2, 0, 3, 1, 2) and the algorithm executes the following steps.

r = 0, c_2 = 2, $\Rightarrow$

h = 1, c_1 = 2, s[3] = 1 c_2 = 3, r = $\binom{1+3}{1} - \binom{1+2}{1} = 1$

$\Rightarrow$

h = 2, c_1 = 3, s[2] = 3 c_2 = 6, r = 1 + $\binom{2+6}{2} - \binom{2+3}{2} = 19$ $\Rightarrow$

h = 3, c_1 = 6 s[1] = 1 $\Rightarrow$

h = 4, c_1 = 6, s[0] = 2 c_2 = 8, r = 19 + $\binom{4+8}{4} - \binom{4+6}{4} = 304$

The computation of the function I_{lex} may be done in the same way, i.e. using the binomial coefficients stored in the table. In this case, described in the Basic Unranking Lex Algorithm below, the process employs the same memory, needing to store $O(m^2 k \log_m (m + k))$ m-ary digits, and a time complexity of $O(mk \log_m (m + k))$ m-ary digit operations.

Algorithm 2 (Basic Unranking Lex Algorithm)

Input: $r \in \{0,1,\ldots,\binom{k+m-1}{m-1} - 1\}$.

Output: $\vec{s}[0:m-1] \in Z_m^k$.

Execute the following steps.

```
c3 ← k
for h ← 1 to m - 1 do {
      s[h - 1] ← 0
      c1 ← (m − h + c3 choose m − h)
      c2 ← (m − h + c3 − 1 choose m − h)
      while c1 - c2 ≤ r do {
            s[h - 1] ← s[h - 1] + 1
            r ← r - c1 + c2
            c3 ← c3 - 1
            c1 ← c2
            c2 ← (m − h + c3 − 1 choose m − h)
      }
```

```
}
s[m - 1] ← k - c3
return ( s⃗ )
```

A significant modification could be introduced with a Second ranking lex algorithm, by executing every time all the computation involved in the process but using, on each loop, the calculations just performed for the previous binomial coefficient and executing only the computations needed to get the actual coefficient. In this algorithm the process can be accomplished by storing $O(m\log_m (m+k))$ m-ary digits and performing $O(mk \log^2_m (m+k))$ m-ary digit operations. As previously specified, we consider multiplications and divisions to have the same complexity given by the product of the sizes of the two numbers involved.

Algorithm 3 (Second Ranking Lex Algorithm)

Input: $\vec{s}[0:m-1] \in Z_m^k$.

Output: $r \in \{0,1,...,\binom{k+m-1}{m-1}-1\}$.

Execute the following steps.

```
c2 ← 1
r ← 0
c4 ← s[m - 1]
for h ← 1 to m - 1 do {
      c3 ← c4
      a ← c2 (h + c3)/h
      c4 ← c4 + s[m - h - 1]
      c1 ← a
      for b ← 1 to s[m - h - 1] do {
            c2 ← c2 (h + b + c3 - 1)/(b + c3)
      }
      c2 ← c2 (h + c4)/h
      r ← r + c2 - c1
}
return (r)
```

In the previous Algorithm, some operations that give no contribution to the global sum to be stored in r may be avoided. Also in this case we can consider a useful embellishment.

```
c2 ← 1
r ← 0
c4 ← s[m - 1]
```

```
for h ← 1 to m - 1 do {
      c3 ← c4
      a ← c2 (h + c3)/h
      if s[m - h - 1] > 0 then {
            c4 ← c4 + s[m - h - 1]
            c1 ← a
            for b ← 1 to s[m - h - 1] do {
                  c2 ← c2 (h + b + c3 - 1)/(b + c3)
            }
            c2 ← c2 (h + c4)/h
            r ← r + c2 - c1
      }
      else {
            c2 ← a
      }
}
return (r)
```

Now we can unravel the previous algorithm to obtain an unranking lex algorithm. In this case the process presented employs the same memory, needing to store $O(m\log_m(m+k))$ m-ary digits, and requires $O(mk\log^2_m(m+k))$ m-ary digit operations. In the following algorithm unnecessary computations are avoided.

Algorithm 4 (Unranking Lex Algorithm)

Input: $r \in \{0,1,\ldots,\binom{k+m-1}{m-1}-1\}$

Output: $\vec{s}[0:m-1] \in Z_m^k$

Execute the following steps.

```
a ← k
c1 ← (m - 1 + a choose m - 1)
c2 ← 2 c1
for h ← 1 to m - 2 do {
      s[h - 1] ← 0
      if a > 0 {
            c2 ← c2 - c1
            c1 ← c2 a/(m - h + a)
            while r - c2 + c1 ≥ 0 do {
                  r ← r - c2 + c1
                  s[h - 1] ← s[h - 1] + 1
```

```
                    a ← a - 1
                    c2 ← c1
                    c1 ← c2 + a/(m - h + a)
                    }
        }
}
s[m - 2] ← r
s[m - 1] ← a - r
return ( s⃗ )
```

3 Inverse Lexicographic Ranking

A further possibility to rank *m*-ary word weight classes is to consider the inverse lexicographic order (or inverse lex order), i.e. if $s = (s_0, s_1,...,s_{m-1})$, $t = (t_0, t_1,..., t_{m-1})$ are elements in S_k^m then $s <_{invlex} t$ if, and only if, $s \neq t$ and it exists $w \in \{0,1,...,m-1\}$ such that $s_i = t_i$ for $i = 0, 1,..., w - 1$ and $t_w < s_w$.

For example, given m = 3 and $k = 4$, the inverse lex order of the $\binom{4+3-1}{3-1} = 15$ weight symbol sequences is listed in Table 2.

Table 2

Inverse lexicographic order for m = 3 and $k = 4$

s	$P_{invlex}(s)$	s	$P_{invlex}(s)$	s	$P_{invlex}(s)$
(4,0,0)	0	(2,0,2)	5	(0,4,0)	10
(3,1,0)	1	(1,3,0)	6	(0,3,1)	11
(3,0,1)	2	(1,2,1)	7	(0,2,2)	12
(2,2,0)	3	(1,1,2)	8	(0,1,3)	13
(2,1,1)	4	(1,0,3)	9	(0,0,4)	14

The main reason that suggests us to consider this order comes out from (5), since the amount of calculation needed to rank a sequence in this case is smaller than the amount needed in the normal lexicographic order.

Remark. For any $s \in S_k^m$, and therefore among lex and inverse lex orders, their associated ranking functions hold the following property

$$P_{lex}(s) + P_{invlex}(s) = |S_k^m| - 1 \tag{4}$$

in fact
$$P_{lex}(s) + P_{invlex}(s) = |\{t \in S_k^m \mid t <_{lex} s\}| + |\{t \in S_k^m \mid t <_{invlex} s\}| =$$
$$= |\{t \in S_k^m \mid t \neq s\}| = |S_k^m| - 1$$

Theorem 1. *The function P_{invlex} can be expressed through the formula*

$$P_{invlex}(s_0,\ldots,s_{m-1}) = \sum_{h=1}^{m-1} \binom{h-1+\sum_{l=1}^{h} s_{m-l}}{h} \tag{5}$$

Proof: Due to the properties of monotonicity already proved on P_{lex}, it suffices to show the validity of the previous remark (4). For any $s \in S_k^m$, it holds

$$P_{lex}(s) + \sum_{h=1}^{m-1} \binom{h-1+\sum_{l=0}^{h-1} s_l}{h} =$$

$$= \sum_{h=1}^{m-1} \left[\binom{h+\sum_{l=1}^{h+1} s_{m-l}}{h} - \binom{h+\sum_{l=1}^{h} s_{m-l}}{h} + \binom{h-1+\sum_{l=1}^{h} s_{m-l}}{h} \right]$$

$$= \sum_{h=1}^{m-1} \left[\binom{h+\sum_{l=1}^{h+1} s_{m-l}}{h} - \binom{h-1+\sum_{l=1}^{h} s_{m-l}}{h-1} \right]$$

$$= \sum_{h=1}^{m-1} \binom{h+\sum_{l=1}^{h+1} s_{m-l}}{h} - \sum_{h=0}^{m-2} \binom{h+\sum_{l=1}^{h+1} s_{m-l}}{h}$$

$$= \binom{m-1+\sum_{l=1}^{m} s_{m-l}}{m-1} - \binom{0+\sum_{l=1}^{1} s_{m-l}}{0} = \binom{k+m-1}{m-1} - 1$$

and so, it remains proved the formula stated in (5).

Example 3. Suppose, as in Examples 1, 2, that $m = 5$, $k = 8$ and $x = 30224402$. Then s(x) = (2, 0, 3, 1, 2) and

$$P_{invlex}(s) = \binom{2}{1} + \binom{4}{2} + \binom{8}{3} + \binom{9}{4} = 2 + 6 + 56 + 126 = 190 \tag{6}$$

Further, from Example 1 and (6), $P_{lex}(s) + P_{invlex}(s) = 494 = \binom{8+5-1}{5-1} - 1 = |S_k^m| - 1$ as stated in (4).

We can design a first algorithm for ranking sequences under inverse lex order, simply applying the remark (4). In other words, through the algorithms described in the previous section for lex order, we can subtract the value r, obtained as output by those algorithms, by the quantity | S | - 1. In general, the remark (4) provides an alternative method of computation for ranking, both for lexicographic and inverse lexicographic rank. A similar strategy could also be employed to do unranking, too.

Nevertheless, here we design direct algorithms for ranking and unranking computation.

Algorithm 5 (Ranking Inverse Lex Algorithm)

Input: $\vec{s}[0:m-1] \in Z_m^k$

Output: $r \in \{0,1,\ldots,\binom{k+m-1}{m-1}-1\}$

Execute the following steps.

```
i ← 1
while s[m- i] = 0 do {
      i ← i + 1
}
c1 ← 0
c2 ← (i − 1 + s[m − i] choose i)
 r ← c2
for h ← i + 1 to m - 1 do {
      c1 ← c1 + s[m - h + 1]
      c2 ← c2  (c1 + s[m - h] + h - 1)
      for j ← 0 to s[m - h] - 1 do {
                  c2 ← c2 (c1 + j + h − 1) / (c1 + j)
                  }
      c2 ← c2 / h
      r ← r + c2
}
return (r)
```

The ranking algorithm just presented requires to store $O(m\log_m(m+k))$ m-ary digits and to perform $O(mk\ \log^2_m\ (m + k))$ m-ary digit operations as in the lexicographic order case.

Similarly, the following unranking algorithm requires $O(m\log_m\ (m + k))$ m-ary digits to store and $O(mk\ \log^2_m(m+ k))$ m-ary digit operations to perform. Also if the theoretical computational cost in terms of digit operations is the same among lex and inverse lex order, in the second case the process is twice faster than in the first case.

Algorithm 6 (Unranking Inverse Lex Algorithm)

Input: $r \in \{0,1,\ldots,\binom{k+m-1}{m-1}-1\}$

Output: $\vec{s}[0:m-1] \in Z_m^k$

Execute the following steps.

```
c1 ← m - 2 + k
c2 ← m - 1
c3 ← (c1 choose c2)
for h ← 0 to m - 3 do {
      s[h] ← 0
      while c3 > r do {
      s[h] ← s[h] + 1
      c3 ← c3 (c1 - c2)/c1
      c1 ← c1 - 1
      }
      r ← r - c3
      if c1 ≠ 0 then {
      c3 ← c3 c2/c1
      c1 ← c1 - 1
      c2 ← c2 - 1
}
}
s[m - 2] ← c1 - r
s[m - 1] ← r
return ( s⃗ )
```

4 Co-lexicographic Ranking

A useful alternative to the lexicographic order is the co-lexicographic order (or co-lex order), i.e. if $s = (s_0, s_1, \ldots, s_{m-1})$, $t = (t_0, t_1, \ldots, t_{m-1})$ are elements in S_k^m then $s <_{colex} t$ if, and only if, $s \neq t$ and it exists $w \in \{0,1,\ldots,m-1\}$ such that $s_{m-1-i} = t_{m-1-i}$ for $i = 0, 1, \ldots, w-1$ and $s_{m-1-w} < t_{m-1-w}$. For example, given $m = 3$ and $k = 4$, the co-lex order is listed in Table 3.

Remark. For any $s = (s_0, s_1, \ldots, s_{m-1}) \in S_k^m$,consider the reverse sequence $s^R = (s_{m-1}, s_{m-2}, \ldots, s_0) \in S_k^m$, where $(s^R)^R = s$. Then the following property holds among lex and colex order ranking functions

$$P_{colex}(s) = P_{lex}(s^R) \tag{7}$$

Table 3

Co-lexicographic order for $m = 3$ and $k = 4$

s	P_{invlex}(s)
(4,0,0)	0
(3,1,0)	1
(2,2,0)	2
(1,3,0)	3
(0,4,0)	4

s	P_{invlex}(s)
(3,0,1)	5
(2,1,1)	6
(1,2,1)	7
(0,3,1)	8
(2,0,2)	9

s	P_{invlex}(s)
(1,1,2)	10
(0,2,2)	11
(1,0,3)	12
(0,1,3)	13
(0,0,4)	14

In fact

$$P_{colex}(s) = |\{t \in S_k^m \mid t <_{colex} s\}|$$
$$= |\{t \in S_k^m \mid \exists w \in Z_m : t_{m-1-w} < s_{m-1-w} \text{ and } t_{m-1-i} = s_{m-1-i}, i \in Z_w\}|$$
$$= |\{t^R \in S_k^m \mid \exists w \in Z_m : t_w^R < s_w^R \text{and } t_i^R =$$
$$= s_i^R, i \in Z_w\}| = |\{t^R \in S_k^m \mid t^R <_{lex} s^R\}| = P_{lex}(s^R)$$

Theorem 2. *The function P_{colex} can be expressed through the formula*

$$P_{colex}(s_0,\ldots,s_{m-1}) = \sum_{h=1}^{m-1}\left[\binom{h+\sum_{l=0}^{h} s_l}{h} - \binom{h+\sum_{l=0}^{h-1} s_l}{h}\right] \tag{8}$$

Proof. The proof is trivial using (1) in connection with (7).

Example 4. Suppose $m = 4$ and $k = 9$. Let it given the word $x = 302121011$. Then $s(x) = s = (2, 4, 2, 1)$, $s^R = (1, 2, 4, 2)$ and

$$P_{colex}(s) = P_{lex}(s^R) = \binom{7}{1} - \binom{3}{1} + \binom{10}{2} - \binom{8}{2} + \binom{12}{3} - \binom{11}{3} = 76$$

In fact $s = (2, 4, 2, 1)$ is preceded by all the sequences $u \in A$ where

$$A = \{u \in S_m^k \mid u = (u_0, u_1, u_2, 0) \text{ or } u = (u_0, u_1, 0, 1) \text{ or } u = (u_0, u_1, 1, 1)\ u =$$
$$= (u_0, 0, 2, 1) \text{ or } u = (u_0, 1, 2, 1)$$
$$\text{or } u = (u_0, 2, 2, 1) \text{ or } u = (u_0, 3, 2, 1)\}$$

and

$$|A| = \binom{11}{2} + \binom{9}{1} + \binom{8}{1} + \binom{6}{0} + \binom{5}{0} + \binom{4}{0} + \binom{3}{0} = 76\ .$$

We are now able to design a direct algorithm to compute the rank in co-lex order of the sequences modifying the Second Ranking Lex Algorithm, i.e. exchanging the quantity $s[m - h - 1]$ with the quantity $s[h]$, in each statement where it appears, because of relations (7) and (8).

Algorithm 7 (Ranking Colex Algorithm)

Input: $\vec{s}[0:m-1] \in Z_m^k$

Output: $r \in \{0,1,\ldots,\binom{k+m-1}{m-1}-1\}$

Execute the following steps.

```
c2 ← 1
r ← 0
c4 ← s[0]
for h ← 1 to m - 1 do {
      c3 ← c4
      a ← c2 (h + c3)/h
      if s[h] > 0 then {
            c4 ← c4 + s[h]
            c1 ← a
            for b ← 1 to s[h] do {
                  c2 ← c2 (h + b + c3 - 1)/(b + c3)
            }
            c2 ← c2 (h + c4)/h
            r ← r + c2 - c1
      }
      else {
            c2 ← a
      }
}
return (r)
```

The ranking co-lexicographic algorithm just presented can be accomplished by storing $O(m\log_m(m+k))$ m- ary digits and performing $O(mk \log_m (m + k))$ m- ary digit operations.

Analogously, a direct algorithm to unrank the word weight classes can be obtained by exchanging in the Unranking Lex Algorithm the quantity s[h - 1] with the quantity s[m - h] and the quantity s[m - 1] with the quantity s[0] whereas they are used. Clearly, with these adjustments, the space and time complexities of co-lexicographic algorithms remain the same as in the lexicographic case; then, unranking weight classes under co-lex ordering requires, as well, $O(m\log_m (m + k))$ m- ary digits to be stored and $O(mk \log_m (m + k))$ m- ary digit operations to be performed.

Algorithm 8 (Unranking Colex Algorithm)

Input: $r \in \{0,1,\ldots,\binom{k+m-1}{m-1}-1\}$

Output: $\vec{s}[0:m-1] \in Z_m^k$

Execute the following steps.

```
a ← k
c1 ← (m − 1 + a choose a)
c2 ← 2  c1
for h ← 1 to m - 1 do {
      s[m - h] ← 0
      if a > 0 {
            c2 ← c2 - c1
            c1 ← c2 a / (m − h + a)
            while r - c2 + c1 > 0 do {
                  r ← r - c2 + c1
                  s[m - h] ← s[m - h] + 1
                  a ← a - 1
                  c2 ← c1
                  c1 ← c2 a / (m − h + a)
                  }
            }
}
s[0] ← a
return ( s⃗ )
```

We could also obtain algorithms for co-lex ranking and unranking functions introducing in the lex algorithms previously described the substitutions $s[0] \rightarrow s[m - 1]$, $s[1] \rightarrow s[m - 2]$,..., $s[m - 1] \rightarrow s[0]$ as first step (respectively as last step) of the ranking lex algorithm (respectively unranking lex algorithm).

5 Concluding Remarks

We have described three efficient methods to rank and unrank word weight classes, taking in consideration different kind of orderings: lexicographic, inverse lexicographic and colexicographic.

In the first approach, investigated for the lexicographic case, it is required to store the binomial coefficients of the form $\binom{a+b}{a}$ with a = 0, 1,..., m - 1, b = 0, 1,...,k, which globally require $O(m^2 k \log_m(m+k))$ m- ary digits storing elements and the whole algorithm requires $O(m^2 \log_m(m+k))$ m- ary digit operations for the ranking process, and $O(mk \log_m(m + k))$ for the unranking one.

In the second approach, investigated for lex, inverse lex and co-lex orderings, the computation of the binomial coefficients is optimized and the whole

algorithms, in each case, require $O(mk \log^2_m(m+k))$ *m*- ary digit operations by storing $S = O(m \log_m(m+k))$ *m*- ary digits, for both ranking and unranking functions.

We invite the reader to note that if we do rank and unrank over the set S_k^m with Cover's method, described in [2], the complexity of the algorithms so obtained is worse than the complexity of the algorithms presented in this paper. In fact, using Cover's plain method, also if we store all the binomial coefficients we need, that means to store $O(m^2 k \log_m(m + k))$ m-ary digits, the algorithm has to perform $O = (mk \log_m(m + k))$ *m*-ary digit operations. Instead, in our algorithms, storing only $O(m \log_m(m + k))$ *m*-ary digits, the time complexity is $O = (mk \log^2_m(m + k))$ *m*-ary digit operations.

References

[1] P. J. CAMERON: *Combinatorics: Topics, Techniques, Algorithms*, Cambridge University Press, 1994.

[2] T. M. COVER: "*Enumerative Source Encoding*", IEEE Transactions on Information Theory, vol. it-**19**, n. 1, pp. 73–77, Jan. 1973.

[3] D. E. KNUTH: *The Art of Computer Programming*, Vol. **1**: Fundamental Algorithms, third edition, Addison Wesley, Boston, US, 1997.

[4] D. L. KREHER, D. R. STINSON: *Combinatorial Algorithms: generation, enumeration and search*, CRC Press, Boca Raton, US, 1999.

[5] R. MASCELLA, D. PELUSI, L.G. TALLINI: "*On computing the indexing function over the set of combinatorial compositions*", ECIT 2004 Conference (CD Proceedings), Romania, 2004.

[6] D. STANTON, D. WHITE: *Constructive Combinatorics*, Springer-Verlag, Berlin, 1986.