Combinatorial Coding and Lexicographic Ordering

Peter Kabal

Department of Electrical & Computer Engineering
McGill University
Montreal, Canada

Version 1, February 2018

You are free:

to Share – to copy, distribute and transmit this work

to Remix – to adapt this work

Subject to conditions outlined in the license.

Revision History:

- 2018-02 v1: Initial release

1 Introduction

This document examines methods to generate a combinatorial index for the selection of items, and the decoding of the index to produce the corresponding selection of items. Marching through the indices produces lexicographically ordered selections. Three cases are considered: selections with no repeated items, selections with repetitions, and selections with prescribed repetition multiplicities.

2 Combinations – No Repetitions

Consider the selection of K items from a list of N items. The order of the items in the selection does not matter, and the selection does not include repeated items. The number of unique ways to select the K items is

$$\begin{aligned} N_c &= \binom{N}{K} \\ &= \frac{N!}{(N-K)!\,K!}. \end{aligned} \tag{1}$$

The N items are indexed from 0 to $N-1$. The K selected indices of the items can be represented as a vector of N binary values (the selected K indices are non-zero):

$$\boldsymbol{a} = [a_0, a_1, \ldots, a_{N-1}], \qquad a_n \in \{0, 1\}. \tag{2}$$

The constraint on having K non-zero binary digits is

$$\sum_{n=0}^{N-1} a_n = K. \tag{3}$$

The vector $\boldsymbol{a}$ interpreted as binary number (least significant bit first) lexicographically orders the selections. However, many of the 2^N possible values of $\boldsymbol{a}$ do not satisfy the constraint of having exactly K non-zero values.

An alternate representation uses a vector of K indices,

$$\boldsymbol{I}_a = [I_a(1), ..., I_a(K)], \qquad \text{where } I_k \in \{0, 1, \ldots, N-1\}. \tag{4}$$

The vector of indices is in (sorted) ascending order. If $\boldsymbol{I}_a$ is considered to be a N-ary number (least significant digit first), it can also be used to lexicographically order the selections. However, many of N^K possible value of $\boldsymbol{I}_a$ are either permutations of allowed (sorted) values or have repeated indices.

As an example, consider choosing 3 items from a collection of 20 items. The number of selections (order does not matter, no repetitions allowed) is 1140. This means that for the vector $\boldsymbol{a}$, only 1140 of the 1 048 576 possible values represent the selection of exactly 3 values. For the vector $\boldsymbol{I}_a$, only 1140 values out of 27 000 possible values represent valid sorted-order selections.

The goal is to create a combinatorial index, taking on values from 0 to $\binom{N}{K} - 1$, from a valid $\boldsymbol{a}$ or $\boldsymbol{I}_a$ vector. Conversely, a valid $\boldsymbol{a}$ or $\boldsymbol{I}_a$ vector will be created from the combinatorial index. Each combinatorial index should correspond to a unique selection of K items – ordered combinatorial indices will correspond to selections in lexicographic order.

An algorithm to index the combinatorial cases was presented by Schalkwijk [1].[1] Adapted to the convention of increasing lexicographic order, the index is calculated as

$$i_c(\boldsymbol{a}) = \sum_{n=0}^{N-1} a_n \binom{n}{k_n}, \tag{5}$$

where k_n is an item count vector,

$$k_n = \sum_{i=0}^{n} a_i. \tag{6}$$

The value k_n is the number of selected items with index less than or equal to n – this count can be updated on the fly, $k_{n+1} = k_n + a_n$. It should be noted that the expression for $\binom{n}{k}$ is defined to be zero for $n < k$.

Using $\boldsymbol{I}_a$, Eq. (5) can also be written as

$$i_c(\boldsymbol{I}_a) = \sum_{k=1}^{K} \binom{I_a(k)}{k}. \tag{7}$$

Note that this expression assumes that $I_a(k)$ has been sorted (increasing values). Calculating the index involves the summation of K combinatorial values.

2.1 Coding trellis – no repetitions

The process of generating the index can be viewed as traversing a trellis. The trellis for $K = 3$ is shown in Fig. 1. Each diagonally upward transition from node (n, k) has an associated combinatorial term $\binom{n}{k}$. Start at the lower left corner with the item index n set to zero, the number

[1] The present author described the combinatorial coding scheme for use in a multipulse speech coder [2]. A combinatorial index was used to code the position of a fixed number of pulses chosen from a set of possible positions. After the presentation of that conference paper, J.-P. Adoul brought the Schalkwijk paper [1] to the this author's attention. Later, combinatorial coding was used in a standardized speech coder (ITU-T G.723.1 [3]). See [4], for a description of the details of the combinatorial coding and decoding used in G.723.1.

of selected items k set to zero, and the combinatorial index set to zero. At each value of n, move horizontally to the right if that item is not selected. If an item is selected, move up diagonally and add the corresponding combinatorial term on the transition to the combinatorial index. The process is terminated after reaching K items.

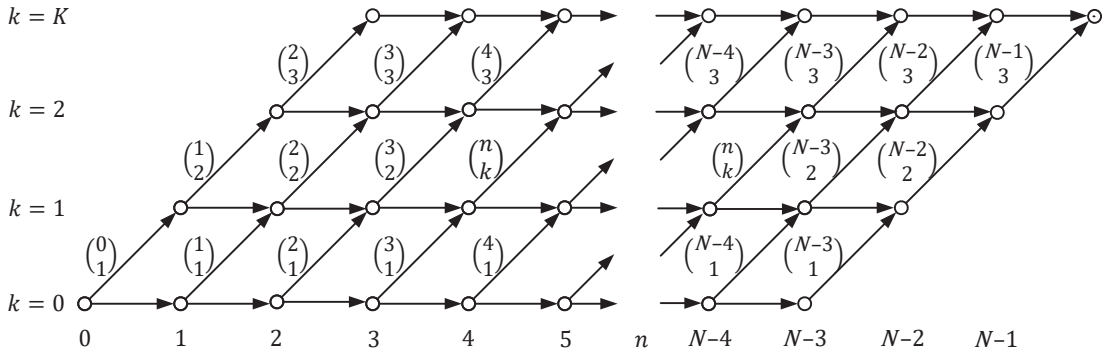

Fig. 1 Trellis diagram for combinatorial coding for ($K = 3$)

2.2 Combinatorial index / lexicographic order

Consider the partial sum of the combinatorial terms for a sequence of diagonal moves starting from the bottom of the trellis at node $(n, 0)$,

$$S_c(L, n) = \sum_{k=1}^{L} \binom{n+k-1}{k}, \qquad 1 \le L \le K. \tag{8}$$

The goal is to find a closed form for this partial sum. A proof by induction will be used:

- Propose a formula for the partial sum of L terms.
- Show that this formula applies for $L = 1$.
- Induction hypothesis: Show that the formula applies when the partial sum is extended from L terms to $L + 1$ terms.

The partial sum formula is

$$S_c(L, n) = \binom{n+L}{L} - 1. \tag{9}$$

A direct direct evaluation for $L = 1$ shows that the formula applies. Now assume the partial sum

formula applies for L terms. For $L+1$ terms,

$$\begin{aligned} S_c(L+1,n) &= \binom{n+L}{L+1} + S_L \\ &= \binom{n+L}{L+1} + \binom{n+L}{L} - 1. \end{aligned} \tag{10}$$

Using the combinatorial identity,

$$\binom{p}{q} = \binom{p-1}{q} + \binom{p-1}{q-1}, \tag{11}$$

then

$$S_c(L+1,n) = \binom{n+L+1}{L+1} - 1. \tag{12}$$

This shows that the partial sum formula applies for all L.

From the trellis diagram, the smallest combinatorial index is generated by moving up the leftmost diagonals. Application of the sum along the diagonals shows that the leftmost diagonal $S_c(K,0)$ is zero.) This is the combinatorial index for choosing the first K items. The largest sum is generated by the rightmost diagonals. This sum is $S_c(K,N-K)$ is $\binom{N}{K}-1$. This is the combinatorial index for choosing the last K items.

The goal is to show that the combinatorial index imposes a lexicographic ordering of the selected items. Consider an induction argument for a vector $\boldsymbol{a}$ of length n and weight k. All such $\boldsymbol{a}$ vectors result in a path through a sub-trellis ending at node (n,k). Assume that the combinatorial indices for all such $\boldsymbol{a}$ correspond to a lexicographic order of the vectors. There are $\binom{n}{k}$ vectors of length n and weight k. Combinatorial indices take on all values from zero (path along the leftmost diagonals in the sub-lattice) to $\binom{n}{k}-1$ (path along the rightmost diagonals in the sub-lattice).

Now increase the length of $\boldsymbol{a}$ by one. If the added element is zero, there is no change in the lexicographic value of the vector $\boldsymbol{a}$, k remains unchanged, and there is no change in the combinatorial index. If the added element is a one, the new $\boldsymbol{a}$ is lexicographically larger than the original vector, and k is increased by one. The corresponding combinatorial index is increased by $\binom{n}{k}$. This value is larger (by one) than the largest possible index produced by the original vector $\boldsymbol{a}$. The combinatorial index is in the lexicographic order of the new $\boldsymbol{a}$ and the largest possible index at the new node is $\binom{n+1}{k+1}-1$.

It is straightforward to verify that the combinatorial indices for small n and k are in proper order. This then completes a proof of the correct one-to-one ordering of the combinatorial index and the lexicographic order of vectors $\boldsymbol{a}$ for a fixed N and K.

2.3 Decoding the combinatorial index – no repetitions

The decoding of a combinatorial index to a $\boldsymbol{a}$ vector is accomplished by following the coding trellis from its terminal point back to the beginning point. In Fig. 2, the trellis has been redrawn by flipping the coding trellis and reversing the transitions. Start with the combinatorial index at the upper left corner. Compare that combinatorial index with the combinatorial term on the diagonally downward transition. If the index is larger than the combinatorial index, the corresponding item has been selected. Subtract the combinatorial term from the index and follow the downward diagonal transition. Otherwise, continue to the right to the next node.

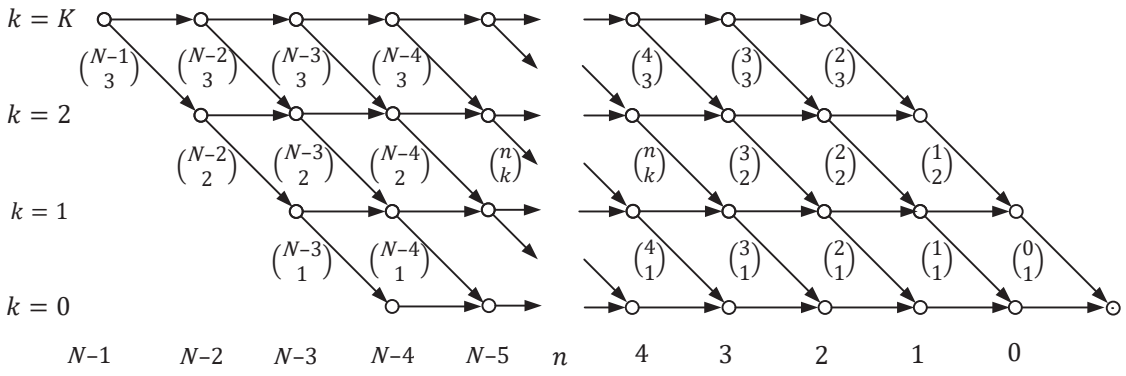

Fig. 2 Trellis diagram for combinatorial decoding for ($K = 3$)

There are $K \times (N - K + 1)$ combinatorial terms in the trellis diagram for coding and decoding. These can be precomputed and stored in a table. An alternative is to update the combinatorial terms as we move along using Eq. (11).

3 Combinations – with Repetitions

The number of combinations of selections of K items from a set of N items allowing repeated items in the selection is denoted as

$$\begin{aligned} N_m &= \left(\!\!\binom{N}{K}\!\!\right) \\ &= \binom{N+K-1}{K}. \end{aligned} \tag{13}$$

The multiset function $\left(\!\!\binom{n}{k}\!\!\right)$ behaves differently from the binomial coefficient $\binom{n}{k}$. For a fixed n, the multiset function is an increasing function of k, whereas the binomial coefficient has its largest value for k near $N/2$. The selection function $\boldsymbol{a}$ is no longer binary – each element can take on

values from 0 to K,

$$\boldsymbol{a} = [a_0, a_1, \ldots, a_{N-1}], \qquad a_n \in \{0, \ldots, K\}, \tag{14}$$

with the constraint,

$$\sum_{n=0}^{N-1} a_n = K. \tag{15}$$

The combinatorial index for combinations with repetitions case is,

$$i_m(\boldsymbol{a}) = \sum_{n=0}^{N-1} \sum_{k=1}^{a_n} \left(\!\!\binom{n}{k_{n-1}+k}\!\!\right), \tag{16}$$

where the item count vector is

$$k_n = \sum_{i=0}^{n} a_i. \tag{17}$$

As before, the combinatorial index can be calculated from the (sorted) vector of K indices $\boldsymbol{I}_a$ for the selected items. With repetitions, the same index can appear more than once in $\boldsymbol{I}_a$. The combinatorial index is computed as

$$i_m(\boldsymbol{I}_a) = \sum_{k=1}^{K} \left(\!\!\binom{I_a(k)}{k}\!\!\right). \tag{18}$$

3.1 Coding trellis – with repetitions

The process of generating the combinatorial index can be viewed as traversing the trellis structure shown in Fig. 3. Start from the lower left corner. If items are present for a given n, move up vertically adding the combinatorial terms on the transitions until all items with index n have been considered. Next, move horizontally to the next value of n.

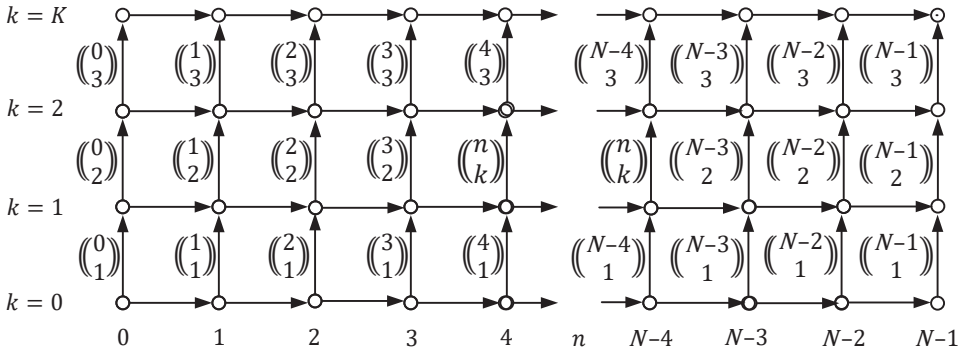

Fig. 3 Trellis diagram for combinatorial coding (with repetitions) for ($K = 3$)

Consider the partial sum of the combinatorial terms for a sequence of upward moves starting from the bottom at node $(n, 0)$,

$$S_m(L, n) = \sum_{k=1}^{L} \left(\!\!\binom{n}{k}\!\!\right). \tag{19}$$

Using an induction argument, the partial sum formula is

$$S_m(L, n) = \left(\!\!\binom{n+1}{L}\!\!\right) - 1. \tag{20}$$

A direct evaluation for $L = 1$ shows that the formula applies. Now assume the partial sum formula applies for L terms. For $L + 1$ terms,

$$\begin{aligned} S_m(L+1, n) &= \left(\!\!\binom{n}{L+1}\!\!\right) + S_L(, n) \\ &= \left(\!\!\binom{n}{L+1}\!\!\right) + \left(\!\!\binom{n+1}{L}\!\!\right) - 1. \end{aligned} \tag{21}$$

Using the multiset identity,

$$\left(\!\!\binom{p}{q}\!\!\right) = \left(\!\!\binom{p}{q-1}\!\!\right) + \left(\!\!\binom{p-1}{q}\!\!\right), \tag{22}$$

then

$$S_m(L+1, n) = \left(\!\!\binom{n+1}{L+1}\!\!\right) - 1. \tag{23}$$

This shows that the partial sum formula applies for all L.

The smallest combinatorial index occurs if all K items are at $n = 0$. That index is zero. The largest combinatorial index occurs if all K items are at $n = N - 1$. That index is $\left(\!\!\binom{N}{K}\!\!\right) - 1$.

Consider the same example used earlier. For no repetitions, the number of choices of 3 items selected from 20 possibilities was 1140. If repetitions are allowed the number goes to 1540, of which 400 have at least one repeated selected index.

An induction argument as used for the combinations with no repetitions case, shows that the combinatorial index for the combinations with repetitions case also imposes a lexicographic ordering to the selected items.

3.2 Decoding the combinatorial index – with repetitions

In Fig. 4, the coding trellis has been redrawn by flipping the coding trellis and reversing transitions. Start with the combinatorial index at the top left. Compare that index with the combinatorial term on the downward transition. If the combinational index is larger than the term on the trellis, subtract that term from the index, and add an item. Continue downward as long as the index is larger than the corresponding combinatorial term on the downward transition. Then proceed one

step to the right.

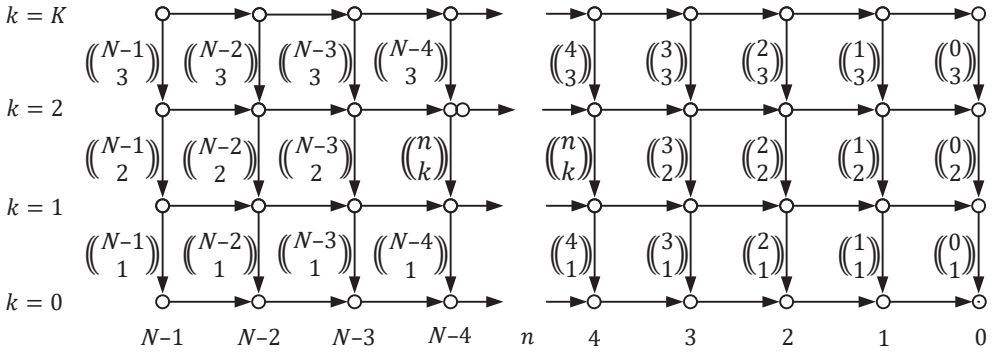

Fig. 4 Trellis diagram for combinatorial decoding (with repetitions) for ($K = 3$)

4 Categories

The selection of combinations with repetition subsumes the selection of combinations with no repetitions. Each K item selection can be categorized. Let the categories be indicated as a vector of K values that shows the multiplicities of a particular selection. The sum of the components of the category vector is K. For instance the category vector $[1\ 1\ 1\ 1\ 1]$ indicates that all K values are of multiplicity 1, i.e. no repetitions. The category vector $[\ \ 1\ 1\ 2]$ has $K-2$ items with no repetitions (multiplicity 1), and 1 item with multiplicity 2. The category vectors are shown as being of length K. The blank elements in the category vector can be considered to be zero.

The generation of the category vectors for a given K is a problem in creating integer partitions. The subject of integer partitions has a voluminous literature. Even determining the number of such category vectors as a function of K, without actually generating the partitions, involves a deep dive into theory.[2] The category vectors can be programmatically generated in lexicographic order starting from the all-ones vector (see for instance [5]).

Once the category vectors are known, the number of selections in each category can be determined using combinatorial arguments. The number of non-zero elements in the category vector gives the number of unique elements K_u. Then the number of K-selections with that category vector is given by

$$N_r = \frac{N!}{(N-K_u)!\,n_1!\,n_2!\cdots n_K!}, \tag{24}$$

[2]The number of partitions is given by the partition function. The number of partitions has been tabulated for moderate values of K.

where n_k is the number of elements in the category vector with multiplicity k. The ratio of the terms involving N gives the number of permutations of the K_u unique items taken from the list of N items. Each $n_k!$ term in the denominator cancels out the fraction of the permutations due to the n_k items of multiplicity k. Note that if $n_k = 0$, then by definition $n_k! = 1$. If all items in the selection are unique (no repetitions, $K_u = K$, $n_1 = N - K$, $n_2, \ldots, n_K$ are zero), the formula gives $\binom{N}{K}$ as expected.

Setting $n_0 = N - K_u$, the number of K item selections in Eq. (24) can be expressed in terms of the multichoose function (multinomial coefficient),

$$\binom{N}{n_0, n_1, \ldots, n_K} = \frac{N!}{n_0!\, n_1! \cdots n_K!}, \tag{25}$$

where $\sum_k n_k = N$.

The category vectors for $K = 5$ are listed in Table 1. The category vectors depend only on K. The number of selections in each category depends on both N and K. The category vectors were generated using Algorithm P in Section 7.2.1.4 of [5]. The number of selections in each category is calculated using Eq. (24) for $N = 20$ and $K = 5$. The first category (no repetitions) has $\binom{N}{K}$ selections (here 15 504). The sum of the number of values in all categories is $\left(\!\binom{N}{K}\!\right)$ (here 42 504).

Table 1 Categories for selecting $K = 5$ items from $N = 20$ items

Category	Number
[1 1 1 1 1]	× 15 504
[1 1 1 2]	× 19 380
[1 2 2]	× 3 420
[1 1 3]	× 3 420
[2 3]	× 380
[1 4]	× 380
[5]	× 20

4.1 Coding category selections

4.1.1 Multilevel symbols

Schalkwijk [1] proposes a scheme to generate a combinatorial index for multi-level index vectors $\boldsymbol{a}$. The index values can take on Q values 0 to $Q - 1$. Symbol count vectors m_n^d are defined as

$$m_n^d = \sum_{i=0}^{n} \delta(a_i, d), \qquad d \in \{0, 1, \ldots, Q-1\}. \tag{26}$$

where $\delta(p,q)$ is the Kronecker delta function,

$$\delta(p,q) = \begin{cases} 0, & p \neq q, \\ 1, & p = q. \end{cases} \tag{27}$$

Since for each value of n, only one of the m_n^d values is set to one,

$$\sum_{d=0}^{Q-1} m_n^d = n+1; \tag{28}$$

The vector $\boldsymbol{a}$ contains n_d instances of symbol d,

$$n_d = m_{N-1}^d. \tag{29}$$

The vector of the n_d values establishes a symbol count vector for vector $\boldsymbol{a}$,

$$C_a = [n_0, n_1, \ldots, n_{Q-1}], \qquad \text{where} \sum_i n_i = N. \tag{30}$$

The combinatorial index for a Q-ary $\boldsymbol{a}$ vector is given in [1] for a given number of each of the multi-valued symbols,

$$i_s(\boldsymbol{a}) = \sum_{n=0}^{N-1} \sum_{i=0}^{a_n-1} \frac{n!}{(m_n^i - 1)! \prod\limits_{\substack{j=0 \\ j\neq k}}^{Q-1} m_n^j!} \tag{31}$$

In this equation, the term $(m_n^i - 1)!$ in the denominator can evaluate to $(-1)!$. To handle that case, the factorial of -1 is defined to be ∞.

Consider the multichoose identity,

$$\binom{n+1}{n_0, n_1, \ldots, n_K} = \sum_{i=0}^{K} \binom{n}{n_0, \ldots, (n_i - 1), \ldots, n_K}, \tag{32}$$

where $n_0 + \cdots + n_K = n+1$. A partial sum version of this identity is

$$\sum_{i=0}^{L-1} \binom{n}{n_0, \ldots, (n_i - 1), \ldots, n_K} = \binom{n+1}{n_0, n_1, \ldots, n_K} \frac{1}{n+1} \sum_{i=0}^{L-1} n_i. \tag{33}$$

Using the partial sum identity, the combinatorial index can be written as

$$i_s(\boldsymbol{a}) = \sum_{n=0}^{N-1} \left[\binom{n+1}{m_n^0, \ldots, m_n^{Q-1}} \frac{1}{n+1} \sum_{i=0}^{a_n-1} m_n^i \right]. \tag{34}$$

Since only one of the elements of the symbol count vector (element $m_n^{a_n}$) is updated for each n, Eq. (34) can also be written as

$$i_s(\boldsymbol{a}) = \sum_{n=0}^{N-1} \left[\binom{n}{m_{n-1}^0, \ldots, m_{n-1}^{Q-1}} \frac{1}{m_n^{a_n}} \sum_{i=0}^{a_n-1} m_n^i \right]. \tag{35}$$

Equation (34) is useful for coding the index (n stepped in an increasing order), while Eq. (35) is useful for generating the vector $\boldsymbol{a}$ from an index (n stepped downward).

As will be shown, the combinatorial index $i_s(\boldsymbol{a})$ takes on values from 0 to $N_s - 1$, where

$$N_s = \binom{N}{n_0, \ldots, n_{Q-1}}. \tag{36}$$

Consider ternary symbols (0, 1, 2) in a vector of length 9 with symbol count vector $[4, 2, 3]$. The symbol count vector specifies the number of each symbol in the vectors. The total number of vectors in this category is $\binom{9}{4,2,3} = 1\,260$. Examples of vectors in this category are shown below.

[2 2 2 1 1 0 0 0 0]. This is the first vector in lexicographic order in the category. Since the vector $\boldsymbol{a}$ has symbols in descending order, each term in the sum in Eq. (35) is zero, giving the combinatorial index 0.

[0 2 2 0 1 0 2 1 0]. This is a vector in the category, giving the combinatorial index 304.

[0 0 0 0 1 1 2 2 2]. This is the last vector in lexicographic order in the category. This vector is the reversal of the first vector in lexicographic order. The combinatorial index for this vector is $N_s - 1 = 1\,259$.

An inductive argument can be used to show that the combinatorial index for a particular symbol count vector is assigned in lexicographic order of the vector $\boldsymbol{a}$. Assume that for a vector of length N, combinatorial indices are assigned in lexicographic order of that vector and that the indices take on values from 0 to $N_s - 1$, where N_s is the multichoose value for a vector of length N and the symbol count vector $[m_{N-1}^0, \ldots, m_{N-1}^{Q-1}]$.

Increase N by one. If element a_N is a zero, the combinatorial index does not change, and the lexicographic order of the new $\boldsymbol{a}$ is the same as the old $\boldsymbol{a}$. There is another possibility for no increase in the combinatorial index: all symbol counts for symbols below the current symbol a_N are zero. That means the vector is the first vector in lexicographic order. As in the example given

earlier, in this case symbols up to a_N are in descending order. For all other cases, the combinatorial index increases, and that increase is larger for increasing a_N. This means that if the a_{N-1} increases the lexicographic order, the combinatorial index also increases.

Let the number of combinations for an N length vector be $N_s(N)$ and let the combinatorial index for an N length vector be $i_s(\boldsymbol{a}, N)$. From Eq. (34).

$$i_s(\boldsymbol{a}, N+1) = i_s(\boldsymbol{a}, N) + N_s(N+1)\frac{1}{N+1}\sum_{i=0}^{a_N-1} m_N^i, \tag{37}$$

where

$$N_s(N+1) = \binom{N+1}{m_N^0, \ldots, m_N^{Q-1}}. \tag{38}$$

The value of $N_s(N)$ in Eq. (36) can be expressed in terms of $N_s(N+1)$ using the fact that only $m_N^{a_N}$ is updated for $n = N$,

$$\begin{aligned} N_s(N) &= \binom{N}{m_{N-1}^0, \ldots, m_{N-1}^{Q-1}} \\ &= \binom{N+1}{m_N^0, \ldots, m_N^{Q-1}} \frac{m_N^{a_N}}{N+1}. \end{aligned} \tag{39}$$

Then the upper bound on $i_s(N+1)$ is

$$\begin{aligned} 0 \le i_s(\boldsymbol{a}, N+1) &\le \left[N_s(N+1)\frac{m_N^{a_N}}{N+1} - 1 \right] + \left[N_s(N+1)\frac{1}{N+1}\sum_{i=0}^{a_N-1} m_N^i \right] \\ &= N_s(N+1)\left[\frac{1}{N+1}\sum_{i=0}^{a_N} m_N^i \right] - 1 \\ &\le N_s(N+1) - 1. \end{aligned} \tag{40}$$

The upper limit is reached if $a_N = Q-1$. To complete the inductive argument, it can be shown that the upper limit applies for $N = 1$.

The coding process generates a combinatorial index *and* a symbol count vector. To decode the index, both the combinatorial index and the symbol count vector are needed. The decoding process proceeds by reversing the coding procedure and peeling off terms from the combinatorial index.

4.1.2 Repeated items

The exposition in [1] is in terms of coding an $\boldsymbol{a}$ vector consisting of signalling values. In keeping with the exposition in terms of selecting items, the multilevel vector $\boldsymbol{a}$ can be reinterpreted as

representing the multiplicity of the selection of item n out of a possible N items. A zero multiplicity means that the item is not selected, and a non-zero multiplicity of d means that the item is selected d times. In keeping with previous discussions, the total number of items selected is K,

$$\sum_{i=1}^{Q-1} n_i = K. \tag{41}$$

If this is the only constraint, the problem is that of combinations with repetitions of Section 3. To use the formalism in this section, the number of repetitions r_d must be specified for each d, Specifying each multiplicities is equivalent to choosing the category vector of Section 4. The coding scheme then becomes one of a lexicographic ordering of vectors $\boldsymbol{a}$ in a specific multiplicity category. The category with no repetitions (vector $\boldsymbol{a}$ has $N - K$ zeros and K ones) leads to the same formulas as were derived in Section 2.

The formulation given in Eq. (35) is calculated from the vector $\boldsymbol{a}$. The calculation can also be based on $\boldsymbol{I}_a$. To this end:

- The selected items are $I_a(k)$, for $1 \leq k \leq K$, with $K = Q - 1$. Repeats are allowed.
- In Eq. (35) the combinatorial index is only updated for $a_n > 0$. Create the vector, $\boldsymbol{I}'_a$, which contains the K_u unique values in $\boldsymbol{I}_a$. The elements $I'_a(k)$ give the values of n for selected items. Record the multiplicities of the items in an auxiliary vector $\boldsymbol{r}$,

$$\sum_{k=1}^{K_u} r_k = K. \tag{42}$$

- The symbol count m_n^d is reinterpreted as a multiplicity count. The multiplicity count m_n^d for $1 \leq d \leq K$ is updated for each selected item. The multiplicity count m_n^0 is updated for unselected items, but is not used until a selected item is encountered. Instead, m_n^0 can be updated for each selected item using Eq. (28),

$$m_n^0 = n + 1 - \sum_{i=1}^{K} m_n^i. \tag{43}$$

- Reindex the multiplicity counts with k as m_k^d with $1 \leq k \leq K_u$. The updates to the multiplicity counts are

$$\begin{aligned} m_k^d &= m_{k-1}^d + \delta(d, r_k), \qquad 1 \leq d \leq K, \\ m_k^0 &= n + 1 - \sum_{i=1}^{K} m_k^i. \end{aligned} \tag{44}$$

Now the calculation of the combinatorial index in Eq. (35) can be rewritten as

$$i_s(\boldsymbol{I}'_a) = \sum_{k=1}^{K_u} \left[\binom{I'_a(k)}{m_{k-1}^0, \ldots, m_{k-1}^K} \frac{1}{m_k^{r_k}} \sum_{i=0}^{r_k-1} m_k^i \right]. \tag{45}$$

One could envisage coding all categories for combinations with repetitions as follows. Identify the category of the vector $\boldsymbol{a}$, as in Section 4. The final code will consist of an offset (sum of the numbers of vectors in preceding categories) plus the combinatorial code for the current category. The total number of codes will be the same as coding combinations with repetitions. Decoding can be done by first identifying the category from a table of offsets for the categories, and then using the difference from the offset as a combinatorial code for the identified category. This procedure produces lexicographically order *within* a category.

5 Summary

Since the coding/decoding algorithms are straightforward, the approaches described here can be used to efficiently generate lexicographically ordered selections. For combinations with repetitions, the calculation of the combinatorial index and its decoding as described here has not, as for as this author is aware, been reported elsewhere.

This report has been consistent in that the selection vector $\boldsymbol{a}$ is defined as having elements going from least significant lexicographically to most significant. The appropriate changes have to be made if other conventions are adopted.

Appendices A through D show the coding/decoding procedures written in the Matlab language. These routines are available on the MathWorks site as a user contribution (File Exchange, keyword: "combinatorial index") [6]. The files that can be downloaded from the MathWorks site also include test files.

Appendix A CombCode / CombDecode - Combinatorial Codes (no repetitions)

The routine CombCode in this appendix implements Eq. (7) in Matlab. The routine sorts the array of selected indices if they are not in ascending order. The routine CombDecode implements the decoding procedure.

```
function CIndex = CombCode(SItems)
%  Cindex = CombCode(SItems)
% Generate a combinatorial lexicographic index from a vector of K items
% from a collection of N items. The items are indexed as 0 to N-1. Each of
% selected items is the index value of the item. The items in the vector
% cannot include repetitions and order does not matter. N is not explitly
% set, but N >= max(SItems)+1.
%
% CIndex - Index for the vector of K items, 0 to C(N,K)-1, where C(N,K) is
%     N choose K
% SItems - Vector of K indices of the selected items. Each element of the
%      vector takes on a value 0 to N-1.

% $Id: CombCode.m,v 1.3 2018/02/12 03:43:37 pkabal Exp $

% Sort the items
if (any(diff(SItems) < 0))
  SItems = InsSort(SItems);
end

% Check SItems
if (SItems(1) < 0) || any(diff(SItems) <= 0)
  error('Invalid item index');
end

% Number of selected items
K = length(SItems);

% nchoosek(n, k) does not work for n < k; it can be set to 0
% In the loop nchoosek(n, k) is evaluated for n = 0 to N-1, k from 1 to K

% Find the combinatorial code
CIndex = 0;
for k = 1:K
  n = SItems(k);
```

```
  if (n >= k)
    CIndex = CIndex + nchoosek(n, k);
  end
end

end

% ------ ------
function X = InsSort(X)
% Sort a vector in ascending order (insertion sort algorithm)
% Y = InsSort(X)
% X - Vector of N values to be sorted
% Y - Output vector of sorted values of X

N = length(X);

for i = 2:N
  Xi = X(i);
  j = i;
  while (j > 1 && X(j-1) > Xi)
    X(j) = X(j-1);
    j = j - 1;
  end
  if (i ~= j)
    X(j) = Xi;
  end
end

end
```

```
function SItems = CombDecode(CIndex, N, K)
%   SItems = CombDecode(CIndex, N, K)
% Decode a combinatorial index to generate a vector of length K. The vector
% elements are in increasing order, with each element taking on a value
% from 0 to N-1.
%
% SItems - Vector of K indices of the selected items. Each element of the
%      vector takes on a value 0 to N-1.
% CIndex - Index for the vector of K items, 0 to C(N,K)-1, where C(N,K) is
%      N choose K
% N - Number of items to choose from (N > 0)
```

```
% K - Number of items in the vector (0 to N)

% $Id: CombDecode.m,v 1.3 2018/02/12 03:43:50 pkabal Exp $

if (CIndex < 0 || CIndex >= nchoosek(N, K))
  error('Invalid combinatorial index value');
end

% nchoosek(n, k) does not work for n < k; it should be set to 0
% In the loop, nchoosek(n, k) is evaluated for n = 0 to N-1, k from 1 to K

SItems = NaN(1, K);
for n = N-1:-1:0
  Cnode = 0;
  if (n >= K)
    Cnode = nchoosek(n, K);
  end
  if (CIndex >= Cnode)
    SItems(K) = n;
    if (K <= 1)
      break;          % Early return
    end
    K = K - 1;
    CIndex = CIndex - Cnode;
  end
end

end
```

Appendix B MultiCombCode / MultiCombDecode - Combinatorial Codes (with repetitions)

The routine MultiCombCode in this appendix implements Eq. (18) in Matlab. The routine sorts the array of selected indices if they are not in ascending order. The routine MultiCombDecode implements the decoding procedure.

```
function CIndex = MultiCombCode(SItems)
%  [CIndex, K] = MultiCombCode(SItems)
% Generate a multi-combinatorial lexicographic index from a vector of K
% items (repetitions allowed). The items are indexed as 0 to N-1. Each of
% selected items is the index value of the item. The items in the vector
% can include repetitions and order does not matter.
%
% CIndex - combinatorial code, 0 to Multiset(max(SItemps)+1, K)
% SItems - List of items, each 0 to N-1

% $Id: MultiCombCode.m,v 1.3 2018/02/12 03:44:39 pkabal Exp $

% Number of combinations with repetition
% MultiSet does not work for n = 0, it should return 0
% In the loop, MultiSet is evaluated for n = 0 to N-1, k from 1 to K
MultiSet = @(n, k) nchoosek(n+k-1, k);      % n > 1

% Sort the selected items
if (any(diff(SItems) < 0))
  SItems = InsSort(SItems);
end

% Check SItems
if (SItems(1) < 0)
  error('Invalid item index');
end

% Number of selected items
K = length(SItems);

% Find the multi-combinatorial code
CIndex = 0;
for k = 1:K
  n = SItems(k);
```

```
  if (n > 0)
    CIndex = CIndex + MultiSet(n, k);
  end
end

end

% ------ ------
function X = InsSort(X)
% Sort a vector in ascending order (insertion sort algorithm)
% [Y] = InsSort(X)
% X - Vector of N values to be sorted
% Y - Output vector of sorted values of X

N = length(X);

for i = 2:N
  Xi = X(i);
  j = i;
  while (j > 1 && X(j-1) > Xi)
    X(j) = X(j-1);
    j = j - 1;
  end
  if (i ~= j)
    X(j) = Xi;
  end
end

end
```

```
function SItems = MultiCombDecode(CIndex, N, K)
%  SItems = MultiCombDecode(CIndex, N, K)
% Decode a multi-combinatorial index to generate a vector of length K.
% The vector elements are in increasing order, with each element taking on
% a value from 0 to N-1. Values can be repeated.
%
% SItems - Vector of K indices of the selected items. Each element of the
%      vector takes on a value 0 to N-1.
% CIndex - Index for the vector of K items, 0 to M(N,K)-1, where M(N,K) is
%      N MultiSet K
% N - Number of items to choose from (N > 0)
```

```
% K - Number of items in the vector (0 to N)

% $Id: MultiCombDecode.m,v 1.4 2018/02/12 03:44:51 pkabal Exp $

% Number of combinations with repetition
% MultiSet does not work for n = 0, it should return 0
% MultiSet is evaluated for n = 0 to N-1, k from 1 to K
MultiSet = @(n, k) nchoosek(n+k-1, k);      % n > 1

if (CIndex < 0 || CIndex >= MultiSet(N, K))
  error('Invalid multi-combinatorial index value');
end

SItems = NaN(1, K);
for n = N-1:-1:0
  while (K >= 1)
    Cnode = 0;
    if (n > 0)
      Cnode = MultiSet(n, K);
    end
    if (CIndex >= Cnode)
      SItems(K) = n;
      K = K - 1;
      CIndex = CIndex - Cnode;
    else
      break;
    end
  end
  if (K <= 0)
    break;          % Early return
  end
end

end
```

Appendix C MultiChooseCode / MultiChooseDecode - Combinatorial Codes (with prescribed repetitions)

The routine MultiChooseCode in this appendix implements Eq. (45) in Matlab. The routine sorts the array of selected indices if they are not in ascending order. The routine MultiChooseDecode implements the decoding procedure.

```
function CIndex = MultiChooseCode(SItems)
%  CIndex = MultiChooseCode(SItems)
% Generate a lexicographically ordered combinatorial code for a vector of
% K items selected from N items for the particular combination of item
% multiplicities. These multiplicities define a category which can be
% found from the selected items using the routine Multiplicity.
%
% CIndex - Combinatorial code taking on values from 0 to Ns-1, where
%    Ns = MultiChoose(N, Category). Increasing values of CIndex correspond
%    to increasing lexicographic order of the selected item vector.
% SItems - Input vector of K items (values 0 to N-1)

% $Id: MultiChooseCode.m,v 1.4 2018/02/19 13:35:31 pkabal Exp $

% Sort the selected items
if (any(diff(SItems < 0)))
  SItems = InsSort(SItems);
end

% Check input vector
if (SItems(1) < 0)
  error('Invalid values in input vector');
end

K = length(SItems);
RepCount = zeros(1, K+1);
SNZ = 0;        % sum(RepCount((1:K)+1))

% Loop over all items
CIndex = 0;
k = 1;
while (k <= K)

  n = SItems(k);
```

```
  % Find the multiplicity of item n
  isym = 1;
  for j = k+1:K
    if (SItems(j) ~= n)
      break;
    end
    isym = isym + 1;
  end
  k = k + isym;

  % Update Repcount for n-1 (just before the current item) to account for
  % the unselected items
  RepCount(0+1) = n - SNZ;     % sum(repCount) = n

  % Sum multiplicity values up to (but not including) the current one
  s = sum(RepCount((0:isym-1)+1));
  if (s > 0)
    CIndex = CIndex + MultiChoose(n, RepCount) * s / ...
                                              (RepCount(isym+1) + 1);
  end

  % Update the repetition count for the current n
  RepCount(isym+1) = RepCount(isym+1) + 1;
  SNZ = SNZ + 1;          % Number of unique items processed

end

end

% ------ ------
function Nm = MultiChoose(N, r)

if (sum(r) ~= N)
  error('Invalid parameters to MultiChoose');
end

% Simplest form (subject to overflow in the numerator)
% N = factorial(n) / prod(factorial(r));

% Cancel out the largest factor in r using nchoosek
```

```
[rmax, I] = max(r);
r(I(1)) = 1;

Nm = nchoosek(N, rmax) * factorial(N - rmax) / prod(factorial(r));

end

% ------ ------
function X = InsSort(X)
% Sort a vector in ascending order (insertion sort algorithm)
% [Y] = InsSort(X)
% X - Vector of N values to be sorted
% Y - Output vector of sorted values of X

N = length(X);

for i = 2:N
  Xi = X(i);
  j = i;
  while (j > 1 && X(j-1) > Xi)
    X(j) = X(j-1);
    j = j - 1;
  end
  if (i ~= j)
    X(j) = Xi;
  end
end

end
```

```
function SItems = MultiChooseDecode(CIndex, N, Category)
%  SItems = MultiLevelDecode(CIndex, N, Category)
% Decode a combinatorial index for a vector of selected items
%
% SItems - Output vector of K items (values 0 to Q-1)
% CIndex - Combinatorial code taking on values from 0 to Ns-1, where
%      Ns = MultiChoose(N, Category). Increasing values of CIndex
%      correspond to increasing lexicographic order of the vector SItems.
% N - Number of items to select from
% Category - K element vector representing the multiplicities of the
%      items in SItems
```

```
% $Id: MultiChooseDecode.m,v 1.5 2018/02/19 13:35:18 pkabal Exp $

% Generate the level count vector
RepCount = Cat2LevCount(N, Category);
K = length(RepCount) - 1;

Ns = MultiChoose(N, RepCount);
if (CIndex < 0 || CIndex >= Ns)
  error('Index out of range');
end

SItems = NaN(1, K);
k = K;
for n = N-1:-1:0

  % The combinatorial index for [n, RepCount] is between 0 and Ns-1, where
  %   Ns = MultiChoose(n+1, RepCount).
  % Calculate Cn, the term due to possible a[n] values - when Cn exceeds
  % CIndex, we have gone too far - back off to previous value

  S = 0;
  CnP = 0;
  for isym = 0:K
    S = S + RepCount(isym+1);
    Cn = Ns * S / (n+1);
    if (Cn > CIndex)

      % Update Ns
      Ns = Ns * RepCount(isym+1) / (n+1);

      % Update the repetition count / index
      RepCount(isym+1) = RepCount(isym+1) - 1;
      CIndex = CIndex - CnP;

      for i = 1:isym
        SItems(k) = n;
        k = k - 1;
      end
      break;
    else
```

```
      CnP = Cn;
    end
  end

  if (k < 1)    % Early exit when all items have been found
    break;
  end

end

end

% ------ ------
function LevCount = Cat2LevCount(N, Category)
% Convert a category vector to a level count vector.
% Category - K element vector with the multiplicity of the items.
%     [0 0 ... 1 1 ... 2 2 2 ... ]
%     The category vector describes the multiplicity of K selected items.
%     The appearance of a value 0 < i <= K, indicates that an item is
%     included in the selected items with multiplicity i. If the value i
%     is repeated, more than one item has multiplicity i. The sum of the
%     values in Category is K. The length of Category is also K. Zeros in
%     Category are used to pad the vector (when items are repeated) to the
%     length K.
% LevCount - Q vector with a level count. Now the multiplicity of an item
%     is interpreted as a level. Levels are numbered from 0 to Q-1.
%     LevCount(i) counts the number of instances of each level. The sum of
%     the elements in LevCount is N.
%
% Category (size K) only counts selected items, while LevCount (size Q)
% counts 0 levels as well. These zero levels can be interpreted to be
% non-selected items. Then
%  Q = K+1
% Consider the K non-zero values in Category. These represent the ensemble
% of selected items. This number Knz will be less than K if any item has
% a multiplicity greater than 1. The number of unselected items is
% LevCount[0] = N-Knz.

K = length(Category);
CatNz = Category(Category > 0);
if (sum(CatNz) ~= K)
```

```
  error('Invalid category vector');
end

% Count the number of instances of multiplicity i in Category
Knz = length(CatNz);
LevCount = [N-Knz, sum(CatNz' == (1:K), 1)];

end

% ------ ------
function Nm = MultiChoose(N, r)

if (sum(r) ~= N)
  error('Invalid parameters to MultiChoose');
end

% Simplest form (subject to overflow in the numerator)
% N = factorial(n) / prod(factorial(r));

% Cancel out the largest factor in r using nchoosek
[rmax, I] = max(r);
r(I(1)) = 1;

Nm = nchoosek(N, rmax) * factorial(N - rmax) / prod(factorial(r));

end
```

Appendix D MultiLevelCode / MultiLevelDecode - Combinatorial Codes (with prescribed numbers of levels)

The routine MultiLevelCode in this appendix implements Eq. (35) in Matlab. The routine sorts the array of selected indices if they are not in ascending order. The routine MultiLeveleDecode implements the decoding procedure.

```
function CIndex = MultiLevelCode(a)
%  CIndex = MultiLevelCode(a)
% Generate a lexicographically ordered combinatorial code for a vector of
% Q-ary symbols
%
% CIndex - Combinatorial code taking on values from 0 to
%      MultiChoose(N, Category)-1. Increasing values of CIndex correspond
%      to increasing lexicographic order of the vector a.
% a - Input vector of symbols (N values, each 0 to Q-1)

% $Id: MultiLevelCode.m,v 1.4 2018/02/19 13:34:57 pkabal Exp $

% Check input vector
if (any(a < 0))
  error('Invalid values in input vector');
end

Q = max(a) + 1;
N = length(a);
SymCount = zeros(1, Q);
Np = 1;      % MultiChoose(n, SymCount) - initialized for n = 0;

% Loop over all items
CIndex = 0;
for n = 0:N-1

  % Current symbol
  isym = a(n+1);

  % Sum symbol values up to (but not including) the current one
  s = sum(SymCount((0:isym-1)+1));
  if (s > 0)
    CIndex = CIndex + Np * s / (SymCount(isym+1) + 1);
  end
```

```
  % Update the symbol count
  SymCount(isym+1) = SymCount(isym+1) + 1;

  % Update Np for the next n
  Np = Np * (n+1) / SymCount(isym+1);
end

end
```

```
function a = MultiLevelDecode(CIndex, SymCount)
%  a = MultiLevelDecode(CIndex, LevelCount)
% Decode a combinatorial index for a vector of Q-ary symbols
%
% a - Output vector of symbols (N values, each 0 to Q-1)
% CIndex - Combinatorial code taking on values from 0 to
%      MultiChoose(N, SymCount)-1. Increasing values of CIndex correspond
%      to increasing lexicographic order of the vector a.
% SymCount - Q element vector containing symbol counts. SymCount(i) is
%      the number of times that symbol i+1 appears in the vector a. The sum
%      of the elements in SymCount is N, the length of the input vector a.

% $Id: MultiLevelDecode.m,v 1.4 2018/02/19 13:34:31 pkabal Exp $

Q = length(SymCount);
N = sum(SymCount);

Ns = MultiChoose(N, SymCount);
if (CIndex < 0 || CIndex >= Ns)
  error('Invalid index');
end

a = NaN(1, N);
for n = N-1:-1:0

  % The combinatorial index for [n, SymCount] is from 0 to Ns-1, where
  %   Ns = MultiChoose(n+1, SymCount)
  % Calculate Cn, the term due to possible a[n] values - when Cn exceeds
  % CIndex, we have gone too far - back off to previous value

  S = 0;
```

```
  CnP = 0;
  for isym = 0:Q-1
    S = S + SymCount(isym+1);
    Cn = Ns * S / (n+1);
    if (Cn > CIndex)

      % Update Ns
      Ns = Ns * SymCount(isym+1) / (n+1);

      % Update the symbol count / index
      SymCount(isym+1) = SymCount(isym+1) - 1;
      CIndex = CIndex - CnP;

      a(n+1) = isym;
      break;
    else
      CnP = Cn;
    end
  end

end

end

% ------ ------
function Nm = MultiChoose(N, r)

if (sum(r) ~= N)
  error('Invalid parameters to MultiChoose');
end

% Simplest form (subject to overflow in the numerator)
% N = factorial(n) / prod(factorial(r));

% Cancel out the largest factor in r using nchoosek
[rmax, I] = max(r);
r(I(1)) = 1;

Nm = nchoosek(N, rmax) * factorial(N - rmax) / prod(factorial(r));

end
```

References

[1] J. P. M. Schalkwijk, "An Algorithm for Source Coding", *IEEE Trans. Inform. Theory*, vol. IT-18, no. 3, pp. 395–399, May 1972.

[2] M. Berouti, H. Garten, P. Kabal, P. Mermeltein, "Efficient Computation and Encoding of the Multipulse Excitation for LPC", *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, (San Diego, CA), pp. 6–9, May 1984.

[3] ITU-T Recommendation G.723.1, *Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s*, March 1996.

[4] P. Kabal, *ITU-T G.723.1 Speech Coder: A Matlab Implementation*, Technical Report, Dept. Electrical & Computer Engineering, McGill University, December 2017.

[5] D. E. Knuth, *The Art of Computer Programming, Volume 4a Combinatorial Algorithms, Part 1A*, Addison-Wesley, ISBN 978-0201038040, May 2011.

[6] P. Kabal, *Routines for Combinatorial Coding and Lexicographic Ordering*, on-line `www.mathworks.com/matlabcentral/fileexchange`, February 2018.