

**Technical Report 95-1-006**

**Algorithms for Unranking Combinations  
and Other Related Choice Functions**

Zbigniew Kokosiński

February 27, 1995

**Department of Computer Software  
The University of Aizu  
Tsuruga, Ikki-Machi, Aizu-Wakamatsu City  
Fukushima, 965-80 Japan**

# Algorithms for Unranking Combinations and Other Related Choice Functions

## Abstract

In this report new unranking algorithms are developed for a large class of choice functions representing various classes of combinatorial objects: combinations, complementary combinations, conjugative nondecreasing choice functions, ordered partitions. Presented algorithms differ with the type of lexicographical order they deal with and the method of binomial coefficient evaluation. The proofs of the correctness provide a unified insight into various existing unranking techniques. All obtained algorithms are the best known within their classes. New unranking algorithms can be applied in parallel systems for distribution subtasks among processors and for parallel generation of choice functions including their ordered subsets and random sequences.

## 1. Introduction

In computer engineering ranking and unranking combinatorial objects is of great importance from the point of view of many practical applications. Algorithms for ranking and unranking combinations, permutations, partitions, trees *etc.*, are widely used in solving combinatorial problems in parallel because they can be applied for parallel generation of combinatorial objects and also play an important role in dividing splittable tasks and distribution of resulting subtasks among cooperating processors [2, 7].

All combinatorial objects we deal with throughout this paper have a common representation in the form of choice functions of indexed families of sets. This concept was proposed in [8] and applied in [2, 3] for the uniform representation of all combinatorial objects. Then this representation was used for representing various classes of combinatorial objects [4, 6, 7]. In this report we restrict our attention to algorithms for unranking set subsets (combinations) and some of their applications.

Unranking combination algorithms can be used to produce  $k$ -subsets of the set  $\{1, 2, \dots, n\}$ , where:  $1 \leq k \leq n$ , for any given sequence of natural numbers including random sequences. On the basis of unranking algorithms adaptive parallel algorithms for generation of all  $C(n, k)$  combinations were given [1, 11], and one of them lay claim to be cost-optimal [11]. One unranking combination method [10] was also applied for indexing ordered partitions [13]. All those algorithms derive elements  $a_i$ , of the sequence  $\langle a_1, \dots, a_i, \dots, a_k \rangle$ , where:  $1 \leq a_1 < \dots < a_i < \dots < a_k \leq n$ , for given

natural numbers  $n$ ,  $k$  and  $N$  ( $1 \leq N \leq C(n,k)$ ), by performing some arithmetic operations on binomial coefficients [1, 3, 7, 10, 11, 12].

Basically, unranking combinations algorithms can be characterized by two main criteria: 1) type of linear order defined on the set subsets we deal with, and 2) method applied for binomial coefficient evaluation. Different characteristics define separate categories of unranking algorithms. Usually, a lexicographical order is assumed, either increasing or decreasing. The method of evaluation of binomial coefficients effects on time and space complexity of given algorithm. Original formulation of early unranking algorithms was focused on giving a general method rather than an optimal one, so we may assume that binomial coefficients were there derived by factorialing [10, 13]. Then new algorithms were proposed with "restricted factorialing", where next consecutive binomial coefficient is obtained by certain modification of the previous one [1, 11]. Since relatively large constants are hidden behind linear assessments for this category of algorithms recently another approach was proposed in order to reduce computation time of multiple unranking: binomial coefficients are picked up from a supplementary table [3, 7]. A generalization of this tabular unranking methods to any class of combinatorial objects can be found in [5].

All known unranking combination algorithms are characterized in Table I.

TABLE I

Algorithm	Evaluation of binomial coefficients	Linear order	Time complexity	Space complexity
CNR [11]	RF	IL	$O(n)$	$O(k)$
RANKCINV [1]	RF	DL	$O(nk)$	$O(k)$
Kth COMBINATION [3]	CT	IL	$O(nk)$	$O(nk)$
UNRANKCOMB [7]	CT	DL	$O(n)$	$O(nk)$

RF - Reduced Factorialing, CT - Coefficient Table,  
 IL - Increasing Lexicographical, DL - Decreasing Lexicographical

Algorithms *CNR* [11] and *UNRANKCOMB* [7] are correct and optimal within their categories. Algorithms *RANKCINV* [1] and *Kth COMBINATION* [3] belong to different categories although they have the same asymptotic time complexity  $O(nk)$ . The algorithm *Kth COMBINATION* was favorable compared in [3] with algorithm *RANKCINV*, however both algorithms are nonoptimal within their classes.

In the recent paper algorithm *Kth MONOFU* was proposed for a class of unranking monotonic and nondecreasing choice functions [4]. This algorithm had been developed on the basis of the *Kth COMBINATION* algorithm and has the same time complexity  $O(nk)$ .

In this report we present new algorithms for unranking combinations belonging to various categories. They have reduced time complexities and have not drawbacks of previous methods. We begin with the algorithm *UNRANKCOMB-A* which first version was published without proof of its correctness [7]. Then we present the algorithm *UNRANKCOMB-B* which is an improved version of the algorithm *Kth COMBINATION* with reduced time complexity and versatile coefficient table. Both algorithms *UNRANKCOMB-A* and *UNRANKCOMB-B* have  $O(n)$  time complexity and use supplementary binomial coefficient tables in the form of Pascal Triangle or certain its modification. The tables can be used for all pairs  $\{n,k\}$  satisfying certain obvious restrictions resulting from the size of a given table. Proofs of correctness for both algorithms are keys for understanding not only all existing but also new algorithms in this area. On the basis of *UNRANKCOMB-A*, another unranking algorithm, *UNRANKCOMB-C*, is developed with  $O(k \log n)$  time complexity which reveals very good performance for large  $n$  and small  $k$ . After minor modifications this algorithm can be used also for unranking combinations if  $n$  is large and  $(n-k)$  is small. Our fourth unranking combination algorithm, *UNRANKCOMB-D*, belongs to the same category as *RANKCINV* but it has only  $O(n)$  time complexity.

All four algorithms can be easily modified for unranking nondecreasing choice functions of an indexed family of sets [4]. The resulting algorithms: *UNRANKNDCF-A*, *UNRANKNDCF-B*, *UNRANKNDCF-C* and *UNRANKNDCF-D* have  $O(n)$ ,  $O(n)$ ,  $O(k \log n)$  and  $O(n)$  time complexities, respectively.

The algorithm *UNRANKCOMB-C* is well suited for unranking lists of ordered partitions and seems to be the best one for this particular application.

## 2. Basic notions

Let  $\langle A_i \rangle_{i \in I}$  denote an **indexed family of sets**  $A_i = A$ , where:  $A = \{1, \dots, n\}$ ,  $I = \{1, \dots, k\}$ ,  $1 \leq k \leq n$ . Any mapping  $f$  which "chooses" one element from each set  $A_1, \dots, A_k$  is called a **choice function** of the family  $\langle A_i \rangle_{i \in I}$  [8].

If a supplementary condition:  $a_i < a_j$ , for  $i < j$ , and  $i, j \in I$ , is satisfied then any choice function  $\kappa = \langle a_i \rangle_{i \in I}$  that belongs to the indexed family  $\langle A_i \rangle_{i \in I}$  is called **increasing choice function** of this family. Sets of all increasing choice functions are representations of the set of all **k-subsets (combinations)** of the set  $A$  (in these cases we deal in fact with indexed sets  $C_i = \{i, \dots, n-k+i\} \subset A_i$ ) [2, 3].

Similarly, if another supplementary requirement:  $a_i \leq a_j$ , for  $i < j$ ,  $i, j \in I$ , holds then any sequence  $\lambda = \langle a_i \rangle_{i \in I}$  that belongs to the indexed family  $\langle A_i \rangle_{i \in I}$  is a **nondecreasing choice function** of this family [4].

For given **increasing choice function**  $\kappa = \langle a_i \rangle_{i \in I}$  that belongs to the indexed family  $\langle A_i \rangle_{i \in I}$  we define a **conjugate nondecreasing choice function**  $\lambda = \langle b_i \rangle_{i \in I}$ , that belongs to the indexed family  $\langle B_i \rangle_{i \in I}$ ,  $B_i = \{1, \dots, n-k+1\} \subset A_i$ , such that  $b_i = a_i - i + 1$ , for any  $i \in I$ . Function  $\kappa$  is conjugate in respect to  $\lambda$  iff  $\lambda$  is conjugate with  $\kappa$ .

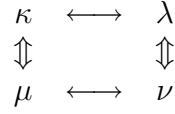


Figure 1: Complement and conjugation relations between choice functions  $\kappa, \lambda, \mu$  and  $\nu$  ( $\longleftrightarrow$  - conjugation relation,  $\Updownarrow$  - complement relation).

For given **increasing choice function**  $\kappa = \langle a_i \rangle_{i \in I}$  that belongs to the indexed family  $\langle A_i \rangle_{i \in I}$  we define a **complementary increasing choice function**  $\mu = \langle a_j \rangle_{j \in J}$ ,  $J = \{1, \dots, n-k\}$ , that belongs to the indexed family  $\langle A_j \rangle_{j \in J}$ , such that  $a_i \neq a_j$ , for any  $i \in I, j \in J$ . Function  $\kappa$  is complementary in respect to  $\mu$  iff  $\mu$  is complementary with  $\kappa$ .

For given choice functions  $\kappa, \lambda$  and  $\mu$  there exists the unique choice function  $\nu$  which is equal to the **conjugate nondecreasing choice function** for  $\mu$  and is called the **complementary nondecreasing choice function** for  $\lambda$ .

Relationships between choice functions  $\kappa, \lambda, \mu$  and  $\nu$  are shown in diagram (Fig.1). The number of different choice functions  $\langle a_1, \dots, a_{k-\alpha}, a_{k-\alpha+1}, \dots, a_k \rangle$  with constant sequence  $\langle a_1, \dots, a_{k-\alpha} \rangle$ ,  $0 \leq \alpha \leq k$  is a binomial coefficient :

- 1)  $C(n-a_{k-\alpha}, \alpha)$ , for increasing choice functions, where  $a_0 = 0$ ;
- 2)  $C(n-a_{k-\alpha} + \alpha - 1, \alpha)$ , for nondecreasing choice functions, where  $a_0 = 0$ ;

Without lost of generality, for the rest of this paper we can put  $\alpha = k$ .

Let us introduce now lexicographical order on the set of all choice functions of the family  $\langle A_i \rangle_{i \in I}$  :

For given choice functions  $\delta = \langle d_1, \dots, d_k \rangle$  and  $\gamma = \langle g_1, \dots, g_k \rangle$ , we say that  $\delta$  is less then  $\gamma$  according to the **increasing lexicographical order**, if and only if there exists  $i \in \{1, \dots, k\}$ , satisfying  $d_i < g_i$ , and  $d_j = g_j$ , for every  $j < i$ .

For given choice functions  $\delta = \langle d_1, \dots, d_k \rangle$  and  $\gamma = \langle g_1, \dots, g_k \rangle$ , we say that  $\delta$  is less then  $\gamma$  according to the **decreasing lexicographical order**, if and only if there exists  $i \in \{1, \dots, k\}$  satisfying  $d_i > g_i$  and  $d_j = g_j$ , for every  $j < i$ .

There is one to one correspondence between any linearly ordered set of choice functions  $S$  with cardinality  $|S| = s$  and linearly ordered set  $\{0, 1, \dots, s-1\}$ . If  $\beta \in S$  then  $\rho(\beta) = x$  is called **rank** of  $\beta$ , where  $\rho$  is the **ranking function**. The function  $\rho^{-1}(x) = \beta$  is called **unranking function**.

Ranks of  $\rho(\beta)$  and  $\rho'(\beta)$  in lexicographical increasing and decreasing orders, respectively, satisfy relation:  $\rho(\beta) + \rho'(\beta) = s-1$ .

### 3. Algorithms for unranking increasing choice functions

In this section we assume combinations to be represented by increasing choice functions.

In the algorithm **UNRANKCOMB-A** a table PTA is used, which includes a part of a modified Pascal Triangle (see Table II). Each binomial coefficient  $C(n,k)$  is mapped to the cell  $PTA[n-k+2,k]$  there.

TABLE II (Table PTA for:  $1 \leq k \leq n \leq n_{max} = 6$ )

$i \setminus j$	1	2	3	4	5	6
1	0	0	0	0	0	0
2	1	1	1	1	1	...
3	2	3	4	5	...	
4	3	6	10	...		
5	4	10	...			
6	5	...				

In the algorithm given below table PTA is already given. The following restrictions on pairs  $\{n,k\}$ ,  $k \leq n$ , should be taken into account during construction of this table (2 and 3 optionally):

- 1)  $n \leq n_{max}$ , where:  $n_{max}$  - is any natural number (size of the triangle);
- 2)  $(1 \leq k \leq k_{max} \leq n_{max})$  (columns from 1 to  $k_{max}$ );
- 3)  $0 \leq n-k=r \leq n_{max} - k_{min}$ , (rows from 1 to  $r+2$ );

An algorithm of construction of the table PTA is obvious and therefore will be omitted. Now we are ready to present our first unranking combination algorithm.

#### *Algorithm UNRANKCOMB-A*

Input :  $n, k, N$  ( $0 \leq N \leq C(n,k)-1$ ) - number of the choice function  $\kappa$  in increasing lexicographical order, table  $PTA[n,n]$  in which element  $PTA[r-k+2,k]$  contains value  $C(r,k)$ , for  $k-1 \leq r \leq n$ .

Output: table  $K[k]$  with choice function  $\kappa$ .

Method: Computations proceed with combinations ranks in decreasing lexicographical order. In order to determine  $K$  a "divide and conquer" technique is used. In each step 2.1 a subset of choice functions containing  $N$ 'th combination is partitioned into two blocks according to object ranks. If condition 2.1 is satisfied then next value  $K[i]$  is obtained in step 2.1.1. In each iteration a new condition 2.1 is derived according to the value  $PTA[t,m]$  as well as the combination rank  $N$  in a new subset. After  $n$  iterations we get a unique output sequence.

```

1. t:=n-k+1; m:=k; N':=PTA[t,m]+PTA[t+1,m-1]-1-N;
2. repeat
  2.1. if PTA[t,m] =< N'
    then
      2.1.1. K[k-m+1]:=n-t-m+2; N':=N'-PTA[t,m]; m:=m-1;
    else
      2.1.2. t:=t-1;
  until m=0;
3. return K.

```

*Example 1*

For  $n=6$  and  $k=4$  find 12th increasing choice function in the increasing lexicographical order.

*Solution:*

Step 1:

$t=3, m=4, N'=PTA[3,4]+PTA[4,3]-1-12=2$

Step 2:

Step 2.1:  $PTA[3,4]=5 > 2$ ; Step 2.1.2:  $t=2$ .

$m>0$ : Step 2.1:  $PTA[2,4]=1 =< 2$ ; Step 2.1.1:  $K[1]=2, N'=1, m=3$ .

$m>0$ : Step 2.1:  $PTA[2,3]=1 =< 1$ ; Step 2.1.1:  $K[2]=3, N'=0, m=2$ .

$m>0$ : Step 2.1:  $PTA[2,2]=1 > 0$ ; Step 2.1.2:  $t=1$ .

$m>0$ : Step 2.1:  $PTA[1,2]=0 =< 0$ ; Step 2.1.1:  $K[3]=5, N'=0, m=1$ .

$m>0$ : Step 2.1:  $PTA[1,1]=0 =< 0$ ; Step 2.1.1:  $K[4]=6, N'=0, m=0$ .

$m=0$ :

Step 3:

return K.

The 12th increasing choice function in lexicographical order for  $n=6, k=4$  is  $\langle 2, 3, 5, 6 \rangle$ .  $\square$

**Theorem 1**

Algorithm *UNRANKCOMB-A* is correct and its asymptotic computational complexity is  $O(n)$ .

**Proof**

The set of all  $C(n,k)$  combinations can be shown in the form of a rooted ordered tree of height  $k$  (see Fig.2). There are  $C(n-k+i,i)$  nodes with depth  $i$ . Each node with depth  $i, 0 \leq i \leq k-1$ , has  $(n-k+i+1-t)$  descendants, where  $t$  is an integer label for edge connecting given node with its ancestor (for root that have no ancestor we

depth (level)	0	1	2	3	4	path rank								
		--3--	1	----	4--	1	----	5--	1	----	6--	1	0	
				--4--	1	----	5--	1	----	6--	1	1		
						--5--	1	----	6--	1	2			
								--6--	1	3				
		--2--	4	----	3--	3	----	4--	2	----	5--	2	4	
				--4--	1	----	5--	1	----	6--	1	5		
						--5--	1	----	6--	1	6			
								--6--	1	7				
			--3--	4	----	4--	2	----	5--	2	8			
				--5--	1	----	6--	1	9					
						--6--	1	10						
					--4--	3	----	5--	2	11				
						--6--	1	12						
						--5--	2	13						
	ROOT	----	1--	15	----	2--	10	----	3--	6	----	4--	3	14

Figure 2: Rooted ordered tree of all  $C(6,4)$  combinations (X - edge label, X - node label, X - useless node label).

assume  $t=0$ ), and edges connecting given node with its descendants are labeled by  $(t+1), (t+2), \dots, (n-k+i)$ , respectively. In this way all nodes with depth  $i$  as well as all paths are ordered in the tree. Traversing the tree in preorder and listing all path from the root to subsequent leaves - by sequences of edge labels - is equivalent to generation of all  $C(n,k)$  combinations in increasing lexicographical order. Let us assign to all such paths their ranks in decreasing lexicographical order. Unranking the combination with rank  $N$  in the tree is equivalent to finding in the tree a path with rank  $N' = C(n,k)-1-N$ . Every node of the tree has an integer label equal to the sum of all leaves of ordered subtrees rooted in this node and all its siblings with depth  $i$  following it. Each node label is a binomial coefficient. We determine the path with rank  $N'$  by determining a proper subtrees on consecutive levels starting from the root. Rooted subtrees on  $i$ th level are viewed in decreasing order of their size (size means in this case : the number of subtree leaves). In order to do this current rank of choice function is compared with node labels  $C(r,k)$  taken from the cell  $PTA[r-k+2,k]$ . In each level no more then  $(n-k+i-t)$  comparisons are made and before the next step rank  $N'$  is modified (step 2.1.1 of the algorithm). Step 2.1 with the complexity  $O(1)$  is repeated  $O(n)$  times. Condition  $PTA[t,m] \leq N$  is satisfied  $k$  times and in each step 2.1.1, next item of the  $N$ th  $k$ -subset is obtained. Hence the total complexity of the algorithm is  $O(n)$ .  $\square$

Algorithm *Kth COMBINATION* [3] was the first algorithm for unranking combinations in which binomial coefficients were picked up from a supplementary table DCM. However, this algorithm reveals two important shortcomings:

1) the presence of inner loop in the step 8.1.1.2 increases its time complexity to  $O(nk)$ , and

2) in order to allow multiple unranking for different pairs  $\{n,k\}$  setting initial values of DCM coordinates in steps 6 and 7 has to be redefined.

Below we present new unranking algorithm UNRANKCOMB-B that belongs to the same category as *Kth COMBINATION* but its time complexity is reduced to  $O(n)$  and the supplementary table PTB is convenient for multiple unranking. Table PTB has the form of the conventional Pascal Triangle:  $PTB[i,j] = PTB[j,i] = C(i+j-2, j-1)$ , for any  $i, j \geq 1$ .

*Algorithm UNRANKCOMB-B*

Input :  $n, k, N$  ( $0 \leq N \leq C(n,k)-1$ ) - number of the choice function  $\kappa$  in increasing lexicographical order, table  $PTB[n,n]$  in which element  $PTB[r-k+1, k+1]$  contains value  $C(r,k)$ , for  $0 \leq k \leq r \leq n$ .

Output: table  $K[k]$  with choice function  $\kappa$ .

Method: Computations proceed with combinations ranks in increasing lexicographical order. In order to determine  $K$  a "divide and conquer" technique is used. In each step 2.1 a subset of choice functions containing  $N$ th combination is partitioned into two

blocks according to object ranks. If condition 2.1 is not satisfied then next value  $K[i]$  is obtained in step 2.1.2. In each iteration a new condition 2.1 is derived according to the value  $PTB[l,m]$  as well as the combination rank  $N$  in a new subset. After  $n$  iterations we get a unique output sequence.

```

1. t:=n-k+1; m:=k;
2. repeat
  2.1. if PTB[t,m] > N
    then
      2.1.1. N:=N-PTB[t,m]; t:=t-1;
    else
      2.1.2. K[k-m+1]:=n-m-t+2; m:=m-1;
  until m=0;
3. return K.

```

*Example 2*

For  $n=6$  and  $k=4$  find 12th increasing choice function in the increasing lexicographical order.

*Solution:*

```

Step 1: t=3, m=4.
Step 2: Step 2.1: PTB[3,4]=10 > 12; Step 2.1.1: N=2, t=2.
m>0: Step 2.1: PTB[2,4]= 4 > 2; Step 2.1.2: K[1]=2, m=3.
m>0: Step 2.1: PTB[2,3]= 3 > 2; Step 2.1.2: K[2]=3, m=2.
m>0: Step 2.1: PTB[2,2]= 2 > 2; Step 2.1.1: N=0, t=1.
m>0: Step 2.1: PTB[1,2]= 1 > 0; Step 2.1.2: K[3]=5, m=1.
m>0: Step 2.1: PTB[1,1]= 1 > 0; Step 2.1.1: K[4]=6, m=0.
m=0: return K.

```

The 12th increasing choice function in lexicographical order for  $n=6,k=4$  is  $\langle 2, 3, 5, 6 \rangle$ .  $\square$

**Theorem 2**

Algorithm UNRANKCOMB-B is correct and its asymptotic computational complexity is  $O(n)$ .

**Proof**

The set of all  $C(n,k)$  combinations can be shown in the form of a rooted ordered tree of height  $k$  (see Fig.3).

There are  $C(n-k+i,i)$  nodes with depth  $i$ . Each node with depth  $i$ ,  $0 \leq i \leq k-1$ , has  $(n-k+i+1-t)$  descendants, where  $t$  is an integer label for edge connecting given

depth (level)	0	1	2	3	4	path	rank
		--3--	1 ---4--	1 ---5--	1 ---6--	1	14
			--4--	1 ---5--	1 ---6--	1	13
				--5--	1 ---6--	1	12
					--6--	1	11
	--2--	4 ---3--	3 ---4--	2 ---5--	1		10
		--4--	1 ---5--	1 ---6--	1		9
			--5--	1 ---6--	1		8
				--6--	1		7
		--3--	3 ---4--	2 ---5--	1		6
			--5--	1 ---6--	1		5
				--6--	1		4
			--4--	2 ---5--	1		3
				--6--	1		2
				--5--	1		1
	ROOT	---1--	10 ---2--	6 ---3--	3 ---4--	1	0

Figure 3: Rooted ordered tree of all  $C(6,4)$  combinations (X - edge label, X - node label).

node with its ancestor (for root that have no ancestor we assume  $t=0$ ), and edges connecting given node with its descendants are labeled by  $(t+1), (t+2), \dots, (n-k+i)$ , respectively. In this way all nodes with depth  $i$  as well as all paths are ordered in the tree. Traversing the tree in preorder and listing all path from the root to subsequent leaves - by sequences of edge labels - is equivalent to generation of all  $C(n,k)$  combinations in increasing lexicographical order. Let us assign to all such paths their ranks in increasing lexicographical order. Unranking combination with rank  $N$  in the tree is equivalent to finding in the tree a path with rank  $N$ . Every node of the tree has an integer label equal to the number of all leaves of ordered subtree rooted in this node. Each node label is a binomial coefficient. We determine the path with rank  $N$  by determining a proper subtrees on consecutive levels starting from the root. Rooted subtrees on  $i$ th level are viewed in decreasing order of their size (size means in this case : the number of subtree leaves). In order to do this current rank of choice function is compared with node labels  $C(r,k)$  are taken from the cell  $PTB[r-k+1, k+1]$ . In each level no more then  $(n-k+i+1-t)$  comparisons are made and before the next step rank  $N'$  is modified (step 2.1.1 of the algorithm). Step 2.1 with the complexity  $O(1)$  is repeated  $O(n)$  times. Condition  $PTB[t, m] \leq N$  is satisfied  $k$  times and in each step 2.1.2 next item of the  $N$ th  $k$ -subset is obtained. Hence the total complexity of the algorithm is  $O(n)$ .  $\square$

Both above algorithms are best in their categories. However their average performance is relatively low if  $k \ll n$ . This fact motivated us to develop an algorithm with smaller asymptotic time complexity. Let us notice that values  $PTA[i, j]$  appear in the  $j$ th column of  $PTA$  table in an increasing order with  $i$ . Therefore the searching technique used for finding  $PTA[t, m]$  which satisfies condition 2.1 in the algorithm *UNRANKCOMB-A* can be replaced by a binary search. In this way a "divide and conquer" technique is to be applied once more on the lower level to determine a parameter required by the "divide and conquer" on the higher level. Due to different construction of the table  $PTB$  the algorithm *UNRANKCOMB-B* can not be modified in this way.

The third unranking combination algorithm presented in this section, *i.e.* the algorithm *UNRANKCOMB-C* is constructed on the basis of the algorithm *UNRANKCOMB-A*. Similarly as in the previous algorithms we assume representation of combinations by increasing choice functions.

#### *Algorithm UNRANKCOMB-C*

Input :  $n, k, N$  ( $0 \leq N \leq C(n,k)-1$ ) - number of the choice function  $\kappa$  in increasing lexicographical order, table  $PTA[n, n]$  in which element  $PTA[r-k+2, k]$  contains value  $C(r, k)$ , for  $k-1 \leq r \leq n$ .

Output: table  $K[k]$  with choice function  $\kappa$ .

Method: Computations proceed with combinations ranks in decreasing lexicographical order. In order to determine  $K$  a "divide and conquer" technique is used. In each step 2.1 a subset of choice functions containing  $N'$ 'th combination is partitioned into two blocks according to object ranks.  $PTA[t,m]$  used in step 2.2 is determined using a binary search recursive procedure in step 2.1. Values  $K[i]$ ,  $1 \leq i \leq k$ , are obtained in  $k$  consecutive steps. After  $i$ 'th step the new combination rank  $N'$  in a new subset is computed. (Notation  $\text{Trunc}(x)$  denotes the integer part of  $x$ ).

```

1.  $t=n-k+1$ ;  $N' := PTA[t,k] + PTA(t+1,k-1) - 1 - N$ ;
2. for  $m=k$  downto 1 do
    2.1. BS ( $N', PTA, m, 1, t, t$ );
    2.2.  $K[k-m+1] := n-t-m+2$ ;  $N' := N' - PTA[t,m]$ ;
3. return  $K$ .

```

```

procedure BS ( $N', PTA, m, w, u, t$ )
4.  $t := \text{Trunc}((w+u)/2)$ ;
5. if ( $N' < PTA[t,m]$ ) then
    5.1. BS( $N', PTA, m, w, t-1, t$ )
    else
    5.2. if ( $N' \geq PTA[t+1,m]$ ) then BS( $N', PTA, m, t, u, t$ );
6. return

```

*Example 3*

For  $n=7$  and  $k=2$  find 14th increasing choice function in the increasing lexicographical order.

*Solution:*

```

Step 1:  $t=6$ ,  $N' = PTA[6,2] + PTA[7,1] - 1 - 14 = 6$ .
Step 2:  $m=2$ ; Step 2.1:  $w:=1$ ;  $u:=6$ .
        Step 2.2: BS(6,PTA,2,1,6,6), return with  $t=4$ .
        Step 2.3:  $K[1]=3$ ,  $N'=0$ .
Step 2:  $m=1$ ; Step 2.1:  $w:=1$ ;  $u:=4$ .
        Step 2.2: BS(0,PTA,1,1,4,4), return with  $t=1$ .
        Step 2.3:  $K[2]=7$ ,  $N'=0$ .
Step 3: return  $K$ .

```

The 14th increasing choice function in lexicographical order for  $n=7$ ,  $k=2$  is  $\langle 3, 7 \rangle$ .  $\square$

### Theorem 3

Algorithm *UNRANKCOMB-C* is correct and its asymptotic computational complexity is  $O(k \log n)$ .

### Proof

General approach is the same as in algorithm *UNRANKCOMB-A*, hence correctness of the method results directly the Proof of Theorem 1. Each binomial coefficient  $C(n,k)$  is taken from the cell  $PTA[n-k+2,k]$ . Step 2 is repeated  $k$  times. Complexity of binary searching among  $n$  items is  $O(\log n)$ . Since in each column of the PTA table we have at most  $\log(n-m)$  binomial coefficients the complexity of BS procedure is  $O(\log(n-k)) \leq O(\log n)$ . In this way the total complexity of the algorithm *UNRANKCOMB-C* is  $O(k \log n)$ .  $\square$

In particular cases of input data we obtain the following assessments of the complexity of the algorithm *UNRANKCOMB-C*:

- a)  $O(\log_2 n)$ , for  $k \leq \log n$ ;
- b)  $O(\log n)$ , for  $k \leq C$ , where  $C$  is a constant.

Unranking algorithms with additional  $O(nk)$  space and preprocessing time used for the construction of the supplementary table are good tools if computations are repeated many times for various input data. For occasional usage only another algorithm with "reduced factorialing" technique can be applied with  $O(n)$  time complexity. Relatively large constant is hidden in this assesment. Presented below algorithm *UNRANKCOMB-D* is a version of the algorithm em *UNRANKCOMB-A* in which binary coefficients are computed in each step using one of the following two formulas:

- 1)  $C(n-1,k) = (n-k)C(n,k)/n$ , and
- 2)  $C(n-1,k-1) = kC(n,k)/n$ .

### Algorithm UNRANKCOMB-D

Input :  $n, k, N$  ( $0 \leq N \leq C(n,k)-1$ ) - number of the choice function  $\kappa$  in increasing lexicographical order.

Output: table  $K[k]$  with choice function  $\kappa$ .

Method: Computations proceed with combinations ranks in decreasing lexicographical order. In order to determine  $K$  a "divide and conquer" technique is used. In each step 3 a subset of choice functions containing  $N$ 'th combination is partitioned into two blocks according to object ranks. If condition 3.1 is satisfied then next value  $K[i]$  is obtained in step 3.1.1. In each iteration a new condition 3.1 is derived according to the value  $e$  and the combination rank  $N'$  in a new subset. After  $n$  iterations we get a unique output sequence.

```

1. e:=C(n,k); N':=e-1-N;
2. e:=(n-k)e/n; t:=n-k+1; m:=k; p:=n-1;
3. repeat
  3.1. if e =< N'
    then
      3.1.1. K[k-m+1]:=n-t-m+2;
      3.1.2. if e > 0 then
        3.1.2.1. N':=N'-e; e:=me/p;
      3.1.3. m:=m-1; p:=p-1;
      else
        3.1.4. e:=(p-m)e/p; t:=t-1; p:=p-1;
    until m=0;
4. return K.

```

*Example 4*

For  $n=7$  and  $k=2$  find 14th increasing choice function in the increasing lexicographical order.

*Solution:*

Step 1:

$e=C(7,2)=21$ ;  $N'=6$ .

Step 2:

$e=15$ ;  $t=6$ ;  $m=2$ ;  $p=6$ .

Step 3:

Step 3.1:  $e=15 > N'=6$ ; Step 3.1.4:  $e=10$ ;  $t=5$ ;  $p=5$ .

$m>0$ : Step 3.1:  $e=10 > N'=6$ ; Step 3.1.4:  $e=6$ ;  $t=4$ ;  $p=4$ .

$m>0$ : Step 3.1:  $e=6 = N'=<6$ ; Step 3.1.1:  $K[1]=3$ ;

Step 3.1.2:  $e=6>0$ :

Step 3.1.2.1:  $N'=0$ ;  $e=3$ .

Step 3.1.3:  $m=1$ ;  $p=3$ .

$m>0$ : Step 3.1:  $e=3 > N'=0$ ; Step 3.1.4:  $e=2$ ;  $t=3$ ;  $p=2$ .

$m>0$ : Step 3.1:  $e=2 > N'=0$ ; Step 3.1.4:  $e=1$ ;  $t=2$ ;  $p=1$ .

$m>0$ : Step 3.1:  $e=1 > N'=0$ ; Step 3.1.4:  $e=0$ ;  $t=1$ ;  $p=0$ .

$m>0$ : Step 3.1:  $e=0 =<N'=0$ ; Step 3.1.1:  $K[2]=7$ ;

Step 3.1.2:  $e=0$ .

Step 3.1.3:  $m=0$ ;  $p=-1$ .

$m=0$ :

Step 4:

return K.

The 14th increasing choice function in lexicographical order for  $n=7$ ,  $k=2$  is  $\langle 3, 7 \rangle$ .  $\square$

#### Theorem 4

Algorithm *UNRANKCOMB-D* is correct and its asymptotic computational complexity is  $O(n)$ .

#### Proof

Correctness of the method results directly from the Proof of Theorem 1. Each binomial coefficient  $C(n,k)$  is obtained from one of the two formulas: 1)  $C(n-1,k)=(n-k)C(n,k)/n$ ; and 2)  $C(n-1,k-1)=kC(n,k)/n$ .

Step 3.1 with the complexity  $O(1)$  is repeated  $n$  times. Value  $C(n,k)$  in the step 1 can be computed in  $O(k)$  time. Formulas 1) and 2) used for computing values  $e$  in steps 2, 3.1.2.1 and 3.1.4 can be evaluated in  $O(1)$  time. The condition  $e \leq N'$  is satisfied  $k$  times and in each step 3.1.1 next item of the  $N$ th  $k$ -subset is obtained. Hence, the total complexity of the algorithm is  $O(n)$ .  $\square$

In comparison to other algorithms of this type the algorithm *UNRANKCOMB-D* is faster than algorithm *RANKCINV* [1] which has  $O(nk)$  time complexity, and has the same complexity and approximately the same constant as the algorithm *CNR* [11]. Both algorithms: *UNRANKCOMB-D* and *RANKCINV* deal with combination ranks in decreasing lexicographical order while the algorithm *CNR* deals with combination ranks in increasing lexicographical order.

Properties of all new algorithms presented above are gathered in Table III.

TABLE III

Algorithm	Evaluation of binomial coefficients	Linear order	Time complexity	Space complexity
UNRANKCOMB-A	CT	DL	$O(n)$	$O(nk)$
UNRANKCOMB-B	CT	IL	$O(n)$	$O(nk)$
UNRANKCOMB-C	CT	DL	$O(k \log n)$	$O(nk)$
UNRANKCOMB-D	RF	DL	$O(n)$	$O(k)$

RF - Reduced Factorialing, CT - Coefficient Table,  
 IL - Increasing Lexicographical, DL - Decreasing Lexicographical

In the next section we show what typical applications of unranking combination algorithms are.

## 4. Applications

Some of applications of combination unranking algorithms were mentioned in the Introduction. Among many usefull applications the following two are of our particular interest:

- 1) unranking some other related combinatorial objects, like complementary increasing choice functions, monotonic choice functions and ordered partitions, and
- 2) generation of combinatorial objects.

We deal with these applications throughout next four subsections.

### 4.1. Unranking complementary increasing choice functions

Increasing lexicographical order of increasing choice functions  $\kappa$  of the family  $\langle A_i \rangle_{i \in I}$  does corresponde to decreasing lexicographical order of complementary increasing choice functions  $\mu$  of the family  $\langle A_j \rangle_{j \in J}$ . Hence,  $\rho(\kappa) = \rho'(\mu)$ .

This means that unranking combination algorithms can also be used for unranking complementary increasing choice functions (complementary combinations). For large  $k$ , we can deal with  $(n-k)$ -subsets rather than with  $k$ -subsets. The approach is straightforward: instead of  $N$ 'th  $k$ -element increasing choice function in increasing lexicographical order we need to determine  $N$ 'th  $(n-k)$ -element increasing choice function in this order and then find the complementary increasing choice function. For  $(n-k) \ll n$ , using algorithm *UNRANKCOMB-C* is the best choice. In order to obtain the complementary increasing choice function  $\mu$  for given increasing choice function  $\kappa$  we use following simple procedure:

```
procedure COMPCOMB (n, k, K[n-k], M[k])
  1. i:=0; j:=1; p:=1;
  2. repeat
    2.1. if j=M[p]
      then
        2.1.1. p:=p+1;
      else
        2.1.2. i:=i+1; M[i]:=j;
    2.2. j:=j+1;
  until i=k;
```

Input choice function is written in the table  $K[k]$  while the output choice function is written in the table  $M[n-k]$ . Although the procedure has  $O(n)$  time complexity the constant of this assessment is low and in most practical applications it does not effect on the total unranking computation time.

## 4.2. Unranking other monotonic choice functions of the family $\{1, 2, \dots, n\}^k$ , $1 \leq k \leq n$

The concept of a monotonic choice function of the family  $\{1, 2, \dots, n\}^k$ ,  $1 \leq k \leq n$ , is introduced in [4] where also some relations between this class of choice functions and choice functions representing number and set partitions are investigated.

Unranking algorithms developed in section 3 can be immediately applied for unranking monotonic nondecreasing choice functions  $\lambda$  of  $\{1, 2, \dots, n\}^k$ ,  $1 \leq k \leq n$ .

There are two ways of solving this problem. The first one : instead of the  $N$ th nondecreasing choice function  $\lambda$  find the  $N$ th combination  $k$  out of  $(n+k-1)$  items using any unranking algorithm given in section 3, and then using the following conversion procedure derive the corresponding nondecreasing choice function (written in a table  $L[k]$ ):

```

procedure CONJNDCF (k, K[k], L[k])
1. for i=1 to k do
    1.1. L[i]:= K[i]-i+1;

```

If one prefers not to use the conversion procedure to the output of an algorithm for unranking combinations we recommend modifying of any presented above unranking algorithm by introducing the conversion into it. It can be done by replacing the substitution  $K[k-m+1]:=n-t-m+2$  performed in steps 2.1.1, 2.1.2, 2.2 and 3.1.1 of the algorithms *UNRANKCOMB-A*, *UNRANKCOMB-B*, *UNRANKCOMB-C* and *UNRANKCOMB-D*, respectively with  $L[k-m+1]=n-l-k+2$ . On the input of each algorithm input variable  $n$  must be replaced by  $n'=(n+k-1)$ . Thus we obtain four new algorithms for unranking nondecreasing choice functions : *UNRANKNDCF-A*, *UNRANKNDCF-B*, *UNRANKNDCF-C* and *UNRANKNDCF-D* with complexities  $O(n)$ ,  $O(n)$ ,  $O(k \log n)$  and  $O(n)$ , respectively.

As an example we present a pseudocode of the *UNRANKNDCF-C* algorithm.

### *Algorithm UNRANKNDCF-C*

Input :  $n, k, N$  ( $0 \leq N \leq C(n+k-1, k)-1$ ) - number of the choice function  $\kappa$  in increasing lexicographical order, table  $PTA[n+k-1, n+k-1]$  in which element  $PTA[r-k+2, k]$  contains value  $C(r, k)$ , for  $k-1 \leq r \leq n+k-1$ .

Output: table  $L[k]$  with choice function  $\lambda$ .

Method: Computations runs with nonincreasing choice function ranks in decreasing lexicographical order. In order to determine L a "divide and conquer" technique is used. In each step 2.1 a subset of choice functions containing nonincreasing choice function is partitioned into two blocks according to object ranks.  $PTA[t,m]$  used in step 2.2 is determined using a binary search recursive procedure in step 2.1 ( the same as in the algorithm UNRANKCOMB-C ). Values  $K[i]$ ,  $1 \leq i \leq k$ , are obtained in  $k$  consecutive steps. After  $i$ th step the new object rank  $N'$  in a new subset is computed.

1.  $t=n$ ;  $N' := PTA[t,k] + PTA(t+1,k-1) - 1 - N$ ;
2. for  $m=k$  downto 1 do
  - 2.1. BS ( $N', PTA, m, 1, t, t$ );
  - 2.2.  $L[k-m+1] := n-t+1$ ;  $N' := N' - PTA[t,m]$ ;
3. return L.

*Example 5*

For  $n=7$  and  $k=2$  find 14th nondecreasing choice function in lexicographical order.

*Solution:*

- Step 1:  $t=7$ ,  $N' = PTA[7,2] + PTA[8,1] - 1 - 14 = 13$ .
- Step 2:  $m=2$ ; Step 2.1: BS(13,PTA,2,1,7,7), return with  $t=5$ .  
 Step 2.2:  $L[1]=3$ ,  $N'=3$ .
- Step 2:  $m=1$ ; Step 2.1: BS(3,PTA,1,1,5,5), return with  $t=4$ .  
 Step 2.2:  $L[2]=4$ ,  $N'=0$ .
- Step 3: return L.

The 14th nondecreasing choice function in lexicographical order for  $n=7$ ,  $k=2$  is  $\langle 3, 4 \rangle$ .  $\square$

All derivative algorithms for unranking nondecreasing choice functions are faster then the algorithm *Kth MONOFU* which has  $O(nk)$  time complexity.

If necessary, for large  $k$ , in order to reduce time of computations we can deal with complementary nondecreasing choice functions.

Increasing lexicographical order of nondecreasing choice function  $\lambda$  of the family  $\langle A_i \rangle_{i \in I}$  does corresponde to decreasing lexicographical order of complementary nondecreasing choice functions  $\nu$  of the family  $\langle A_j \rangle_{j \in J}$ . Hence,  $\rho(\lambda) = \rho'(\nu)$ .

This means that for  $(n-k) \ll n$  method described in section 4.1. for unranking complementary increasing choice functions is easy applicable for unranking nondecreasing choice functions: instead of  $N$ th nondecreasing choice function in increasing lexicographical order we need to determine  $N'$ th nondecreasing choice function in this order and then find the complementary nondecreasing choice function.

### 4.3. Unranking ordered partitions

In [13] a method of ranking ordered  $m$ -block partitions of the set  $\{1, 2, \dots, n\}$  is presented. A multinomial index for these partitions is there defined that consists of  $m$  subindices corresponding to each block of partition. Each  $i$ th subindex is rank of combination  $k_i$  out of  $m_i = (n - (k_1 + \dots + k_{i-1}))$  items, where  $k_i$  is the power of the  $i$ th block of partition. Ranking and unranking algorithms for ordered partitions are given there using existing algorithms for ranking and unranking increasing choice functions. Algorithms presented in this paper may be efficiently applied for this aim. In many cases if  $m$  is large enough and the condition  $m_i \gg k_i$  is satisfied the algorithm *UNRANKCOMB-C* will be the best choice. In this case the complexity of the algorithm for unranking ordered partitions can be significantly reduced to  $O(m \log^2 n)$  or even  $O(m \log n)$ .

Since unranking of ordered partitions consists of  $m$  independent steps each step devoted to unranking exactly one block index we can easily obtain an unranking parallel algorithm employing  $m$  processors. This parallel version runs with the time assessment  $O(\log^2 n)$  or even  $O(\log n)$ .

### 4.4. Generation of choice functions

One particular but very important application area is generation of choice functions representing various classes of combinatorial objects. We point out three applications of this type:

- 1) possibility construction adaptive and cost-optimal parallel algorithms for generation choice functions with any number of processors (obtained algorithms will be better than existing algorithms for generation by unranking [1, 11]);
- 2) generation of "irregular" subsets of choice functions [3, 4];
- 3) random generation of choice functions.

Generation choice functions by unranking is an alternative for generation methods applying other generation techniques. The choice of the proper method depends on the particular task. Since basic generation techniques are already known all details of particular applications will be omitted in this paper. Positions given as references are a good source for further study in this area.

## References

- [1] Akl S.G.: Design and analysis of parallel algorithms, Prentice Hall, Englewood Cliffs, N.J., 1989, pp. 148-150.
- [2] Kapralski A.: Binary matrices and their applications and processing in sequential and parallel dedicated processors, Monografia 95, Politechnika Krakowska, Kraków, Poland, 1989 (in Polish)
- [3] Kapralski A.: New methods for generation permutations, combinations and other combinatorial objects in parallel, J. Parallel and Distrib. Computing, 17 (1993), pp. 315-326.
- [4] Kapralski A.: Representation and parallel generation of number and set partitions, decompositions and related combinatorial objects, TR 94-1-038, University of Aizu, 1994.
- [5] Kapralski A.: Models and parallel generation of any subset of partitions, compositions, decompositions and other combinatorial objects (submitted for publication)
- [6] Kokosiński Z.: On generation of permutations through decomposition of symmetric groups into cosets, BIT, 30 (1990), pp. 583-591.
- [7] Kokosiński Z.: Circuits generating combinatorial configurations for sequential and parallel computer systems, Monografia 160, Politechnika Krakowska, Kraków, Poland, 1993, 106 pp. (in Polish)
- [8] Mirsky L.: Transversal theory, Academic Press, N.Y. 1971.
- [9] Lehmer D.H.: Teaching combinatorial trick to a computer, [in:] Proc. of Symposium Appl. Math., Combinatorial Analysis 10, Amer. Math. Society, Providence, R.I. 1960, pp. 179-193.
- [10] Lehmer D.H.: The machine tools of combinatorics, [in:] Beckenbach E.F. (editor): Applied combinatorial mathematics, John Wiley, N.Y. 1964, pp. 5-31.
- [11] Tang C.Y., Du M.W. and Lee R.C.T.: Parallel generation of combinations, [in:] Proc. Int. Computer Symposium, Taipei, Taiwan 1984, pp. 1006-1010.
- [12] Wells M.B.: Elements of combinatorial computing, Pergamon Press, N.Y. 1971.
- [13] Williamson S.G.: Ranking algorithms for lists of partitions, SIAM J. Comp. 5 (1976), No.4, pp. 602-617.