algorithms

MDPI

Article

Unranking Small Combinations of a Large Set in Co-Lexicographic Order

Vladimir Kruchinin [1], Yuriy Shablya [2,*], Dmitry Kruchinin [3] and Victor Rulevskiy [3]

[1] Institute of Innovation, Tomsk State University of Control Systems and Radioelectronics, 634050 Tomsk, Russia; kru@2i.tusur.ru
[2] Department of Complex Information Security of Computer Systems, Tomsk State University of Control Systems and Radioelectronics, 634050 Tomsk, Russia
[3] Department of Computer Control and Design Systems, Tomsk State University of Control Systems and Radioelectronics, 634050 Tomsk, Russia; kruchinindm@gmail.com (D.K.); rvm@tusur.ru (V.R.)
* Correspondence: shablya-yv@mail.ru

Abstract: The presented research is devoted to the problem of developing new combinatorial generation algorithms for combinations. In this paper, we develop a modification of Ruskey's algorithm for unranking m-combinations of an n-set in co-lexicographic order. The proposed modification is based on the use of approximations to make a preliminary search for the values of the internal parameter k of this algorithm. In contrast to the original algorithm, the obtained algorithm can be effectively applied when n is large and m is small because the running time of this algorithm depends only on m. Furthermore, this algorithm can be effectively used when n and m are both large but $n - m$ is small, since we can consider unranking $(n - m)$-combinations of an n-set. The conducted computational experiments confirm the effectiveness of the developed modification.

Keywords: combinatorial generation; combination; unranking; co-lexicographic order; combinatorial algorithm

Citation: Kruchinin, V.; Shablya, Y.; Kruchinin, D.; Rulevskiy, V. Unranking Small Combinations of a Large Set in Co-Lexicographic Order. *Algorithms* **2022**, *15*, 36. https://doi.org/10.3390/a15020036

Academic Editor: David F. Manlove

Received: 6 January 2022
Accepted: 23 January 2022
Published: 25 January 2022

1. Introduction

Combinatorial algorithms are algorithms for investigating combinatorial structures. The development of combinatorial algorithms is one of the basic tasks in computer science [1,2]. Such algorithms provide efficient methods for processing information presented as a discrete structure. For example, it can be combinatorial objects like combinations, permutations, partitions, graphs, trees, etc. [1].

In designing combinatorial algorithms, special attention is paid to the procedure for traversing all possible elements of a given combinatorial set. This problem can be studied as enumerating (counting the total number of elements), listing (making and recording the complete list of all elements), and generating (constructing all the required elements and their sequential visiting) combinatorial objects [2]. Computer programs that use combinatorial algorithms often need to present combinatorial objects one-by-one in the computer memory in the form of a certain data structure and process them. With this approach, there is no need to store the entire list of elements of combinatorial sets, which often have a huge cardinality. Hence, look at the generation of combinatorial objects in more detail since combinatorial generation algorithms are able to solve this problem [3].

In combinatorial generation, the following four problems are distinguished [4]:

1. Listing is the process of generating all elements of a combinatorial set sequentially.
2. Ranking is the process of getting the rank r of a combinatorial object a (the position of the object in their ordering).
3. Unranking is the inverse process of ranking, and unranking algorithms allow to generate the object a that has a rank r.

4. Random selection is the process of generating elements of a combinatorial set in a random order.

In this paper, we consider the combinatorial generation of m-combinations of an n-set ($0 \leq m \leq n$). An m-combination of an n-set is a subset of m distinct elements selected from a set of n elements. The total number of all m-combinations of an n-set can be calculated using the binomial coefficient $\binom{n}{m}$. There are two main ways to represent such combinations. One possible way of representing an m-combination of an n-set is the increasing sequence $c = (c_1, c_2, \ldots, c_m)$ of the selected elements in this combination, where $1 \leq c_1 < \cdots < c_m \leq n$. Another way of representing an m-combination of an n-set is the binary sequence $b = (b_1, b_2, \ldots, b_n)$, where $b_1 + b_2 + \ldots + b_n = m$, $b_i = 1$ encodes that the i-th element is selected and $b_i = 0$ encodes that the i-th element is not selected in this combination.

There are many different algorithms for generating all combinations with different orderings. Akl [5] conducted a computational experiment to compare the speed of such algorithms that were developed before 1980. During the experiment, the following nine algorithms for generating all combinations were implemented in Fortran: algorithms of Bitner et al., Chase, Ehrlich, Kurtzberg, Lehmer, Liu and Tang, Mifsud, Nijenhuis and Wilf, Payne and Ives. The results of the experiment showed that Mifsud's algorithm [6] was the fastest. The next stage in obtaining new algorithms for the fast generation of combinations is shown in the active use of parallel computing (see [7–16]). Such parallel algorithms use m processors and allow to produce combinations in constant time per combination. In addition to algorithms that provide fast generation of combinations, there are studies aimed at reducing the required memory space for calculations. For example, Itai [17] presented an algorithm to produce combinations in lexicographic order that required only $O(1)$ extra storage, but non-constant time.

There are also studies on the development of algorithms for generating all combinations that provide some useful properties due to applying a special ordering of the generated combinations. For example, there are algorithms that use a special ordering of the generated combinations in which the strong minimal change property holds (see algorithms of Eades and McKay [18], Xiang [19], Torres-Jimenez and Izquierdo-Marquez [20]). The main goal of these algorithms is to reduce the number of changes when generating combinations represented as increasing sequences of the selected elements in the combination and, as a result, to produce such combinations in constant time per combination or in constant space. Another interesting way to order the generated combinations was proposed by Ruskey and Williams [21]. This order called cool-lex order is a Gray code order that is based on the use of prefix shift operations.

However, there are situations in which it is necessary to generate a certain object again without generating all the objects that precede it. For this purpose, there are special combinatorial generation algorithms for ranking and unranking combinatorial objects. Such algorithms for ranking and unranking m-combinations of an n-set in lexicographic order were first presented by Knott [22]. Next, Er [23] proposed new ranking and unranking algorithms with $O(n - m)$ and $O(n)$ time complexity (without taking into account the time for calculating the binomial coefficients), respectively. These algorithms are shorter and simpler than Knott's corresponding algorithms due to representing combinations as binary sequences. The following algorithms for unranking m-combinations of an n-set were developed by Kokosinski [24]:

- UNRANKCOMB-A: decreasing lexicographical order, $O(n)$ time complexity and $O(nm)$ space complexity;
- UNRANKCOMB-B: increasing lexicographical order, $O(n)$ time complexity and $O(nm)$ space complexity;
- UNRANKCOMB-C: decreasing lexicographical order, $O(m \log n)$ time complexity and $O(nm)$ space complexity;
- UNRANKCOMB-D: decreasing lexicographical order, $O(n)$ time complexity and $O(m)$ space complexity.

Kokosinski [25] also presented the first parallel algorithm for unranking combinations. This algorithm called UNRANKCOMB-E requires n processors and has $O(m)$ time complexity. Additionally, this parallel algorithm has the preprocessing step for calculating and storing the binomial coefficients with $O(n)$ time complexity and $O(nm)$ space complexity.

There are also studies on the development of algorithms for ranking and unranking combinations by applying some general approach. For example, Ryabko [26] described methods for encoding and decoding combinatorial objects represented as sequences by calculating the number of all sequences prefixes. Applying this approach, new algorithms for ranking and unranking m-combinations of an n-set in lexicographical order were obtained. These algorithms use binary sequences to represent combinations and have $O(n \log^3 n \log \log n)$ time complexity and $O(n \log^2 n)$ space complexity. Shablya et al. [27] presented another general approach for developing new combinatorial generation algorithms. This method, which is based on the use of AND/OR trees for representing combinatorial sets, allows the development of new algorithms for listing, ranking and unranking combinations in co-lexicographic order. Such ranking and unranking algorithms also use binary sequences to represent m-combinations of an n-set and have $O(n + m^2)$ and $O(nm)$ time complexity, respectively. Genitrini and Pepin [28] proposed a way to improve ranking and unranking algorithms that are based on calculating binomial coefficients. The main idea is to calculate new binomial coefficients using the binomial coefficients obtained in the previous steps. The effectiveness of this approach was tested on several well-known algorithms for unranking combinations and a new algorithm for unranking combinations in lexicographic order, which is based on the factoradic numeral system.

In addition, there are studies dedicated to the special case of developing new efficient algorithms for ranking and unranking m-combinations of an n-set where the obtained algorithms are independent of n. Hence, such algorithms are effective when n is large and m is small. For example, Shimizu et al. [29] presented $O(m^{3m+3})$-time ranking and unranking algorithms. These algorithms use several bijections from the set of combinations to other finite sets. Moreover, the order of the generated combinations is not one of the well-known variants for ordering the elements of combinatorial sets.

Parque and Miyashita [30,31] introduced another approach for unranking small combinations of large sets by using a gradient-based optimization algorithm. The order of the generated m-combinations of an n-set is the revolving door order starting with $(1, 2, \ldots, m-1, m)$ and ending with $(1, 2, \ldots, m-1, n)$. To search for each element c_i of the combination $c = (c_1, c_2, \ldots, c_m)$, it is necessary to solve a constrained minimization problem by a gradient-based optimization. This unranking algorithm has $O(m^2)$ time complexity when using a single processor, and $O(m \log m)$ when using at most $O(m/\log m)$ processors. However, there is no information about any algorithm for ranking combinations in this order.

This paper focuses on unranking small m-combinations of a large n-set with using co-lexicographic order. The organization of this paper is as follows. Section 2 presents a brief description of a way for ranking and unranking combinations in co-lexicographic order and a detailed description of the proposed modification of the original unranking algorithm. In Section 3 we confirm the effectiveness of the proposed modification by performing several computational experiments and obtaining the computational complexity for the presented algorithm.

2. Materials and Methods

2.1. Algorithms for Ranking and Unranking Combinations in Co-Lexicographic Order

There are several variants for ordering any list of strings or sequences, for example, the well-known lexicographic and co-lexicographic orders [2]. Lexicographic (lex) order is a such order in which $a_1, a_2, \ldots, a_n < b_1, b_2, \ldots, b_m$ if $n = m$ and there exists $k \in \{1, \ldots, n\}$ satisfying $a_k < b_k$ and $a_i = b_i$ for every $i < k$ or if $n < m$ and $a_i = b_i$ for $i \leq n$. Co-lexicographic (colex) order is a such order in which $a_1, a_2, \ldots, a_n < b_1, b_2, \ldots, b_m$ if $a_n, \ldots, a_2, a_1 < b_m, \ldots, b_2, b_1$ in lex order.

Lex order is a natural and clear way of ordering. At the same time, colex order can make combinatorial algorithms shorter, faster, more elegant and natural [4]. For ranking a combination $c = (c_1, c_2, \ldots, c_m)$ in colex order, we can use the next formula (Equation (4.10) in [4]):

$$\texttt{RankColex}(c) = \sum_{i=1}^{m} \binom{c_i - 1}{i}. \tag{1}$$

Applying (1), it is possible to calculate the rank of a given m-combination of an n-set fast, even for large n since it is independent of n. Let us consider the following algorithm for unranking combinations in colex order (Algorithm 4.10 in [4]):

For a given m-combination of an n-set, Algorithm 1 takes the number m of the selected elements and the rank $r < \binom{n}{m}$ of the combination in colex order as input. As a result, the values of the selected elements c_i in the combination will be obtained in the form of a sequence $(c_1, c_2, \ldots, c_m)$. The main idea of this algorithm for unranking combinations in colex order is as follows. First, for the case $k = m$, we search for the interval that contains the rank r among the following binomial coefficients:

$$\binom{m}{m} + \binom{m+1}{m} + \binom{m+2}{m} + \ldots + \binom{n}{m} = \binom{n+1}{m+1}. \tag{2}$$

Algorithm 1: Algorithm for unranking combinations in colex order

```
   algorithm: UnrankColex(r,m)
   input: A nonnegative integer m and a rank r
   output: A sequence c = (c_1, c_2, ..., c_m)
1  begin
2      for i := m to 1 do
3          k := i
4          while r ≥ C(k, i) do k := k + 1
5          c_i := k
6          r := r − C(k−1, i)
7      end
8      return c
9  end
```

It is easy to see that the number of terms on the left in (2) is $n - m + 1$. If we know the rank r of an m-combination of an n-set and the solution for

$$\binom{k-1}{m} \leq r < \binom{k}{m}, \tag{3}$$

then the m-th selected element in the combination is equal to k. Next, we need to get the $(m-1)$-th selected element in the combination by searching for the interval that contains the rank

$$r := r - \binom{k-1}{m}. \tag{4}$$

Thus, solving (3) and applying (4), we can sequentially find the values of the selected elements of the m-combination with the rank r. For $m < n - m$, the computational complexity of Algorithm 1 is $O\big(m^2 \cdot (n-m)\big)$ which was obtained on the basis of the following information:

- the number of iterations in the for-loop (Line 2 in Algorithm 1) is m;
- the minimum number of comparisons in the while-loop (Line 4 in Algorithm 1) is $n - m + 1$ (when $i = m$) and the maximum number of comparisons is n (when $i = 1$);
- the computational complexity of calculating the value of a binomial coefficient $\binom{n}{m}$ is $O(m)$ when $m < n - m$ and $O(n - m)$ when $m > n - m$.

For large n and small m, the number of intervals in (3) that can contain the required rank r is large due to the dependence on n. The goal of our improvements of Algorithm 1 is to reduce the number of comparisons in the while-loop (Line 4 in Algorithm 1). For this purpose we need to find the solution of (3) fast.

2.2. Modification of Unranking Algorithm

Let us change Algorithm 1 by adding a function `FindK`(r,m) for preliminary search for the value of k. This function should calculate the approximate value of k for solving the inequality (3); however, the obtained value of k must not be greater than the true value of the solution of this inequality. Thus, we get an algorithm for unranking combinations in colex order (Algorithm 2).

Algorithm 2: Modification of algorithm for unranking combinations in colex order

```
   algorithm: UnrankColexNew(r,m)
   input: A nonnegative integer m and a rank r
   output: A sequence c = (c_1, c_2, ..., c_m)
1  begin
2      for i := m to 1 do
3          k := FindK(r,i)
4          while r ≥ C(k,i) do k := k + 1
5          c_i := k
6          r := r − C(k−1,i)
7      end
8      return c
9  end
```

Next, we consider the development of the function `FindK`(r,m) for Algorithm 2 in more detail.

A binomial coefficient can be presented as

$$\binom{n}{m} = \frac{n(n-1)(n-2)\cdots(n-m+1)}{m!}. \tag{5}$$

Using (3) and (5), we get

$$r < \frac{k(k-1)(k-2)\cdots(k-m+1)}{m!}.$$

After some transformations, we obtain the following inequality:

$$\sqrt[m]{k(k-1)(k-2)\cdots(k-m+1)} > \sqrt[m]{r\,m!}. \tag{6}$$

If we consider the inequality of arithmetic and geometric means [32] for the list of numbers $k, (k-1), (k-2), \ldots, (k-m+1)$, then we get

$$\frac{k+(k-1)+(k-2)+\ldots+(k-m+1)}{m} \geq \sqrt[m]{k(k-1)(k-2)\cdots(k-m+1)}.$$

After some simplifications on the left, we obtain

$$k - \frac{m-1}{2} \geq \sqrt[m]{k(k-1)(k-2)\cdots(k-m+1)}. \tag{7}$$

Combining (6) and (7), we get

$$k - \frac{m-1}{2} > \sqrt[m]{r\,m!}$$

or

$$k > \sqrt[m]{r\,m!} + \frac{m-1}{2}.$$

Hence,

$$k \approx \left\lceil \sqrt[m]{r\,m!} + \frac{m-1}{2} \right\rceil. \tag{8}$$

Thus, applying (8), we can find an approximate value of k for which the next inequality is true:

$$r < \binom{k}{m}. \tag{9}$$

We also use the following Stirling's approximation [33] for calculating the factorial in (8):

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\frac{1}{12n} - \frac{1}{360n^3}}. \tag{10}$$

Using the approximate Formula (10) in (8) and making some transformations, we obtain the following approximate formula for k:

$$k \approx \left\lceil m\, e^{\frac{\log r}{m} + \frac{\log 2\pi m}{2m} + \frac{1}{12m^2} - \frac{1}{360m^4} - 1} + \frac{m-1}{2} \right\rceil. \tag{11}$$

This approximate formula can be used in the preliminary search for the value of k. That is, the Formula (11) can be applied as the basis of the function `FindK`(r, m) (Line 3 in Algorithm 2).

3. Results

We have performed a computational experiment for checking the obtained Formula (11). For this purpose we set the number of selected elements m in the range of 10 to 200 and the rank r in the range of 10^{100} to 10^{200}. For these values of m and r, we have the distribution of values of n in the range of 332 (when $m = 170$ and $r = 10^{100}$) to $4.5 * 10^{20}$ (when $m = 10$ and $r = 10^{200}$). Then we calculate the difference between the value of k obtained by using the approximate Formula (11) and the value of k obtained by using the while-loop (Line 4 in Algorithm 1). The results of the computational experiment are presented in Table 1.

Table 1. Errors in determining the value of k depending on the values of m and r.

m \ r	10^{100}	10^{110}	10^{120}	10^{130}	10^{140}	10^{150}	10^{160}	10^{170}	10^{180}	10^{190}	10^{200}
10	0	0	0	0	0	0	0	0	0	1	1
20	0	0	0	0	0	0	0	0	0	1	1
30	0	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0	0
60	1	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0	0
80	1	0	0	0	0	1	0	0	0	0	0
90	1	1	0	1	0	0	0	1	0	0	0
100	1	1	1	1	1	1	0	0	0	0	0
110	2	1	1	0	1	1	1	0	0	0	0
120	2	1	1	1	1	1	0	0	1	0	1
130	3	2	2	1	1	1	1	0	0	1	0
140	3	3	2	1	1	1	1	1	1	1	0
150	4	3	3	3	2	1	2	2	1	1	1
160	4	4	3	2	2	2	2	1	1	1	1
170	5	4	4	4	3	2	3	2	2	1	2
180	5	5	5	4	3	3	2	2	2	2	2
190	7	6	5	5	3	3	3	3	2	2	2
200	7	6	6	5	5	4	3	3	3	2	2

From Table 1 we can see that the errors are small. Note that increasing r reduces the errors, but increasing m raises the errors. It is explained by the use of the approximate Formulas (8) and (10) for calculations. Moreover, we can not use the approximate Formula (11) for small m because the application of (10) to such values of m leads to an increase in errors. Therefore, for small m we can find the values of k by solving an equation obtained from (9):

If $m = 1$, then we get

$$r < \binom{k}{1}.$$

In this case, we can use the following initial condition for the value of k:

$$k = r.$$

If $m = 2$, then we get

$$r < \binom{k}{2}$$

or

$$k^2 - k - 2r > 0.$$

In this case, we can use the following initial condition for the value of k:

$$k = \left\lceil \frac{1 + \sqrt{1 + 8r}}{2} \right\rceil.$$

For other small values of m we can use other approximate formulas for solving (9), for example, it can be the approximate Formula (8). Hence, we obtain an algorithm for preliminary search for the values of k (Algorithm 3). In this algorithm, the parameter s shows the boundary value of m for applying the approximate Formula (11).

Algorithm 3: Preliminary search for the values of k for Algorithm 2

algorithm: `FindK(r,m)`
input: A nonnegative integer m and a rank r
output: A nonnegative integer k

1 **begin**
2 **if** $r = 0$ **then** $k := m$
3 **else if** $m = 1$ **then** $k := r$
4 **else if** $m = 2$ **then** $k := \left\lceil \frac{1+\sqrt{1+8r}}{2} \right\rceil$
5 **else if** $m < s$ **then** $k := k \approx \left\lceil \sqrt[m]{r\,m!} + \frac{m-1}{2} \right\rceil$
6 **else** $k := \left\lceil m\, e^{\frac{\log r}{m} + \frac{\log 2\pi m}{2m} + \frac{1}{12m^2} - \frac{1}{360m^4}} - 1 + \frac{m-1}{2} \right\rceil$
7 **return** k
8 **end**

Algorithm 2 with the use of Algorithm 3 has $O(m \cdot (m + \varepsilon m)) \approx O(m^2)$ time complexity (assuming algebraic operations with numbers in $O(1)$). This is determined by the number m of iterations in the for-loop (Line 2 in Algorithm 2) where each loop requires:

- preliminary search for the value of k (Line 3 in Algorithm 2) by using Algorithm 3 that has $O(m)$ time complexity;
- search for the true value of k by making ε iterations in the while-loop (Line 4 in Algorithm 2);
- calculation of the value of a binomial coefficient $\binom{n}{m}$ that has $O(m)$ time complexity when $m < n - m$.

The parameter ε is the error in determining the value of k that is substantially smaller than n (see Table 1). Hence, the modified Algorithm 2 has the polynomial complexity that depends only on m. In the case when n is large and m is small, this algorithm will be better than the original Algorithm 1 due to the independence of n in its complexity.

We have also performed a computational experiment aimed at comparing the obtained algorithm for unranking small combinations of a large set in colex order with Parque and Miyashita's unranking algorithm [30,31]. For this purpose we set the number of selected

elements m in the range of 10 to 100 and generate 20 different m-combinations of an n-set with uniformly distributed ranks r in the range of 0 to $\binom{n}{m} - 1$. We implemented these algorithms in the computer algebra systems Maxima on a laptop (Intel i7-9750H, 2.6 GHz, Windows 10, 64 bit). Figures 1 and 2 present average time to generate each combination for $n = 500$ and $n = 1000$. Note that both algorithms show similar results. However, there is a difference in the way the combinations are ordered and in the availability of the ranking algorithm.

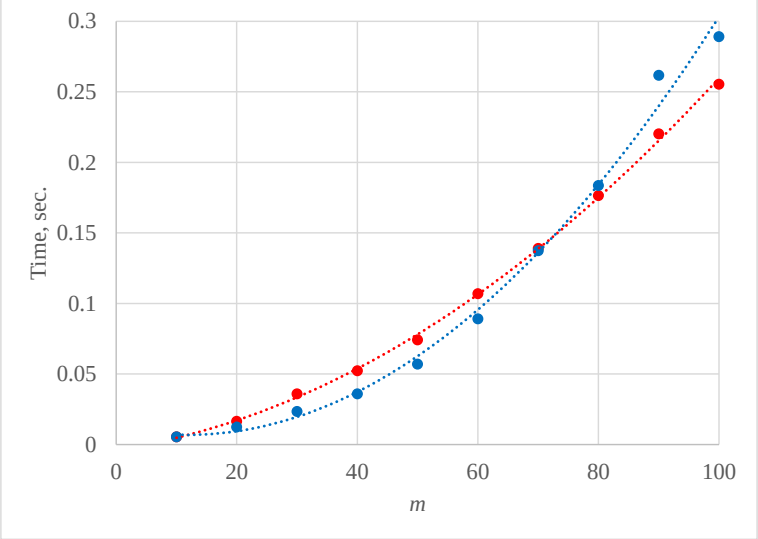

Figure 1. Average time to generate combinations for $n = 500$: (**red line**) Parque and Miyashita's unranking algorithm. (**blue line**) The obtained algorithm for unranking combinations.

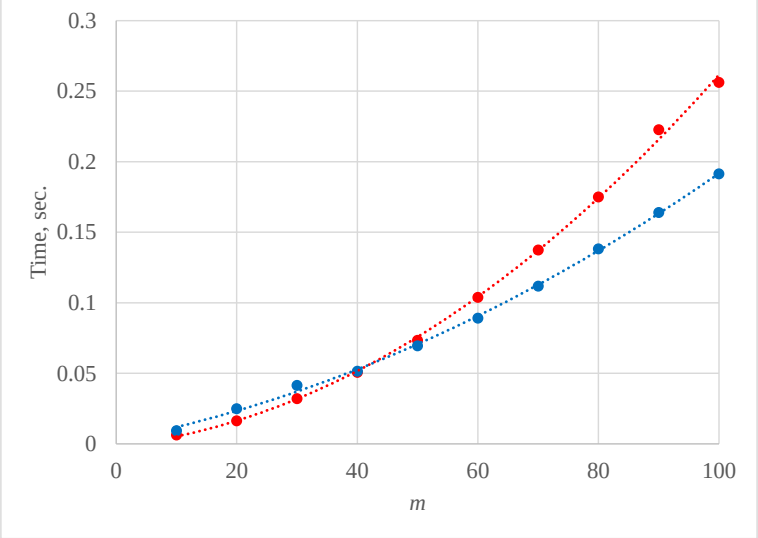

Figure 2. Average time to generate combinations for $n = 1000$: (**red line**) Parque and Miyashita's unranking algorithm. (**blue line**) The obtained algorithm for unranking combinations.

4. Conclusions

In this paper, we have presented a modification of the algorithm for unranking m-combinations of an n-set in colex order described in [4]. The computational complexity of the modified Algorithm 2 for unranking combinations in colex order with the use of Algorithm 3 for preliminary search for the values of k is $O(m \cdot (m + \varepsilon m)) \approx O(m^2)$ (assuming algebraic operations with numbers in $O(1)$).

In contrast to the original Algorithm 1, the obtained Algorithm 2 can be effectively applied when n is large and m is small because the running time of this algorithm depends

only on m. When n and m are both large but $n - m$ is small, then we can consider unranking $(n - m)$-combinations of an n-set. If we mark the selected elements in the $(n - m)$-combination of an n-set as the unselected elements in the m-combination of an n-set, then we obtain the original m-combination.

Author Contributions: Investigation, V.K., Y.S., D.K. and V.R.; methodology, D.K.; writing—original draft preparation, D.K. and Y.S.; and writing—review and editing, Y.S. and V.K. All authors have read and agreed to the published version of the manuscript.

Funding: The development of the combinatorial generation algorithm was funded by the Russian Science Foundation grant number 18-71-00059. The computational experiments were performed within the project FEWM-2020-0046.

Data Availability Statement: Data is contained within the article.

Acknowledgments: The authors would like to thank the referees for their helpful comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kreher, D.L.; Stinson, D.R. *Combinatorial Algorithms: Generation, Enumeration, and Search*; ACM: New York, NY, USA, 1999.
2. Knuth, D.E. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*; Addison-Wesley Professional: Boston, MA, USA, 2011.
3. Stojmenovic, I. *Handbook of Applied Algorithms: Solving Scientific, Engineering and Practical Problems*; Chapter Generating all and Random Instances of a Combinatorial Object; John Wiley and Sons, Inc.: Hoboken, NJ, USA, 2007; pp. 1–38. [CrossRef]
4. Ruskey, F. Combinatorial Generation. Working Version (1j-CSC 425/520). Available online: http://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf (accessed on 1 December 2021).
5. Akl, S.G. A comparison of combination generation methods. *ACM Trans. Math. Softw.* **1981**, *7*, 42–45. [CrossRef]
6. Mifsud, C.J. Algorithm 154: Combination in lexicographical order. *Comm. ACM* **1963**, *6*, 103. [CrossRef]
7. Chan, B.; Akl, S.G. Generating combinations in parallel. *BIT Numer. Math.* **1986**, *26*, 1–6. [CrossRef]
8. Chen, G.H.; Chern, M.S. Parallel generation of permutations and combinations. *BIT Numer. Math.* **1986**, *26*, 277–283. [CrossRef]
9. Akl, S.G.; Gries, D.; Stojmenovic, I. An optimal parallel algorithm for generating combinations. *Inform. Process. Lett.* **1989**, *33*, 135–139. [CrossRef]
10. Lin, C.-J. A parallel algorithm for generating combinations. *Comput. Math. Appl.* **1989**, *17*, 1523–1533. [CrossRef]
11. Tsay, J.C.; Lin, C.J. A systolic design for generating combinations in lexicographic order. *Parallel Comput.* **1990**, *13*, 119–125. [CrossRef]
12. Stojmenovic, I. A simple systolic algorithm for generating combinations in lexicographic order. *Comput. Math. Appl.* **1992**, *24*, 61–64. [CrossRef]
13. Elhage, H.; Stojmenovic, I. Systolic generation of combinations from arbitrary elements. *Parallel Process. Lett.* **1992**, *2*, 241–248. [CrossRef]
14. Kapralski, A. New methods for the generation of permutations, combinations, and other combinatorial objects in parallel. *J. Parallel Distrib. Comput.* **1993**, *17*, 315–326. [CrossRef]
15. Xu, C.W.; Ma, X.; Shiue, W.K. A new parallel combination generator. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Sunnyvale, CA, USA, 9–11 August 1996; pp. 25–28.
16. Kokosinski, Z. On parallel generation of combinations in associative processor architectures. In Proceedings of the IASTED International Conference on Parallel and Distributed Systems, Barcelona, Spain, 9–11 June 1997; pp. 283–289.
17. Itai, A. Generating permutations and combinations in lexicographical order. *J. Braz. Comput. Soc.* **2001**, *7*, 65–68. [CrossRef]
18. Eades, P.; McKay, B. An algorithm for generating subsets of fixed size with a strong minimal change property. *Inform. Process. Lett.* **1984**, *19*, 131–133. [CrossRef]
19. Xiang, L.; Ushijima, K. On $O(1)$ time algorithms for combinatorial generation. *Comput. J.* **2001**, *44*, 292–302. [CrossRef]
20. Torres-Jimenez, J.; Izquierdo-Marquez, I. A low spatial complexity algorithm to generate combinations with the strong minimal change property. *Discret. Math. Algorithms Appl.* **2019**, *11*, 1950060. [CrossRef]
21. Ruskey, F.; Williams, A. The coolest way to generate combinations. *Discret. Math.* **2009**, *309*, 5305–5320. [CrossRef]
22. Knott, G.D. A numbering system for combinations. *Comm. ACM* **1974**, *17*, 45–46. [CrossRef]
23. Er, M.C. Lexicographic ordering, ranking and unranking of combinations. *Int. J. Comput. Math.* **1985**, *17*, 277–283. [CrossRef]
24. Kokosinski, Z. Algorithms for unranking combinations and their applications. In Proceedings of the IASTED/ISMM International Conference on Parallel and Distributed Computing and Systems, Washington, DC, USA, 19–21 October 1995; pp. 216–224.
25. Kokosinski, Z. Unranking combinations in parallel. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Sunnyvale, CA, USA, 9–11 August 1996; pp. 79–82.

26. Ryabko, B.Y. Fast enumeration of combinatorial objects. *Discret. Math. Appl.* **1998**, *8*, 163–182. [CrossRef]
27. Shablya, Y.; Kruchinin, D.; Kruchinin, V. Method for developing combinatorial generation algorithms based on AND/OR trees and its application. *Mathematics* **2020**, *8*, 962. [CrossRef]
28. Genitrini, A.; Pepin, M. Lexicographic unranking of combinations revisited. *Algorithms* **2021**, *14*, 97. [CrossRef]
29. Shimizu, T.; Fukunaga, T.; Nagamochi, H. Unranking of small combinations from large sets. *J. Discret. Algorithms* **2014**, *29*, 8–20. [CrossRef]
30. Parque, V.; Miyashita, T. Towards the succinct representation of m out of n. In Proceedings of the Internet and Distributed Computing Systems, Tokyo, Japan, 11–13 October 2018; pp. 16–26. [CrossRef]
31. Parque, V.; Miyashita, T. Unranking combinations using gradient-based optimization. In Proceedings of the International Conference on Tools with Artificial Intelligence, Volos, Greece, 5–7 November 2018; pp. 579–586. [CrossRef]
32. Roberts, A.W.; Varberg, D.E. *Convex Functions*; Academic Press: New York, NY, USA, 1973.
33. Arfken, G.B.; Weber, H.J.; Harris, F.E. *Mathematical Methods for Physicists*; Elsevier Academic Press: Burlington, MA, USA, 2012. [CrossRef]