Educational Technologies
Образователни технологии

ON SOME RANDOMIZED ALGORITHMS AND THEIR EVALUATION

Krasimir Yordzhev
South-West University – Blagoevgrad (Bulgaria)

Abstract. The paper considers implementations of some randomized algorithms in connection with a random $n^2 \times n^2$ Sudoku matrix with the programming language C++. For this purpose we describe the set Π_n of all $(2n) \times n$ matrices, consisting of elements of the set $\mathbb{Z}_n = \{1,2,\ldots,n\}$, such that every row is a permutation. We emphasize the relationship betweenΠ_n-matrices and the $n^2 \times n^2$ Sudoku matrices. An algorithm to obtain random Π_n matrices is presented. Several auxiliary algorithms that are related to the underlying problem have been described. We evaluated all algorithms according to two criteria - probability evaluation, and time for the generation of random objects and checking a belonging to a specific set. These evaluations are interesting from both theoretical and practical points of view because they are particularly useful in the analysis of computer programs.

Keywords: randomized algorithm; random object; permutation; binary matrix; algorithm evaluation; Sudoku matrix

1. Introduction

This article is intended for anyone who studies programming, as well as for teachers. The work is a continuation and addition of (Yordzhev, 2012). Here, along with the basic definitions and ideas for constructing randomized algorithms outlined in the cited publication, we will also describe specific implementations of these algorithms in the C++ programming language.

To demonstrate the ideas outlined in the article, we have chosen the C ++ programming language (Todorova, 2002), (Todorova, 2011a), (Todorova, 2011b), but they can be implemented in any other algorithmic language. We hope that students who prefer to write in Java (Hadzhikolev & Hadzhikoleva, 2016) or any other modern programming language will have no problems with the implementation of the algorithms we have proposed.

The presented in the article source codes have been repeatedly tested with various input data and they work correctly.

Let $\mathfrak{M}$ be a finite set. *A Random objects generator of* $\mathfrak{M}$ is every algorithm $\mathcal{A}_{\mathfrak{M}}$ randomly generating any element of $\mathfrak{M}$, while elements generated by a random objects generator will be called *random elements* of $\mathfrak{M}$, for example random num-

bers, random matrices, random permutations, etc. We take for granted that probabilities to obtain different random elements of $\mathfrak{M}$ by means of $\mathcal{A}_{\mathfrak{M}}$ are equal, and are also equal to $\frac{1}{|\mathfrak{M}|}$. We denote the time that the random objects generator needs to obtain a random element of $\mathfrak{M}$ with $T(\mathcal{A}_{\mathfrak{M}})$.

By *randomized algorithm* we will mean any algorithm which essentially uses a random object generator in its work.

The randomized algorithms are very often used to solve problems, which are proved to be NP-complete. For detailed information about NP-complete problems and their application see (Garey & Jonson, 1979) or (Hopcroft et al., 2001). A proof that a popular Sudoku puzzle is NP-complete is given in (Yato, 2003) and (Yato & Seta, 2003).

In this study, we will solve some particular cases from the following class of problems:

Let n and $m = m(n)$ be natural numbers. Let us consider the set $\mathcal{U}$, consisting of objects every of which dependent on m parameters, and every parameter belongs to the finite set $\mathfrak{M}$. We assume that there is a rule that uniquely describes object $u \in \mathcal{U}$, if all m parameters are specified. Let $\mathcal{V} \subset \mathcal{U}$. The problem is to obtain (at least one) object, which belongs to the set $\mathcal{V}$. The number of the elements of the sets $\mathcal{U}$ and $\mathcal{V}$ depends only on the parameter m, which is an integer function of the argument n.

The standard algorithm that solves the above problem is briefly described as follows:

Algorithm 1

1) *We obtain consequently* $m = m(n)$ *random elements of* $\mathfrak{M}$ *using random objects generator* $\mathcal{A}_{\mathfrak{M}}$ *and so we get the object* $u \in \mathcal{U}$*;*

2) *We check if* $u \in \mathcal{V}$*. If the answer is no, everything is repeated.*

In other words, if we already have a random objects generator, a randomized algorithm can be used as a generator of more complex random objects. The efficiency of Algorithm 1 depends on the particular case in which it is used and can be evaluated according to the following criteria:

Probability evaluation: If $p(n)$ denotes the probability after generating $m = m(n)$ random elements of $\mathfrak{M}$ of obtaining an object of $\mathcal{V}$, then according to the classical probability formula:

$$p(n) = \frac{|\mathcal{V}|}{|\mathcal{U}|} \tag{1}$$

Time for generating and checking: We denote by $\tau(n)$ the time needed to execute one iteration (repetition) of Algorithm 1. Then

$$\tau(n) = m(n)T(\mathcal{A}_{\mathfrak{M}}) + \theta(n), \tag{2}$$

where $\theta(n)$ is the time to examine if the obtained object belongs to the set $\mathcal{V}$.

It is obvious that the efficiency of Algorithm 1 will be proportional to $p(n)$ and inversely proportional to $\tau(n)$.

Of course, time for generating and checking does not give us the total time to execute the algorithm, since the number of repetitions is not known in advance. However, the characteristic $\tau(n)$ is essential to the effectiveness of any randomized algorithms.

The cases in which probability evaluation is equal to 1, i.e. the cases in which the algorithm is constructed directly to obtain element of the set $\mathcal{V}$ and there is no need of belonging examination, are of great interest, as only one iteration is implemented then, i.e. there is no repetition. Let n be a positive integer. We denote by $\mathbb{Z}_n$ the set of the integers

$$\mathbb{Z}_n = \{1,2,\dots,n\}.$$

There are standard procedures for obtaining random numbers of the set $\mathbb{Z}_n$ in most of the programming environments. We take this statement for granted and we will use it in our examinations. Let $\mathcal{A}_n$ be a similar procedure. In the current study, we will consider that for $n \neq l$

$$T(\mathcal{A}_n) \approx T(\mathcal{A}_l) = t_0 = Const. \tag{3}$$

Below we show an example of a C ++ function that generates a random positive integer belonging to the set $\mathbb{Z}_n = \{1,2,\dots,n\}$:

Algorithm 2.

```
int rand_Zn(int n)
{
    return rand() % n + 1;
}
```

In order for the function rand_Zn(int) to work so that every time we execute the program in which we will use it to obtain various random numbers, we must add the procedure

```
srand(time(0));
```

before first accessing this function, for example, at the beginning of the main() function. The functions rand() and srand(s) are from the library <cstdlib>, and the function time(t) is from the library <ctime>. For more details, see for example (Azalov & Zlatarova, 2011, p. 75).

Let P_{ij}, $0 \leq i,j \leq n-1$ be n^2 in number square $n \times n$ matrices, whose elements belong to the set $\mathbb{Z}_{n^2} = \{1,2,\dots,n^2\}$. Then $n^2 \times n^2$ matrix

$$P = [P_{ij}] = \begin{bmatrix} P_{0\,0} & P_{0\,1} & \cdots & P_{0\,n-1} \\ P_{1\,0} & P_{1\,1} & \cdots & P_{1\,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ P_{n-1\,0} & P_{n-1\,1} & \cdots & P_{n-1\,n-1} \end{bmatrix}$$

is called a *Sudoku matrix*, if every row, every column and every submatrix P_{ij}, $0 \leq i, j \leq n-1$ make *permutation* of the elements of set $\mathbb{Z}_{n^2}$, i.e. every integer $s \in \{1,2, \ldots, n^2\}$ is present only once in every row, every column and every submatrix P_{ij}. Submatrices P_{ij} are called *blocks* of P.

In this paper we will illustrate the above mentioned ideas by analyzing some randomized algorithms for obtaining an arbitrary permutation of n elements, an arbitrary $n^2 \times n^2$ Sudoku matrix and an arbitrary $(2n) \times n$ matrix with $2n$ rows and n columns, every column of which is a permutation of n elements.

We will prove that the problem for obtaining ordered n^2 - tuple of $(2n) \times n$ matrices, every row of which is a permutation of elements of $\mathbb{Z}_n$ is equivalent to the problem of generating a Sudoku matrix. We will analyze some possible algorithms for generating a random Sudoku matrix.

How to create computer program for Sudoku solving (a mathematical model of the algorithm), using the concept set combined with the trial and error method is described in (Yordzhev, 2018).

2. Random permutations

Let n be an positive integer. We denote by $\mathcal{S}_n$ the set of all *permutations* $\langle a_0, a_1, \ldots, a_{n-1} \rangle$, where $a_i \in \mathbb{Z}_n$ and $a_i \neq a_j$ when $i \neq j$, $0 \leq i, j \leq n-1$.

If $\sigma = \langle a_0, a_2, \ldots, a_{n-1} \rangle \in \mathcal{S}_n$ is a permutation of all elements of the set $\mathbb{Z}_n = \{1,2, \ldots, n\}$ then obviously σ depends of $m(n) = n$ parameters a_0, a_1, $\ldots$, a_{n-1}.

As it is well known, the number of all n-tuples of integers $\langle x_1, x_2, \ldots, x_n \rangle$, $x_i \in \mathbb{Z}_n$ is equal to

$$|\underbrace{\mathbb{Z}_n \times \mathbb{Z}_n \times \cdots \times \mathbb{Z}_n}_{n}| = n^n \tag{4}$$

and the number of all permutations of *n* elements is equal to

$$|\mathcal{S}_n| = n! = 1 \cdot 2 \cdot 3 \cdots n. \tag{5}$$

We denote by $p_1(n)$ the probability to obtain a random permutation of $\mathcal{S}_n$ with the help of Algorithm 1. Then according to equations (1), (4) and (5) we obtain:

$$p_1(n) = \frac{n!}{n^n} \tag{6}$$

The next algorithm works in time $O(n)$ and checks if ordered n-tuple $\langle a_0, a_1, \ldots a_{n-1} \rangle$, $a_i \in \mathbb{Z}_n$, $i = 0,1, \ldots, n-1$ is a permutation.

Algorithm 3. *Check if given integer array a[n] is a permutation of the elements from the set* $\mathbb{Z}_n = \{1,2,\ldots,n\}$.

```
bool alg3(int a[], int n)
{
    int v[n+1];
    for (int i=1; i<=n; i++) v[i] = 0;
    for (int i=0; i<n; i++)
    {
if ( (a[i] > n) || (a[i] < 1) ) return false;  /* Incorrect data - there is an element that does not belong to the set Zn.*/
    v[a[i]]++;
          if (v[a[i]] > 1)  return false; /* Since the number a[i] occurs more than once in the n-tuple. */
    }
    return true;
}
```

Let $\tau_1(n)$ denote the time for implementing one iteration of Algorithm 1 when a random permutation of elements of $\mathbb{Z}_n$ is obtained, and let $\theta_1(n)$ denote the time for checking whether an arbitrary n-tuples of numbers of $\mathbb{Z}_n$ belongs to $\mathcal{S}_n$. Analyzing Algorithm 3, it is easy to see that

$$\theta_1(n) \in O(n). \tag{7}$$

Then, having in mind the equations (2), (3) and (7) we obtain the following time for generating and checking:

$$\tau_1(n) = nT(\mathcal{A}_n) + \theta_1(n) \in nt_0 + O(n) = O(n). \tag{8}$$

The following algorithm is also randomized (random integers are generated), but its probability evaluation is equal to 1, i.e. in Algorithm 1 step 2 is not implemented, because when the first random n numbers are generated the obtained ordered n-tuple is a permutation.

Algorithm 4л *Obtaining random permutation* $\sigma = \langle a_0, a_1, \ldots, a_{n-1}\rangle \in \mathcal{S}_n$, $\sigma = \langle a_0, a_1, \ldots, a_{n-1}\rangle \in \mathcal{S}_n$, *where* $a_i \in \mathbb{Z}_n$, $i = 1,2,\ldots,n$, $a_i \neq a_j$ *when* $i \neq j$.

```
void alg4(int a[], int n)
{
    int v[n];
    for (int i=0; i<n; i++) v[i] = i+1;
    int r;
    for (int i=0; i<n; i++)
    {
```

```
            r = rand_Zn(n-i)-1;
            a[i] = v[r];   /* We remove the element v[x] and reduce the number
of the elements of the array with 1. */
   for (int j=r; j<=n-i; j++) v[j] = v[j+1];
        }
   }
```

Analyzing the work of Algorithm 4, we see that in the end, n different elements of the set $\mathbb{Z}_n = \{1,2,\dots,n\}$ are filled in the array a[n]. We consistently randomly take these elements from the array v with length n where all integers $1,2,\dots,n$ are filled, such that v[i] = i+1, $i = 0,1,\dots,n-1$. After each choice we remove the selected integer from the array v, i.e. from the integers which we will randomly take in the next steps of the algorithm. Therefore Algorithm 4 which obtains random permutation has probability evaluation:

$$p_2(n) = 1 \tag{9}$$

In Algorithm 4, there are two nested loops, so that the outer loop is repeated exactly n times. Internal loop in the worst case (if randomly selected integer equal to the value of the item v[0]) will be repeated as many times as is the length of the array v, which in the first iteration of the outer loop is equal to n and decreased each time by 1. Thus, we obtain the following time evaluation for generating and checking in one iteration of Algorithm 4.

$$\tau_2(n) \in t_0[O(n) + O(n-1) + \cdots + O(1)] = O(n^2). \tag{10}$$

We see that Algorithm 1 applied for obtaining random permutations is more efficient than Algorithm 4 in terms of the time for generating and checking. But on the other hand, the probability evaluation of Algorithm 4 is equal to $p_2(n) = 1$. The probability evaluation of Algorithm 1 is equal to $p_1(n) = \frac{n!}{n^n} < 1$ for $n \geq 2$. Considering that the number of iterations is previously unknown and $\lim_{n\to\infty} p_1(n) = 0$, then this makes Algorithm 4 overall much more effective than the Algorithm 1 when applied to generate random permutations.

3. Random (2n) X n matrices, every row of which is a permutation of elements of $\mathbb{Z}_n$

Let Π_n denote the set of all $(2n) \times n$ matrices, which are also called Π_n matrices, in which every row is a permutation of all elements of $\mathbb{Z}_n$. In this case $\mathfrak{M} = \mathbb{Z}_n$ and $m(n) = 2n^2$. It is obvious that

$$|\Pi_n| = (n!)^{2n} \tag{11}$$

It is easy to see that when we obtain random Π_n matrix with the help of Algorithm 1 the following evaluations can be observed:

Probability evaluation:

$$p_3(n) = \frac{|\mathcal{V}|}{|\mathcal{U}|} = \frac{(n!)^{2n}}{n^{2n^2}} \tag{12}$$

Time for generating and checking:

$$\tau_3(n) = m(n)T(\mathcal{A}_n) + 2n\tau_1(n) \in 2n^2 t_0 + 2nO(n) = O(n^2), \tag{13}$$

where $\tau_1(n)$ is obtained according to equation (8).

The next randomized algorithm will be more efficient than Algorithm 1 in obtaining random Π_n matrix according to the probability evaluation.

Algorithm 5. *Obtaining a random matrix* $M = [\pi_{ij}]_{2n\times n} \in \Pi_n$ *with probabilitv evaluation equal to 1. The algorithm will write the elements of the matrix* $M = [\pi_{ij}]_{2n\times n}$ *in the array* p[] *with size* $2n^2$*, such that* π_{ij} =p[i*n+j]*, where* $0 \leq i < 2n$, $0 \leq j < n$.

```
void alg5(int p[], int n)
{
    int m=2*n;
    int a[n];
    for (int i=0; i<m; i++)
    {
        alg4(a,n);
        for (int j=0; j<n; j++)
        {
            p[i*n+j] = a[j];
        }
    }
}
```

Practically, Algorithm 5 repeats $2n$ times Algorithm 4. As Algorithm 4 has a probability evaluation equal to 1, then Algorithm 5 which obtains random Π_n matrix also has probability evaluation

$$p_4(n) = 1p_4(n) = 1 \tag{14}$$

It is easy to see that Algorithm 5 works in time for generating and checking

$$\tau_4(n) = 2n\tau_3(n) \in 2nO(n^2) = O(n^3). \tag{15}$$

As we can see below Π_n matrices can successfully be used to create algorithms that are efficient in developing Sudoku matrices.

4. S-permutation matrices

A *binary* (or *boolean*, or *(0,1)-matrix*) is called a matrix whose elements belong to the set $\mathfrak{B} = \{0,1\}$. With $\mathfrak{B}_{n\times m}$ we will denote the set of all $n \times m$ binary matrices. The set of all square $n \times n$ binary matrices we will denote with $\mathfrak{B}_n = \mathfrak{B}_{n\times n}$

Two binary matrices $A = [a_{ij}]_{n\times m} \in \mathfrak{B}_{n\times m}$ and $B = [b_{ij}]_{n\times m} \in \mathfrak{B}_{n\times m}$ will be called *disjoint* if there are not elements $a_{ij} \in A$ and $b_{ij} \in B$ with one and the same indices i and j, such that $a_{ii} = b_{ii} = 1$, i.e. if $a_{ij} = 1$ then $b_{ij} = 0$ and if $b_{ij} = 1$ then $a_{ij} = 0$, $1 \le i \le n$, $1 \le j \le m$.

A square binary matrix $A \in \mathfrak{B}_n$ is called *permutation*, if there is only one 1 in every row and every column of the matrix A.

Let Σ_{n^2} denote the set of all permutation $n^2 \times n^2$ matrices of the following type

$$A = \begin{bmatrix} A_{0\,0} & A_{0\,1} & \cdots & A_{0\,n-1} \\ A_{1\,0} & A_{1\,2} & \cdots & A_{1\,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n-1\,0} & A_{n-1\,1} & \cdots & A_{n-1\,n-1} \end{bmatrix}, \tag{16}$$

where for every $s, t \in \{0,1,2,\dots,n-1\} A_{st}$ is a square $n \times n$ binary submatrix (*block*) with only one element equal to 1, and the rest of the elements are equal to 0. The elements of Σ_{n^2} will be called *S-permutation*.

Geir Dahl introduces the concept of S-permutation matrix [Dahl, 2009] in relation to the popular *Sudoku puzzle* giving the following obvious proposition:

Proposition 4.1. (Dahl, 2009) *A square $n^2 \times n^2$ matrix P with elements of $\mathbb{Z}_{n^2} = \{1,2,\dots,n^2\}$ is Sudoku matrix if and only if there are mutually disjoint matrices $A_1, A_2, \dots, A_{n^2} \in \Sigma_{n^2}$ such that P can be presented as follows:*

$$P = 1 \cdot A_1 + 2 \cdot A_2 + \cdots + n^2 \cdot A_{n^2}$$

■

As it is proved in [Dahl, 2009] and with other methods in (Yordzhev, 2013)

$$|\Sigma_{n^2}| = (n!)^{2n} \tag{17}$$

Therefore, if a random Σ_{n^2} matrix obtained by means of Algorithm 1 the following probability evaluation is valid:

$$p_5(n) = \frac{(n!)^{2n}}{2^{(n^2)^2}} = \frac{(n!)^{2n}}{2^{n^4}} \tag{18}$$

In order to obtain a random Σ_{n^2} matrix, we have to generate $m(n) = (n^2)^2 = n^4$ random integers, which belong to the set $\mathfrak{B} = \{0,1\}$. Hence, whatever randomized algorithm is used, the result is the following time for generating and checking:

$$\tau_5(n) = m(n)T(\mathcal{A}_2) + \theta_5(n) = n^4 t_0 + \theta_5(n) \in O(n^z), \quad z \ge 4, \tag{19}$$

where $\theta_5(n)$ is time for checking that the given binary matrix belongs to the set Σ_{n^2}.

Below we will give an algorithm (Algorithm 6), which checks that the given binary matrix is Σ_{n^2} and works in time $O(n^4)$, i.e. $\theta_5 = O(n^4)$ and therefore $z = 4$.

Let $A_{st} = [(a_{st})_{kl}]_{n\times n}$ be an arbitrary $n \times n$ matrix, $0 \leq k, l, s, t < n$. Let $A = [\alpha_{ij}]_{n^2\times n^2}$, $0 \leq i, j < n^2$ be an $n^2 \times n^2$ matrix such that for every i, j, k, l, s, t, $0 \leq i, j < n^2$ and $0 \leq k, l, s, t < n$, the equality

$$\alpha_{ij} = (a_{st})_{kl}$$

is satisfied. Then, taking into account that the numbering of indexes starts from 0, it is easily seen that

$$i = sn + k \tag{20}$$

and

$$j = tn + l. \tag{21}$$

Algorithm 6. *Checking that a binary* $n^2 \times n^2$ *matrix* $B = [\beta_{ij}] \in \Sigma_{n^2}$. *The algorithm gets the elements of the matrix* $B = [\beta_{ij}]$ *from the array* a[] *with size* n^4, *such that* β_{ij} =a[i*n2+j], *where* $n2 = n^2$, $0 \leq i < n^2 - 1$, $0 \leq j < n^2 - 1$.

For ease, we will not check for correctness of the input data, i.e. whether the input matrix is binary, in other words whether all its elements are 0 or 1. We leave this check to the reader for an exercise.

```
bool alg6(int a[], int n)
{
    int n2 = n*n;
    int r;
    int i,j,k,l,s,t;
    for (i=0; i<n2; i++)
    {
        r=0;
        for (j=0; j<n2; j++)
        {
            r = r+a[i*n2+j];
           if(r>1) return false; /* There are more than one
1 in the i-th row. */
}
          if(r==0) return false;/* i-th row is completely
zero.*/
}
    for (j=0; j<n2; j++)
    {
        r=0;
        for (i=0; i<n2; i++)
{
            r = r+a[i*n2+j];
           if(r>1) return false; /* There are more than one
1 in the i-th column. */
}
```

```
            if(r==0) return false;/* i-th column is completely
zero.*/
   }
        for (s=0; s<n; s++)
        {
            for (t=0; t<n; t++)
            {
                r=0;
                for (k=0; k<n; k++)
                {
                    for (l=0; l<n; l++)
                    {
                        i = s*n+k;/* According to equation (20) */
   j = t*n+l; /* According to equation (21)*/
   r= r+a[i*n2+j];
                    }
                }
                if(r!=1) return false;
            }
        }
        return true;
   }
```

When we compare (18) with (12) and (14), as well as (19) with (13) and (15) we may assume that algorithms which use random Π_n matrices are expected to be more efficient (regard to probability and time for generating and checking) than algorithms using random Σ_{n^2} matrices to solve similar problems. This gives grounds for further examination of the Π_n matrices' properties.

5. Random Sudoku matrices

We will give a little bit more complex definition of the term "disjoint" regarding Π_n matrices. Let $C = [c_{ij}]_{2n\times n}$ and $D = [d_{ij}]_{2n\times n}$ be two Π_n matrices. We regard C and D as *disjoint* matrices, if there are no natural numbers $s, t \in \{1,2,\dots n\}$ such that the ordered pair $\langle c_{st}, c_{n+ts}\rangle$ is equal to the ordered pair $\langle d_{st}, d_{n+ts}\rangle$.

The relationship between Π_n matrices and Sudoku matrices is illustrated by the following theorem, considering Proposition 4.1.

Theorem 5.1. *There is a bijective map from Π_n to Σ_{n^2} and the pair of disjoint matrices of Π_n corresponds to the pair of disjoint matrices of Σ_{n^2}*

Proof. Let $P = [\pi_{ij}]_{2n\times n} \in \Pi_n$. We obtain an unique matrix of Σ_{n^2} from P by means of the following algorithm:

Algorithm 7.

From a given matrix $M = [\pi_{ij}]_{2n\times n} \in \Pi_n$ we obtain a unique matrix $A = [\alpha_{ij}]_{n^2\times n^2} \in \Sigma_{n^2}$. The algorithm gets the elements of the matrix $M = [\pi_{ij}]_{2n\times n}$ from the array p[] with size $2n^2$, such that p[i*n+j]= π_{ij}, $0 \leq i < 2n$, $0 \leq j < n$ and obtained using Algorithm 5. The algorithm will write the elements of the matrix $A = [\alpha_{ij}]_{n^2\times n^2}$ in the array a[] with size n^4, such that α_{ij} =a[i*n*n+j], $0 \leq i, j < n^2$.

```
void alg7(int p[], int a[], int n)
{
    int i,j,s,t;
    int n2 = n*n;
    int n4 = n2*n2;
    for (i=0; i<n4; i++) a[i] = 0;

    int b[2][n][n];

    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
          b[0][i][j] = p[i*n+j];/*  (22)  */

    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        b[1][j][i] = p[n2+i*n+j];/*  (23)  */

    for (s=0; s<n; s++)
        for (t=0; t<n; t++)
        {
            i = s*n+b[0][s][t]-1; /* According to equation (20) */
j = t*n+b[1][s][t]-1;/*  According to equation (21) */
a[i*n2+j] = 1;
}

}
```

For convenience, at the beginning of Algorithm 7, we constructed an auxiliary $2 \times n \times n$ three-dimensional array b such that,according to equation (22)

$$b[0][i][j] = \pi_{ij} \tag{24}$$

and according to equation (23)

$$b[1][j][i] = \pi_{n+ij} \tag{25}$$

is satisfied for each $i, j \in \{0,1, \ldots, n-1\}$.

For example, from the matrix

$$P = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \in \Pi_3, \tag{26}$$

we get the array b. For more clearly, to the left of the comma we will write the elements of b[0], and to the right of the comma – the elements of b[1]:

$$\mathrm{b} = \begin{bmatrix} 1,3 & 3,1 & 2,1 \\ 2,1 & 3,2 & 1,2 \\ 1,2 & 2,3 & 3,3 \end{bmatrix} \tag{27}$$

From the array b, Algorithm 7 obtains a binary matrix $A = [\alpha_{ij}]_{n^2 \times n^2}$, $0 \leq i, j < n^2$ of type (16) such that if $A_{st} = [\gamma_{ij}]_{n \times n}$, $0 \leq s, t < n$ is a block in A, then

$$\gamma_{ij} = \begin{cases} i = s * n + b[0][s][t] - 1 \\ j = t * n + b[1][s][t] - 1 \end{cases} \tag{28}$$

From (24), (25) and (28) follows that there is a single 1 in each block $A_{st} \in A$.

Let $s \in \{0,1, \ldots, n-1\}$. Since s-th row of the matrix P is a permutation, then there is only one 1 in every row of $n \times n^2$ matrix

$$R_s = [A_{s0} \quad A_{s1} \quad \cdots \quad A_{sn-1}].$$

Similarly, since ordered n-tuple $\langle p_{n+t\ 1}, p_{n+t\ 2}, \ldots, p_{n+tn} \rangle$ which is $(n+t)$-th row of P is a permutation for every $t \in \mathbb{Z}_n$, then there is only one 1 in every column of $n^2 \times n$ matrix

$$C_t = \begin{bmatrix} A_{1t} \\ A_{2t} \\ \vdots \\ A_{nt} \end{bmatrix}.$$

Therefore, the matrix A which is obtained with the help of Algorithm 7 is a Σ_{n^2} matrix.

For example, from the array B, which corresponds to the array (27) and using Algorithm 7, we obtain the Σ_9-matrix

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Since a unique matrix of Σ_{n^2} is obtained for every $P \in \Pi_n$ by means of Algorithm 7, then this algorithm provides a description of the map $\varphi : \Pi_n \to \Sigma_{n^2}$. It is easy to see that if there are given different elements of Π_n, we can use Algorithm 7 to obtain different elements of Σ_{n^2}. Hence, φ is an injection. But according to formulas (11) and (17) $|\Sigma_{n^2}| = |\Pi_n|$, whereby it follows that φ is a bijection.

Analyzing Algorithm 7, we arrived at the conclusion that P and Q are disjoint matrices of Π_n if and only if $\varphi(P)$ and $\varphi(Q)$ are disjoint matrices of Σ_{n^2} according to the above mentioned definitions. The theorem is proved. ■

Let $\mathcal{M}_{n^2}$ be the set of all square $n^2 \times n^2$ matrices with elements of the set $\mathbb{Z}_{n^2} = \{1,2, ..., n^2\}$, and let σ_n be the number of all $n^2 \times n^2$ Sudoku matrices. Obviously, $|\mathcal{M}_{n^2}| = (n^2)^{n^2} = n^{2n^2}$. Then if we use Algorithm 1 to obtain random Sudoku matrix. According to equation (1) there is the following probability evaluation:

$$p_6(n) = \frac{\sigma_n}{n^{2n^2}} \tag{29}$$

When n = 2, $\sigma_2 = 288$ [Yordzhev, 2013].

When $n = 3$, there are exactly

$$\sigma_3 = 6\ 670\ 903\ 752\ 021\ 072\ 936\ 960 \approx 6.671 \times 10^{21}$$

in number Sudoku matrices [Felgenhauer and Jarvis, 2006; Yordzhev, 2018]. As far as the author of this study knows, there is not a universal formula for the number σ_n of Sudoku matrices with every natural number n. We consider it as an open problem in mathematics.

If we employ random methods to create the matrix $P \in \mathcal{M}_{n^2}$ with elements of $\mathbb{Z}_{n^2}$, then according to Algorithm 1 we need to verify if every row, every column and every block of P is a permutation of elements of $\mathbb{Z}_{n^2}$. According to evaluation of Algorithm 3, every verification can be done in time $O(n)$ (see equation (8)). Hence, when we employ Algorithm 1 to obtain a random Sudoku matrix we will obtain the following time for generating and checking:

$$\tau_6(n) \in T(\mathcal{A}_n)(n^2)^2 + 2nO(n) + n^2O(n) = t_0n^4 + 2nO(n) + n^2O(n) \in O(n^4). \tag{30}$$

Here we will present a more efficient algorithm for obtaining random Sudoku matrix, based on the propositions and algorithms which are examined in the previous sections of this paper. The main point is to obtain n^2 in number random Π_n matrices (Algorithm 5). For every Π_n matrix which is obtained, it has to be checked if it is disjoint with each of the above obtained matrices. The criteria, described in Theorem 5.1 and Proposition 4.1 are used in the verification. If the obtained matrix is not disjoint with at least one of the above

mentioned matrices, it has to be replaced with another randomly generated Π_n matrix.

Algorithm 8. *Obtaining random Sudoku matrix*

```
\include <iostream>
#include <cstdlib>
#include <ctime>
#define N 3
#define N2 9   // N2 = N*N
#define N4 81  // N4 = N2*N2
#define d 18   // d = 2*N2
using namespace std;
int main()
{
    srand(time(0));
    int S[N4];
    int P[d];
    int A[N4];
    int z;
    for (int i=0; i<N4; i++) S[i] = 0;
    int K=1;
    while (K <= N2)
    {
        alg5(P,N);
        alg7(P,A,N);
        z=0;
        for (int i=0; i<N4; i++)
        {
z += A[i]*S[i];
            if (z != 0) break;
        }
        if (z==0)
        {
            for (int i=0; i<N4; i++) S[i] += K*A[i];
            K++;
        }
    }

    for (int i=0; i<N4; i++)
    {
        cout<<S[i]<<" ";
        if ((i+1)%N2 == 0) cout<<endl;
    }
    return 0;
}
```

With the help of Algorithm 8, we received a lot of random 9×9 random Sudoku matrices, for example the next one:

$$\begin{bmatrix} 6 & 4 & 2 & 3 & 1 & 7 & 8 & 9 & 5 \\ 5 & 3 & 1 & 8 & 2 & 9 & 4 & 7 & 6 \\ 7 & 8 & 9 & 4 & 5 & 6 & 2 & 3 & 1 \\ 9 & 6 & 7 & 2 & 4 & 5 & 1 & 8 & 3 \\ 3 & 2 & 4 & 6 & 8 & 1 & 9 & 5 & 7 \\ 1 & 5 & 8 & 9 & 7 & 3 & 6 & 2 & 4 \\ 8 & 9 & 5 & 1 & 3 & 4 & 7 & 6 & 2 \\ 2 & 1 & 3 & 7 & 6 & 8 & 5 & 4 & 9 \\ 4 & 7 & 6 & 5 & 9 & 2 & 3 & 1 & 8 \end{bmatrix}$$

REFERENCES

Azalov, P. & Zlatarova, F. (2011). *C++ v primeri, zadachi i prilozheniya*. Sofia: Prosveta.

Dahl, G. (2009). Permutation matrices related to sudoku. *Linear Algebra and its Applications*, 430 (8-9): 2457 – 2463.

Felgenhauer, B.& Jarvis, F. (2006). Mathematics of sudoku. *Mathematical Spectrum*, 39 (1): 15 – 22.

Garey, M. R. & Jonson, D. S. (1979). *Computers and Intractability.* A *Guide to the Theory of* NP-*Completeness*. Bell Telephone Laboratories.

Hadzhikolev, E. & Hadzhikoleva, S. (2016). *Osnovi na programiraneto s Java*. Plovdiv: Paisiy Hilendarski (ISBN 978-619-202-108-5).

Hopcroft, J. E., Motwani, R. & Ullman, J. D. (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Todorova, M. (2002). *Programirane na* C++, volume I and II. Sofia: Ciela (ISBN 954-649-454-2(1), 954-649-480-1(2)).

Todorova, M. (2011a). *Obektho-orientirano programirane na bazata na ezika* C++. Sofia: Ciela (ISBN 978-954-28-0909-8).

Todorova, M. (2011b). *Strukturi ot danni i programirane na* C++ Sofia: Ciela (ISBN 978-954-28-0909-6).

Yato, T. (2003). *Complexity and Completeness of Finding Another Solution and Its Application to Puzzles – Masters thesis*. Univ. of Tokyo, Dept. of Information Science.

Yato, T. & Seta, T. (2003). Complexity and completeness of finding another solution and its application to puzzles. *IEICE Trans. Fundamentals*, E86-A(5): 1052 – 1060.

Yordzhev, K. (2012). Random permutations, random sudoku matrices and randomized algorithms. *International Journal of Mathematical Sciences and Engineering Applications*, 6 (VI): 291 – 302.
Yordzhev, K. (2013). On the number of disjoint pairs of s-permutation matrices. *Discrete Applied Mathematics*, 161 (18): 3072 – 3079.
Yordzhev, K. (2018). How does the computer solve sudoku – a mathematical model of the algorithm. *Mathematics and Informatics*, 61 (3): 259 – 264.

✉ **Assoc. Prof. Krasimir Yordzhev, DSc.**
Researcher ID: F-2628-2014
ORCID: 0000-0002-0432-8025
Department of Informatics
Faculty of Natural Sciences
South-West University "Neofit Rilski"
Blagoevgrad, Bulgaria
E-mail: yordzhev@swu.bg