Wearable Technology (2023) Volume 4 Issue 1
doi: 10.54517/wt.v4i1.2681

ORIGINAL RESEARCH ARTICLE

Mathematical aspects of digital technologies: The problem of the absent-minded passenger

Andrey Shcherbakov[1,2]

[1] *Department of Cognitive Analytical and Neuro-Applied Technologies, Russian State Social University, 129226 Moscow, Russian Federation; x509@ras.ru*

[2] *State University of Management, 109542 Moscow, Russian Federation*

ABSTRACT

The example of the absent-minded passenger problem illustrates the fundamental difficulty of estimating practical values of probabilistic parameters on the basis of general theorizing and allows us to give more accurate and subtle estimates of various parameters, as well as their practical computable approximation. This article is important for building systems of artificial intelligence and artificial consciousness, as well as for modern didactics of higher education, especially for the disciplines of "applied mathematics" and "theoretical computer science". It is shown that a detailed study of a random process provides more significant applied materials than formally correct theoretical calculations.

Keywords: logistics; artificial consciousness; Monte Carlo methods (MCm); cryptography; probability; digital technologies

1. Informal introduction

In the second half of the 20th century, the problem of the "crazy old lady" appeared and became well-known, although in rather narrow circles of fans of mathematical problems.

Its wording is as follows. The plane is boarding at the airport. Exactly 100 passengers lined up in the queue. The crazy old lady stands first (in formal problem statements, the "crazy old lady" is replaced by an "absent-minded passenger" or "scientist"). After entering the salon, she sits down in any randomly selected seat that she liked (perhaps on her own). Each subsequent passenger, entering the cabin, sits down in his (indicated on the ticket) seat, if it is free, and on any of the free ones otherwise. For false simplicity, it can be assumed that the passenger sits in the first of the available seats.

What is the probability that the last passenger in the queue will sit in his seat (indicated on his ticket)?

A more difficult question is how many passengers are not in their seats on average?

The problem of an absent–minded passenger is a good way to think in general about the global issue of the interaction of information technology with mathematics, and mathematics with practice, its solution can be useful for transport and warehouse logistics, help a mathematics teacher expand the horizons of his students with new mathematical concepts, throw bridges not only to building the foundations of the

ARTICLE INFO
Received: 15 April 2024 | Accepted: 6 May 2024 | Available online: 29 May 2024

CITATION
Shcherbakov A. Mathematical aspects of digital technologies: The problem of the absent-minded passenger. *Wearable Technology* 2023; 4(1): 2681. doi: 10.54517/wt.v4i1.2681

methodology of artificial intelligence thinking, but also even the basics of cryptography.

2. Formal defenition and solution of the problem of an absent-minded passenger

Setting of the problem:

In the study[1], the condition and solution of the problem are formulated. There are n seats on the bus, and all tickets have been sold to n passengers. An Absent—minded Scientist (hereinafter referred to as an absent-minded passenger) gets on the bus first and, without looking at the ticket, takes the first available seat. Then the passengers enter one at a time. If the newcomer sees that his seat is empty, he takes his seat. If the seat is occupied, then the person who enters takes the first available seat. Find the probability that the passenger who entered last will take a seat according to his ticket?

2.1. Solution

Let's number all the passengers, starting with the absent-minded passenger, in the order in which they boarded the bus. The last passenger has the number n. For simplicity, we will number the seats in the same way. Let everyone except the last passenger have already entered and taken their seats. There is one free seat left. If it was the second seat, then the second passenger (or the absent-minded passenger) I would have occupied it already. The same is true for seats numbered 3, 4, 5, …, $n - 1$. This means that this seat belongs either to the last passenger or to the absent-minded passenger.

It is clear that it can belong to both the first and the last with equal chances. In the first case, the last passenger will not sit in his seat, and in the second—in his own. This means that the probabilities of both events are equal to ½.

2.2. Basic concepts and procedures for solving the problem of an absent-minded passenger

Consider the concept of substitution[2] of degree or order n — this is the mapping of a vector (0, 1, ..., $n - 1$) to a vector (p_0, p_1, ..., p_{n-1}), where p_i belongs to the set of integers from 0 to $n - 1$ and p_i are not repeated. It is easy to see that this mapping is bijective (unambiguous).

It is by such a construction (substitution) that it is advisable to represent the queue of passengers. The only clarification is to abandon the 0 and zero elements and consider vectors with elements from 1 to n. The number n is called the degree of substitution.

Let's try to generate a code to generate random substitutions (Algorithm 1).

Algorithm 1 GenPermit Random Substitution generation procedure

```
#define N_PRM 250
// Errors
// -1 - the size of the substitution is exceeded
// -2 - the number of attempts is exceeded
// -3 - error during generation (the sum of elements does not match)
int GenPermit(int n, unsigned char *prm)
    {
     int i,j,k,sum1,sum2,tg,c;
     unsigned char p;
     if(n>=N_PRM) return(-1);
     sum1=0;
     for(i=0;i<n;i++) sum1=sum1+i;
     NextRandom16(rnd,rnd_1,rnd);
     prm[0]=rnd[0]%n;
     c=0;
     for(i=1;i<n;i++)
```

Algorithm 1 (*Continued*)

```
        {
         m:;
         c++;
         if(c>N_PRM*20) return(-2);
         NextRandom16(rnd,rnd_1,rnd);
         p=rnd[0]%n;
         tg=0;
         for(j=0;j<i;j++) if(p==prm[j]) tg=1;
 // There is no generated random number among the previous elements
         if(tg==0) prm[i]=p;
         else goto m;
        }
     sum2=0;
     for(i=0;i<n;i++) sum2=sum2+prm[i];
 // Checking the checksum
     if(sum1!=sum2) return(-3);
     for(i=0;i<n;i++) prm[i]=prm[i]+1;

     return(0);
    }
```

The GenPermit substitution generation procedure contains the input parameter *n*—the degree of the generated substitution and the output—an array of numbers from 0 to 255, limited in length by the value N_PRM, which is set to 250.

The procedure uses unsigned integers from 0 to 255 (one byte long). When analyzing a for queues exceeding 256, it is necessary to use arrays of integers.

This procedure uses the procedure for generating a random vector with a length of 16 bytes NextRandom16(rnd,rnd_1,rnd), we will talk about it below.

For our purposes, we will select the zero element of a random rnd array and to obtain elements from 0 to n, we will reduce each byte modulo n and use division with remainder for this.

Mainly for educational purposes, we generate the first substitution element separately in the specified way (we get a zero random byte from a random number generator and bring it modulo n).

Next, we need to get another $n - 1$ substitution element in such a way that the elements are not repeated, which is what two cycles in the main body of the procedure are devoted to. It is easy to see that the procedure uses an undesirable goto construct, but it is secured by a cycle counter for obtaining the next substitution element.

Let's pay attention to a moment that is fundamentally important for conducting such experiments—it is ensuring the correctness and reliability of calculations.

First of all, it is the control of the size of the substitution: the construction

```
if(n>=N_PRM) return(-1);
```

Next, protection against looping the procedure

```
if(c>N_PRM*10) return(-2);
```

And checking for the correct generation of the substitution

```
sum2=0;
    for(i=0;i<n;i++) sum2=sum2+prm[i];
    if(sum1!=sum2) return(-3);
```

The sum1 value is calculated in advance:

```
sum1=0;
    for(i=0;i<n;i++) sum1=sum1+i;
```

However, you can also use the arithmetic progression sum formula instead of performing an additional loop.

Suppose, after performing the described procedure, a substitution of degree 20 is obtained (a queue of 20 passengers):

001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020
008 016 014 003 009 019 013 007 015 005 001 011 012 006 010 017 018 020 002 004

The first passenger must take the seat no.8, the second—the seat no.16, the twentieth—the seat no.4.

Now let's consider a procedure that simulates boarding with the first absent-minded passenger (Algorithm 2).

Algorithm 2 Boarding simulation procedure

```
int boarding(int n, unsigned char *qu,int echo)
    {
     unsigned char mm[N_PRM], mm1[N_PRM], pl[N_PRM],p;
     int i,j,k,c,tg;
     for(i=0;i<n;i++) {mm[i]=0;mm1[i]=1;}
// the list of seats is a reverse substitution
     GenPermit_1(n,qu,pl);
     NextRandom16(rnd,rnd_1,rnd);
     p=rnd[0]%n;
// The first absend-minded passenger
     mm[0]=p+1;
     mm1[p]=0;
     if(echo==1)
       {
        printf("Pass %03d Tic %03d Seat %03d\n",1,qu[0],p+1);
        getch();
       }
    if(echo==2) AppLogD1(LOGNAME,n,mm);
// Boarding of other passengers, starting from the second
     for(i=1;i<n;i++)
        {
         tg=0;
         for(j=0;j<i;j++)
            {
             if(mm[j]!=qu[i]) continue;
             else {tg=1;break;}
            }
// If the seat is free
          if(tg==0)
            {
             mm[i]=qu[i];
             mm1[mm[i]-1]=0;
             if(echo==1){
             printf("Pass %03d Tic %03d Seat %03d - Normal\n",i+1,qu[i],mm[i]);
             for(k=0;k<n;k++) printf("%03d ",mm[k]);
             printf("\n");
             getch();}
            }
// If the seat is occupied
        else
           {
            for(j=0;j<n;j++) if(mm1[j]==1) {mm[i]=j+1;mm1[j]=0;break;}
             if(echo==1){
             printf("Pass %03d Tic %03d Seat %03d - Wrong\n",i+1,qu[i],mm[i]);
             for(k=0;k<n;k++) printf("%03d ",mm[k]);
```

Algorithm 2 (*Continued*)

```
                printf("\n");
                getch();}
              }
            if(echo==2) AppLogD1(LOGNAME,n,mm);
          }
  // Seating control
      c=0;
      for(i=0;i<n;i++) c=c+(mm[i]-1);
      tg=0;
      for(i=0;i<n;i++) tg=tg+i;

  // How many are not in their seat
      c=0;tg=0;
      for(i=0;i<n;i++) if(qu[i]!=mm[i]) c++;

      if(qu[n-1]!=mm[n-1]) tg=1;

      if(tg==1) return(-c);
      else     return(c);
     }
```

Let's demonstrate how the procedure works on test data.

So, the first absent-minded passenger took the seat no.19 instead of his seat no.8.

The 6th passenger in the queue should take the seat no.19.

1) 019 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
2) 019 016 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
3) 019 016 014 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
4) 019 016 014 003 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
5) 019 016 014 003 009 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000

When boarding the 6th passenger, he takes a free seat instead of his occupied one and now the passenger who should have been sitting in the seat no.1 (11th in line) will also change seats.

6) 019 016 014 003 009 001 000 000 000 000 000 000 000 000 000 000 000 000 000 000
7) 019 016 014 003 009 001 013 000 000 000 000 000 000 000 000 000 000 000 000 000
8) 019 016 014 003 009 001 013 007 000 000 000 000 000 000 000 000 000 000 000 000
9) 019 016 014 003 009 001 013 007 015 000 000 000 000 000 000 000 000 000 000 000
10) 019 016 014 003 009 001 013 007 015 005 000 000 000 000 000 000 000 000 000 000

The 11th passenger takes the second vacant seat, and the 19th passenger who should be sitting in the 2nd will have to change seats.

11) 019 016 014 003 009 001 013 007 015 005 002 000 000 000 000 000 000 000 000 000
12) 019 016 014 003 009 001 013 007 015 005 002 011 000 000 000 000 000 000 000 000
13) 019 016 014 003 009 001 013 007 015 005 002 011 012 000 000 000 000 000 000 000
14) 019 016 014 003 009 001 013 007 015 005 002 011 012 006 000 000 000 000 000 000
15) 019 016 014 003 009 001 013 007 015 005 002 011 012 006 010 000 000 000 000 000
16) 019 016 014 003 009 001 013 007 015 005 002 011 012 006 010 017 000 000 000 000
17) 019 016 014 003 009 001 013 007 015 005 002 011 012 006 010 017 018 000 000 000
18) 019 016 014 003 009 001 013 007 015 005 002 011 012 006 010 017 018 020 000 000

The 19th passenger sits in the seat no.4, and the twentieth moves to the seat no.8.

19) 019 016 014 003 009 001 013 007 015 005 002 011 012 006 010 017 018 020 004 000
20) 019 016 014 003 009 001 013 007 015 005 002 011 012 006 010 017 018 020 004 008

The last passenger is not sitting in his seat.

Now we can use the Monte Carlo method to estimate the number of passengers not in their seats and the statistical probability for the last passenger of not taking his seat.

The Monte Carlo methods (MCm)[3] are a group of numerical methods for studying various kinds of random processes. The essence of the methods is as follows: the process is described by a mathematical model (in our case, the generation of random substitutions of degree *n*, which sets the length of the queue and the number of seats) using a random variable generator, the model is repeatedly implemented on given data sets (sometimes the term model is “calculated” is used), based on the data obtained, the probabilistic characteristics of the process under consideration are calculated (Algorithm 3).

Algorithm 3 Estimation of the number of passengers not sitting in their seats

```
for(i=1;i<11;i++)
    {
     pr_n=i*10;
     sum1=0;sum2=0;
     for(j=0;j<100;j++)
       {
        GenPermit(pr_n,prm1);
        rez=boarding(pr_n,prm1,0);
        sum1=sum1+abs(rez);
       if(rez<0) sum2++;
       }
     printf("\nN= %03d Med1 =%f       Med2=%f\n",pr_n,(float)(sum1/100.),(float)(sum2/100.));
     AppLogD2(LOGNAME,pr_n,sum1,sum2);
    }
```

In this fragment, the passenger seating procedure from fragment 2 is called sequentially a hundred times and the degree of substitution changes from 10 to 100 in increments of ten.

We get the following results.

The first experiment:

```
N= 010 Med[pass] =2.680000 P[last] = 0.510000
N= 020 Med[pass] =3.710000 P[last] = 0.530000
N= 030 Med[pass] =4.150000 P[last] = 0.470000
N= 040 Med[pass] =4.340000 P[last] = 0.410000
N= 050 Med[pass] =4.520000 P[last] = 0.500000
N= 060 Med[pass] =4.880000 P[last] = 0.400000
N= 070 Med[pass] =4.460000 P[last] = 0.280000
N= 080 Med[pass] =5.050000 P[last] = 0.360000
N= 090 Med[pass] =5.050000 P[last] = 0.320000
N= 100 Med[pass] =5.370000 P[last] = 0.300000
```

The second experiment:

```
N= 010 Med[pass] =2.910000 P[last] = 0.470000
N= 020 Med[pass] =3.630000 P[last] = 0.530000
N= 030 Med[pass] =3.630000 P[last] = 0.460000
N= 040 Med[pass] =4.300000 P[last] = 0.410000
N= 050 Med[pass] =4.070000 P[last] = 0.440000
```

```
N= 060 Med[pass] =4.620000 P[last] = 0.420000
N= 070 Med[pass] =5.020000 P[last] = 0.300000
N= 080 Med[pass] =4.970000 P[last] = 0.410000
N= 090 Med[pass] =5.160000 P[last] = 0.340000
N= 100 Med[pass] =5.070000 P[last] = 0.340000
```

You can see that the results are quite stable—for ten passengers, the probability that the last one will not be in his seat is about 0.5 and decreases with an increase in the number of passengers to about 0.3.

The number of passengers not sitting in their seats increases from 2–3 (the number of passengers, of course, an integer) for a queue of 10 passengers to 5 for a queue of 100 passengers.

As it is easy to see, a formally correct mathematical solution does not withstand a collision with reality very well.

Another small remark: the boarding simulation procedure returns the number of passengers who are not sitting in their seats, and if the last passenger is not in his seat, then the procedure returns a negative value.

3. Approaches to the approximation of the obtained results

Now let's try to study the properties of sequences of changes in the number of passengers who are not sitting in their seats, and the probability that the last passenger was not in his seat.

We use the approximation function by a polynomial of the second degree. After the approximation, we can simply use the calculations according to the formula, without making time-consuming iterative calculations (Algorithm 4).

Algorithm 4 Calculation of the coefficients of a polynomial of the second degree based on its three arguments and values

```
int a3(float *x, float *y, float *abc)
{
 float m,r,n,z,t,v;
 if(x[1]==0.) return(-1);
 m=x[2]-(x[2]*x[2])/x[1];
 r=y[2]-y[1]*(x[2]*x[2])/(x[1]*x[1]);
 n=1-(x[2]*x[2])/(x[1]*x[1]);
 z=x[3]-(x[3]*x[3])/x[1];
 t=1-(x[3]*x[3])/(x[1]*x[1]);
 v=y[3]-y[1]*(x[3]*x[3])/(x[1]*x[1]);
 if(m==0.) return(-2);
 if((m*t-z*n)==0.) return(-3);
 abc[2]=(m*v-r*z)/(m*t-z*n);
 abc[1]=(r-n*abc[2])/m;
 abc[0]=(y[1]-x[1]*abc[1]-abc[2])/(x[1]*x[1]);
 return(0);
 }
```

As it is easy to see, the approximation is reduced to solving a system of three linear equations with three unknowns, and in the variable abc we obtain, respectively, the coefficients of the approximating polynomial (Algorithm 5).

Algorithm 5 Calculating the value of a polynomial of the second degree

```
float p3(float x, float *abc)
{
 float r;
 r=abc[0]*x*x+abc[1]*x+abc[2];
 return(r);
}
```

Let’s calculate the MCm value for *n* = 200, approximate it for values 20, 100 and 200 (Algorithm 6).

Algorithm 6 Approximation

```
pr_n=200;
        for(j=0;j<100;j++)
            {

             GenPermit(pr_n,prm1);
             rez=boarding(pr_n,prm1,0);
             sum1=sum1+abs(rez);
            if(rez<0) sum2++;
            }
         printf("\nN= %03d Med1 =%f Med2 = %f\n",pr_n,(float)(sum1/100.),(float)(sum2/100.));
         AppLogD2(LOGNAME,pr_n,sum1,sum2);
         ss[200]=(float)(sum1/100.);pp[200]=(float)(sum2/100.);
      xx[1]=20.;
      xx[2]=100.;
      xx[3]=200.;
      yy[1]=ss[20];
      yy[2]=ss[100];
      yy[3]=(float)(sum1/100.);
      r=a3(xx,yy,ab);
      AppLogD3(LOGNAME,ab[0],ab[1],ab[2]);
      for(i=1;i<21;i++)
        AppLogD4(LOGNAME,ss[i*10],p3((float)(i*10),ab),ss[i*10]-p3((float)(i*10),ab));
      xx[1]=20.;
      xx[2]=100.;
      xx[3]=200.;
      yy[1]=pp[20];
      yy[2]=pp[100];
      yy[3]=(float)(sum2/100.);
      r=a3(xx,yy,ab);
      AppLogD3(LOGNAME,ab[0],ab[1],ab[2]);
      for(i=1;i<21;i++)
      AppLogD4(LOGNAME,pp[i*10],p3((float)(i*10),ab),pp[i*10]-p3((float)(i*10),ab));
```

We get the following result

```
N= 010 Med[pass] =2.820000 P[last] = 0.420000
N= 020 Med[pass] =3.490000 P[last] = 0.510000
N= 030 Med[pass] =3.960000 P[last] = 0.470000
N= 040 Med[pass] =4.050000 P[last] = 0.410000
N= 050 Med[pass] =4.700000 P[last] = 0.480000
N= 060 Med[pass] =4.570000 P[last] = 0.340000
N= 070 Med[pass] =4.460000 P[last] = 0.300000
N= 080 Med[pass] =4.800000 P[last] = 0.440000
N= 090 Med[pass] =5.260000 P[last] = 0.360000
N= 100 Med[pass] =4.970000 P[last] = 0.350000
N= 110 Med[pass] =5.520000 P[last] = 0.360000
N= 120 Med[pass] =5.560000 P[last] = 0.420000
N= 200 Med[pass] =11.300000 P[last] = 0.690000
```

A very interesting result: the hypothesis that the probability decreases with increasing N turns out to be incorrect in practice.

The number of passengers not in their seats is determined by the values of the following approximating polynomial:

$\text{Med[pass]} = 0.000249N^2 + -0.011367N + 3.617776$

The first is the argument N, the second value is calculated by MCm, the third is the value of the approximating polynomial, the fourth is the difference between them.

```
10 2.820000 3.528999 −0.708999
20 3.490000 3.490000 0.000000
30 3.960000 3.500778 0.459222
40 4.050000 3.561335 0.488666
50 4.700000 3.671668 1.028332
60 4.570000 3.831780 0.738220
70 4.460000 4.041669 0.418331
80 4.800000 4.301335 0.498665
90 5.260000 4.610780 0.649220
100 4.970000 4.970002 −0.000002
110 5.520000 5.379001 0.140999
120 5.560000 5.837779 −0.277779
130 0.000000 6.346333 −6.346333
140 0.000000 6.904665 −6.904665
150 0.000000 7.512776 −7.512776
160 0.000000 8.170664 −8.170664
170 0.000000 8.878328 −8.878328
180 0.000000 9.635772 −9.635772
190 0.000000 10.442992 −10.442992
200 11.300000 11.299991 0.0000
```

Approximation of the probability that the last passenger is not in his seat. The column legend is similar to the one above.

$P[last] = 0.000030N^2 + -0.005600N + 0.610000$

```
10 0.420000 0.557000 −0.137000
20 0.510000 0.510000 0.000000
30 0.470000 0.469000 0.001000
40 0.410000 0.434000 −0.024000
50 0.480000 0.405000 0.075000
60 0.340000 0.382000 −0.042000
70 0.300000 0.365000 −0.065000
80 0.440000 0.354000 0.086000
90 0.360000 0.349000 0.011000
100 0.350000 0.350000 −0.000000
110 0.360000 0.357000 0.003000
120 0.420000 0.370000 0.050000
130 0.000000 0.389000 −0.389000
140 0.000000 0.414001 −0.414001
150 0.000000 0.445001 −0.445001
160 0.000000 0.482001 −0.482001
170 0.000000 0.525001 −0.525001
180 0.000000 0.574001 −0.574001
190 0.000000 0.629001 −0.629001
```

200 0.690000 0.690001 −0.000001

It is easy to see that in both cases, the values for arguments 20, 100 and 200 do not differ from the obtained MCm’s, since they are used as base points for approximation.

Algebraic objects related to the problem of the absend-minded passenger.

So, the queue of passengers is described by substitution, and the seating of passengers is described by reverse substitution (Algorithm 7).

Algorithm 7 Getting the reverse substitution

```
int GenPermit_1(int n, unsigned char *params, unsigned char *prm_1)
{
int i;
    if(n>=N_PRM) return(-1);
    for(i=0;i<n;i++)prm_1[prm[i]-1]=i+1;
    return(0);
    }
```

Let the direct substitution of degree 20 be as follows:

002 014 004 015 012 007 017 005 003 019 010 009 020 011 018 001 013 016 006 008

Then the reverse substitution is as follows:

016 001 009 003 008 019 006 020 012 011 014 005 017 002 004 018 007 015 010 013

The 16th passenger took the seat no.1, the 1st passenger—the seat no.2, etc.

If the queue and seating substitutions are multiplied by the following procedure, then a single substitution is obtained, that is, a statement in which *pi* = *i* (Algorithm 8).

Algorithm 8 Multiplication of substitutions

```
int MulPermit( int n, unsigned char *prm1, unsigned char *prm2, unsigned char *prm_r)
   {
    int i;
    if(n>=N_PRM) return(-1);
    for(i=0;i<n;i++) prm_r[i]=prm2[prm1[i]-1];
    return(0);
   }
```

Substitutions of degree n form a noncommutative group with the multiplication operation described by fragment 5 of order (order is the number of elements in the group) n! (n factorial). It is precisely because of the large variety of different queues that it is advisable to use MCm.

4. Some thoughts on the implementation in artificial consciousness systems

Currently, the theory and practice of artificial intelligence is at a certain “crossroads”.

Some authors suggest continuing to build simulation models based on neural networks and their training, while others move on to more conceptual objects with complex architecture such as artificial consciousness (AC)[4].

In particular, to solve the described problems, the mechanism of artificial consciousness associated with the formulation, solution and setting of problems seems appropriate and promising.

The problem of an absent-minded passenger could be a kind of Turing test for artificial consciousness. It contains important elements of cognitive logic that make AC’s work constructive:

- methods for generating and representing fundamental mathematical objects (in this case, substitutions);
- methods of implementing operations with them, both basic (multiplication and reversal of substitutions) and expanded (imitation of boarding passengers or placing goods in a queue, taking into account warehouse logistics errors caused, as a rule, by the human factor, for example, by theft);
- Monte Carlo methods for solving numerical problems and obtaining data for applied approximation;
- methods of internal and external control of the obtained results (checking substitutions, protection against looping, exceeding the boundary values of parameters).

The combination of these mechanisms makes AC creative enough to solve a very wide range of practical problems and, in addition, to integrate into educational processes in the role of “interlocutor”, “opponent” and, possibly, “mentor”.

The considered problem is also important for modern didactics, especially for the disciplines of “applied mathematics” and “theoretical computer science”.

A rather important and new fact is that a detailed study of a random process using programming and modeling methods gives more significant applied results than formally correct theoretical calculations.

Conflict of interest

The author declares no conflict of interest.

References

1. MCCME project with the participation of school 57, Task 65310. Available online: https://problems.ru/view_problem_details_new.php?id=65310 (accessed on 12 February 2024).
2. Substitutions. Available online: http://www.algebraical.info/doku.php?id=glossary:group:permutation (accessed on 12 February 2024).
3. The Monte Carlo method. Available online: https://ru.wikipedia.org/wiki/Method_Monte Carlo (accessed 12 February 2024).
4. Shcherbakov A, Uryadov A. Philosophical and technical view of artificial consciousness. Wearable Technology. 2023; 4(1). doi: 10.54517/wt.v4i1.2498