UDC 519.854.2

PERMANENT DECOMPOSITION ALGORITHM FOR THE COMBINATORIAL OBJECTS GENERATION

Turbal Y. V. – Dr. Sc., Professor of Computer Science and Applied Mathematics Department, National University of Water and Environmental Engineering, Rivne, Ukraine.

Babych S. V. – Programming Department, College of National University of Life and Environmental Sciences of Ukraine, Rivne, Ukraine.

Kunanets N. E. – Dr. Sc., Professor of Department of Information Systems and Networks, National University "Lviv Polytechnic", Lviv, Ukraine.

ABSTRACT

Context. The problem of generating vectors consisting of different representatives of a given set of sets is considered. Such problems arise, in particular, in scheduling theory, when scheduling appointments. A special case of this problem is the problem of generating permutations.

Objective. Problem is considered from the point of view of a permanent approach and a well-known one, based on the concept of lexicographic order.

Method. In many tasks, it becomes necessary to generate various combinatorial objects: permutations, combinations with and without repetitions, various subsets. In this paper we consider a new approach to the combinatorial objects generation, which is based on the procedure of the permanent decomposition. Permanent is built for the special matrix of incidence. The main idea of this approach is including to the process of the algebraic permanent decomposition by row additional function for the column identifiers writing into corresponding data structures. In this case, the algebraic permanent in not calculated, but we get a specific recursive algorithm for generating a combinatorial object. The computational complexity of this algorithm is analyzed.

Results. It is investigated a new approach to the generation of complex combinatorial objects, based on the procedure of decomposition of the modified permanent of the incidence matrix by line with memorization of index elements.

Conclusions. The permanent algorithms of the combinatorial objects generation is investigated. The complexity of our approach in the case of permutation is compared with the lexicographic algorithm and the Johnson-Trotter algorithm.

The obtained results showed that our algorithm belongs to the same complexity class as the lexicographic algorithm and the Johnson-Trotter method. Numerical results confirmed the effectiveness of our approach.

KEYWORDS: algorithm, permutation, permanent, decomposition, complexity.

ABBREVIATIONS

JSP is a job-shop problem;

NP-Complete is a nondeterministic polynomial-time complete;

PD-algorithm is a permanent decomposition algorithm;

PD-approach is a permanent decomposition approach;

SDR is a system of different representatives.

NOMENCLATURE

i, j are indices of vectors and matrix elements;

a_j is element of the sets;

S_i is set of the elements;

n_{ij} is the number of occurrences of the element a_j in the set S_i;

R_i is a row of the schedule matrix;

permod is modified permanent;

n is size of the array ;

$Q(n)$ is computational complexity;

O() is complexity class;

$(v_1, v_2, ..., v_m)$ is SDR;

v_{ij} is element of the schedule;

n! is factorial number 1*2*..*n;

e is natural number;

class SDR is special class in C++ notation for storing information about SDR;

SDR() is constructor of the class SDR;

s, p, next, sizes, sizep, n are fields of the class SDR;

_p,_n, psize are parameters of the constructor SDR;

head is first element of the list;

generic() is recursive function for the permutation generation;

s1, p1 are additional arrays in the function generic();

–> is class field access operator via pointer.

INTRODUCTION

Task planning can be defined as a procedure of allocation of resources for a specific job at a specific time.

The most important goal of planning is use of resources. The goal is to minimize waiting time planning. A good time algorithm provides a good system productivity. Problems of combinatorial object generation often arise in computer modeling, cryptography, theory of schedules.

In this paper we consider a new approach to the generation of generalized combinatorial objects of special structure that are well suited for some scheduling tasks (schedule of meetings). Scheduling problems is the most widely studied problems in computer science. There are well known Job-shop scheduling or the job-shop problem (JSP) , the nurse operations research problem of finding

DOI 10.15588/1607-3274-2022-4-10

an optimal way to assign nurses to shifts, typically with a set of hard and soft constraints. The complexity of the corresponding algorithms in such problems is a critical parameter that allows us to assess the possibility of using a particular algorithm in practice. [1]

At the heart of our approach are procedures for the permanent decomposition of incidence matrices with memorization of identifier elements. We called our approach PD-methods.

The object of study is combinatorial objects generation in the task of Job-shop scheduling.

Despite the large number of publications on the generation of combinatorial objects, the development of new algorithms and approaches is relevant due to their computational complexity.

The subject of study is permanent decomposition algorithms for the combinatorial objects generation.

The purpose of the work is to develop methods for generating combinatorial objects that can be extended to solving complex scheduling problems.

1 PROBLEM STATEMENT

Suppose we have *n* elements $(a_1, a_2, ..., a_m)$, that can be part of *m* sets $(S_1, S_2, ..., S_m)$ and the occurrence of the same element several times is allowed. Information about which elements are included in the corresponding sets will be given in the form of an incidence matrix:

$$\begin{pmatrix} & a_1 & a_2 & ... & a_n \\ S_1 & n_{11} & n_{12} & ... & n_{1n} \\ S_2 & n_{21} & n_{22} & ... & n_{2n} \\ ... & ... & ... & ... & ... \\ S_m & n_{m1} & n_{m2} & ... & n_{mn} \end{pmatrix} \qquad (1)$$

The elements $(a_1, a_2, ..., a_m)$ will be called the identifiers of the columns of the incidence matrix. The system of different representatives (SDR) will be called a vector of the form:

$$(v_1, v_2, ..., v_m),\ v_i \in S_i, i = \overline{1,m}, v_i \neq v_j, i \neq j.$$

We divide identifier elements on regular and “stream”. If the element a_i is “stream”, then it must be simultaneously written in all positions of the sample vector, where the correspondent incidence matrix column contains non-zero elements. An arbitrary vector of samples (or a matrix, the rows of which are samples) will be called a schedule.

The schedule

$$((v_{11}, v_{12}, ..., v_{1m}), (v_{21}, v_{22}, ..., v_{2m}), ..., (v_{k1}, v_{k2}, ..., v_{km}))$$

will be considered correct under the conditions:

1. $\forall j \in \{1,2,...,m\}$:
$\{v_{1j} \bigcup v_{2j} \bigcup ... \bigcup v_{kj}\} = \{n_{j1} * a_1 \bigcup n_{j2} * a_2 \bigcup ... \bigcup n_{jn} * a_n\}$,
$l * a = \{a_1, a_2, ..., a_l\}, a_i = a, i = \overline{1,l}$.

2. $\forall i \in \{1,2,...,k\}: v_{ij} \neq v_{ir}, j \neq r.$, elements v_{ij}, v_{ir}, are non-stream.

Obviously, in the case when each element is included in each set only once and all elements are non-stream, matrix of incidence is

$$\begin{pmatrix} & a_1 & a_2 & ... & a_n \\ S_1 & 1 & 1 & ... & 1 \\ S_2 & 1 & 1 & ... & 1 \\ ... & ... & ... & ... & ... \\ S_n & 1 & 1 & ... & 1 \end{pmatrix} \qquad (2)$$

Then one of the variants of the correct schedule can be written in the form:

$$\begin{pmatrix} a_1 & a_2 & a_3 & ... & a_n \\ a_2 & a_3 & a_4 & ... & a_1 \\ a_3 & a_4 & a_5 & ... & a_2 \\ ... & ... & ... & ... & ... \\ a_n & a_1 & a_2 & ... & a_{n-1} \end{pmatrix}. \qquad (3)$$

The rows of schedule matrix consist of *n* permutations of the corresponding elements. The algorithm of cyclic shift of column or row elements is implemented here. Obviously, the number of correct schedules constructed by the cyclic shift algorithm is n!. The task of scheduling is very complex, NP-complete. In the general case, to construct all possible variants of correct schedules, it is necessary to analyze all possible SDR variants. Therefore, using any algorithm for solving the problem of generating permutations in the “same place”, it is necessary to additionally store each variant of permutations in memory. Therefore, we must use the most optimal algorithm for generating all possible permutations. In this paper, from the point of view of complexity, an approach is investigated that is based on the use of the procedure for decomposing the permanent of the incidence matrix. It requires the development of new approaches, in particular, to the problems of generating combinatorial objects.

2 REVIEW OF THE LITERATURE

Despite the fact that a significant number of algorithms have been developed to generate various combinatorial objects, such as permutations, permutations with repetitions of different types, systems of subsets of some sets of elements [1–4, 6–13], new approaches and algorithms are still emerging.

© Turbal Y. V., Babych S. V., Kunanets N. E., 2022
DOI 10.15588/1607-3274-2022-4-10

Given the novelty of the combination of permanent and decomposition solutions within the calendar calculation – it is difficult to rely on similar literature.

Consider the most relevant areas of application of such solutions.

A significant amount of most recent research has focused on the tasks of scheduling within cloud computing. There are numerous and excellent resources available in the cloud. The cost of performing tasks in the cloud depends on what resources are used. Cloud planning is different from traditional planning. In the environment of cloud computing, the task of scheduling is the biggest and most difficult issue. Task scheduling problem is the NP-complete problem. Many heuristics have introduced scheduling algorithms, but more improvements are needed to make the system faster and more responsive [5].

A detailed overview of the combinatorial algorithms can be given by Knuth [6], Ruskey [7] which considers the concept of combinatorial generation and distinguishes the following tasks: listing-generating elements of a given combinatorial set sequentially, ranking – numbering elements of a given combinatorial set, unranking – generating elements of a given combinatorial set in accordance with their ranks and random selection–generating elements of a given combinatorial set in random order.

General methods for developing combinatorial generation algorithms were studied by such researchers as S. Bacchelli [1, 2], E. Barcucci [2], A. Del Lungo [3, 4], V.V. Kruchinin [8, 9], P. Flajolet [10] and others. It is wellknown algorithms for the permutation generation [11–14], such as Bottom-Up, Lexicography, Johnson-Trotter [8], PIndex [15], Inversion [15].

3 MATERIALS AND METHODS

The main idea that we use in our approach to the problem of generating combinatorial objects is based on
the using of the modified permanent properties.

Definition: Modified permanent of the incidence matrix will be the sum of all possible products of the numerical elements of the matrix, each of which contains one element from each row and column, and the element of the flow column (the column corresponding to the flow element) cannot be in the product together with the elements. Other rows corresponding to the same stream element (the corresponding rows will be crossed out in the schedule or with elements of other columns corresponding to the same element).

In the case of flow elements absence, the modified permanent is a normal permanent. The procedure for finding a permanent can easily be implemented recursively in the same way as finding the determinant of a matrix by decomposition on any line. Based on the definition, the decomposition procedure will be as follows: a nonzero row element is multiplied on a modified permanent matrix formed by the following rules – if a row element belongs to a stream column, the matrix is formed by deleting the column where this element is located and all rows corresponding to all elements of this stream. If a row element does not belong to a stream column, then the matrix is formed by deleting the row and column where this element is situated, as well as all stream columns at the intersection of which are non-zero elements.

Obviously, the permanent of the square incidence matrix consisting of 1 is equal to n! (we decompose on the first line, then we have n components that already contain matrices of dimension n–1, etc.).

The main idea of our algorithm construction: in the process of permanent decomposition can be memorized the identifiers of the current elements columns.

Consider an example. Let's incidence matrix present in the form:

$$\begin{pmatrix} & 1 & 2 & 3 \\ R_1 & 1 & 1 & 1 \\ R_2 & 1 & 1 & 1 \\ R_3 & 1 & 1 & 1 \end{pmatrix} \tag{4}$$

Thus, we have:

$$per\,\mathrm{mod}\begin{array}{r|ccc|} & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 1 \\ 3 & 1 & 1 & 1 \end{array} = 1^1\, per\,\mathrm{mod}\begin{vmatrix} 2 & 3 \\ 1 & 1 \\ 1 & 1 \end{vmatrix} +$$

$$+1^2\, per\,\mathrm{mod}\begin{vmatrix} 1 & 3 \\ 1 & 1 \\ 1 & 1 \end{vmatrix} + 1^3\, per\,\mathrm{mod}\begin{vmatrix} 1 & 2 \\ 1 & 1 \\ 1 & 1 \end{vmatrix} =$$

$$=1^1 1^2\, per\,\mathrm{mod}\begin{vmatrix} 3 \\ 1 \end{vmatrix} + 1^1 1^3\, per\,\mathrm{mod}\begin{vmatrix} 2 \\ 1 \end{vmatrix} +$$

$$+1^2 1^1\, per\,\mathrm{mod}\begin{vmatrix} 3 \\ 1 \end{vmatrix} + 1^2 1^3\, per\,\mathrm{mod}\begin{vmatrix} 1 \\ 1 \end{vmatrix} +$$

$$+1^3 1^1\, per\,\mathrm{mod}\begin{vmatrix} 2 \\ 1 \end{vmatrix} + 1^3 1^2\, per\,\mathrm{mod}\begin{vmatrix} 1 \\ 1 \end{vmatrix} =$$

$$=1^1 1^2 1^3 + 1^1 1^3 1^2 + 1^2 1^1 1^3 + 1^2 1^3 1^1 + 1^3 1^1 1^2 + 1^3 1^2 1^1.$$

As we see, in the case of a square matrix all elements of which equals to 1, we can get all permutations by decomposing the permanent with "memorization". Based on the decomposition procedure, the following general recursive PD-algorithm for forming systems of different representatives of sets is obvious:

1. The initial matrix of incidence is formed.

2. The first row of the matrix is viewed and all non-zero elements are found.

3. For each non-zero element of the first line:

a) the corresponding identifier element is added to the corresponding permutation;

b) a new incidence matrix is formed from the initial one by deleting the column and row where the found non-

© Turbal Y. V., Babych S. V., Kunanets N. E., 2022
DOI 10.15588/1607-3274-2022-4-10

zero element stands (memory is allocated and data is copied);

c) is called recursively the generation function for the new matrix.

4 EXPERIMENTS

Let’s consider in more detail the problem of generating permutations. The specificity of our approach to permutation generation is that we need to keep in mind all permutations for their further use, in particular, in scheduling tasks for scheduling generation. Note that the incidence matrix has all the elements 1 and it is square. In this case, during the decomposition of the permanent, there is no need to store the incidence matrix in memory, it is enough to know only the identifier elements. So, we consider a permanent vector. We will use a singly linked list to store all elements. Each item in the list will contain information about the SDR. We can use two arrays – one to represent the already written part of the SDR, another – to place the elements that will still be used for decomposition. We can use special class in C++ notation for storing information about SDR:

```
class SDR {
public:
char *s;
char* p;
SDR* next;
 int sizes;
int sizep;
int n;
SDR(char* _p, int _n, int psize)
{       n=_n; sizep=psize; sizes=_n-psize;
        p=new char[psize];
        for(int i=0;i<psize;i++) p[i]=_p[i];
        next=NULL; }
SDR(SDR* head, int k) {
        sizes=1+head->sizes;
        n=head->n;
        sizep=n-sizes;
        next=NULL;
        s=new char[sizes];
        p= new char[sizep];
        for(int j=0;j<sizes-1;j++) s[j]=head->s[j];
                s[sizes-1]=head->p[k];
        int l=0;
        for(int i=0;i<sizep+1;i++)
                if(i!=k) p[l++]=head->p[i]; } };
```

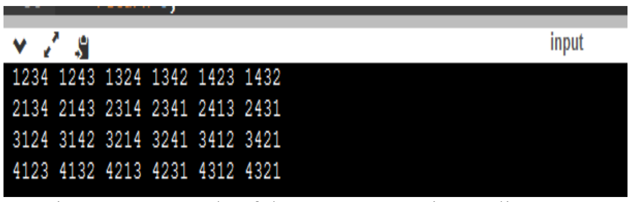
In this class we create two constructors: SDR(char* _p, int _n, int psize) for the first initialization end SDR(SDR* head, int k) for the creation new class member on the base of head in which k-th element of array p writes to the arrays. Obviously, using a singly linked list, we must correctly insert the newly created element after the head element in the list:

```
SDR *tmp=new SDR(head,i);
tmp->next=head->next;
head->next=tmp;
```

© Turbal Y. V., Babych S. V., Kunanets N. E., 2022
DOI 10.15588/1607-3274-2022-4-10

Thus we can construct recursive function for the permutation generation according to our approuch:

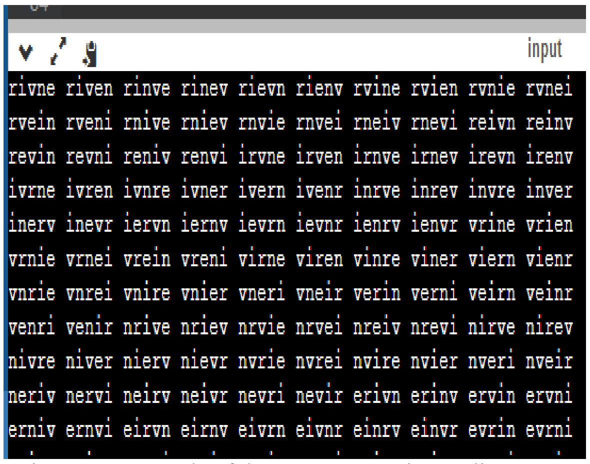
```
void generic(SDR* head) {
     if (head->sizes<head->n) {
          for(int i=head->sizep-1;i>0;i--) {
               SDR *tmp=new SDR(head,i);
               tmp->next=head->next;
               head->next=tmp;
          generic(tmp); }
          char* s1=new char[head->sizes+1];
          char* p1= new char[head->sizep-1];
          for(int j=0;j<head->sizes;j++)
               s1[j]=head->s[j];
          s1[head->sizes]=head->p[0];
          for(int i=0;i<head->sizep-1;i++)
               p1[i]=head->p[i+1];
          delete head->s;
          delete head->p;
          head->s=0;
          head->p=0;
          head->s=s1;
          head->p=p1;
          head->sizes++;
          head->sizep--;
          generic(head); } }
```



```
1234 1243 1324 1342 1423 1432
2134 2143 2314 2341 2413 2431
3124 3142 3214 3241 3412 3421
4123 4132 4213 4231 4312 4321
```

Figure 1 – Example of the program running, online GDB C++compiler

We can consider small examples of the our program running, results is on the Fig.1, where initial array is char p[]={'1','2','3','4'} and on the Fig. 2, where initial array is char p[]={'r','i','v','n',’e’}.



```
rivne riven rinve rinev rievn rienv rvine rvien rvnie rvnei
rvein rveni rnive rniev rnvie rnvei rneiv rnevi reivn reinv
revin revni reniv renvi irvne irven irnve irnev irevn irenv
ivrne ivren ivnre ivner ivern ivenr inrve inrev invre inver
inerv inevr iervn iernv ievrn ievnr ienrv ienvr vrine vrien
vrnie vrnei vrein vreni virne viren vinre viner viern vienr
vnrie vnrei vnire vnier vneri vneir verin verni veirn veinr
venri venir nrive nriev nrvie nrvei nreiv nrevi nirve nirev
nivre niver nierv nievr nvrie nvrei nvire nvier nveri nveir
neriv nervi neirv neivr nevri nevir erivn erinv ervin ervni
erniv ernvi eirvn eirnv eivrn eivnr einrv einvr evrin evrni
```

Figure 2 – Example of the program running, online GDB C++ compiler

5 RESULTS

Let us consider complexity of the PD-approach for permutation generation (see function generic()). In order to compare the computational complexity of our approach with the known methods, we will take into account that in most cases, are used algorithms that do not require recording all variants of permutations, the so-called generation algorithms "in the same place". When calculating the number of copies, we assume that copying to the array s is constructive in the case when all permutations are stored in memory, and copying to the array p creates an additional computational load and must be taken into account:

```
 p= new char[sizep];// ~n-1 assignments
 int l=0;// 1 assignment
for(int i=0;i<sizep+1;i++)//n increments
if(i!=k) p[l++]=head->p[i]; // n-1 assignments, 1 //increment, n-
1 class field access, n comparisons
```

The size of the array p sizep will decrease from *n* (initial iteration) to 0.

Let $Q(n)$ be a number of assignments and increments, *n*-size of array p. Obviously, that $Q(1)=1$. Then we have:

$$Q(n)=n(Q(n-1)+3n)=nQ(n-1)+3n^2=$$
$$=n(n-1)Q(n-2)+3n(n-1)^2+3n^2=$$
$$=n(n-1)(n-2)Q(n-3)+3n(n-1)(n-2)^2+$$
$$+3n(n-1)^2+3n^2=\ldots=n(n-1)\ldots(n-k)Q(n-k-1)+$$
$$+3n(n-1)\ldots(n-k+1)(n-k)^2+$$
$$+3n(n-1)\ldots(n-k+2)(n-k+1)^2+\ldots+$$
$$+3n(n-1)(n-2)^2+3n(n-1)^2+3n^2=$$
$$=n!+3n(n-1)\ldots 32^2+\ldots+3n(n-1)(n-2)^2+$$
$$+3n(n-1)^2+3n^2=n!+3n!(2+\frac{3}{2!}+\frac{4}{3!}+\ldots+$$
$$+\frac{n-k-2+1}{(n-k-2)!}+\ldots+\frac{n-1+1}{(n-1)!})=n!+3n!(1+1+\frac{1}{2!}+\ldots+$$
$$+\frac{1}{(n-k-1)!}+\ldots+\frac{1}{(n-2)!})+3n!(1+\frac{1}{2!}+\ldots+\frac{1}{(n-1)!})\le$$
$$\le n!+3n!(1+1+\frac{1}{2!}+..+\frac{1}{n!}+\ldots)+$$
$$+3n!(1+\frac{1}{2!}+\ldots+\frac{1}{(n-1)!}+\ldots)=n!(1+3e+3(e-1))=$$
$$=n!(6e-2)=O(n!).$$

Consider the case of an arbitrary incidence matrix. In this case, it is necessary to prepare a new incidence matrix in the process of recursively calling the generation function, allocate memory and copy data. Thus we'll have minimum $2(n-1)^2$ additional arithmetic operations. Let all elements in the matrix be nonzero. Then we have:

$$Q(n)=n(Q(n-1)+2(n-1)^2)=nQ(n-1)+2n(n-1)^2=$$
$$=n(n-1)Q(n-2)+2n(n-1)(n-2)^2+2n(n-1)^2=$$
$$=n(n-1)(n-2)Q(n-3)+2n(n-1)(n-2)(n-3)^2+$$
$$+2n(n-1)(n-2)^2+2n(n-1)^2=\ldots=$$
$$=n!+2n(n-1)(n-2)\ldots 21^2+\ldots+$$
$$2n(n-1)(n-2)\ldots 32^2+\ldots+2n(n-1)(n-2)^2+$$
$$+2n(n-1)^2=n!+2n!(1^2+\frac{2^2}{2!}+\ldots+\frac{(n-1)^2}{(n-1)!})=$$
$$=n!+2n!(1^2+\frac{2}{1!}+\ldots+\frac{(n-1)}{(n-2)!}))=n!+2n!\sum_{k=0}^{n-2}\frac{k+1}{k!}=$$
$$=n!+2n!(\sum_{k=0}^{n-3}\frac{1}{k!}+\sum_{k=0}^{n-2}\frac{1}{k!})\le n!(1+4e)=O(n!).$$

6 DISCUSSION

Thus, the use of a permanent approach for generating permutations has made it possible to obtain an algorithm whose computational complexity is comparable to the fastest known algorithms. It is known [1] that, for example, Johnson Trotter's algorithm PMin(n) or lexicographic algorithm Plex(n) also have computational complexity $O(n!)$.

Obvious, that PD-algorithm generates the list of permutation in lexicographic order. This order can be defined by the initial array. If this array have standart lexicographically ordered elements, we'll get lexicographically ordered perturbations. We can consider small example of the our program running (See Fig.1), where char p[]={'1','2','3','4'}. If we use initial array char p[]={'r','i','v','n','e'} than we define special order:

'r'<'i'<'v'<'n'<'e'. Our perturbations are ordered according to this order (see Fig. 2).

An essential feature of our approach is the possibility of modifying it to generate combinatorial objects of a more complex structure. To do this, it is enough to specify the appropriate incidence matrix. If it is necessary to consider some additional conditions, then it may be necessary to modify the definition of the permanent and, accordingly, the procedure for decomposition by string (for example, the impossibility of the presence in one SDK the same elements, unless they are streamed). The following modification can be considered. If the element is not "stream" then we delete the column where it stands, the row and all "stream" columns if non-zero elements intersect with this row. If the running element is streaming, then all rows where the streaming element and all columns with the same identifiers to the current one is crossed out. If there are non-zero elements of other streams at the intersection with the stream rows, the corresponding columns are also crossed out. Thus, we obtain a solution to the problem of correctness of the SDR: on the one hand in the SDR is not possible the presence of two identical elements, on the other hand such a presence is possible if the element is streaming.

If the number of non-zero elements in each line of the matrix of incidence is less than n or equal to k <n, the number of arithmetic operations can be significantly less. For example, for k=1 the number of recursive calls will not exceed n. However, when viewing a row of such a matrix, will be necessary comparison with 0 of each elements. And that's why we still have n! comparisons. However, this problem is easily solved by considering the rows of the incidence matrix as dynamic arrays with numbers of nonzero elements. Then the complexity of the algorithm at $k = 1$ will be $O(n)$.

CONCLUSIONS

Thus, the paper considers a new approach to the generation of complex combinatorial objects, based on the procedure of decomposition of the modified permanent of the incidence matrix by line with memorization of index elements. The specificity of this approach is that certain additional conditions imposed on the relevant SDRs are taken into account at the stage of permanent decomposition procedures. Thus, in the case when the matrix consists of only 1, we obtain the decomposition procedure of the ordinary permanent. If we set the condition of the presence of "stream" elements in the SDR and the arbitrary configuration of the structure of the incidence matrix, the decomposition procedure must be modified.

The paper evaluates the complexity of the PD-algorithm for generating permutation and shows that it is equal to O (n!). Such complexity is in the fastest algorithms, such as lexicographic, Johnson-Trotter. However, in practice PD-algorithm will obviously work slower, in particular due to the large number of memory operations and data coping. However, the recursive PD algorithm can easily be modified to generate much more complex objects, while the mentioned known approaches exclusively use the specifics of permutations.

The scientific novelty of this paper lies in the fact that in the work it was possible to use the algebraic properties of special modifications of matrices permanets to construct efficient algorithms for generating combinatorial objects**.**

The practical significance of obtained results is that the software realizing the proposed methods is developed, as well as experiments to study their properties are conducted. The PD-algorithm can be used in software development where the generation of combinatorial objects is used, in particular, in information security systems.

Prospects for further research are to study the proposed methods for a broad class of scheduler problems, job-shop problem (JSP), the nurse operations research problem .

REFERENCES

1. Bacchelli S., Barcucci E., Grazzini E., Pergola E. Exhaustive generation of combinatorial objects by ECO, *Acta Informatica*, 2004, Vol. 40, pp. 585–602.
2. Bacchelli S., Ferrari L., Pinzani R., Sprugnoli R. Mixed succession rules: The commutative case, *Journal of Combinatorial Theory,* 2010, Vol. 117, Series A, pp. 568–582. DOI: 10.1016/j.jcta.2009.11.005
3. Barcucci E., Del Lungo A., Pergola E., Pinzani R. ECO: A methodology for the enumeration of combinatorial objects, *Journal of Difference Equations and Applications*, 1999, Vol. 5, pp. 435–490.
4. Del Lungo A., Duchi E., Frosini A., Rinaldi S. On the generation and enumeration of some classes of convex polyominoes, *The Electronic Journal of Combinatorics,* 2011, Vol. 11, № 1, pp. 1–46. DOI: 10.37236/1813
5. Arnaw Wadhonkar, Theng Deepti A Task Scheduling Algorithm Based on Task Length and Deadline in Cloud Computing, *International Journal of Scientific & Engineering Research,* 2016, Vol. 7, № 4, pp. 1905–1909.
6. Knuth D. E. The Art of Computer Programming, Vol. 4A : Combinatorial Algorithms Part 1. Boston, Addison-Wesley Professional, 2011.
7. Ruskey F. Combinatorial Generation. Working Version (1j-CSC 425/520) [Electronic resourse], *Department of Computer Science University of Victoria,* 2003. Access mode: http://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf. Accessed 1 May 2020.
8. Kruchinin V. V. Methods for Developing Algorithms for Ranking and Unranking Combinatorial Objects Based on AND/OR Trees. Tomsk, V-Spektr, 2007.
9. Shablya Y., Kruchinin D., Kruchinin V. Method for Developing Combinatorial Generation Algorithms Based on AND/OR Trees and Its Application, *Mathematics,* 2020, Vol. 8, № 962. DOI: 10.3390/math8060962
10. Flajolet P., Zimmerman P., Cutsem B. A calculus for the random generation of combinatorial structures, *Theoretical Computer Science*, 1994, Vol. 132, pp. 1–35.
11. Xin Chen, Yan Lan, Attila Benkő, György Dósa, Xin Han Optimal algorithms for online scheduling with bounded rearrangement at the end, *Theoretical Computer Science.* - 2011, Vol. 412, No. 45, pp. 6269–6278. DOI: 10.1016/j.tcs.2011.07.014
12. Do P. T., Tran T. T. H, Vajnovszki V. Exhaustive generation for permutations avoiding (colored) regular sets of patterns, *Discrete Applied Mathematics,* 2019, Vol. 268. pp. 44–53.
13. Mirshekarian Sadegh, Šormaz Dušan N. Correlation of job-shop scheduling problem features with scheduling efficiency, *Expert Systems with Applications,* 2016, Vol. 62. pp. 131–147. DOI: 10.1016/j.eswa.2016.06.014
14. Humble Travis S. Yuma Nakamura, Kazuki Ikeda Application of Quantum Annealing to Nurse Scheduling Problem, *Scientific Reports,* 2019, Vol. 9, No. 1, P. 12837. DOI: 10.1038/s41598-019-49172-3
15. Fedoriaeva T. I. Combinatorial algorithms. Novosobirsk, Novosibirsk State University, 2011.

Received 08.09.2022.
Accepted 12.11.2022.

DOI 10.15588/1607-3274-2022-4-10

УДК 004.93

АЛГОРИТМ ДЕКОМПОЗИЦІЇ ПЕРМАНЕНТУ ДЛЯ ГЕНЕРАЦІЇ КОМБІНАТОРНИХ ОБ'ЄКТІВ

Турбал Ю. В. – д-р техн. наук, професор кафедри комп'ютерних наук та прикладної математики Національного університету водного господарства та природокористування, Рівне, Україна.

Бабич С. В. – викладач відділення Програмування, Рівненського Фахового Коледжу Національного університету біоресурсів і природокористування України, Рівне, Україна.

Кунанець Н. Е. – д-р наук із соціальних комунікацій, професор кафедри Інформаційних систем та мереж, Інституту комп'ютерних наук та інформаційних технологій, Національного університету «Львівська політехніка», Львів, Україна.

АНОТАЦІЯ

Актуальність. Розглядається задача генерування векторів, що складаються з різних представників заданої множини. Такі проблеми виникають, зокрема, в теорії складання розкладів, при плануванні зустрічей. Окремим випадком цієї задачі є задача генерування перестановок. Мета роботи – розглянути проблему з точки зору постійного та загальновідомого підходу, виходячи з концепції лексикографічного порядку.

Метод. У багатьох завданнях виникає необхідність генерувати різноманітні комбінаторні об'єкти: перестановки, комбінації з повтореннями і без них, різноманітні підмножини. У цій роботі розглядається новий підхід до генерації комбінаторних об'єктів, який базується на процедурі постійної декомпозиції. Перманент будується для спеціальної матриці інцидентності. Основна ідея цього підходу полягає в включенні до процесу алгебраїчної перманентної декомпозиції за допомогою додаткової функції рядка для запису ідентифікаторів стовпців у відповідні структури даних. У цьому випадку алгебраїчний перманент не обчислюється, а отримуємо конкретний рекурсивний алгоритм генерації комбінаторного об'єкта. Проаналізовано обчислювальну складність цього алгоритму.

Результати. В межах PD-підходу розглянуто задачі генерації комбінаторних об'єктів, зокрема, перестановок. Досліджено обчислювальну складність запропонованих алгоритмів у порівнянні з відомими підгодами. Розглянуто варіант програмної реалізації розроблених алгоритмів.

Висновки. У роботі розглянуто новий підхід до генерації складних комбінаторних об'єктів, що грунтується на процедурі декомпозиції модифікованого перманенту матриці інцидентності за рядком із запам'ятовуванням елементів індексу. Специфіка цього підходу полягає в тому, що певні додаткові умови, що накладаються на відповідні системи різних представників, враховуються на етапі процедур декомпозиції. Досліджено складність розглянутих алгоритмів. У разі більш складних варіантів матриці інцидентності пропонується відповідна модифікація поняття перманенту і, відповідно, процедура його декомпозиції .

КЛЮЧОВІ СЛОВА: алгоритм, перманент, декомпозиція, складність обчислення.

ЛІТЕРАТУРА / ЛИТЕРАТУРА

1. Exhaustive generation of combinatorial objects by ECO / [S. Bacchelli, E. Barcucci, E. Grazzini, E. Pergola] // Acta Informatica. – 2004. – Vol. 40. – P. 585–602.
2. Mixed succession rules: The commutative case / [S. Bacchelli, L. Ferrari, R. Pinzani, R. Sprugnoli] // Journal of Combinatorial Theory. – 2010. – Vol. 117, Series A. – P. 568–582. DOI: 10.1016/j.jcta.2009.11.005
3. ECO: A methodology for the enumeration of combinatorial objects / [E. Barcucci, A. Del Lungo, E. Pergola, R. Pinzani] // Journal of Difference Equations and Applications. – 1999. – Vol. 5. – P. 435–490.
4. On the generation and enumeration of some classes of convex polyominoes / [A. Del Lungo, E. Duchi, A. Frosini, S. Rinaldi] // The Electronic Journal of Combinatorics. – 2011. – Vol. 11, № 1. – P. 1–46. DOI: 10.37236/1813
5. Arnaw Wadhonkar A Task Scheduling Algorithm Based on Task Length and Deadline in Cloud Computing / Arnav Wadhonkar, Deepti Theng // International Journal of Scientific & Engineering Research. – 2016. – Vol. 7, № 4. – P. 1905–1909.
6. Knuth D. E. The Art of Computer Programming / D. E. Knuth. – Vol. 4A : Combinatorial Algorithms Part 1. Boston : Addison-Wesley Professional, 2011.
7. Ruskey F. Combinatorial Generation. Working Version (1j-CSC 425/520) [Electronic resourse] / F. Ruskey. – Department of Computer Science University of Victoria, 2003. Access mode: http://page.math.tu-berlin.de/~felsner/SemWS17-18/Ruskey-Comb-Gen.pdf. Accessed 1 May 2020.
8. Kruchinin V. V. Methods for Developing Algorithms for Ranking and Unranking Combinatorial Objects Based on AND/OR Trees / V. V. Kruchinin. – Tomsk: V-Spektr, 2007.
9. Shablya Y. Method for Developing Combinatorial Generation Algorithms Based on AND/OR Trees and Its Application / Y. Shablya, D. Kruchinin, V. Kruchinin // Mathematics. – 2020. – Vol. 8, № 962. DOI: 10.3390/math8060962
10. Flajolet P. A calculus for the random generation of combinatorial structures / P. Flajolet, P. Zimmerman, B. Cutsem // Theoretical Computer Science. – 1994. – Vol. 132. – P. 1–35.
11. Optimal algorithms for online scheduling with bounded rearrangement at the end / Chen Xin, Lan Yan, Benkő Attila, Dósa György, Han Xin // Theoretical Computer Science. -2011. – Vol. 412, № 45. – P. 6269–6278. DOI: 10.1016/j.tcs.2011.07.014
12. Do P. T. Exhaustive generation for permutations avoiding (colored) regular sets of patterns // P. T. Do, T.T.H Tran, V. Vajnovszki // Discrete Applied Mathematics. – 2019. – Vol. 268. – P. 44–53.
13. Mirshekarian Sadegh Correlation of job-shop scheduling problem features with scheduling efficiency // Mirshekarian Sadegh, N. Šormaz Dušan // Expert Systems with Applications. – 2016. – Vol. 62. – P. 131–147. DOI: 10.1016/j.eswa.2016.06.014
14. Humble Travis S. Application of Quantum Annealing to Nurse Scheduling Problem / S. Humble Travis, Nakamura Yuma, Ikeda Kazuki. // Scientific Reports. – 2019. – Vol. 9, № 1. – P. 12837. DOI: 10.1038/s41598-019-49172-3
15. Fedoriaeva T. I. Combinatorial algorithms / T. I. Fedoriaeva – Novosobirsk : Novosibirsk State University, 2011.

© Turbal Y. V., Babych S. V., Kunanets N. E., 2022
DOI 10.15588/1607-3274-2022-4-10