Modification of the Permanent Decomposition Method for the Meeting Schedule Problem

Yurii Turbal[a], Sergii Babych[b], Larysa Bachyshyna[a], Nataliia Kunanets[c] and Nataliia Kovalchuk[a]

[a] *National university of water and environmental engineering, Soborna st.,11, Rivne, 33022, Ucraine*
[b] *College of National University of Life and Environmental Sciences of Ukraine, Software engineering, Kopernyka St., 44, Rivne, 33000, Ukraine*
[c] *Lviv Politechnic National University, 12 Bandera street, Lviv, Ukraine, 79013*

Abstract

In this paper are considered meeting schedule problem and permanent decomposition methods (PD), which was investigated in reseant years. PD-approach to the combinatorial objects generetion is based on the procedure of the incidence matrices permanent decomposition with "memorization" of the columns identifier elements – process of the algebraic permanent decomposition by row includes addidional function for the column identifiers writing into corresponding data structures. In fact, algebraic permanent in not calculated, but we get a specific recursive algorithm for generating a combinatorial object. In this paper we present a new modification of the PD-algorithn which includes modification of the incidence matrix and procedure of the permanent decomposition. Our modification allows to build all possible variants of meeting schedules according to the system of different representatives, which cad be formed by the base PD-algorithm. For effective software implementation of the proposed modifications special additive-disjunctive forms is proposed. It it considered data structures and some aspect of the program realisation.

Keywords 1

Algorithm; permanent ; decomposition; sheduling ; ADF.

1. Introduction

The tasks of calendar planning have not lost their relevance in recent years, despite the presence of a large number of theoretical results and practical approaches to their solution. Sheduling problems is the most widely studied problems in computer sciense. There are wellknown Job-shop scheduling or the job-shop problem, the nurse operations research problem of finding an optimal way to assign nurses to shifts. We will consider only the most popular problem areas for the use of planning tasks.

Job shop scheduling problem (JSP) is the core part of production management in job shop system. The efficient completion of production tasks requires reasonable job shop scheduling. Therefore, the study of job shop scheduling problem has very important theoretical significance and application value. Job shop scheduling problem is one of the typical complex scheduling problems with high complexity and difficulty. It has a strong engineering application background [1],[5].

Task scheduling problem is the one of the most critical issues in cloud computing environment because cloud performance depends mainly on it. There are various types of scheduling algorithms; some of them are static scheduling algorithms that are considered suitable for small or medium scale cloud computing; and dynamic scheduling algorithms that are considered suitable for large scale cloud computing environments. Worth mentioning the most popular three static task scheduling

ITTAP'2021: 1nd International Workshop on Information Technologies: Theoretical and Applied Problems, November 16–18, 2021, Ternopil, Ukraine
EMAIL: turbaly@gmail.com (A. 1); asha.maneteren@mail.com (A. 2); l.d.bachyshyna@nuwm.edu.ua (A. 3); n.s.kovalchuk@nuwm.edu.ua (A. 4); nek@lviv.ua (A. 5)
ORCID: 0000-0002-5727-5334 (A. 1); 0000-0002-8669-7288 (A. 2); 0000-0002-7060-1747 (A. 3); 0000-0002-4723-4934 (A. 4); 0000-0003-3007-2462 (A. 5)

algorithms performance there are: first come first service (FCFS), short job first scheduling (SJF), MAX-MIN. The CloudSim simulator has been used to measure their impact on algorithm complexity, resource availability, total execution time (TET), total waiting time (TWT), and total finish time (TFT) [2],[4].[8].

In a Big data computing, the processing of data requires a large amount of CPU cycles and network bandwidth and disk I/O. Dataflow is a programming model for processing Big data which consists of tasks organized in a graph structure.Scheduling these tasks is one of the key active research areas which mainly aims to place the tasks on available resources. It is essential to effectively schedule the tasks, in a manner that minimizes task completion time and increases utilization of resources. In recent years, researchers have discussed and presented different task scheduling algorithms. Especially often there are scientific works in the areas the state-of-art of various task scheduling algorithms, scheduling considerations for batch and streaming processing, and task scheduling algorithms in the wellknown open-source big data platforms. Furthermore, this study proposes a new task scheduling system to alleviate the problems persists in the existing task scheduling for big data [3],[6],[8]].

These are just a few areas of use of scheduling tasks, but among them we can trace the use of algorithms, analogous to which is the algorithm in this article.

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]. In particular, the complexity of scheduling algorithms (NP-complete problems) was pointed out in [1], where is proposing a solution by a method close to the exact method for a certain criterion of optimality, because "the implementation of the tasks of scheduling by the method of search, as well as search with return, is not effective" [2]. Therefore, in modern scientific works, heuristic approaches are preferred, because the real size of the obtained problems does not yet allow to solve them in practice with existing precise methods for quadratic programming problems. The results that this or that ordering leads to differ significantly. In some practical cases, these differences are determined by another value, or take a cost nature.

In the paper [1] were considered meeting schedule problem and was proposed the algorithm of the meeting schedule construction which is based on the procedure of incidence matrix decomposition. In the paper [2] were considered a similar aprouch to the generation of generalized combinatorial objects of special structure that are well suited for some scheduling tasks (schedule of meetings). At the heart of this approach are procedures for the permanent decomposition of incidence matrices with memorization of identifier elements. This approuch was called PD-method. Main rezult of PD-procedure was creation the system of different representatives of the list of sets. In this paper we present a new modification of thePD-algorithn which includes modification the incidence matrix construction and procedure of the permanent decomposition. The new modification allows to build all possible variants of schedules according to the system of different representatives. For effective software implementation of the proposed modifications, the use of special additive-disjunctive forms is proposed.

2. Main idea of the PD-algorithm modification

Let consider m teams which must met whith the n single persons (teachers, project managers, SCRAM-masters ect.) $P_1, P_2, ..., P_n$ during any period of time T. It is known the times of possible meetings for every day and numbers of meetings for every person whith appropriated teams. Obviously, the schedule can be represented in the form of the matrix, in which the columns can be matched by teams, rows-time indicators, and the elements of the matrix will be identifiers (numbers) of persons:

$$S=\begin{pmatrix} & C_1 & C_2 & C_3 & C_4 & C_5 & \dots & C_m \\ t_1 & P_1 & P_1 & P_1 & P_2 & P_3 & \dots & P_{i_1} \\ t_2 & P_1 & P_2 & P_3 & P_4 & P_5 & \dots & P_{i_2} \\ t_3 & P_1 & P_2 & P_3 & P_5 & P_6 & \dots & P_{i_3} \\ t_4 & P_5 & P_5 & P_5 & P_2 & P_1 & \dots & P_{i_4} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ t_6 & P_6 & P_7 & P_6 & P_6 & P_7 & \dots & P_{i_6} \end{pmatrix} \qquad (1)$$

We will call such a matrix a matrix of schedules. If we consider the columns of the matrix S, we get sets $S_1, S_2, \dots, S_m$ 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 incidence matrix of the form:

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

where $n_{ij} = \begin{cases} 1, a_j \in S_i \\ 0, a_j \overline{\in} S_i \end{cases}$. The elements $a_1, a_2, \dots, a_n$ will be called the identifiers of the columns of the incidence matrix.

Identifier elements are devided 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.

. The system of different representatives (SDR) will be called a vector of the form: $(v_1, v_2, \dots, v_m), v_i \in S_i, i = 1,2,\dots,m, v_i \neq v_j\ if\ this\ elemen\ are\ non-stream, i \neq j.$

An arbitrary vector of samples(or a matrix, the rows of which are samples) will be called a schedule .

The schedule $((v_{11}, v_{12}, \dots, v_{1m}), (v_{21}, v_{22}, \dots, v_{2m}), \dots, (v_{k1}, v_{k2}, \dots, v_{km}))$ will be considered correct under the conditions :

1. $\forall j \in \{1,2,..,m\}: \{v_{1j} \cup v_{2j} \cup \dots \cup v_{kj}\} = \{n_{j1} * a_1 \cup n_{j2} a_2 \cup \dots n_{jn} * a_n\},$

$$l * a = \{a_1, a_2, .., a_l\}, a_i = a$$

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

Note that incidence matrix (1) does not provide comprehensive information for scheduling. Therefore, consider a modification of the incidence matrix, whyку n_{ij} is the number of occurrences of the element a_j in the set S_i .

Suppose, for example, that we have a schedule matrix of the form:

$$R = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & 3 & 4 \end{pmatrix} \qquad (3)$$

Than we have such incidene matrix:

$$\begin{pmatrix} & 1 & 1^n & 2 & 3 & 4 \\ R_1 & 1 & 1 & 1 & 0 & 0 \\ R_2 & 0 & 1 & 0 & 2 & 0 \\ R_3 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \qquad (4)$$

Let's build the process of decomposition of the modified permanent with "memorization" on the first line:

$$per\text{mod} \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{pmatrix} 1 & 1^n & 2 & 3 & 4 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix} = 1_1^1 * per\text{mod} \begin{array}{c} \\ 2 \\ 3 \end{array}\begin{pmatrix} 2 & 3 & 4 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} + 1_1^{1^n} * 1 + 1_1^2 * per\text{mod} \begin{array}{c} \\ 2 \\ 3 \end{array}\begin{pmatrix} 1 & 3 & 4 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

$$= 1_1^1 * (2_2^3\, per\text{mod} \begin{array}{c} \\ 3 \end{array}\begin{pmatrix} 24 \\ 01 \end{pmatrix}) + 1_1^{1^n} * 1 + 1_1^2 * (2_2^3\, per\text{mod} \begin{array}{c} \\ 3 \end{array}\begin{pmatrix} 14 \\ 11 \end{pmatrix}) = 1_1^1 2_2^3 1_3^4 + 1_1^{1^n} + 1_1^2 2_2^3 1_3^1 + 1_1^2 2_2^3 1_3^4 .$$

Thus, SDR are:{(1,3,4),(1^n),(2,3,1),(2,3,4)}.

Consider in more detail the procedure of decomposing the permanent. Note that the non-zero element in the row of the matrix in the first step means that the corresponding component must be present in the schedule. The first group must contain elements 1,2, 1^n .,. Then in the process of decomposition is branching and various situations arise. In the process of calculating the permanent, we simply use the operation of adding +. However, from the point of view of constructing a schedule, the logic of the addition operation is completely different here, it actually means "disjunction", the possibility of choosing one of the options for SDR. If an element with a value greater than 1 is multiplied, the situation of inclusion of several components is possible. For example, in the expression $2_2^3\, per\text{mod} \begin{array}{c} \\ 3 \end{array}\begin{pmatrix} 14 \\ 11 \end{pmatrix} = 2_2^3 * 1_3^1 + 2_2^3 * 1_3^4$ addition means the mandatory inclusion of both components, because 3 is included twice, the element of the incidence matrix is 2. At the same time, in the expression $1_2^3\, per\text{mod} \begin{array}{c} \\ 3 \end{array}\begin{pmatrix} 14 \\ 11 \end{pmatrix} = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4$ addition means choice of one of two options, because element 3 is used only once.

3. Additive-disjunctive forms (ADF) and they properties.

Thus, we have two operations: selection and mandatory inclusion. An expression containing elements(string) using the specified operations will be called an additive-disjunctive form(ADF). In the future, the selection operation will be marked with the $\vee$ icon. In addition, when using the corresponding column element, which corresponds to a non-zero element of the incidence matrix, for mandatory inclusion, we can reduce the value of the element of the incidence matrix by 1. Therefore,

$$1_2^3\, per\text{mod} \begin{array}{c} \\ 3 \end{array}\begin{pmatrix} 14 \\ 11 \end{pmatrix} = 1_2^3 * 1_3^1 \vee 1_2^3 * 1_3^4 ,$$

$$2_2^3\, per\text{mod} \begin{array}{c} \\ 3 \end{array}\begin{pmatrix} 14 \\ 11 \end{pmatrix} = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4 ,$$

$$1_2^3\, per\text{mod} \begin{array}{c} \\ 3 \end{array}\begin{pmatrix} 145 \\ 111 \end{pmatrix} = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4 + 1_2^3 * 1_3^5 = ADF = 1_2^3 * 1_3^1 \vee 1_2^3 * 1_3^4 \vee 1_2^3 * 1_3^5 ,$$

$$2_2^3\, per\mathrm{mod} \begin{array}{c|c} & 1\,4\,5 \\ \hline 3 & 1\,1\,1 \end{array} = 2_2^3 * 1_3^1 + 2_2^3 * 1_3^4 + 2_2^3 * 1_3^5 = ADF = (1_2^3 * 1_3^1 + 1_2^3 * 1_3^4) \vee$$

$$(1_2^3 * 1_3^1 + 1_2^3 * 1_3^5) \vee (1_2^3 * 1_3^4 + 1_2^3 * 1_3^5),$$

$$3_2^3\, per\mathrm{mod} \begin{array}{c|c} & 1\,4\,5 \\ \hline 3 & 1\,1\,1 \end{array} = 3_2^3 * 1_3^1 + 3_2^3 * 1_3^4 + 3_2^3 * 1_3^5 = ADF = 1_2^3 * 1_3^1 + 1_2^3 * 1_3^4 + 1_2^3 * 1_3^5.$$

Therefore, the value of the incidence matrix element is equal to the number of inclusions of the corresponding index in the final ADF in the expressions of mandatory inclusion. Therefore, in the case when the value of the multiplier element is less than the number of non-zero elements of the decomposition line, we have options for interpreting the initial sum in the form of ADF. Thus, ADF is an expression that includes strings elements and two operations: + mandatory inclusion and binary selection $\vee$.

Let us consider a simple properties of ADF:

1. $a \vee a = a$;
2. $a + a = \{a, a\}$;
3. $a + b = b + a$;
4. $a \vee b = b \vee a$;
5. $a + b \vee c = (a + b) \vee (a + c)$;
6. $(a + b) \vee (a + c) = a + b \vee c$;
7. $$a + b \vee c + d \vee e = (a + b) \vee (a + c) + d \vee e = (a + b + d \vee e) \vee (a + c + d \vee e) =$$
   $$= (a + b + d) \vee (a + b + e) \vee (a + c + d) \vee (a + c + e).$$

4. Algorithms of the correct schedule ADF construction.

Now we can modify the permanent decomposition procedure to get the correct ADF. To do this, at each iteration of the permanent decomposition we will analyze the occurrence of the same elements in all components of the mandatory inclusion. Obviously, the condition must be met - the total number of occurrences of the element in the block of mandatory inclusion should be equal to its index. If this condition is found to be violated, the mandatory inclusion block will be considered invalid and should be removed. Let consider previous example and build correct ADF:

$$per\mathrm{mod} \begin{array}{c|ccccc} & 1 & 1^n & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 & 2 & 0 \\ 3 & 1 & 1 & 0 & 0 & 1 \end{array} = 1_1^1 * per\mathrm{mod} \begin{array}{c|ccc} & 2 & 3 & 4 \\ \hline 2 & 0 & 2 & 0 \\ 3 & 0 & 0 & 1 \end{array} + 1_1^{1^n} * 1 + 1_1^2 * per\mathrm{mod} \begin{array}{c|ccc} & 1 & 3 & 4 \\ \hline 2 & 0 & 2 & 0 \\ 3 & 1 & 0 & 1 \end{array} =$$

$$= 1_1^1 * (\, 2_2^3\, per\mathrm{mod} \begin{array}{c|c} & 2\,4 \\ \hline 3 & 0\,1 \end{array}) + 1_1^{1^n} * 1 + 1_1^2 * (\, 2_2^3\, per\mathrm{mod} \begin{array}{c|c} & 1\,4 \\ \hline 3 & 1\,1 \end{array}) =$$

$$1_1^1 * (1_2^3\, per\mathrm{mod} \begin{array}{c|c} & 2\,4 \\ \hline 3 & 0\,1 \end{array}) + 1_1^{1^n} * 1 + 1_1^2 * (1_2^3\, per\mathrm{mod} \begin{array}{c|c} & 1\,4 \\ \hline 3 & 1\,1 \end{array}) = 1_1^1 1_2^3 1_3^4 + 1_1^{1^n} + 1_1^2 1_2^3 1_3^1 \vee 1_1^2 1_2^3 1_3^4 .=$$

$$= (1_1^1 1_2^3 1_3^4 + 1_1^{1^n} + 1_1^2 1_2^3 1_3^1) \vee (1_1^1 1_2^3 1_3^4 + 1_1^{1^n} + 1_1^2 1_2^3 1_3^4) = 1_1^1 1_2^3 1_3^4 + 1_1^{1^n} + 1_1^2 1_2^3 1_3^1.$$

In the process of decomposition, in the last step, incorrect mandatory inclusion blocks are removed. In last example such block is $1_1^1 1_2^3 1_3^4 + 1_1^{1^n} + 1_1^2 1_2^3 1_3^4$.

If we apply a recursive algorithm for the decomposition of the permanent, then in this case it is difficult to implement control of the occurrence of the corresponding elements in the blocks of mandatory inclusion. Therefore, the approach described above in the recursive organization of the program is difficult to implement. Consider a slightly different approach. We will build ADF on the basis of the already built system of PSA. With this approach, it is good to adhere to the OOP paradigm.

We can modified procedure of ADF creation using list of all correct SDR, creating by base procedure of permanent decomposition. Thus we can propose algorithm of generation all correct variants of schedule using ADF (we call this algarithm ADF-algorithm):

1. Formation of the initial general block.

1.1 Review the first row of the incidence matrix and find all non-zero elements and their identifiers (corresponding elements of the zero row of the incidence matrix). The list of SDR is divided into groups, each SPRS from one group contains the first identical elements-identifiers.

1.2 The sign of obligatory inclusion + is put between groups.

1.3 If the index of the corresponding element of the first line is 1, then a selection operation is placed between all elements of the group. Otherwise, all possible combinations of group elements are formed, which are connected by the oleration +, the number of SDR in which is equal to the index of the element, between which the selection operation is placed.

2. Second line analysis.

2.1 Consider the second row of the incidence matrix, similarly determine the non-zero elements and their identifiers.

2.2 For each non-zero element :

2.2.1 Determine in which SDR list it is present. Then we determine all possible options so that it is included depending on its index. To do this, number all SDR, where the second is our running element, consider all possible combinations with SDRS, containing our second element and the number of which is equal to its index and check the required number of its occurrences, taking into account the operations of mandatory inclusions. index.

2.2.2 For each selected correct combination we form a new general block, which is built in the following way - from each disjunctive block containing SDR with our second element, include only selected in this combination SDR, and all others containing our second element are removed.

3. Next, similarly analyzed the third line, fourth, etc. Moreover, the condition of the correct occurrence of each element is analyzed for each common block.

Thus, the modification of the procedure of decomposition of the permanent incidence matrix by constructing ADP allows to obtain all possible correct variants of schedules. The proposed procedure solves not only the problem of matching the schedule elements in the rows, but also immediately obtain the required number of identical elements in the columns in accordance with the input data recorded in the modified incidence matrix.

Let's consider an example. We have system of SDR: {(1,3,4),(1,1,1), (2,3,1), (2,3,4)}, incidence matrix (5).

1. Analysis of the first row of the incidence matrix:

   $\{[(1,3,4)]+[(1,1,1)]+[(2,3,1)\vee(2,3,4)]\}$.

2. Second row:

   $1^{n}:\{[(1,3,4)]+[(1,1,1)]+[(2,3,1)\vee(2,3,4)]\}$,

   $3: \{[(1,3,4)]+[(1,1,1)]+[(2,3,1)]\}\vee\{[(1,3,4)]+[(1,1,1)]+[(2,3,4)]\}$.

3. Third row:

   $1: \{[(1,3,4)]+[(1,1,1)]+[(2,3,1)]\}\vee\{[(1,3,4)]+[(1,1,1)]+[(2,3,4)]\}$,

   $1^{n}:\{[(1,3,4)]+[(1,1,1)]+[(2,3,1)]\}\vee\{[(1,3,4)]+[(1,1,1)]+[(2,3,4)]\}$,

   $4: \{[(1,3,4)]+[(1,1,1)]+[(2,3,1)]\}$.

5. Data structures and program realisation

In accordance with our algorithm, to store information about all schedule options, we will use a list of instances of the class, which should contain information about the corresponding SDR, as well as symbols of ADF operations and possible parentheses that define disjunctive blocks. We can use C++ notation and create class ADF:

```
class ADF
{
  public:
  SDR* context;
  char * after;
  char * before;
  ADF *next;
  int delflag;
  ADF()
   {after=new char[2];
    before=new char[2];
    delflag=0;
   }
 };
```

```
class SDR
{     public:
      int nlast;
      int nmax;
      char *s;
      SDR *next;
      SDR()
      {
              s=new char [20];
              nlast=0;
              next=NULL;
      }
}
```

For the matrix of incidence, we can use class incydent :

```
class incydent
{     public:
      int nmaxrow, nmaxcol, **r;
      incydent(int nrow, int ncol)
      { // class body
      }
      incydent(void){}
      ~incydent()
      {//class body     }
};
```

In class ADF we must store additional information about symbols of ADF operations and possible parentheses and use arrays before[2] and after[2]. Field delflag we can use as a flag,that informs about deleting this element at last step of our algorithm. Let’s consider the main functions of our software implementation.

Function ADF* startgener (SDR *head) generates the initial ADP from the list of SDR, analyzing the first elements of the lines of SDR:

```
ADF* startgener (SDR *head)
```

```
{
ADF *rez;
ADF* last=new ADF;
SDR* lasthead=head;
last->before[1]='(';
if(head!=NULL) last->context=head;
last->next=NULL;
rez=last;
head=head->next;
while(head!=NULL)
  {
  ADF* current=new ADF;
  current->context=head;
  current->next=NULL;
  int i=0,j=0;
  while(lasthead->s[i]==lasthead->s[i+1]) i++;
  while(head->s[j]==head->s[j+1]) j++;
     if((lasthead->s[0]==head->s[0]) && (i==j))
     {
     last->after[1]='v';
     current->before[0]='v';
     }
     else
     { last->after[0]=')';
     last->after[1]='+';
     current->before[0]='+';
     current->before[1]='0';
     }
last->next=current;
last=current;
lasthead=head;
head=head->next;

if(head->next==NULL) current->after[0]=')';
}
return rez;
}
```

Function int ident(int m,int j,SDR* si, incydent* pi) analizes SDR and checks stream correctness:

```
int ident(int m,int j,SDR* si, incydent* pi)
{ if(si->s[j]==pi->r[0][j])
{if((j>0)&&(pi->r[0][j]!=pi->r[0][j-1] )|| (j==0))
return 1;
else
{for(int k=1;k<pi->nmaxrow;k++)
if((pi->r[k][j]>0)&&(si->s[k-1]!=pi->r[0][j])) return 0;
return 1;
}}}
```

Function int generlexc(int n,int k, int *mas) generates the next closest combination of the mas in in lexicographic order.

Function int goodblock(ADF* head, int *mas, int k, int m,int j,incydent* pi) checks whether 2 SDRs falls into one disjunctive subblock and also whether the index for quantity of correct occurrences is not bigger.

Function ADF* copyblock(ADF* startblock) copies the block of obligatory inclusion at once for startblock .

Function void genernewblock (int m,int j, int* mas,int size,ADF* startblock, incydent* pi) generates a new block with the inclusion of elements in accordance with the selected combination of mas numbers of the SDRs elements to be included. It works after goodblock .

Function ADF* nextblockadr(ADF* head) returns the start address of the next block of mandatory inclusion or NULL

Function int mjgeneric(ADF* head ,incydent* pi,int m,int j) generates all blocks of mandatory inclusion on the m-j element of the matrix incidence and writes after head:

```
 int mjgeneric(ADF* head ,incydent* pi,int m,int j)
{
    if(pi->r[m][j]>0)
    {
    while(head!=NULL)
    {ADF* tmp=head;
    ADF* nexthead =nextblockadr(head);
    int kv=0;
    while(tmp!=nexthead)
    {
    if(ident(m,j,tmp->context, pi)==1) kv++;
    tmp=tmp->next;
    }
    int index=pi->r[m][j];
    int *nu=new int[index];
    for(int fi=0;fi<index;fi++) nu[fi]=fi;
    for(int i=0;i<factorial(kv)/( factorial(index)* factorial(kv-index));i++ )
    {tmp=head;
    if(goodblock(tmp, nu, index, m, j, pi))
     genernewblock( m,j,nu,index,tmp,pi);
    generlexc(kv,index,nu);
    } //end for
    head=nexthead;
    head->delflag=1;
    }}}
```

Function ADF* ostgeneric(SDR* head, incydent* pi) realizes full procedure of the schedule generation:

```
ADF* ostgeneric(SDR* head, incydent* pi)
{ADF* rozklad=startgener(head);
for(int j=0;j<pi->nmaxcol;j++)
for(int m=1;m< pi->nmaxrow;m++)
if(pi->r[m][j]>0)
{mjgeneric(rozklad,pi,m,j);
if((j>0)&&(pi->r[0][j]==pi->r[0][j-1]))
{for(int k=m;k<pi->nmaxrow;k++)
if(pi->r[k][j]>0) pi->r[k][j]=0;}
}
ADF* current=rozklad;
while(current->next->next!=NULL)
{
if(current->next->delflag==1) {
ADF* tmp= current->next->next;
delete current->next;
```

```
    current->next=tmp;}
    }
    return rozklad;
    }
```

6. Conclusions

Thus, the paper considers a modification of the PD algorithm for generating systems of different representatives, which allows to immediately obtain all possible admissible schedules. Our modification is based on the use of special forms, which we called additive-disjunctive forms. Such forms can be useful for a wide range of tasks of generating combinatorial objects, scheduling. Clear algorithms are proposed for constructing meeting schedules based on ADF, one of which uses a system of different representatives (SDR) as input data, which is built on the basis of a basic PD-algorithm. This approach easily fits into the object-oriented programming paradigm [9] ,[10] and can be easily implemented using any object-oriented programming language.

7. References

[1] Yu Yingchen, A Research Review on Job Shop Scheduling Problem. E3S Web of Conferences. International Conference on Environmental and Engineering Management 253 (2021). doi:10.1051/e3sconf/202125302024.

[2] T. Aladwani, Types of Task Scheduling Algorithms in Cloud Computing Environment, in: Rodrigo da Rosa Righi, Scheduling Problems - New Applications and Trends, IntechOpen, United Kingdom, London, 2020. doi:10.5772/intechopen.86873.

[3] K. Govindarajan, S. Kamburugamuve, P. Wickramasinghe, V. Abeykoon, G. Fox, Task Scheduling in Big Data - Review, Research Challenges, and Prospects, in: Ninth International Conference on Advanced Computing, 2017. doi:10.1109/ICoAC.2017.8441494.

[4] A. Khalfay, A. Crispin, K. A. Crockett, A review of technician and task scheduling problems, datasets and solution approaches, in: Intelligent Systems Conference (IntelliSys), Computer Science, 2017. doi: 10.1109/INTELLISYS.2017.8324306

[5] B. Peng, Z. Lu, T.C.E. Cheng, A tabu search/path relinking algorithm to solve the job shop scheduling problem- Computers & Operations Research, Elsevier Science Publishing Company, United Kingdom, London, 2016. doi:10/1016/j.cor.2014.08.006.

[6] L. Williams, CPU Scheduling Algorithms in Operating Systems, 2021. URL: https://www.guru99.com/cpu-scheduling-algorithms.html.

[7] L. Kuang, L. Zhang, A new task scheduling algorithm based on value and time for cloud platform, AIP Conference Proceeding 1864 (2017). doi:10/1063/1.4992834.

[8] E. C. da Silva, P. H. R. Gabriel, A Comprehensive Review of Evolutionary Algorithms for Multiprocessor DAG Scheduling, Computation, MPDI, 2020, 8, 26 p.

[9] B. Stroustrup: A Tour of C++ (Second Edition), Addison-Wesley, 2018, 240 p.

[10] B. Stroustrup: Programming --Principles and Practice Using C++ (Second Edition), Addison-Wesley, 2014, 1312 p.