Sergey NOVIKOV

Institute of Mathematics and Computer Science,
The John Paul II Catholic University of Lublin,
Konstantynów Street 1H, 20-708 Lublin, Poland

Parallel constructing of the shortest coverings of large Boolean matrices

Abstract: The paper presents two options of the parallel algorithm for finding the shortest covering of a large Boolean matrix, where the decomposition of the initial matrix into matrices of smaller sizes is based on the partitioning of rows. The parallel algorithm ***COVMB*** contains sequential algorithms for partitioning of initial $\boldsymbol{m \times n}$ Boolean matrix on submatrices, building special matrices, summering of the shortest coverings of special Boolean matrices with smaller sizes as well as the sequential algorithm ***SECNOP*** for finding the shortest coverings of smaller Boolean matrices.

Keywords: Boolean matrix, shortest covering, decomposition, sequential algorithm, parallel algorithm, cluster.

1. Introduction and preliminary observations

The NP-hard shortest covering problem can be expressed as follows. It is required to find such a subset of rows of the given Boolean $\boldsymbol{m \times n}$ matrix (BM), where $\boldsymbol{m \geq n}$, that each of the columns has one in some row from this subset, the number of these rows being minimal [1].

The algorithms and the corresponding computer programs for solving the shortest covering problem are widely applied in the theory of complex systems, computing systems and also for planning of resources distribution in GRID [2]. That's why various algorithms (Petric's method, greedy algorithm, *minimax* algorithm and other [1]) have been proposed. Corresponding computer programs were developed and used in the design of discrete devices (ESPRESSO-II, GANP, Tie) [3].

As a rule, the effectiveness of the proposed algorithms and computer programs depends on the specific of input data.

To particular, the so-called sparse BMs, were researched sparse BM, where the probability of the occurrence of 1s among the elements of BM is less than ***0,05.*** The sequential programs for finding of the shortest coverings of sparse Boolean matrices were developed and published in [3].

Unfortunately, most of the BMs are not sparse. This fact implies continuing development of algorithms and computer programs for finding the shortest covers of large Boolean matrices with different properties.

To solve a NP-hard task of large dimension, approximate heuristic algorithms are used. They do not guarantee an optimality but allow to get decisions sufficiently close to the optimal ones in a reasonable time.

However, an even approximate algorithm requires too much computing for solving of the NP- hard large-scale problems. Therefore increasing the efficiency of solutions of the above mentioned tasks by using modern multiprocessor computing systems (computer clusters) arises as an emergent problem.

A computer cluster consists of a set of loosely or tightly connected computing processors that work together for solving tasks of smaller dimensions. The computing processors send the obtained results to the control processor in order to "summarize" them into a task solution of larger dimension. Computer clusters are controlled and scheduled by special software.

In order to bring solving difficult problem of large size to solving several tasks of smaller sizes using a computer cluster it is necessary to solve the problem of decomposition of input data and to develop a parallel algorithm for solving the problem on the cluster.

In this way, along with a parallel algorithm, an abstract computing system with one control processor $\boldsymbol{p_0}$ and several computing processors $\boldsymbol{p_i}$, are proposed. The sequential algorithms are components of our parallel algorithm. The sequential algorithms in the parallel algorithm interact in accordance with the timetable drawn up (computer schedule).

To address the issue of finding the shortest covering of large Boolean matrices using decomposition of initial large matrices into matrices of smaller sizes, a parallel algorithm was proposed by the author [4]. This parallel algorithm is based on the partitioning of the Boolean matrix into blocks of columns (column minors). A column minor of a matrix is the part of the matrix formed by some subset of columns.

The corresponding parallel program ***POKRMB*** was written by Adam Adamus in C++ in the integrated development environment Dev-C++ version 4.9.9.2 for Windows and Linux systems using environment MPI to communicate between nodes of our cluster [5].

After the testing of the program (at first on a typical PC, then on the cluster of the Siedlce University of Natural Sciences and Humanities) the efficiency of our parallel program ***POKRMB*** was researched. The results of these studies are published in [6].

We propose two options of a parallel algorithm for finding the shortest covering of a large Boolean $\boldsymbol{m \times n}$ matrix $\boldsymbol{M}$ based on other principles. The decomposition of the initial large matrix into matrices of smaller sizes is based on the partitioning of rows.

The parallel algorithm ***COVMB*** uses the sequential algorithm ***SECNOP*** for finding the shortest coverings of smaller Boolean matrices. The algorithm ***SECNOP*** was proposed and programmed by Adrian Nogal in C++ [7]. The corresponding program ***SECNOP*** proved to be more effective than the program ***POKRMB*** for finding the shortest coverings of smaller Boolean matrices [7].

2. Parallel algorithm COVMB

To parallel the computations with the help of the parallel algorithm ***COVMB(M;P(M))*** it is necessary to perform the following 11 steps:

1) Partitioning the initial matrix $\boldsymbol{M}$ *into matrices* $\boldsymbol{M_1, M_2, ..., M_T}$

The control processor $\boldsymbol{p_0}$ partitions the Boolean $\boldsymbol{m \times n}$ matrix $\boldsymbol{M}$ into $\boldsymbol{T}$ blocks of rows (row minors) with the help of the algorithm $\boldsymbol{A1(M; M_1, M_2, ..., M_T)}$. A row minor of a matrix is the part of the matrix formed by some subset of rows. In other words the row minors of the $\boldsymbol{m \times n}$ Boolean matrix $\boldsymbol{M}$ are its submatrices $\boldsymbol{M_1, M_2, ..., M_T}$ of smaller sizes $\boldsymbol{q \times n}$, where $\boldsymbol{q=[m/T]}$ for the matrices $\boldsymbol{M_1, M_2, ..., M_{T-1}}$ and $\boldsymbol{q=m-[m/T]*(T-1)}$ for the matrix $\boldsymbol{M_T}$. It's conveniently to put $\boldsymbol{T = [\sqrt{m}\,]}$.

Then the control processor $\boldsymbol{p_0}$ sends the matrices $\boldsymbol{M_1, M_2, ..., M_T}$ to processing processors $\boldsymbol{p_1, \ldots, p_T}$ as input data. The transition to p. *2*.

2) Parallel summation rows in the matrices $\boldsymbol{M_1, M_2, ..., M_T}$

Each processing processor $\boldsymbol{p_i}$, by using the algorithm $\boldsymbol{A2(M_i; s_i)}$, executes the logical summation of rows in the matrix $\boldsymbol{M_i}$, where $\boldsymbol{i \in \{1,2,\ldots,T\}}$, with the help of the operation *disjunction*. The result of this summation is the $\boldsymbol{n}$-component Boolean vector $\boldsymbol{s_i = r_{i1} \vee r_{i2} \vee \ldots \vee r_{iq}}$, where $\boldsymbol{r_{ij}}$ is a row of the matrix $\boldsymbol{M_i}$ **.**

The processing processor $\boldsymbol{p_i}$ sends $\boldsymbol{s_i}$ to the control processor $\boldsymbol{p_0}$.

The transition to p. *3*.

3) Building the support Boolean matrix $\boldsymbol{M^*}$

The control processor $\boldsymbol{p_0}$, using the algorithm $\boldsymbol{A3(s_1, s_2, ..., s_T; M^*)}$, firstly analyzes the vectors $\boldsymbol{s_1, s_2, \ldots, s_T}$ obtained from the processing processors $\boldsymbol{p_1, p_2, \ldots, p_T}$.

If each component of the vector $\boldsymbol{s_i}$ (corresponding to the matrix $\boldsymbol{M_i}$) is equal to $\boldsymbol{1}$, then the process of finding the shortest covering of the Boolean $\boldsymbol{m \times n}$ matrix $\boldsymbol{M}$ boils down to finding the shortest covering of the Boolean $\boldsymbol{q \times n}$ matrix $\boldsymbol{M_i}$, where $\boldsymbol{q \leq m-[m/T]*(T-1)}$. After that $\boldsymbol{p_0}$ puts $\boldsymbol{Mq := M_i}$ and moves to p. *10*.

However, this situation is a particular incident, which may happen for "tight" Boolean matrices.

In the general case, the control processor $\boldsymbol{p_0}$ builds the support $\boldsymbol{T \times n}$ matrix $\boldsymbol{M^*}$, the rows which are vectors $\boldsymbol{s_1, s_2, ..., s_T}$, sends $\boldsymbol{M^*}$ to the processing processor $\boldsymbol{p_1}$ and moves to p. *4*.

4) Finding of the shortest covering of the support Boolean matrix $\boldsymbol{M^*}$

The processor $\boldsymbol{p_1}$, using the sequential algorithm ***SECNOP(M*; P(M*))***, finds the shortest covering of the Boolean $\boldsymbol{T \times n}$ matrix $\boldsymbol{M^*}$, where $\boldsymbol{T = [\sqrt{m}\,]}$.

The result of the implementation of the algorithm ***SECNOP(M*; P(M*))*** is the subset of row names from $\boldsymbol{M^*}$. The elements of $\boldsymbol{P(M^*)}$ determine what $\boldsymbol{q \times n}$ matrices of the much smaller sizes (in comparison with the size of MB $\boldsymbol{M = M_1 \cup M_2 \cup \ldots \cup M_T}$) is necessary to explore to find the shortest covering of the Boolean matrix $\boldsymbol{M}$. The program ***SECNOP*** finds the shortest covering of the Boolean $\boldsymbol{16000 \times 16000}$ matrix [7].

Obviously, $\boldsymbol{|\, P(M^*) \,| = l \leq T = [\sqrt{m}\,]}$.

The transition to p. *5*.

5) Construction of the special $(q+1) \times n$ *matrices* M_i'

The processor p_0, using the algorithm $A4(P(M^*), M^*, M_1, M_2, ..., M_T; M_{i1}', M_{i2}', \ldots, M_{il}')$, constructs the $(q+1) \times n$ matrices $M_{i1}', M_{i2}', \ldots, M_{il}'$.

To construct the matrix M_i', p_0 first finds the corresponding Boolean vector s_i in the M^* and the corresponding Boolean $q \times n$ matrix M_i. After that p_0 inverts the vector s_i and writes the $\neg s_i$ to the matrix M_i as the additional row ($r_0 = \neg s_i$) to complete the construction of this special matrix M_i'. At last, p_0 sends the $(q+1) \times n$ matrices M_{ij}' to processing processors $p_1, p_2, \ldots, p_l$. The transition to p. *6*.

6) Parallel finding of the shortest coverings of the matrices M_i'

Each processing processor p_i with $i \in \{1,2,\ldots,l\}$, by using the sequential algorithm $SECNOP(M_i';P(M_i'))$, finds the shortest covering of the special $(q+1) \times n$ matrix M_i'.

After that p_i sends the solution (the shortest covering $P(M_i')$) to the control processor and moves to p. *7*.

7) Summation of shortest coverings of the matrices M_i'

By using the algorithm $A5(P(M_1'),\ldots, P(M_l'); P'(M), Mr, r_1,\ldots, r_t)$, the processor p_0 adds together the solutions obtained by the processing processors $p_1, p_2, \ldots, p_l$, i.e. $P'(M) = P(M_1') \cup \ldots \cup P(M_l')$, and deletes the item r_0 from it. The covering $P'(M)$ may contain redundant elements (row numbers of the initial matrix M). To eliminate redundant elements from $P'(M)$, the processor p_0 constructs the Boolean matrix Mr. The rows of the matrix Mr are the rows of the initial matrix M with row numbers included in the set $P'(M)$. The processor p_0 writes a new row r^* with the number $j(r^*)$ from $P'(M)$ into Mr, if the condition $r^* \wedge r_i \neq r^*$ for each row r_i from Mr is satisfied. Otherwise, p_0 removes $j(r^*)$ from the set $P'(M)$ and does not writes the row r^* into Mr. The Boolean $t \times n$ matrix Mr will be used for the reduction of the redundant elements in the covering of M. The remaining elements of the set $P'(M)$ form the set $P'(Mr)$. Obviously, $P'(Mr) \subseteq P'(M)$ and $|P'(Mr)|=t$.

Then the control processor p_0 sends to the processing processors $p_1, p_2, \ldots, p_t$ the following input data: 1) the set $P'(Mr)$, 2) the $t \times n$ matrix Mr, 3) the row name r_i for checking of the redundancy of the corresponding element of $P'(Mr)$. The transition to p. *8*.

8) Parallel elimination of redundant elements from $P'(Mr)$

An element of $P'(Mr)$ is redundant for building the shortest covering of the matrix Mr, if after cancellation the corresponding row from Mr all columns of Mr' can be covered by the disjunction of the remaining rows.

Each processing processor p_i, where $i \in \{1,2,\ldots,t\}$, using the sequential algorithm $A6(P'(Mr), Mr, r_i; P_i(Mr))$, first modernizes the matrix Mr by replacing each 1s by 0s in the row r_i. Then it executes logical summation rows in the modernized matrix Mr' with the help of the operation *disjunction*. The result of this summation is the n-component Boolean vector $s_i = r_{i1} \vee r_{i2} \vee \ldots \vee r_{it}$.

After that $\boldsymbol{p_i}$ analyzes the vector $\boldsymbol{s_i}$. If some component of the vector $\boldsymbol{s_i}$ is equal to 0, then the corresponding to $\boldsymbol{r_i}$ element of the set ***P'(Mr)*** is not redundant. In this case, the processor $\boldsymbol{p_i}$ sends the set $\boldsymbol{P_i(Mr)= P'(Mr)}$ to the control processor $\boldsymbol{p_0}$ and moves to p. *9*.

Otherwise, if $\boldsymbol{s_i =11\dots1}$, the corresponding to the $\boldsymbol{r_i}$ element of the set ***P'(Mr)*** is redundant. The processor $\boldsymbol{p_i}$ removes it from the set ***P'(Mr)*** and selects a new row ***r**** from ***Mr'.*** Then $\boldsymbol{p_i}$ modernizes ***Mr'*** by replacing each 1 by 0 in the row ***r****, summarizes the rows of the modernized matrix ***Mr''*** and analyzes the new vector $\boldsymbol{s_i^*}$.

If $\boldsymbol{s_i^* =11\dots1}$, then the corresponding to ***r**** element of the set ***P'(Mr)*** is redundant and it must be removed from the set ***P'(Mr)***. After that $\boldsymbol{p_i}$ selects the new row ***r**** from ***Mr''.***

Otherwise, if some component of the vector $\boldsymbol{s_i^*}$ is equal to 0, then the corresponding to ***r**** element of the set ***P'(Mr)*** is not redundant and it must be in the set ***P'(Mr)***. In this case, $\boldsymbol{p_i}$ "restores" each 1s in ***r**** and selects the new row ***r**** from ***Mr'***.

This process ends when all ***t*** the rows of the matrix ***Mr*** are investigated**.** Then $\boldsymbol{p_i}$ sends the set $\boldsymbol{P_i(Mr) \subseteq P'(Mr)}$ to the control processor $\boldsymbol{p_0}$ and moves to p. *9*.

9) The finding the shortest covering of the initial Boolean matrix ***M***

The control processor $\boldsymbol{p_0}$, using the algorithm $\boldsymbol{A7(P_1(Mr), P_2(Mr),\dots, P_t(Mr); P(M))}$, where $\boldsymbol{t= |\,P'(M)| < m}$, selects the shortest covering from the obtained sets $\boldsymbol{P_1(Mr), P_2(Mr),\dots, P_t(Mr)}$. The transition to p. *11*.

10) The finding the shortest covering of the tight initial Boolean matrix

By using the sequential algorithm ***SECNOP(Mq; P(M)),*** the control processor $\boldsymbol{p_1}$ finds the shortest covering of the Boolean matrix ***Mq***, i.e. the set ***P(Mq)***, and puts ***P(Mq)=P(M).*** The transition to p. *11*.

11) The ending of the computing

The control processor $\boldsymbol{p_0}$ ends the finding the shortest covering of our initial Boolean $\boldsymbol{m \times n}$ matrix ***M.***

The proposed parallel algorithm ***COVMB*** for finding the shortest covering of a large Boolean matrix using rows-decomposition implements the following computer schedule:

$$\boldsymbol{H(COVMB)=((A1,p_0),(A2,p_1,\dots,p_T),(A3,p_0),(SECNOP,p_1),(A4,p_0), (SECNOP, p_1,\dots,p_l), (A5,p_0), (A6, p_1, p_2 ,\dots, p_t), (A7, p_0), (SECNOP,p_1), (A8, p_0))}\ ,$$

where a record $\boldsymbol{(Aj,p_i)}$ indicates that the processor $\boldsymbol{p_i}$ performs the algorithm ***Aj***; ***A1*** - algorithm for partitioning of the initial matrix ***M*** into ***T*** submatrices; ***A2*** - algorithm for logical summation of rows in the matrices; ***A3*** - algorithm for building of the support Boolean matrix ***M*;*** ***SECNOP*** - sequential algorithm for finding the optimal covering of a Boolean matrix; ***A4*** - algorithm for construction of the special $\boldsymbol{(q+1) \times n}$ matrices $\boldsymbol{M_i'}$; ***A5*** - algorithm for summation shortest covers of the special matrices $\boldsymbol{(P'(M) = P(M_1') \cup \dots \cup P(M_l'))}$, construction of the matrix ***Mr*** and preparation of the data for elimination of the redundant elements; ***A6*** – algorithm for elimination of the redundant elements from the covering ***P'(Mr); A7*** – algorithm for selecting the shortest coverings from $\boldsymbol{P_1(Mr)}$**,**

$P_2(Mr)$,…, $P_t(Mr)$, A8 – algorithm for ending the finding the shortest covering of our initial Boolean $\boldsymbol{m\times n}$ matrix ***M.***

3. Example

Let us find the shortest covering of the ***20×12*** matrix ***M***

$$\begin{array}{r|cccccccccccc}
 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} \\
r_1: & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
r_2: & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
r_3: & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
r_4: & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
r_5: & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
r_6: & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
r_7: & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
r_8: & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
r_9: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
r_{10}: & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
r_{11}: & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
r_{12}: & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
r_{13}: & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
r_{14}: & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
r_{15}: & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
r_{16}: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
r_{17}: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
r_{18}: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
r_{19}: & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
r_{20}: & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{array}$$

1. The control processor performs a partition of ***M*** into four blocks and sends the matrices M_1, M_2, M_3, M_4 to the processing processors p_1, p_2, p_3, p_4.

M_1

r_1:	0	0	0	0	1	0	0	0	0	0	1	0
r_2:	0	1	0	0	0	0	0	0	0	0	1	0
r_3:	0	0	0	0	1	1	0	0	0	0	0	1
r_4:	0	0	1	0	1	0	1	0	0	0	0	0
r_5:	0	0	0	0	0	1	0	0	0	0	1	0

M_2

r_6:	0	0	0	0	1	0	0	0	0	0	0	0
r_7:	1	0	1	0	0	0	0	0	0	0	0	0
r_8:	0	1	0	0	1	0	0	1	1	0	0	0
r_9:	0	0	0	0	0	0	0	0	1	0	0	0
r_{10}:	0	0	1	0	1	0	0	0	1	0	0	0

M_3

r_{11}:	0	0	1	0	1	0	0	0	0	1	0	0
r_{12}:	1	0	0	0	0	1	0	0	1	0	0	0
r_{13}:	0	0	0	0	0	1	0	1	0	0	1	0
r_{14}:	1	0	1	0	0	0	0	0	1	0	0	0
r_{15}:	1	0	0	0	1	0	0	0	0	0	0	0

M_4

r_{16}:	0	0	0	0	0	0	0	0	1	0	0	0
r_{17}:	0	0	0	0	0	0	0	1	0	1	1	0
r_{18}:	0	0	0	0	0	0	0	0	1	1	0	0
r_{19}:	1	0	0	0	0	0	0	0	0	0	0	0
r_{20}:	0	0	0	1	0	0	0	0	1	0	1	0

2. Each processing processor p_i, where $i \in \{1,2,\ldots,4\}$, executes the logical summation of the rows of the matrix M_i, using the operation disjunction, and sends the vector s_i to p_0.

$s_1 = r_1 \vee r_2 \vee r_3 \vee r_4 \vee r_5$ = 0 1 1 0 11 1 0 0 0 1 1

$s_2 = r_6 \vee r_7 \vee r_8 \vee r_9 \vee r_{10}$ = 1 1 1 0 1 0 0 1 1 0 0 0

$s_3 = r_{11} \vee r_{12} \vee r_{13} \vee r_{14} \vee r_{15}$ = 1 0 1 0 1 1 0 1 1 1 1 0

$s_4 = r_{16} \vee r_{17} \vee r_{18} \vee r_{19} \vee r_{20}$ =1 0 0 1 0 0 0 1 1 1 1 0

3. The control processor p_0 constructs the 4×12 matrix M^* :

s_1	0	1	1	0	1	1	1	0	0	0	1	1
s_2	1	1	1	0	1	0	0	1	1	0	0	0
s_3	1	0	1	0	1	1	0	1	1	1	1	0
s_4	1	0	0	1	0	0	0	1	1	1	1	0

4. The processor p_1 finds the shortest covering of the Boolean 4×12 matrix M^*.

As a result, we obtain $P(M^*) = \{s_1, s_4\}$.

5. The control processor $\boldsymbol{p_0}$ constructs the matrices $\boldsymbol{M_1'}$ and $\boldsymbol{M_4'}$, using matrices $\boldsymbol{M_1}, \boldsymbol{M_4}$ and vectors $\boldsymbol{s_1}$, $\boldsymbol{s_4}$.

$$\boldsymbol{M_1'} \qquad \begin{array}{lcccccccccccc}
\boldsymbol{r_0}: & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
\boldsymbol{r_1}: & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\boldsymbol{r_2}: & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\boldsymbol{r_3}: & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
\boldsymbol{r_4}: & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{r_5}: & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

$$\boldsymbol{M_4'} \qquad \begin{array}{lcccccccccccc}
\boldsymbol{r_0}: & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
\boldsymbol{r_{16}}: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\boldsymbol{r_{17}}: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
\boldsymbol{r_{18}}: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
\boldsymbol{r_{19}}: & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\boldsymbol{r_{20}}: & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{array}$$

6. The processing processors $\boldsymbol{p_1}$ and $\boldsymbol{p_4}$ find the shortest coverings of $\boldsymbol{M_1'}$ and $\boldsymbol{M_4}$' and obtain $\boldsymbol{P(M_1')=\{ r_0 , r_2, r_3, r_4\}}$, $\boldsymbol{P(M_4')=\{ r_0 , r_{17}, r_{19}, r_{20}\}}$.

7. The control processor $\boldsymbol{p_0}$ adds together $\boldsymbol{P(M_1')}$ and $\boldsymbol{P(M_4')}$ and deletes the item $\boldsymbol{r_0}$ from it. We have $\boldsymbol{P'(Mr) = P'(M)=\{ r_2, r_3, r_4, r_{17}, r_{19}, r_{20}\}}$.

8,9**.** To exclude redundant elements, it is necessary to analyze the rows of ***Mr***:

$$\begin{array}{lcccccccccccc}
r_2: & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
r_3: & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
r_4: & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
r_{17}: & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
r_{19}: & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
r_{20}: & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0
\end{array}$$

Each row of ***Mr*** is important. Each row contains the element 1, which is the only one in the corresponding column of ***Mr***. Deleting at least one element from ***P'(Mr)***, we'll obtain a set, which is not the shortest covering of the BM ***Mr***.

We conclude that our $\boldsymbol{P'(Mr) =\{ r_2, r_3, r_4, r_{17}, r_{19}, r_{20}\}= P(M)}$ is the shortest covering of the initial $\boldsymbol{20{\times}12}$ matrix ***M***.

4. An interesting option of the parallel algorithm COVMB

We can propose also the option of the parallel algorithm ***COVMB(M;P(M))*** for finding the optimal covering of a $\boldsymbol{m{\times}n}$ Boolean matrix.

The algorithm ***COVMB'(M;P(M))*** finds the solution performing the following 7 steps:

1) Partitioning the initial matrix $\boldsymbol{M}$ *into* $\boldsymbol{T}$ *matrices* $\boldsymbol{M_1, M_2, ..., M_T}$

Similarly as in p.*1* of the algorithm ***COVMB(M;P(M))***, the control processor $\boldsymbol{p_0}$ partitions the $\boldsymbol{m \times n}$ matrix $\boldsymbol{M}$ into blocs of rows (row minors) with the help of the algorithm ***A1(M; M₁, M₂,..., M_T)***.

After that $\boldsymbol{p_0}$ sends the $\boldsymbol{q \times n}$ matrices $\boldsymbol{M_1, M_2, ..., M_T}$ to processing processors $\boldsymbol{p_1,\dots,p_T}$ as input data. The transition to p. *2*.

2) Parallel construction of the special $\boldsymbol{(q+1) \times n}$ *matrices* $\boldsymbol{M_i'}$

Each processing processor $\boldsymbol{p_i}$, with $\boldsymbol{i \in \{1,2,\dots,T\}}$, by using the algorithm ***A2M(Mᵢ; Mᵢ')***, first executes the logical summation of rows in the matrix $\boldsymbol{M_i}$, where $\boldsymbol{i \in \{1,2,\dots,T\}}$, with the help of the operation *disjunction*. Then $\boldsymbol{p_i}$ analyzes the vector $\boldsymbol{s_i == r_{i1} \vee r_{i2} \vee \dots \vee r_{iq}}$, where $\boldsymbol{r_{ij}}$ is a row of the matrix $\boldsymbol{M_i}$. If $\boldsymbol{s_i = 11\dots1}$, then the process of finding the shortest covering of the Boolean $\boldsymbol{m \times n}$ matrix $\boldsymbol{M}$ boils down to the finding of the shortest covering of the Boolean $\boldsymbol{q \times n}$ matrix $\boldsymbol{M_i}$, where $\boldsymbol{q \le m-[m/T]*(T-1)}$. In this case, the processing processor $\boldsymbol{p_i}$ sends $\boldsymbol{M_i' = M_i}$ to $\boldsymbol{p_0}$ and moves to p. *3*. Otherwise, it constructs the $\boldsymbol{(q+1) \times n}$ matrix $\boldsymbol{M_i'}$. The processor $\boldsymbol{p_i}$ inverts the vector $\boldsymbol{s_i}$ and writes $\boldsymbol{\neg s_i}$ into the matrix $\boldsymbol{M_i}$ as an additional row ($\boldsymbol{r_0 = \neg s_i}$) to complete the construction of a special matrix $\boldsymbol{M_i'}$. After that $\boldsymbol{p_i}$ sends $\boldsymbol{M_i'}$ to $\boldsymbol{p_0}$ and moves to p. *3*.

3) Preparation of data for processing processors

The control processor $\boldsymbol{p_0}$ prepares the data for processing processors, using the algorithm ***A3(Mᵢ, Mᵢ'; Mr, M₁',…, M_T')***.

If $\boldsymbol{M_i' = M_i}$, the control processor $\boldsymbol{p_0}$ puts $\boldsymbol{Mr := M_i}$ and moves to p. *6*.

Otherwise, it sends $\boldsymbol{M_i'}$ to $\boldsymbol{p_i}$ and moves to p. *4*.

4) Parallel finding of the shortest coverings of the matrices $\boldsymbol{M_i'}$

Each processing processor $\boldsymbol{p_i}$, with $\boldsymbol{i \in \{1,2,\dots,T\}}$, by using the sequential algorithm ***SECNOP(Mᵢ';P(Mᵢ'))***, finds the shortest covering of the special $\boldsymbol{(q+1) \times n}$ matrix $\boldsymbol{M_i'}$.

After that $\boldsymbol{p_i}$ sends the solution (the shortest covering $\boldsymbol{P(M_i')}$) to the control processor $\boldsymbol{p_0}$ and moves to p. *5*.

5) Construction of the covering $\boldsymbol{P'(Mr)}$

Similarly as in p.*7* of the algorithm ***COVMB(M;P(M))***, by using the algorithm ***A5(P(M₁'),…, P(M_T'); P'(Mr), Mr)***, the control processor $\boldsymbol{p_0}$ adds together the solutions obtained by the processing processors $\boldsymbol{p_1, p_2, \dots, p_T}$, deletes the item $\boldsymbol{r_0}$, builds the Boolean matrix $\boldsymbol{Mr}$ and constructs the covering $\boldsymbol{P'(Mr) \subseteq P'(M) = P(M_1') \cup \dots \cup P(M_T')}$.

The transition to p. *6*.

6) The finding of the shortest coverings of the matrix $\boldsymbol{Mr}$

The control processor p_0 and processing processors $p_1,\ldots, p_k$ by using the parallel algorithm ***PSECNOP(Mr; P(Mr))***, finds the shortest covering ***P(Mr)*** of the $t \times n$ matrix ***Mr,*** where $t= | P'(Mr)| < m$, $k \le Max(t,n)$, and puts ***P(Mr)=P(M).*** The transition to p. *7.*

7) The ending of the computing

The control processor p_0 ends the finding the shortest covering ***P(M)*** of our initial Boolean $m\times n$ matrix ***M***.

The parallel algorithm ***COVMB'*** for finding the shortest covering of a Boolean matrix of large dimension using rows-decomposition implements the following computer schedule:

$H(COVMB')=((A1,p_0),(A2M,p_1,\ldots,p_T), (A3,p_0), (SECNOP,p_1,\ldots,p_T), (A5,p_0), (PSECNOP, p_0 ,p_1,\ldots, p_k))$.

Using the parallel algorithm ***COVMB'*** for considered above the 20×12 matrix ***M***, we obtain $P(M_1')=\{r_0, r_2, r_3, r_4\}$, $P(M_2')=\{ r_0, r_7, r_8\}$, $P(M_3')=\{ r_0, r_{11}, r_{12}, r_{13}\}$, $P(M_4')=\{ r_0, r_{17}, r_{19}, r_{20} \}$ and $P'(M)=\{r_2, r_3, r_4, r_7, r_8, r_{11}, r_{12}, r_{13}, r_{17}, r_{19}, r_{20} \}$. To eliminate the redundant elements from the ***P'(M)***, it is necessary to find the shortest covering of the Boolean 11×12 matrix ***Mr***. As a result, we obtain $P(M)=\{ r_3, r_4, r_7, r_8, r_{11}, r_{20}\}$.

5. Conclusion

The computational complexity of the NP-hard problem of finding the shortest covering of a $m \times n$ Boolean matrix equals $O(2^m)$.

Labour-consuption of an algorithm, or time complexity, is estimated by the number of conditional elementary operations to be performed to solve the problem. For our problem, by conditional elementary operations one usually understands the *disjunction, conjunction* and *comparison* of *n*-component Boolean vectors.

The decomposition of an initial large $m \times n$ Boolean matrix on row minors allows to reduce finding of the shortest covering of the large Boolean matrix to finding the shortest coverings of several Boolean matrices with smaller sizes.

The proposed parallel algorithms ***COVMB*** and ***COVMB'***, which use rows-decomposition, are particularly effective in relation to tight Boolean matrices.

The finding of the shortest covering for some tight Boolean $m \times n$ matrix may be reduced to the finding the shortest covering of one corresponding Boolean $[\sqrt{m}] \times n$ matrix.

According to the algorithm ***COVMB***, we must build shortest coverings for $l < T$ special matrices $M'_{i1}, \ldots, M'_{il}$ only. This reduces the cost of the main task. The advantages of the ***COVMB*** algorithm should also include the ability to obtain not one but several solutions of the main task.

The ***COVMB'*** algorithm is simpler because it does not require to build the auxiliary Boolean matrix ***M**** and to find the corresponding shortest covering. By using the ***COVMB'***, it is required to build the shortest coverings for all $(q+1)\times n$ matrices M_1', ..., M_T'. All this makes possible to find a more effective solution.

Thus, we can conclude that the algorithms ***COVMB*** and ***COVMB'*** are competitive.

References

1. A. Zakrevskij, Yu. Pottosin, L. Cheremisinova. Combinatorial Algorithms of Discrete Mathematics. -Ed. A. Keevalik.- Tallin: TUT Press, 2008, 193 pages.

2. V.S. Ponomarenko, S.V. Listrovoj. Metod resheniya zadachi o minimal'nom pokrytii kak sredstvo planirovaniya v GRID. Problemy upravleniya. 2008, t. 3, pp. 78–84 (in Russian).

3. Leonchik P.V. Algoritm pokrytiya razrezhennyh bulevyh matric. Informatika, 2007, 2, pp. 53-61 (in Russian).

4. Novikov S.V. Rasparallelivanie vychislenij pri reshenii dvuh zadach postroeniya optimal'nyh pokrytij. Vestnik Grodnenskogo universiteta, Grodno, seriya 2, 1(92), 2010, pp. 30-36 (in Russian).

5. Adam Adamus. Równoległy program znalezienia minimalnego pokrycia macierzy Boole'a o dużych rozmiarach. Praca magisterska. Akademia Podlaska, Siedlce, 2010, 99 pages.

6. Sergey Novikov, Adam Adamus. Investigations of the efficiency of a parallel program for construction of the shortest covering of a Boolean matrix. Studia Informatica, Systemy i technologie informacyjne, VOLUME 1-2(14)2010, Wyd. UPH, Siedlce, 2011, pp. 67-76.

7. Adrian Nogal. Algorytm znalezienia minimalnego pokrycia macierzy Boole'a i implementacja równoległego programu na komputerze wysokiej wydajności. Praca dyplomowa inżynierska. Uniwersytet Przyrodniczo-Humanistyczny w Siedlcach, Siedlce, 2014, 40 pages.