Lovrenčić Alen
THT Čakovec

UDC: 007.5
Original Scientific Paper

The Problem of Optimization of the Process of Decomposition of an Information System

*While developing larger information systems, it is necessary to decompose them into subsystems for the purpose of easier designing. The intention is, for decomposition to be performed in a way that results in as small losses in system quality as possible. Conse quently, there is, a need for defining the system decomposition quality parameters. Once the parameters are defined, the obvious question arises of an optimal system decomposition under particular conditions as well as of its time complexity. The respective research shows that the system decomposing algorithm should, in the worst case, be of exponential time complexity, the task belonging to the class of the **NP-complete** tasks.*

Key words: algorithm, complexity, process, information system design.

1. Introduction

An information system should reflect the processes within the object system it supports. This does not mean that, when informatization is implied, the existing object system should be converted as a whole. However, the changes within an informatization system that are certain to improve it should be accompanied by respective changes in the object system. If we want to perform such changes, our aim or ideal should be a completely optimized object system and, accordingly, information system as well. It is to be pointed out that improvement in a part of the system need not result in improvement of the system as a whole. This is why each analysis by means of which the system is broken down into its elements should be performed cautiously and be fully justified. The decomposition is to be performed in a way that should result in as small loss in the system quality as possible. This is why the decomposition quality parameters are to be defined, as well as the algorithm for the optimal division of the system into its subsystems.

2. The definition of terms and the hypothesis

Let us define the term "process" first.

Definition 1: A **process** is an array of actions which, by using particular sources within a business system and a particular time-span, attain a particular objective.

Processes within a system use and create information stored in different media inside or outside the system. The processes are thus related to information and respective media and it is natural, therefore, to define a data class as:

Definition 2: A **data class** is a document or a record of any format created by a process inside or outside the system ,used by one or more processes inside or outside the system.

The processes and their relations with data classes can be shown by means of a matrix in which the rows present particular processes within the system, whereas the columns present the data classes used and created by the processes. A "C" shall mark the field in which a process line intersects the column of a data class it creates, whereas a "U" shall mark the field in which a process line intersects the column of a data class it uses. The theory of making and usage of a process-data class matrix has been shown by (Brumec, 1993)

Definition 3: Processes $\mathbf{p_i}$ and $\mathbf{p_j}$ are said to be **connected** by data class $\mathbf{c_m}$ if one of them creates the class and the other uses it.

Connections among processes within a system are not of the same importance, so each connection can be attached the respective weight. We assume that a doubled weight of a connection implies its doubled importance for the system. On the basis of the connections among processes that have just been defined, the parameters of system decomposition quality are to be defined . A system is decomposed into a certain number of subsystems in a way that provides for each process to correspond exactly to one subsystem. The processes within the same subsystems are called **clusters**.

Definition 4: Let $\mathbf{p_i}$ and $\mathbf{p_j}$ be two processes and let $\mathbf{c_1,..,c_k}$ be classes being created or used by both processes. Let connections that make up these classes have weights $\mathbf{w_1,...,w_k}$, $\mathbf{w_i \geq 0, i=1,...,k}$. Then we define the **adhesion between processes** $\mathbf{p_i}$ **i** $\mathbf{p_j}$ as

$$\mathbf{Ad(p_i,p_j) = \sum_{i=1}^{k} w_i} \qquad (1)$$

Definition 5: Let $\mathbf{k_i}$ and $\mathbf{k_j}$ be two clusters and let the cluster $\mathbf{k_i}$ be composed of the processes $\mathbf{p^{(i)}{}_1,...,p^{(i)}{}_m}$, and the cluster $\mathbf{k_j}$ of the processes $\mathbf{p^{(j)}{}_1,...,p^{(j)}{}_n}$. Then the **adhesion between clusters** $\mathbf{k_i}$ and $\mathbf{k_j}$ is

$$\mathbf{Ad(k_i,k_j) = \sum_{q=1}^{m}\sum_{r=1}^{n} Ad(p_q^{(i)},p_r^{(j)})} \qquad (2)$$

Definition 6: Let **S** be a system decomposed into clusters $\mathbf{k_1,...,k_m}$. Its total **system adhesion** is

$$\mathbf{Ad(S) = \sum_{i=1}^{m}\sum_{\substack{j=1,\\ j\neq i}}^{m} Ad(k_i,k_j)} \qquad (3)$$

As the opposite of adhesion, the subsystem cohesion stands for connections among the processes within the same subsystem. However, with the number of connections within a system being unchangeable regardless of the system decomposition, it is clear that adhesion and cohesion are complementary, i.e. an increase in adhesion implicates a decrease in cohesion and vice versa.

After the definition of adhesion that provides for the parameters of connections between two processes and two subsystems (clusters) has been given, it is possible to set the criteria for system decomposition quality. The system decomposition quality is reversely proportional to adhesion among clusters.

When decomposing a system, our objective shall be to break down the system into subsystems in a way that is to provide for the minimal adhesion possible. Attention should be drawn to the following facts:

- As to minimal adhesion, the most optimal is not to decompose the system at all. Namely, in that case there is only one subsystem and it is identical to the whole system so all connections are internal and cohesion is absolute, i.e. there is no adhesion. This is why optimization requires defining the minimal number of clusters we want to create.
- If we determine the minimal number of clusters we want to create, the optimal decomposition on the basis of adhesion can always be attained even with the minimal cluster number allowed. If a system with more clusters than defined is implied, two clusters can easily be merged into one, which reduces the number of outside connections and increases the number of internal connections, i.e. cohesion is increased (or, at worst, remains the same). Cohesion is increased by exactly those connections that connect the processes in the two merged clusters.

Having defined adhesion, a hypothesis is made to prove it.

Hypothesis 1: There is no class **P** (polynomial) algorithm of a system decomposition into a defined number of clusters in an optimal way regarding minimal system adhesion.

3. Mathematical terms

The adhesion properties need are contained in the following theorem:

Theorem 1: Let $\mathbf{k_i}$ and $\mathbf{k_j}$ be clusters

(1) $\mathbf{Ad(k_i, k_j) \geq 0}$

(2) $\mathbf{Ad(k_i, k_j) = Ad(k_j, k_i)}$

Proof: Trivial. Directly in accordance with definitions 5 and 6.

Q.E.D.

In order to simplify the calculation of adhesion between two clusters, let us prove the following proposition:

Proposition 1: Let there be two clusters, $\mathbf{k_i}$ and $\mathbf{k_j}$ and let the cluster $\mathbf{k_i}$ be composed of processes $\mathbf{p^{(i)}_1,\ldots,p^{(i)}_m}$, and the cluster $\mathbf{k_j}$ be composed of processes $\mathbf{p^{(j)}_1,\ldots,p^{(j)}_n}$. Besides, let $\mathbf{c_1,\ldots,c_k}$ all be mutual classes for any two processes from two clusters. Let there be the weights of connections, marked as $\mathbf{w}^k_{ij}$, among the processes in accordance weith class $\mathbf{c}_k$. Here we assume that, when two processes do not have class in common, the weight of their connection in accordance with the class equals 0. The following is correct then:

$$\mathbf{Ad(k_i,k_j) = \sum_{p=1}^{m}\sum_{q=1}^{n}\sum_{r=1}^{k} w^r_{pq}} \tag{4}$$

Proof: In accordance with the equation (2)

$$\mathbf{Ad(k_i,k_j) = \sum_{p=1}^{m}\sum_{q=1}^{n} Ad(p^{(i)}_q, p^{(j)}_r)}$$

However, in accordance with equation (1), this means

$$\mathbf{Ad(k_i,k_j) = \sum_{p=1}^{m}\sum_{q=1}^{n}\sum_{r=1}^{k} w^r_{pq}}$$

because the weights of non-existing connections, i.e. connections for data classes that are not mutual for two processes, equal 0.

Nevertheless, the sum commutativity implies that

$$\mathbf{Ad(k_i,k_j) = \sum_{p=1}^{m}\sum_{q=1}^{n}\sum_{r=1}^{k} w^r_{pq}}$$

Q.E.D.

This proof has provided for an important fact: the weight of connection between two clusters established by means of a class can be determined by summing the weights of connections established by the processes within the two clusters by means of the class.

Let us prove another proposition, whose expression is of notable significance.

Proposition 2: If the minimal number of clusters **m** to be created is given, then the optimal solution can always be attained with exactly **m** clusters.

Proof: Let us suppose that a solution with more than **m** clusters is attained. In that case we can take any two clusters $\mathbf{k_i}$ and $\mathbf{k_j}$ among the solution clusters and merge them into one, let us say $\mathbf{k_{ij}}$.

In this way all the outside connections, if there are were any, which connected the two clusters have been turned into inside connections. In other words, if the newly originated system is marked with $\mathbf{S_1}$, then it is

$$\mathbf{Ad(S_1) = Ad(S) - Ad(k_i,k_j) \leq Ad(S)}$$

That means that reducing clusters in this way does not increase the adhesion of the system.

By this procedure, consisting of a finite number of steps emerging from any system containing more than **m** clusters, we reach a system of exactly **m** clusters, the adhesion of which amounts to less or is equal to the adhesion of the original system.

Q.E.D.

4. Proof of NP-hardness

Reffering to (Aho, Hopcroft, Ullman, 1987) and (Knuth, 1973), in which the definitions have been given for an algorithm, nondeterministic algorithm, the time complexity of an algorithm and drawing on (Horowitz, Sahni, 1978) and (Garey, Johnson, 1979), where, besides the aforementioned definitions, the theory of **NP-hardness** and **NP-completeness** have been worked out, we can shortly Summarize under algorithm we can be consider a finite sequence of instructions in which every instruction is followed by an unambiguously conceived one; this sequence is executable in turn within a finite time-span. A nondeterministic algorithm differs from a deterministic in absence of the imperative of an unambiguously conceived instruction to follow the previous one; after an instruction is performed, there is choice as to the instruction that is to follow. Nondeterministic algorithms cannot be executed on computer, but their theoretical importance is significant. All the above-mentioned books also offer ways to work out asymptotic evaluations of the time complexity of algorithms. Only the basic definitions are given here.

From the very definition of algorithm it is obvious that in each instance of a problem it must be solved within a finite time. Our point of interest is determining the measure for the time an algorithm of an **n** size needs to be solved. The measure is given by the following definitions:

Definition 7: (Time complexity of an algorithm)
Let there be algorithm **A** that solves the problem. The **time complexity** of the algorithm **A** is the total of all particular time-spans of instructions as related to the quantity of input data that the algorithm is to perform in order to solve an each particular instance of the problem and to release the output into the surroundings.

Note that different instances of the same problem and of the same size can display different time complexities. This is why in different cases the top and bottom limits of the time complexity can be of special interest, as well as of the average complexity.

Definition of algorithm complexity approximations follows:

Definition 8: (algorithm complexity approximation)
If algorithm **A** is given and function $\mathbf{f_A(n)}$ is the function of algorithm complexity in relation to n, the measure of input size, it can be said that:

(1) $f_A(n) = o(g(n))$ if it is
$|f_A(n)| < c|g(n)|$
for a constant **c**

(2) $f_A(n) = O(g(n))$ if it is
$|f_A(n)| \leq c|g(n)|$
for a constant **c**

(3) $f_A(n) = \omega(g(n))$ if it is
$|f_A(n)| > c|g(n)|$
for a constant **c**

(4) $f_A(n) = \Omega(g(n))$ if it is
$|f_A(n)| \geq c|g(n)|$
for a constant **c**

(5) $f_A(n) = \Theta(g(n))$ if it is
$|f_A(n)| = c|g(n)|$
for a constant **c**

The next step defines the problem classes in accordence with time complexity. It also defines the basic theorem of **NP-completness**.

Definition 9: (polynomial reducibility)

Let problems P_1 and P_2 be given. We say that problem P_1 **can be reduced in polynomial time** to problem P_2, and we write down $P_1 \propto P_2$ if there is an algorithm **A** of the time complexity **O(p(n))**, where **p** is a polynomial, and if the algorithm provides transforming the problem P_1 into the problem P_2, i.e. if the solution of problem P_1 can be reduced to the solution of problem P_2 by means of algorithm **A.**

There is a proposition emerging directly from this definition:

Proposition 3:(transitivity)

If $P_1 \propto P_2$ and $P_2 \propto P_3$ then it is also $P_1 \propto P_3$

Proof: Trivial. Directly by definition 10.

Q.E.D.

Definition 10: (*NP* class)

We say that problem $P_1 \in$ **NP** if and only if it can be solved polynomially by means of a nondeterministic algorithm.

If **P** stands for the class containing all problems that can be solved in polynomial time by means of ordinary deterministic algorithms, the basic question it implies is: is **P=NP**? This question has not been answered yet, but there are problems that cannot be solved polynomially by presently known tools. These are the so-called **NP-complete** and **NP-hard** problems. The basic motivation for the previous statement is the basic theorem for a series of the **NP-completeness** theories. The theorem is based on reducing the polynomial solving of whole **NP** class problems to the polynomial solving of a single problem - problem of Conunctive normal form formula satisfiability (CNF):

Theorem 2: (Cook)

P=NP if and only if **CNF** $\in$ **P**.

The theorem shall not be proved here because its proof is by all means non-trivial and voluminous. Two different proofs of the theorem have been published in (Garey, Johnson, 1979) and (Horowitz, Sahni, 1978).

On the basis of this theorem, the following two definitions can be given:

Definition 11: (NP-hard problems)

Problem **P** is said to be **NP-hard** if **CNF** $\propto$ **P.**

Definition 12: (NP-complete problems)

Problem **P** is **NP-complete** if **CNF** $\propto$ **P** and if **P** $\in$ **NP.**

The previous definitions imply that it is sufficient to prove that a problem, already known to be **NP**-hard, can be polynomially reduced to the original problem in order to prove that the original problem is also **NP-hard.**

We shall prove that Clustering Problem (CP) is **NP-hard** and that there is no programming method known today by means of which it would be possible to solve a problem which is of the polynomial complexity. We shall also show that the known **NP-hard** problem of the maximal graph cut (MC) can be reduced to CP problem. The **NP-hardness** of **MC** problem has been taken from (Horowitz, Sahni 1978), page 546.

Let us define the problem first:

Definition 13: (Clustering Problem - CP)

m processes and **n** data classes have been given. Each process creates and uses a certain number of classes. It is important that there can not be two processes creating the same class. The connection between two processes is the data class they both use (or one of them uses the class and the other creates it). The weight of each connection is defined and it is not negative. **k** disjunct subsets (clusters) of the **P** process set are to be defined in a way which provides for the adhesion, i.e. the total of connections between connected clusters, to be minimal.

Definition 14: (Graph Minimal Clustering problem - GMC)

A non-oriented weight graph with positive weights is given. A partition of set **V** graph vertices is set to be made into **k** disjunct subsets,

$\mathbf{V_1, V_2, \ldots, V_k}$,

$\mathbf{V_i \cap V_j = \emptyset}$, $\mathbf{V_1 \cup V_2 \cup \ldots \cup V_k = V}$ so that the sum

$$\sum_{\substack{i \in V_p \\ j \in V_q \\ p \neq q}} \mathbf{w(i,j)} \rightarrow min \qquad (5)$$

Theorem 3: GMC $\propto$ CP

Proof: Each graph vertix represents one process. A new data class is created for each graph edge. In some way, the graph vertices are sorted and the vertix with a smaller mark value creates the data class defined as belonging to the edge and the class is used by a vertix with a greater mark value. Loops are not presented. The connection is attached the weight that the graph edge used to have.

Suppose that the graph has **n** vertices and **m** edges (**m≤n(n-1)<n^2**). Let, for example, vertices be situated in a sequence and edges in a two-dimensional array. It is not needed for the vertices to be presented once more, whereas at most **O(n^2)** readings and writings are necessary for the edges to be presented as a process-data class matrix. We can, therefore restructure the **GMC** into the **CP** problem in polynomial time.

Regarding that a separate class is created for each connection, each data class will have only one process that creates it and one that uses it. So, the **CP** problem is well-defined.

Let us show in addition that the solution of the **CP** problem provides at the same time the solution to the **GMC** problem. The clusters obtained by solving the **CP** problem are at the same time the clusters of the **GMC** problem, whereas the outside connections in the solution of the **CP** problem are the cut edges of the graph in the **GMC** problem; The graph edges and respective system connections are of the same weights and the total outside connections weights in the **CP** problem shall, consequently, make up the edges in the cut edges in the **GMC** problem.

Q.E.D.

Definition 15: (Max-Cut Problem - MC)

Let the weight graph **G=(V,E)** be given. A **S⊆V** set is to be defined so that:

$$\sum_{\substack{i \in S \\ j \notin S}} \mathbf{w(i,j) \rightarrow max} \qquad (6)$$

Theorem 4: MC $\propto$ GMC

Proof: Graph **G'(V,E')** is created, where the weight **w'(i,j) = -w(i,j)**. It is obvious that, in case of **w(i,j) ≥ 0**, it is **w'(i,j) ≤ 0**. It is also obvious, regarding that only the edges of the graph are altered (and their maximal number is $\frac{n(n-1)}{2}$) that, within the $O(n^2)$ life-span, a **G'** graph can be created out of the **MC** problem. It is also obvious that, if

$$\sum_{i\in S}\sum_{j\notin S} w(i,j) \to \max$$

then $S'\subseteq E$, $S'\neq S$

$$\sum_{i\in S}\sum_{j\notin S} w(i,j) \geq \sum_{i\in S'}\sum_{j\notin S'} w(i,j)$$

which implicates:

$$-\sum_{i\in S}\sum_{j\notin S} w(i,j) \leq -\sum_{i\in S'}\sum_{j\notin S'} w(i,j)$$

$$\sum_{i\in S}\sum_{j\notin S} -w(i,j) \leq \sum_{i\in S'}\sum_{j\notin S'} -w(i,j)$$

$$\sum_{i\in S}\sum_{j\notin S} w'(i,j) \leq \sum_{i\in S'}\sum_{j\notin S'} w'(i,j)$$

S is, therefore, the solution of the **GMC** problem at the same time.

Q.E.D.

In (Horowitz, Sahni, 1979), page 546, there is a theorem according to which the **MC** problem is **NP-complete**, i.e. **CNF** $\propto$ **MC** (**CNF** - Conunctive Normal Form Formula Satisfiability). In (Garey, Johnson, 1978) it is said that the **GM** $\propto$ **MC** (**GM** - Graph Matching) problem is solvable in its general form, whereas the planar graphs problem can be solved polynomially. The proof of the theorem has been given in the almanac **Complexity of Computer Algorithms**, Plenum Press, New York, 1972. in the article by Karp,R.: **Reducibility among combinatorial problems**. It is further elaborated in Hadlock, F.O.: **Finding a Maximal Cut in Planar Graphs in Polinomial Time**, SIAM J. Comput. 4, 1975., pp. 221-225, as well as in Orlova, G. I.; Dorfman, Y. G.: **Finding a maximal cut in graphs**, Eng. Cybernetics 10, 1972, pp. 502-506.

This has proved the **NP-hardness** of the clustering problem. It means that, by means of programming methods known today, it is not possible to find a polymomial algorithm that would solve this problem. In other words, each algorithm which is polynomial and that does not solve the problem of nondeterministic choice in a deterministic and polynomial way cannot be optimal.

It is not hard to show that the **CP** problem is **NP-complete.** A nondeterministic algorithm is obtained by adding a **Choice(S)** function to a standard programming language (Pascal, for example). **S** denotes a set from which on each step, the command, picks an element within the **O(1)** time so that the final result is the optimal solution of the problem (if it exists). With the **Choice** function defined in such a way, there is the following algorithm that solves the clustering problem:

Algorithm 1:

```
CONST maxproc = ...; {Large enough integer}

VAR S:ARRAY [1..maxproc,1..maxproc]; {Array which contains process adhesions}

VAR C:ARRAY [1..maxproc] OF INTEGER;

PROCEDURE NedetClu(Pr:S;
                   VAR Cl:C);
VAR i,j:INTEGER;

BEGIN
    FOR i :=1 TO NoProc DO
        Cl[i]:=Choice(Pr,i);
  END;
```

With the Choice function defined above, the algorithm of the time complexity is:

$$\mathit{Complex}(\mathrm{n}) = \mathrm{O}(\mathrm{n}) \tag{7}$$

5. Creating an algorithm for optimal clustering problem solving

It has been shown in the previous chapter that the **CP** problem is **NP-complete**. The algorithm we are going to create shall in the worst case be of an exponential time complexity. However, it shall solve a great number of the instances of the problem at a speed significantly exceeding the speed as it would be if the worst-case time were implied. The algorithm shall be based on a well-known method of programming, the so-called backtracking method. This method always yields an optimal solution with no need of proving its optimality. The point of the method is in presenting the problem as a sequence of decisions, i.e. as a tuple $(\mathbf{x_1,x_2,...,x_n})$. There are two approaches used by the method, Depth First Search and Breath First Search. More general terms regarding method can be found in (Aho, Hopcroft, Ullman, 1987) and (Horowitz, Sahni, 1978). An effective algorithm that uses the method can be found in (Syslo, Deo, Kowalik, 1983).

The problem can as well be presented in the following way. A tuple is created for an informational system with **n** processes; The **i**th component of tuple is a number representing the cluster to which the **i**th process belongs.

Algorithm 2: We shall choose Depth First search approach. The method shall be as follows:

1. The first system adhesion approximation is calculated first. It shall be done by means of finding the connection of the greatest weight and

supposing that all processes are connected by connections whose weights are equal to the weight of the connection with the greatest weight in the system. If the weight of the connection with the greatest weight is marked **M**, the first approximation shall be $\mathbf{M \cdot (n-1) \cdot n/2}$

2. The first process is placed into the first cluster and shall not be moved because these would not be any changes except the change in the cluster numbering.
3. Then the Depth First Search (Preorder) of the problem space tree is used. Unpromising branches are cut, i.e. the branches whose adhesion has already exceeded the least adhesion previously found. The branches that do not provide for at least one process per cluster are also not to be expanded. In addition, if **m** is the aimed number of clusters and $\mathbf{k<m}$, the kth process should be placed only in the first **k** clusters; as to other clusters, it shall result only in a different cluster numbering.

We shall perform an implementation of the algorithm in Pascal in order to assess its complexity in the easier way. The algorithm, as it is usual with this method shall, be performed recursively, which provides for better readability. Naturally, each recursive programme can be converted into a programme without recursion (the general conversion scheme is given in (Horowitz, Sahni, 1978), page 20).

```
CONST MaxProc=...; {Maximal number of processes}
TYPE Adh = RECORD
                        ad:ARRAY [1..MaxProc,1..MaxProc] OF INTEGER;
                        NoProc:INTEGER;
          END;
     cl = RECORD
                        clst:ARRAY [1..MaxProc] OF INTEGER;
                        NoProc:INTEGER;
          END;
     NoCl = ARRAY [1..MaxProc] OF INTEGER;

VAR mad:INTEGER;
    cc:cl;

PROCEDURE Recur(P:Adh;
                m,k,mc:INTEGER;
                VAR ad:INTEGER;
                VAR c:Cl;
                VAR NC:NoCl;
                ne:INTEGER);

{The procedure recursively looks for the optimal solution in the way described above}
```

```
VAR j,I,L,aa,ne1:INTEGER;
BEGIN
    ne1:=ne;
    {If k is less than m, the process is to be placed only in the first k clusters}
    IF mc<m THEN
        j:=mc+1
    ELSE
        j:=m;
    {The k-th process is placed into cluster by cluster with the procedure Recur
    being recurrently called for.Care should be taken not to expand the branches of
    the tree which are not promising, i.e. those whose adhesion value exceeds the
    one already found, and those that do not provide for at least one process per
    cluster}
    FOR i:=1 TO j DO
    BEGIN
        {Checking whether the branch still provides for at least one process in
            each cluster}
        IF NOT ((nc[i]>0) AND (ne1>=P.NoProc-k+1)) THEN
        BEGIN
            aa:=ad;
            {Calculating adhesion  when the k-th process is placed into i-th
                cluster}
            FOR L:=1 TO k-1 DO
                IF c.clst[L]<>i THEN aa:=aa+p.ad[L,k];
            {Leaving a non-promising branch of the tree whose adhesion
            value exceeds the one already found}
            IF aa<mad THEN
            BEGIN
                c.clst[k]:=i;
                Inc(nc[i]);
                IF nc[i]=1 THEN Dec(ne1);
                    IF k<P.NoProc THEN
                    BEGIN
                        IF i<j THEN
                            Recur(P,m,k+1,mc,aa,c,nc,ne1)
                        ELSE
                            Recur(P,m,k+1,mc+1,aa,c,nc,ne1);
                    END
                ELSE
                BEGIN
                    FOR L:=1 TO p.NoProc DO cc.clst[L]:=c.clst[L];
                        mad:=aa;
                END;
                IF nc[i]=1 THEN Inc(ne1);
```

```
                        Dec(nc[i]);
                    END;
                END;
            END;
END;
PROCEDURE BackTrck (P:Adh;
            m:INTEGER;
            VAR c:cl);
```

{The entrance of the algorithm is an orthogonal list P presented as a two-dimensional array. The list contains the weights of connections among the clusters and the natural number m that represents the projected number of clusters. The exit from the procedure is a list c, presented by means of an array, containing the inscriptions showing the cluster in which each process is placed}

```
VAR  i,j,Max:INTEGER;
     ad:INTEGER;
     c1:cl;
     nc:NoCl;
BEGIN
     {Calculating the first approximation}
     Max:=0;
     FOR i:=1 TO P.NoProc-1 DO
          FOR j:=i+1 TO P.NoProc DO
          BEGIN
               IF P.Ad[i,j]>Max THEN Max:=P.Ad[i,j];
               c1.clst[i]:=0;
               nc[i]:=0;
          END;
     {Initializing the array}
     mad:=(Max*(P.NoProc-1)*p.NoProc)/2;
     ad:=0;
     c1.clst[1]:=1;
     nc[1]:=1;
     {Calling for the recursive procedure}
     Recur(P,m,2,1,ad,c1,nc,(m-1));
END;
```

Let us remark that the algorithm is not dependent on the definition of the weights among processes and thus does not have to be used with the weights previously defined by means of adhesion. Namely, the algorithm provides for the optimal solution regardless of how the weights among clusters are defined, as long they as are defined simmetrically. On the other hand, the algorithm provides for the most general form of the solution of the problem. The problem restriction conditions can easily be added (for example, the condition of minimally k processes per cluster).

After some calculating procedures are applied, it can be seen that the algorithm has complexity

$$\mathbf{Complex(n) = O(2^n)} \qquad\qquad (11)$$
$$\mathbf{Complex(n) = \Omega(n^2)}$$

The calculating of algorithm complexity has been presented in (Lovrenčić, 1996).

6. Conclusion

The intention of this paper was to prove that the clustering problem is, **NP-hard** in terms of duration, i.e. the one that cannot be solved polynomially by means of the existing programme tools.

Having proved that the problem is **NP-hard**, we have as well indirectly proved that each algorithm, regardless of the language it is written in, providing for a solution for all instances in the polynomial time, is not optimal in terms of minimal correlation of the system. It is to be pointed out that the proof did not depend on the definition of the weights of the connections among processes. The only condition is that the weight function is commutative.

Two ways of the further problem analysis are possible. The first is to record the problem in such a way that would provide for the polynomial solving. Due to reducibility of the maximal graph cut problem to the clustering problem, it is possible to search for such restrictions that would provide for a planar graph as the system graph, which could be solved polynomially by analogy with the maximal cut problem.

The second way is finding a valid approximative algorithm. If the algorithm is to be valid, it should allow limiting the error absolutely or relatively, regarding the volume scale of the solution. It is easy to see that an algorithm that would yield an absolute approximative algorithm for this problem is also **NP-hard**. The solution is to find a relative approximative algorithm or to prove that it is also **NP-hard.** Otherwise, only stochastically good algorithms are left. They yield good solutions in most cases.

References:

[1] Aho, V.A.; Hopcroft, J.E.;Ullman, J.D. (1987), **"Data Structures and Algorithms"**, Addison-Wesley, Reading, Massachusetts.

[2] Brumec, J. (1993), **"Optimizacija strukture složenih informacijskih sustava"**, Zbornik radova Fakulteta organizacije i informatike Varaždin, pp. 1-23.

[3] Garey, M.R.; Johnson, D.S. (1979), **"A Guide to the Theory of NP-Completeness"**, Freeman & Co., San Francisco, (translation into Russian: Mir, Moskva 1982).

[4] Horowitz, E.; Sahni S. (1978), **"Fundamentals of Computer Algorithms"**, Computer Science Press, Rockville, Maryland.

[5] Knuth, D.E. (1973), **"The Art of Computer Programming - Fundamental Algorithms"**, Addison Wesley, Reading, Massachusetts.

[6] Lovrenčić, A. (1996), **"Problem optimizacije procesa clusteriranja"**, Seminar Paper, Varaždin.

[7] Red.: Nečepirenko, M. I.: **Algoritmi i programmy rešenija zadač na grafah i setjah**, Nauka, Novosibirsk, 1990.

[9] Syslo, M.M.; Deo, N.; Kowalik, J.S. (1983), **"Discrete Optimization Algorithms with Pascal Programs"**, Prentice-Hall International, Englewood Cliffs, New Jersey.

[10] Veljan, D. (1990), **"Kombinatorika s teorijom grafova"**, Školska knjiga, Zagreb.

Received: 1996-07-10

Lovrenčić A. Problem optimizacije procesa podsustava u informacijskom sustavu

Sažetak

Prilikom izgradnje većih informacijskih sustava nužno je, radi lakšeg projektiranja, sustav podijeliti na podsustave. Pritom se teži tome da se rastav napravi tako da sustav što manje izgubi na kvaliteti. Potrebno je stoga definirati mjeru za kvalitetu rastava sustava. Kad se ona definira, prirodno se postavlja pitanje algoritma za optimalno dijeljenje sustava pod zadanim uvjetima te njegove vremenske složenosti. Razmatranja u tom pravcu dovode do zaključka da algoritam za dijeljenje sustava u najgorem slučaju mora biti nadpolinomijalne vremenske složenosti, s time da se zadaća svrstava u grupu **NP-kompletnih** zadaća.