UDC: 007.5
Original scientific paper

AN EFFICIENT ALGORITHM FOR INFORMATION SYSTEM DECOMPOSITION

Alen Lovrenčić
University of Zagreb, Faculty of Organization and Informatics, Varaždin, Croatia
e-mail: alovrenc@foi.hr

A general context for the problem of decomposition within an information system is described in [3]. *The problem has been classified as a* NP*-complete problem, which excludes the possibility of an optimal solution of the problem in polynomial time. This paper has two goals: first, to solve the problem within some additional limitations that are usual in the praxis of decomposition and, second, to provide for an on average faster algorithm to calculate the optimal decomposition of an information system into subsystems.*

Keywords: information systems, algorithm, optimization, complexity.

1. INTRODUCTION

For some 30 years, software engineering has dealt with the problems of increasing application packages' quality and of the standardisation of procedures for designing applications. The development of standard procedures and methods, as well as methodologies as a whole were moving bottom-up. The "lower" methods were the first to develop closer links to computer engineers and be farther from the packages' users; later on, the point of interest started ascending higher and earlier in a package's development stage. This is why more and more attention has been paid lately not only to the methods that are applied at very early stages of the development of a package, but as well to the very designing and systematisation of the application domain the package is to cover. The methods at this stage of the package development process are usually referred to as *the strategic planning methods.*

The problem of decomposition within an information system is one of the basic problems of planning the information systems design. As has already been shown in [3], the optimality of the solution to this problem significantly influences the further course of designing an information system and its final quality and the complexity of its inner relations.

2. CHARACTERISTICS OF THE PROBLEM

The description of the problem of decomposition of an information system has been shown in detail in [2]. Here, only the basic characteristics of the problem, which are important for the development of our algorithm, will be shown. The way into the problem is through a *data processes-classes matrix*. The processes denote the dynamic structure (algorithms, programs, etc.), whereas data processes denote the static characteristics (data structure, databases) of the future system. Each row of the matrix is defined by a process and each column is defined by a data class. Signs are entered into the matrix body and they indicate which operations the process in whose row the sign is performed on the data class, thus denoting the column in which the sign is. Two kinds of relations between processes and a data class are to be distinguished here. At most, one process within the system can create data belonging to a certain data class, whereas several processes can use the data from the data class. See [2] for argumentation on this assumption. Here is an example of the data processes-data classes matrix:

Table 1. Process-data class matrix

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
p_1	C						
p_2		C	U			U	
p_3	U		C	C	C		U
p_4		U		U			C
p_5		U		U			

Let us point out that it is possible for a process to create data for several data classes (process p_3, for example), and that there may exist a so-called *outer data class* whose data is not created within the system but in its surroundings instead (data class c_6, for example). It is also possible for a process to be "passive" within the system, i.e. to create no data (process p_5, for example) and that the data of a data class is created within the system, but it is not used in it (data class c_5, for example) and the data class is most likely to be created as a sort of link between the system and its surroundings.

The first step is, by means of a certain function, to convert this matrix into a *processes-processes matrix* that defines the relations between the processes. To develop the algorithm, we assume that this table with certain properties is given. The property we request for this matrix is symmetry. A part of the argumentation to support this request has been given in [3], but we are going to try to argue the request here. In order to argue it, it is necessary to remember the meaning of this matrix and the goal we want to attain through it. Our goal is to reduce the relations among subsystems as much as possible. The case of the asymmetrical *processes-processes matrix* leads us to the measure of the weight of the relations between processes which is, in turn, dependent on the process from within, from which the observed relation originates. However, in this case, this measure by itself does not suffice to define the quality of decomposition. Namely, there is the amended question of which relation is stronger in this case, the one with (0.5, 0.5) weight or the one with (0.7, 0.1). There are two obvious answers to this. We could suggest that this problem be solved by means of taking the larger of the two weights, or adding them, or averaging them. This could also be determined by a more complex procedure. However, all the procedures come down to one: defining the symmetrical weight of the relation. In other words, no matter which way the afore-mentioned problem is resolved, the solution shall be the answer to the following two questions: which of the relations is stronger and by how much is it stronger? The very answer, if defined generally, shall present the symmetrical measure of the weights of the relationship.

Finally, the central problem we are to deal with in this paper is obtaining the processes' allocation to clusters in such a way that, at the given limitations, the weight of the relationship between clusters is minimal. The weight of the relationship between clusters is defined here as in [3]. To solve this part of the problem, the branch-and-bound method is to be used, i.e. its least-cost variation.

3. PROBLEM DEFINITION AND THEORETICAL DISCUSSION

The first problem to deal with is determining the way to measure the quality of decomposition. The inclination to determine the means of measuring is to use the function $\mathbf{d:D\times D\rightarrow R^+}$ which is metric (for the definition and basic properties of metrics see [4]). The trouble is, however, that the structure defined by our problem cannot be defined naturally as a metric space. Namely, the measure determining the quality of decomposition is based on the quality and the quantity of relationships between clusters, i.e. subsystems. However, it is easy to see that the measurement of a system defined in such a way cannot be a metric.

Let there be a function $\mathbf{d:D\times D\rightarrow R^+}$ measuring the quality and the quantity of relationships between subsystems. The natural properties of the function are as follows:

1. $\forall c_i \forall c_j (d(c_i, c_j) \geq 0)$
2. $\forall c_i \forall c_j (d(c_i, c_j) = 0 \Leftrightarrow c_i = c_j)$
3. $\forall c_i \forall c_j (d(c_i, c_j) = d(c_j, c_i))$

The problem arises with the fourth property of metrics, i.e. with the samenessless of triangles. Let us consider three subsystems of the observed system: $\mathbf{c_1}$, $\mathbf{c_2}$ and $\mathbf{c_3}$. If the decomposition is performed in the standard way, by means of processes, the subsystems shall be related by mutual data. However, in that case $\mathbf{d(c_1, c_3)}$ does not depend on $\mathbf{d(c_1, c_2)}$ and $\mathbf{d(c_2, c_3)}$ values. We therefore cannot guarantee that the following will be valid:

4. $\mathbf{d(c_1, c_2)+d(c_2, c_3) \geq d(c_1, c_3)}$.

Let us give an intuitive example:

Example 1. Let there be a system whose graph is

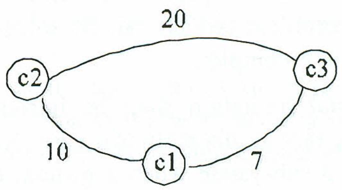

Figure 1. Process connectivity graph

It is clear that the situation presented in Fig. 1 is highly real, where the weight of the relationship between the clusters $\mathbf{c}_1$ and $\mathbf{c}_2$ amounts to 10, between $\mathbf{c}_1$ and $\mathbf{c}_3$ 7 and between $\mathbf{c}_2$ and $\mathbf{c}_3$ 20. On the other hand, it is clear as well that the weight function is not metrics in this example, considering that it is valid that

$$\mathbf{d(c_{1,}\ c_2)+d(c_2\ ,\ c_3)< d(c_1\ ,\ c_3).}$$

The lack of this property makes the problem more complex. Namely, this makes using the dynamic programming method [7] to solve this problem impossible. The reason for this is that the method requires the problem to comply with the *optimality retaining principle*. We shall not be formally proving here that the dynamic programming method is not suitable for solving the problem being discussed because that would take us into an analysis straying away from the goal of this paper.

Let us now describe the input data and the problem-solution requirements. The input data consists of the processes-processes matrix as described in the previous section, of the number of clusters the decomposition is to achieve and of two positive whole number constants that determine the minimal and the maximal number of processes to be found in a single cluster.

We must point out that if the borders of an information system are well-defined, then the graph of the system shall be integral, i.e. it will not be possible to break it down into two or more unrelated graphs. The opposite result would indicate that there might have been a certain error in defining the input data. If the processes and data classes are defined properly, we can then speak of a failure to define the borders of the system. Namely, it means that we have overlooked the fact that we are dealing with two systems independent one of the other when defining the system and have considered the two of them to be one.

4. EFFICIENT ALGORITHM OF OPTIMAL SYSTEM DECOMPOSITION

4.1 Algorithm description

The problem is to be solved by the least-cost variant of the branch-and-bound method. Let us write down the basic items of the problem. The system is described by the graph **G=(V, E)**, where the vertices represent the processes of the system while the edges represent the value of the relations between processes. [3] has shown that the problem of optimal system decomposition boils down to the problem of the minimal cut of the already defined graph. At this point, a possible misunderstanding needs to be avoided. In the graph theory, the problem to which our problem has been boiled down is not a graph cut problem (which can be solved polynomially) but a graph partition problem which is NP-complete.

However, the additional limitation shall be introduced now. It had not been introduced in [3] and it appears in this particular case. Namely, we required that there be at least one process in a subsystem when a general case is implied, which is too weak a condition in our particular case. Thus when generating the system decomposition, the cases in which a subsystem has only one process are not interesting. We shall therefore reinforce this need by requiring each subsystem to have minimally **m** and maximally **M** processes. The second decomposition parameter was the **s** number of subsystems which we want the system to be decomposed into. We shall continue to use the fact that the optimal composition may be attained at the exact minimal number of subsystems allowed (see [3]).

The branch-and-bound method requires defining two functions: the function that evaluates the optimality of the branch development and the "intelligent" function that ranks the state. The first function enables us to cut off non-perspective branches, i.e. the branches that certainly will not yield the optimal result, whereas the second function enables us to determine the development perspective of a field branch and thus always develop the most perspective branches. The first function of the two has already been given in [3] and it has been defined there as follows: First, two processes are defined whose relation weight is maximal and they are marked with **M.** The first approximation is the number **Cmax=M(n-1)n/2**, where **n** is the number of processes within the system. It is clear that, regardless of decomposition, the sum of all the relations between the processes in different subsystems cannot exceed this number because the number **(n-1)n/2** is the maximal number of edges of the graph with **n** vertices and **M** is the number that is equal to or larger than the weight of any edge of the graph. After the algorithm has found the solution, **Cmax** is replaced by the sum of weights of all relations among the processes within different subsystems. Furthermore, by searching the field tree, we can "cut" the branches where the sum of the weights of all relations among processes already allocated to different subsystems exceeds **Cmax.** It is clear that we are not going to cut off the minimal solution because further allocation of processes to subsystems makes the sum of the weights of all relations among the processes allocated to different subsystems rise, or, favourably, the sum remains the same. Besides that, we can, as well, cut the branches that do not provide for the defined minimal number of processes within a subsystem or the branches that exceed the defined maximal number of the processes within a subsystem.

The function is to be somewhat improved now. First, regarding the ranking function calculation which requires the evaluation of relation weight improvement among the processes within different clusters that is caused by expanding a branch, the same function can be used to calculate the first approximation if the whole field is considered to be one branch for which the estimation is being performed. In this way, quite a good approximation is instantly obtained and the approximation shall cut off some non-perspective branches even before the first solution is obtained. Furthermore, we shall not be limited to cutting only those branches that have already reached a cluster relation weight which is larger than the cluster relation weight of any already obtained solution, for in this way some branches are kept whose non-perspectiveness is easily detected. In other words, the function of estimating the optimality of the field is to be improved in such a way that to the presently attained cluster relation weight the minimal attainable weight of every as yet unallocated process is added, assuming that that process is next to be observed. In other words, the branch development estimating function is to be calculated as follows:

- First, the weight of relationships between clusters is taken (in relation to the processes that have already been allocated up to this moment). Let this value be marked as **O.**
- Each process of as yet unallocated processes is to be observed in relation to already allocated ones, and the cluster to which it should be allocated in order to increase the relation weight minimally is to be determined. When the cluster is determined, the process is not allocated to it. Instead. only the **O** value is increased by the value by which the weight would increase if the process were allocated to the cluster. When the next unallocated process is observed, it is again observed only in relation to already allocated processes. In other words, the branch development optimality function does not allocate unallocated processes to clusters.
- Finally, after the calculation has been done in the previous step for all processes as a yet unallocated, the obtained value is compared to the relation weight value of the solution that is best to this moment; if in this way a value larger or equal to the already attained minimal relation weight is found, the branch is proclaimed to be non-optimal and it is cut off.

The ranking function can be defined in various ways. When choosing the function, two points should be kept in mind: It has to evaluate the final solution to the best possible extent and it must be as easy as possible to calculate for it is calculated for each node of the field we enter. This function will help us to pass through as few tree nodes as possible when calculating the solution.

Therefore, the sum of the weights of already allocated processes shall be known at any field node. This figure will be the basis for the ranking function. However, this is not enough. Namely, it is not unimportant at which level of the tree the observed node is placed. The higher on the field tree the node is located, the more processes are allocated to it and it is natural that its respective sum of the weight of the processes allocated to different subsystems is larger. This has to be taken into consideration so that the development of the higher-level nodes is made possible. This is why the greedy algorithm is to be used, so that the increase in the weight sum in further developments of a branch can be estimated.

The ranking function will be defined in a way which also feeds back (aside from the estimation of perspectiveness of a branch) some suboptimal solutions. The function shall thus be used not only for ranking the branches but for searching for the solution as well.

It will also be shown that the algorithm is still NP-complete, regardless of the introduced limitations. This means that in the worst case the algorithm is not speeded up by adding the ranking function, although it is speeded up for some less extreme cases.

Naturally, the remarks on the unimportance of numerating a subsystem from [3] are still valid and thus the basic program frame remains similar, only the new functions of assessing optimality and ranking will be added enabling us to terminate further development of a branch at any stage and shift to another branch. We will, therefore, allocate the processes by their order.

4.2. Data structures

Let us first define the structure of the data to be used. This time, the structures are to be described by means of parameters and that will result in a dynamic quality for the structure sizes, which also means better memory usage.

The weights of relationships between processes are defined as an orthogonal linked list. The list is to be implemented by means of an opened hash table. It means that there will be a tied header list with one header per each process. Each header is to show a linked list with the processes to which the process from the header is related and their respective weights. Let us give an example. Let there be a system defined by the following process table:

Table 2. Process-Process matrix

	p1	p2	p3	p4	p5
p1		2,1	1		
p2	2,1		1,4		
p3	1	1,4			
p4					2,2
p5				2,2	

The structure describing a system that is defined in this way will look as follows:

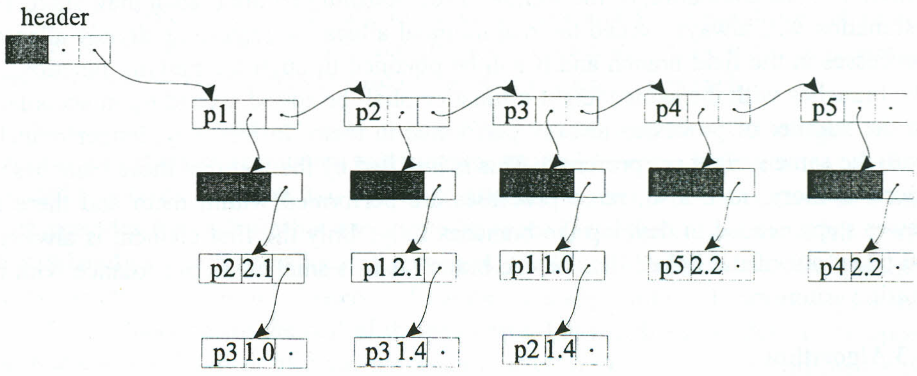

Figure 2. Orthogonal linked list

The allocation of processes to subsystems will be entered in a linked list. The same structure shall be used to keep the best solution found. As for the example from Table 2, for 2 clusters and the requirement that there are at least two and at most 3 processes within each cluster, the optimal distribution will be as shown in the following table:

Table 3. Cluster definition

Number of processes	**Cluster number**
1	1
2	1
3	1
4	2
5	2

which will be written in the structure as shown in Fig. 2

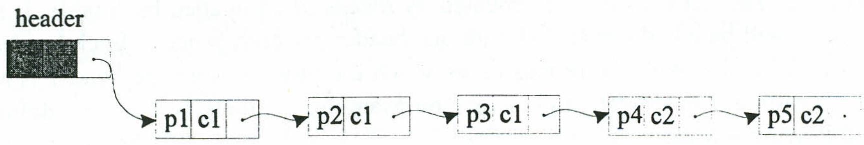

Figure 3. Linked list

Apart from these structures, one structure more is needed to retain the data on all the branches that are currently being observed. This structure will also be an opened hash table, while the linked list defined in the previous structure is to be used for the table nodes lists.

This structure will be filled and emptied in a specific way. When a new element enters the structure, its placement ensures the list being sorted by two criteria. The first criterion is the estimation of the weight of relationship that the branch may yield. This estimation will always exceed the real minimal allocation regarding already allocated processes in the field branch and it will be obtained through the ranking function. All the branches with the same weight estimation shall be sorted ascending in accordance to the number of processes already performed in them. In this way, longer branches with the same weight are preferred. This is justified by the fact that these branches use more memory, and, also, more processes are performed within them and there are fewer steps needed to develop the branches fully. Only the first element is always to be taken out of the linked list, i.e. the branch that is smallest in accordance with our sorting criterion.

4.3 Algorithm

The first task to be completed is the developement of a function that will calculate the weight of relationships between two processes. The processes are marked by natural numbers and the weights of relationships within a system are contained in the orthogonal list. The procedure will initially find the first of the two processes in the header list and, after that, it will try to find the second given process in its body list. If the second process is found, the weight of relationship between the two processes will be read; if not, the procedure returns 0.

This procedure is fairly simple. In the orthogonal list **A** containing the weights of the relationships between particular processes, the procedure calculates the weight of relation between two given processes. The procedure takes for granted that the number of processes is properly given, i.e. the procedure does not check whether the respective processes exist. The procedure searches the header list of a hash table until it finds the first process. After that the first process node hash table is entered for the purpose of searching for the second process. If the process is found, the weight inscribed in that node returns, and if it is not found, 0 returns, given that the hash table is designed in a way that it contains values for existing relations only.

It is easy to note that this function has the time complexity **T(n) = O(n)**, where **n** represents the number of processes within a system.

As to the structure that keeps the allocation of processes at subsystems, there will be some operations to insert a new process as well as the operations that will calculate the function of evaluating the optimality and the ranking function.

The operation that is to calculate the ranking function will imply two subfunctions. One of them will calculate the optimality function from parameters and the other will, by means of the greedy procedure, calculate the evaluation of the increase in the whole weight of relationships between the subsystems in the solution. Let us therefore develop first the greedy algorithm for evaluating the future increase in the whole weight of relationships.

The greedy algorithm places a process in the cluster in which that process causes the least momentary increase in adhesion. The input parameters for this function shall be the list of already allocated processes, the table of relations, the number of processes for which the values are to be calculated and the present weight of the relations among the clusters; it will feed back the estimation of the increase in weight of relations among the clusters in case of the development of a branch.

The goal of this function is to estimate in a quick way how the weight of the relationships between the clusters will increase taking into consideration some already allocated processes. The function will try to allocate the rest of the processes in a way which provides for each further process to be placed into the cluster in which it will cause, related to all already allocated processes, the least increase in the whole weight of relationsips between the clusters. Of course, this algorithm will not always guess the optimal allocation for the rest of the processes, but it will surely denote the upper limit that the development of branches is not to exceed. It will thus have influence on the function of estimating the optimality for that function shall always, when the ranking function feeds back the value that amounts to less than the least whole weight of relationships possible, boil down to the estimated value because this value, being suboptimal, always exceeds or equals the value that it is possible to attain by means of the development of the branch for which the estimation is performed.

As has already been pointed out, this procedure creates some suboptimal solutions as well. Each suboptimal solution is checked in terms of being or not being better than the currently best found, and, if it is better, it becomes the best-obtained solution.

If **n** is the number of processes within a system and **m** the number of clusters, then the time complexity of this procedure amounts to $\mathbf{T(n,m)=n^2 + n \cdot m}$. The time complexity of the procedure is therefore squared.

Let us define the branch development estimating function now. First, this function has to be initialised by some value. As in [3], it can be initialised in such a way that a link between two processes that has the greatest weight is chosen and then multiplied by the largest possible number of vertices in the graph. Of course, this first approximation does not give a real limitation, given that no solution to the problem can give such a great weight of relationships among the clusters. In order to improve this initial approximation, i.e., in order to make it work even before the first solution is found, the ranking function is to be used. The ranking function yields approximate results that are always suboptimal and thus the optimal solution shall not be "cut off".

The other way to improve the branch development estimating function consists of replacing the limit value with a new one whenever the ranking function feeds back a value that amounts to less than the previously obtained upper-limit value. The reason for it is again based on the fact that the greedy algorithm yields suboptimal solutions that always amount to equal or greater than the optimal solution and there is no danger of "cutting off" the optimal solution.

The branch development estimating function is, therefore, to be implemented by means of the variable **WMax** which is composed of the following properties:

- The variable is initialised by the value **WMax:=Greedy(NIL, A, 1, 0)**
- Whenever a solution is found whose whole weight of the relationships between the clusters is **Cm<WMax**, then **WMax:=Cm** is to be used.
- Whenever the ranking function feeds back **Greedy(?)<WMax**, **WMax:=Greedy(?)** is to be used.

When defined in this way, the branch development estimating function shall evaluate the optimal solution better and it shall be cutting non-perspective branches faster.

The procedure is quite simple because all that is important has been done within the **Greedy** procedure. Here, from the structure that describes the problem space, the weight yielded by already allocated processes is read prior to recalling the **Greedy** function that estimates the increase within a particular branch. Finally, if the obtained estimation amounts to less than the value of the Wmax variable, than the estimation value is entered into the variable.

The time complexity of this procedure is equal to the time complexity of the **Greedy** procedure.

Now, when the ranking function has been defined, the rest of the procedure is similar to [3]. Let us define the recalling procedure that accepts as its input the table of weights of relationships between processes, the requested number of processes and the minimal and the maximal number of processes per cluster.

The estimation of the complexity of this procedure shall be performed combinatorially. The complexity of one step of the main loop is to be determined first.

Its time complexity is $\mathbf{T(n,m)= n^2 + n \cdot m}$.

Now, the calculation of the number of tree nodes for the worst case is to be performed, where "the worst case" means the case when the branch development estimating function does not cut any branch.

It is clear that the main loop is repeated maximally the same number of times as the maximal number of nodes that the field tree can contain, assuming that these nodes are not leaves given that the loop does not have to be repeated for them because they can be simply evaluated as solutions. As to the minimal and maximal number of the processes per cluster, the worst case is when a cluster may contain a minimum of **min=1** and a maximum of **max=n-m** processes. If this is the case, all the allocation schedules with at least one process per cluster will be considered regular ones and thus no one branch shall be "cut off" in accordance with this criterion.

The number of nodes in the whole tree should be calculated. Without trying our hand at the precise calculation of the number of nodes (which, moreover, is not of crucial importance in estimating complexity), we shall assume that we are dealing with a complete **m**-ary tree (which is wrong only for the first **m** levels if **n** is large enough) so the number of inner nodes is strictly less at $\mathbf{n^m}$. However, it has been said that the main loop is to be repeated for each node and that its complexity is estimated at $\mathbf{m \cdot n+n^2}$, which shall yield the whole complexity

$$\mathbf{T(n,m)<(m \cdot n+n^2) \cdot n^m}$$

But, as the following is valid

$$\lim_{n\to\infty} \frac{(m \cdot n + n^2) \cdot n^m}{n^{m+2}} = 1$$

it can be formulated as

$$\mathbf{T(n,m)=o(n^{m+2})}$$

On the other hand, as the branches are kept as a whole in the data structure and as each node takes a fixed space, it can be seen that the spatial complexity of the algorithm is, at worst, equal to the number of tree nodes (taking into consideration only the first n-1 levels, given that completed branches are not kept in the structure). The size of the structure that keeps incomplete branches may be of a size equal to the size of the number of inner nodes. All other structures are of a polynomial size in relation to the number of processes. The space complexity of the algorithm can thus be evaluated by

$$\mathbf{S(n)=o(n^m)}$$

5. PROBLEM COMPLEXITY

In [3] it has been shown that the problem of information system decomposition in **c** clusters is NP-complete. We will show that this problem can be reduced to the problem dealt with in this paper, which means that the problem is NP-hard. It is easy to see that our problem is NP-complete. We can use the non-deterministic algorithm given in [3] with some minor modifications.

Definition 1: **(m clusters decomposition problem – SD(m))**
There is an information system with **n** connected processes. Let the weight of the relationship between processes **i** and **j** be **$w(i,j)=w(j,i)\geq 0$**. We have to find the partition of **$P=\{1,\ldots,n\}$** to **m** disjunctive non-empty subsets **$P_1,\ldots,P_m$**, so

that $\sum\limits_{i=1}^{n-1} \sum\limits_{\substack{j=i+1\\ i\in P_k\\ j\in P_l\\ k\neq l}}^{n} w(i,j)$ is minimal.

The problem defined in Definition 1 is NP-complete as shown in [3].

Definition 2.: **(m clusters decomposition problem with maximal max and minimal min processes per cluster - SD(m,min,max))**
There is an information system with **n** connected processes. Let the weight of the relationships between processes **i** and **j** be **$w(i,j)=w(j,i)\geq 0$. Let min** and **max** be natural for such numbers that **min**$\leq$ **$\lfloor n/m \rfloor$** and **max**$\geq$ **$\lceil n/m \rceil$**. We have to find the partition of **$P=\{1,\ldots,n\}$** to **m** disjunctive non-empty subsets **$P_1,\ldots,P_m$**, so that $\sum\limits_{i=1}^{n-1} \sum\limits_{\substack{j=i+1\\ i\in P_k\\ j\in P_l\\ k\neq l}}^{n} w(i,j)$ is minimal, with minimal **min** and maximal **max** processes per cluster.

Theorem 1. **SD(m)$\propto$SD(m,1,n-m)**

Proof: We take **SD(m,1,n-m+1)**. In such a problem, there is a minimum of 1 and a maximum of **n-m** processes per cluster.

Let partition $P=\{P_1,\ldots,P_m\}$ be the solution to problem **SD(m,min,max)**. Then, in every set **P_i**, **$1\leq i\leq m$**, there must be at least one process, which gives **m** processes. Each of the **n-m** processes that are not placed in the first step can be placed in any cluster. In the worst case they can be all placed in the same cluster, in which case this cluster wil contain **n-m+1** processes.

So, P is a suboptimal solution of **SD(m)**. We have to prove that P is the optimal solution for **SD(m)**. Suppose that P is not the optimal solution. Then, there is the solution $P'=\{P_1,\ldots,P_m\}$ so that

$$\sum_{i=1}^{n-1} \sum_{\substack{j=i+1 \\ i\in P_k' \\ j\in P_l' \\ k\neq l}}^{n} w(i,j) < \sum_{i=1}^{n-1} \sum_{\substack{j=i+1 \\ i\in P_k \\ j\in P_l \\ k\neq l}}^{n} w(i,j)$$

But, partition P' is the solution for **SD(m,min,max)**, and it is better then P. That is in contradiction to the assumption that P is the optimal solution to the **SD(m,min,max)** problem.

So, it can be concluded that P is the optimal solution of **SD(m)**.

Both problems are defined using the same input, (except **min** and **max** numbers), so these two numbers can be calculated in **Θ(1)**, and thus the problem **SD(m)** can be reduced to **SD(m,min,max)** in Θ(1) time.

Q.E.D.

Theorem 2. **SD(m)∝SD(m,1,n-m)**

Proof: The proof of this theorem is similar to the proof of the above theorem

Q.E.D.

The two theorems proved above shall give the next corollary:

Corollary 1.: **SD(m,min,max)** problem is NP-complete

Proof: From theorem 1. it can be concluded that **SD(m,min,max)** is NP-hard. But, if the problem **SD(m)** can be solved in polynomial time by a non-deterministic algorithm [3], and if the problem **SD(m,min,max)** can be reduced to **SD(m)** in polynomial time by a deterministic algorithm, which is true by theorem 2., then we could solve the problem **SD(m,min,max)** in polynomial time with a non-determninistic algorithm for **SD(m)**. So, **SD(m,min,max)** is NP-complete.

Q.E.D.

6. CONCLUSION

This paper reviews the problem of optimising the structure of information systems and it uses the results of paper [3], to which this paper is a sequel.

After that, an efficient algorithm is developed and it takes into consideration the specific features of the problem. The algorithm is based on the least-cost variant of the branch-and-bound method. The algorithm yields the optimal solution to the problem in exponential time and in the exponential memory area.

Further direction in this work may be oriented toward improving this algorithm by means of additional heuristics and toward lowering the required memory area, or toward an approximate scheme of this problem that would solve the problem within polynomial time and provide for as small a relative error as desired. Of course, if the latter direction is chosen, it is obligatory to evaluate the relative error for each approximate algorithm so that the algorithm gains value.

REFERENCES

[1] M. J. Atallah. *Algorithms and Theory of Computation Handbook.* CRC Press, Boca Raton, 1998.

[2] J. Brumec. Optimizacija strukture složenih informacijskih sustava. *Zbornik radova,* Vol. 17, Fakultet organizacije i informatike, Varaždin, 1993, pp. 1-23.

[3] A. Lovrenčić. The problem of optimization of the process of decomposition of an information system. *Zbornik radova,* Vol 21, No. 1, Fakultet organizacije i informatike, Varaždin, 1997, pp. 27-41.

[4] S. Mardešić. *Matematička analiza 1.* Školska knjiga, Zagreb, 1988.

[5] M. I.Nečerimenko. *Algoritmy i programmy rešenija zadač na grafah i setjah.* Nauka, Novosibirsk, 1990.

[6] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1994.

[7] S. Skiena. *The Algorithm Design Manual.* Springer Verlag, Heidelberg, 1997.

[8] D. Veljan. *Kombinatorika s teorijom grafova.* Školska knjiga, Zagreb, 1989.

Received: 24 June 1998
Accepted: 30 August 1998

Alen Lovrenčić

EFEKTIVNI ALGORITAM ZA DEKOMPOZICIJU INFORMACIJSKOG SUSTAVA

Sažetak

U radu [3] dan je općenit kontekst problema dekompozicije informacijskog sustava. Dokazano je da je taj problem NP-potpun, čime se isključuje mogućnost nalaženja polinomijalnog algoritma koji bi rješavao taj problem. U ovom radu postavljena su dva cilja: da se u algoritam uključe stvarna ograničenja koja su prirodna pri dekompoziciji informacijskog sustava u podsustave, te da se razvije algoritam za izračunanje optimalne dekompozicije informacijskog sustava sa što manjim vremenom obrade u prosječnom slučaju.

Ključne riječi: informacijski sustav, algoritam, optimizacija, složenost.