JOURNAL OF APPLIED QUANTITATIVE METHODS

SPECIFICATION METHODS OF ECONOMIC PROCESSES

Marian Pompiliu CRISTESCU
Lucian Blaga" University of Sibiu

Email: marian.cristescu@ulbsibiu.ro

Ion IVAN
A.S.E. Bucharest

Email:ionivan@ase.ro

Laurentiu CIOVICĂ
A.S.E. Bucharest

Email:laurentiu.ciovica@gmail.com

Liviu CIOVICĂ
A.S.E. Bucharest

Email:liviu.ciovica@gmail.com

ABSTRACT

This paper present the stages by which we associate a formal language of a economic process. It proposes the use of a mathematical model, described as a graph, for the specification of business processes such as: investments, organization of production, activities of economic analysis. It deals with those economic problems that may be put into formal practice when there is no possibility of using one of the known techniques, or when we are modeling a business process, a manufacturing process etc.

Keywords: optimization algorithm, economic modeling, formal language, graph

1. INTRODUCTION

A production process can be simulated or modeled effectively using linear bounded automated. However, if every time stock elements are bounded, then a finite state automaton can simulate the complexity of machine depends essentially on the size of graphs that describe technology products recipes. The unit of time is greater, the more the grammars associated with a system are simple, and the system is easier modeling.

Operation of many devices due to continuous processes and signal deviations from normality can be simulated using sequential transducers. Through the practical consequences of these results can be listed: the ordering, production planning and

programming, formal demonstration of the need for top-down design and implementation of information systems for manage the economic systems, necessity hierarchical management of socio-economic systems.

Economic problems are put into formal practice when we wish to model a business process, a manufacture process, a part, etc. Each action of the respective business process is marked with: *a, b, c, d,* such actions having clearly defined periods of time, and the action pairs are those composed of strings of the form: *ab*cab*bc…, ab, bb, bc,* so that there are no other restrictions.

2. USING FORMAL ELEMENTS FOR ECONOMIC SYSTEMS SPECIFICATION

The model by which a language *L* is assigned to a business process *Pi* involves the following steps:

1) For the beginning, we seek the elementary events of a process, namely the actionable atoms, by means of which any process development can be written as a string. Most times these events are naturally suggested by the process development. In some cases, we cannot make a natural separation of these elementary actions, but we have to make a cut out of the process with respect to a conveniently chosen time unit. The description of this process will be interpreted as elementary events;

2) In both cases, the essential problem is represented by the finitude/ finiteness of the elementary events inventory, or otherwise of the vocabulary we are working on;

3) Once the vocabulary has been found, we pass on to the identification of the language which describes the process. This requires knowing the process and the rules of process development;

4) The purpose is to build effectively a grammar that generates strings that satisfy all these conditions imposed by fairness. Formally expressed, it can be written as [ATAN07][1]:

Given $c_1, c_2, \ldots, c_n$, the conditions that a string of elementary events must satisfy in order to be correct. For each $i=1,2, \ldots, n$ we build a language $L(c_i)$ of all strings that comply with the condition c_i ignoring the others. The language that is sought is the intersection of all languages, namely:

$$L(P) = \bigcap_{i=1}^{n} L(c_i) \qquad (1)$$

Obtaining the languages $L(c_i)$ is carried out so that each condition corresponds to another method of building an associated language, namely its grammar. This way we can obtain an automat or a grammar used for simulating the system under consideration, and the conclusions on system behavior are obtained with the help of the automat.

A mathematic model, called *graph*, can be used successfully in investments, organization of production, economic analysis activities, transport, etc. The graph is "a figure composed of points connected by arrows. The points symbolize different elements depending on the modeled phenomenon, and the arrows represent the connections that are established between the elements".

"Given $X=\{x_1,x_2,...,x_n\}$ a finite set and $U:X\to P(X)$ which attaches to any $x_i\in X$ a subset $U(x_i)\subseteq X$. The pair $\Gamma=(X,U)$ is called *graph*. The elements of X are called the graph's *vertices*. A pair of vertices (x_1,x_j) forms an *arc* if $x_j\in U(x_i)$."

"A graph $\Gamma=(X,U)$ is *marked*, if there is a set $\sum$ and a function $\varphi:U\to\sum$ which associates to each arc an element from $\sum$ called *label*" [ATAN02], [ATAN07], [CREA04], [JALO06].

From the definition of graph, for two nodes a_i, a_j of X, we cannot consider more than two arcs, differently oriented so that they might connect to each other. Waiving this restriction leads to consider the concept of *multigraph*. Multigraph is the pair *(X, U)*, where *X* is the finite set of nodes, and $U\subseteq X\times\sum\times X$, $\sum$ is the finite set of labels. Between two nodes of *X*, there may be more arcs, differently labeled. By formalizing, we may notice that the path *d* has been identified by a string of the vocabulary $X=\{x_1, x_2, ..., x_n\}$. Just as well, the path *d* may be identified by the string: (a_{i1}, a_{i2}) (a_{i1}, a_{i2})... (a_{i1}, a_{i2}), therefore like a string of arcs, a string of *U**. If the function is injective, namely a string of $\sum^*$ can be assigned to each path of Γ, but the same string of $\sum^*$ might correspond to more distinct paths of the graph Γ. This shift from paths in graphs to strings of symbols allows any problem regarding paths in graphs to be formulated and solved as a linguistic problem. In fact, there is an isomorphism between multigraph and finite automats. First of all, we naturally associate to each finite automat a multigraph, defined as:

$$\Gamma(A)=(K,\{(s_i,a,s_j)?\mid s_j\in\delta(s_i,a),a\in V\}), \tag{2}$$

Vocabulary V is considered like lots of labels. To view the original state machine on this multigraph, draw an arrow from the outside toward the node associated with the initial state and the final states they encircle it with two lines.. Even if automatic A is deterministic, multigraph $\Gamma(A)$ can actually be a multigraph and not a marked graph. This think can be explained such.

Whether regular language: L = {a, b}{c*}{a, b}. This language can be recognized by the next deterministic finite automaton.

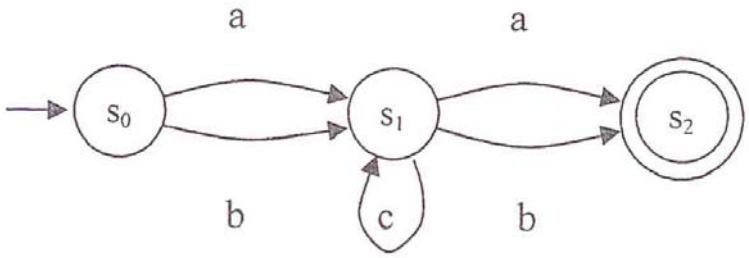
$$A=(K,V,\delta,s_0,F), \tag{3}$$

where:

$$K=\{s_0,s_1,s_2\},V=\{a,b\},F=\{s_2\},\ \delta(s_0,a)=\delta(s_0,b)=s_1, \tag{4}$$
$$\delta(s_1,c)=s_1,\ \delta(s_0,a)=\delta(s_1,b)=s_2,$$

and associated multigraph is:

$$\Gamma(A)=(K,\{(s_0,a,s_1),(s_0,b,s_1),(s_1,c,s_1),(s_1,a,s_2),(s_1,b,s_2)\}). \tag{5}$$



Multigraph $\Gamma(A)$ associated with a finite automaton $A=(K,V,\delta,s_0,F)$ is a graph marked only if either as: $s_1,s_2\in K, a_1,a_2\in V$, if $s_2\in\delta(s_1,a_1)$ and $s_2\in\delta(s_1,a_2)$, then $a_1=a_2$. Thus, it is concluded that: the language of $\Gamma(A)$ is infinite if only multigraph $\Gamma(A)$ contains circuits.

Demonstrating backwards: if given a multigraph $\Gamma=(X,U)$ with $U\subseteq X\times\sum\times X$, we can build the following associated finite automats: and $A(\Gamma,s_0)=(X,\sum,\delta,s_0,X), s_0\in X$ for $s\in X$ and $a\in\sum$, $\delta(s,a)=\{s'\in X,(s,a,s')\in U\}$. Any path in the multigraph Γ with the initial node in point s_0, corresponds to a string of the language $T(A(\Gamma,s_0))$ and vice versa.

For finding all roads that unite a node s_i with a node sj, will take s_i like initial state and s_i, like final state (F = s_i),), then the language $T(A(\Gamma,s_1,s_2))$ indicate exactly the set of all roads in the form of rows of labels.

If $\Gamma=(X,U)$ is a graph, then we may consider $\sum=U$ and $\varphi(s_i,s_j)$, and the previous automat will recognize paths written under the form of arcs. In order to obtain the paths of an unmarked graph $\Gamma=(X,U)$ written as strings over X – the set of nodes, with the help of a grammar, we may get the following result:

Given $G(\Gamma)=(\{S\}\cup U,X,S,P)$ where S is a new symbol and P:

$$P=\{S\to s_i(s_i,s_j)\mid s_i,s_j\in U\}\cup\{(s_i,s_j)\\ \to s_j(s_j,s_k)\mid(s_i,s_j),(s_j,s_k)\in U\}\cup\{(s_i,s_j)\to s_j\mid(s_i,s_j)\in U\} \quad (6)$$

Obviously, the language $L(G(\Gamma))$ comprises all paths of the graph Γ written as strings of nodes. In order to generate only the paths that leave, for instance, from a given node s_i, we will eliminate all rules of the form: $S\to s_k(s_k,s_j)$ for any $k\neq i$.

Similarly, in order to generate only paths which end in a given node s_j, we will eliminate all end rules of the form: $(s_i,s_k)\to s_k$, with $k\neq j$.

It is worth noting the fact that, if the graph Γ comprises circuits, then the language $L(G(\Gamma))$ is infinite.

The demonstration is the following:

Given $s_{i1},\ldots,s_{ip},s_{i1}$ in the graph Γ and the associated string x. Obviously, any sequence x^n represents a path in Γ, so it is in the language $L(G(\Gamma))$ and, therefore, this language is infinite. Vice versa, if $L(G(\Gamma))$ is infinite, then according to the lemma *uvw* for

regular languages, it results that for any string z long enough to be written under the form of *uvw* with $|v| \geq 0$, so that $uv^i w \in L(G(\Gamma))$, for any $i \geq 0$. If $v = s_{i1}s_{i2}...s_{ip}$ because $vv = s_{i1}s_{i2}...s_{ip}s_{i1}s_{i2}...s_{ip}$ represents a path in Γ, it result that vs_{i1} represents a *circuit*.

Another way to address these problems and namely, to obtain the roads from a graph using the grammar is used to *bypass trees*.

Whether G = (V, VT, S, P), a context-free grammar, so that every derivation D in grammar G, is associated with a derivation tree, such:

- is marked with the S tree root;
- if a tree node is marked with an unfinished A and in the derivation D this nonterminal is rewritten using rules : $A \to x_1x_2...x_r$, x_i being the terminal or non-terminal symbols, then node A has r descendants mark from left to right with the symbols $x_1x_2...x_r$.
- According [ATAN02], [ATAN07], [JALO06], we may assign to a graph $\Gamma = (X,U)$, the grammar: $G(\Gamma) = (\{S\} \cup X, \{b\}, S, P)$, where:

$$P = \{S \to s_i \mid s_i \in X\} \cup \{s_i \to s_j \mid (s_i, s_j) \in U\} \cup \{s_j \to b \mid s_i \in X\} \quad (7)$$

Any derivation in the grammar $G(\Gamma)$ will be of the form: $S \Rightarrow s_{i1} \Rightarrow s_{i2} \Rightarrow ... \Rightarrow s_{ik} \Rightarrow b$, with $s_{i1}s_{i2}...s_{ik}$. We must make clear the fact that, when we refer only to paths from the node s_i to the node s_j, then we keep the initial and end rules, namely $S \to s_i$ and $s_j \to b$.

We must mention the fact that all derivations of *b* in the grammar $G(\Gamma)$ will indicate such paths. Based on the building of grammar $G(\Gamma)$ we may find all optimal paths, in terms of transport, on certain itineraries, with certain restrictions, in time units.

In order to generate all possible itineraries that meet the conditions of a problem, it is necessary to be constructed by a linear grammar $G(\Gamma)$ for each race, so to have all itineraries that start with a point of departure, according to that race.

For finding a minimum total road duration, going once through each point of the graph, there is at least one solution is found through Hamiltonian paths of a graph Γ.

Given $\Gamma = (X,U)$, a marked graph, so that to any arc of *U* we assign a positive real number expressing the duration of passing it, or otherwise, the cost of passing that arc. This problem may have interpretation in transport issues, in the technological flow of processing some parts $p_1, p_2, ..., p_n$, on a certain machine, by replacing a part p_i and adapting the machine in order to process another part p_j, so that the time $c(p_i,p_j)$ for the preparation thereof be minimal. In order to solve these problems, we can use optimal and heuristic algorithms. In order to generate all Hamiltonian paths in a graph, there's the method of Latin multiplication, mentioning the fact that it is difficult to program in order to be executed on the computer, assuming the memorizing of all paths with the length *I* in order to be able to generate paths with the length *i+1*.

An algorithm that requires little memory, and which can be applied to any graph, is the algorithm based on the algorithm for generating permutations of a set in a lexicographical order.

Be an oriented graph $\Gamma = (X,U)$, $X=\{a_1, a_2, \ldots, a_n\}$, X orderly crowd with the orderly indices of the nodes, extending this to X^* introducing the orderly lexicographer between the strings so that $x, y \in X^*$:

If $x \in \Pr ef(y)$, than it says that $x < y$

If $x \notin \Pr ef(y)$ and $y \notin \Pr ef(x)$, but $x = x_1 a_i x_2$, $y = x_1 a_j x_3$ with cu $x_1, x_2, x_3 \in X^*$ and $a_i < a_j$ than it says that $x < y$. We build the following grammar, independent from the context:

$G = (\{S\} \cup X, \{b\}, S, P)$, where:

$$P = \{S \to a_1 a_2, \ldots, a_n\} \cup \{a_i \to b \mid i = 1,2,\ldots,n\} \cup \quad (8)$$

$$\{a_i \to a_{i1} a_{i2} \ldots a_{ik}, i \notin \{i_1, i_2, \ldots, i_k\}, (a_i, a_{ij}) \in U, \forall j \text{ și } i_1 < i_2 < \ldots < i_k\}.$$

It observed that exists an unique derivation shift A, in report with the grammar G, with the following proprieties:

- Any way in A, has the length at most $n+1$;
- Any way contains different nodes;
- The shift is maximal, in the sense that no rule $a_i \to b$ can't be replaced with an undetermined rule without violate the integrate to one of the proprieties 1 or 2.

There are removed from this shift all the ways, which have the length less than $n+1$ after its eliminated all the terminals arch, so the one who has the form $a_i \to b$, resulting a new shift A'.

Starting from the following theory [ATAN07], [CREA04], [JALO06]: " The shift A' contains all the Hamiltonians ways from graph Γ and the order endings ways from A', match with the lexicography order of the Hamiltonian's ways from Γ ", resulting that the engendering algorithm of the Hamiltonian's ways, from graph Γ, it reduce to engendering from left to right of the shifts ways A'. Being $\Gamma' = (\{0,1,2,\ldots,n,\infty\}, U')$ the extending graph, where:

$$U' = \{(i,j) \mid (a_i, a_j) \in U\} \cup \{(o,i) \mid \text{ exist } (a_i, a_j) \in U\} \cup \{(i,\infty) \mid i = 1,2,\ldots,n\} \quad (9)$$

Considering the matrix

$$m(i,j) = \min\{k \mid k > j \text{ și } (i,k) \in U',\ i,j = 0,1,2,\ldots,n\}. \quad (10)$$

Being the graph:

$$\Gamma = (\{a_1, a_2, a_3, a_4\}\{(a_1,a_2),(a_1,a_3),(a_1,a_4),(a_2,a_4),(a_3,a_2)\}). \quad (11)$$

Consider the associate graph:

$$\begin{aligned}
&\Gamma = (\\
&\qquad \{0,1,2,3,4,\infty\}, \\
&\qquad\quad \{ \\
&\qquad\quad (1,2),(1,3),(1,4), \\
&\qquad\quad (2,4),(3,2), \\
&\qquad\quad (0,1),(0,2),(0,3),(0,\infty), \\
&\qquad\quad (1,\infty),(2,\infty),(3,\infty),(4,\infty) \\
&\qquad \} \\
&\qquad),
\end{aligned} \quad (12)$$

And the corresponding matrix is the following:

$$\begin{bmatrix} 1 & 2 & 3 & \infty & \infty \\ 2 & 2 & 3 & 4 & \infty \\ 4 & 4 & 4 & \infty & \infty \\ 2 & 2 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \end{bmatrix},$$

To get the crowd $H(\Gamma)$ and not *P(X),* the necessary modifications which are made, are necessary data like $(a_{rk}, a_{rk+1}) \in U$ for any *k*. The verification is made by using the matrix *m*.

Considering the extension of the concept of Hamiltonian's way. Being $\Gamma = (X, U)$ un orientated graph, $X=\{a_1, a_2, ..., a_n\}$.

Any n with form v = (p_1, p_2, ... ,p_n), where p_i is un positive whole number for any i, it is called frequency vector. A way from Γ passing through the node a_i exactly on p_i or, it is called Hamiltonian way engender associated vector v. I this case, if *v* = (1 ... n), then determine the classical notion of Hamiltonian way. The problem of searching the Hamiltonian's ways, generalized, associated a vector $v=(p_1,p_2,...,p_n)$, can be reduced by one usual Hamilton way so that, for each node a_i we insert p_i - *1* new distinct nodes.

3. MODELING THE ECONOMIC SYSTEMS WITH HELP OF THE GRAPH THEORY

If the graph is seen like a image of a system, the nodes representing the components of the system, than the immediate interpretation of an arch (x_i, xj) is that the component x_i, influences directly the component x_j. If the nodes are described as possible moods for the economic system, it can be said that an arch (x_i, *xj*) signify the fact that the system can pass directly from status xi, in status x_j. In both of the cases it has to do only with information about direct links; tough if a component xi doesn't influence directly the component x_j it can be influenced by other components, existing a series of intermediate components : x_1, *x2,...,*x_k, each a direct influence on the next and x_i, directly on x_1, while x_k has directly influence on x_j. So, if it can be realized the through from stage x_i, directly in stage x_j, it could still go through several stages and through other intermediate states. Since finding these influences or possible transitions , is usually very important, this thing isn't quite simple to realize for the case of a system with many components, therefore it is necessary to formalize the notion of possible ''influences" and '' crosswalks'', not necessarily directly. It is obvious that „x_i influences x_j" or "the crosswalk from stage xi in stage x_j" is equivalent to saying that there in a graph exists a way from node x_i to node x_j.

3.1 Search algorithms based on graph theory

For modelling, the economic system it are used several techniques. A method used frequently, is the one that appeals to search algorithms.

In [CORM02], [RADE02], is described the algorithm with help of which it makes searching for, in a directed graph with a finite number of nodes, to find all the possible ways.

So:

1 Step: it is build the Boolean matrix is built of direct adjacent, corresponding to the graph, noted with *A*. In this situation all the ways have the length 1.

It must be noticed that there is a connection between this matrix and the roads of length 2.

Being two nodes x_i and x_j from graph. The existence of a road of length 2 between them implies the existence of a node x_k, from the graph, with the property that there are the arch (x_i, x_k) as well as arch and arc (x_i, x_k) . To see if this exists, it takes at a run every node of the graph and check if there is or not both arcs $((x_i, xk)$ and $(x_i, xk))$. This is equivalent with checking if the directly adjacent of the Boolean matrix, exists any index *k* so that the k-line element *i* and the element *k* of the column *j* are both equal to 1:

$\dot{+}$	0	1
0	0	1
1	1	1

$\dot{\times}$	0	1
0	0	0
1	0	1

then the verifications, described above, appear to be equivalent to the process of verification of the situation in which the element from the position (i,j) of A^2 equal to 1. The value 1 only say that there is at least one way, whose length is 2, between x_i, and x_j. If you want to determine the number of ways of length 2, there are used the rules of multiplication and addition.

Also, it can be observed that if there is a road of length 3 situated between nodes x_i, and x_j, , this implies the existence of a node x_k so that you can determine the existence of a road with length 2 from x_i, to x_k , and an arch from x_k to x_j, , which is equivalent with the check of existence to at least of an index *k* so that the element *k* placed on line *i* of the matrix A^2 and the element *k* on the j-th column in the matrix *A* are at the same time equal with 1 or, more simple, if the element (i,j) from A^3 is 1.

Starting from the ones presented above shows that the existing roads with length *k*, is given by the matrix values A^k , if you have used rules of Boolean algebra and their number are given by A^k if the usual rules were used.

Step 2: we calculate, in succession, the power of *A* till the power of A^{n-1}.

If between the nodes x_i, and x_j exists a road with length $\geq n$, then he will contain a number of nodes at least equal to $n+1$ and, as in the graph are just *n* peaks, it is clear that at least one, for example x_k,, will appear twice. Here will be, in this case, a road from the x_i, to the first appearance of the x_k, and a road from the last apparition of the x_k and x_j. Eliminating all the nodes of the first appearance of x_k and the last one, it results a way from x_i, to x_j, xj, in which x_k appears only once. Applying the method described above, all nodes with multiple appearances on the road, it will get a road from x_i, to x_j, in which each node appeared only once, which is obviously less than n arches. In conclusion, if there is at least one way from x_i, to x_j , then there is an elementary way and there will exist a power of *A*, between A^1 and A^{n-1}, in which the position (i,j) is different from 0. In order to demonstrate the existence of a road between any two nodes it is necessary to calculate only the first $n-1$ powers of *A*.

Step 3: we calculate the matrix $D = A + A^2 + ... + A^{n-1}$

If you want only the determination of roads between the nodes, and not their number, you use multiplication and the Boolean adding, and in accordance with the above observation, we obtain:

$$d_{ij}=\begin{cases}1 & ,dacă\ există\ cel\ putin\ un\ drum\ de\ la\ x_i\ la\ x_j\\ 0 & ,dacă\ nu\ există\ nici\ un\ drum\ de\ la\ x_i\ la\ x_j\end{cases} \quad (13)$$

In this case you can observe that:

$$A\cdot(A+I)^{n-2}=C_{n-2}^{0}\cdot A+C_{n-2}^{1}\cdot A^2+C_{n-2}^{2}\cdot A^3+...+C_{n-2}^{n-2}\cdot A^{n-1}= \quad (14)$$

$$=A+A^2+A^3+...+A^{n-1}=D$$

As a result it is enough to calculate only the power n-2 of the matrix $A+I$, and then the multiplication with A . The advantage of this method, in terms of the economy, is supported by the following observation: if D contains all pairs of arches, among which there exists a road , then:

$$D=(A+A^2+...+A^{n-1})+A^n+A^{n+1}+...+A^{n+k}=D \text{ being any } k\geq 0\Rightarrow$$

$$\Rightarrow A\cdot(A+I)^{n-2+k}=A(A+A^2+...+A^{n-1})+A^n+A^{n+1}+...+A^{n+k-1}=D= \quad (15)$$

$$A\cdot(A+I)^{n-2}\Leftrightarrow A\cdot(A+I)^{n-2+k}=A\cdot(A+I)^{n-2} \text{ being any } k\geq 0$$

Therefore, starting with power k = n-2, all matrices A^k are equal. As such, it goes directly to the calculation of any powers of $A+I$ which is greater than or equal to n-1.

For example, you can calculate:

$$(A+I)^{2^1},(A+I)^{2^2},(A+I)^{2^3},...,(A+I)^{2^r}, \quad (16)$$

where *r* represents the first power of 2 for which: $2^r\geq n-2$.

The above procedure allows you to determine if there is or not at least a road between two nodes, possibly what length he has and how many are this long. However, in practical problems, the most important is to know which actually these roads are. Considering that, all roads can be decomposed into elementary roads, and in the practical problems, they are generally matters of interest, the following steps of the algorithm will be dedicated to finding them and their decomposition. In order to find them, we use the representation of the graph through the Latin matrix from the case F.

The Latin matrices attend to the relation for defining a graph. Sequences of peaks from a graph can be characterized by certain properties. According to [MINU02]"the peaks from an orientated graph which have the same proprieties and, which succeed in a one compatible order with the order from the graph, it is called sequence". The operation can be realized with the sequences, which have the same property, is called concatenation.

The 4th step: building the Latin matrix *L* associated to the graph, where:

$$l_{ij}=\begin{cases}x_ix_j & ,if\ exist\ arch(x_i,x_j)\\ 0 & ,if\ not\ exist\ arch(x_i,x_j)\end{cases} \quad (17)$$

and matrix $\tilde{L}$ is defined by:

$$\tilde{l}=\begin{cases}x_j & ,if\ exist\ arch(x_i,x_j)\\ 0 & ,if\ not\ exist\ arch(x_i,x_j)\end{cases} \quad (18)$$

named the Latin matrix reduced.

The process for finding a way with length size 2, from x_i, to x_j implies finding a node with the property that the arches exist (x_i,x_k) and (x_k,x_j) and memorize the vector (x_i, x_k, x_j) .This is equivalent for finding a index *k* so that the element on the *k* position of *i* line, from the *L* matrix should be x_i,x_k and the element on the *k* position of column *j*, from the matrix $\tilde{L}$ should be x_j. It will be multiplied matrix *L* with the matrix $\tilde{L}$,but using special calculation rules, named Latin multiply and addition.

It is called *alphabet*, a set of signs named *symbols* or *letters* $\{s_i / i \in I\}$ where I is a ordinary *set* of indexes, defined or undefined.

It is called *word* a set defined by symbols named: $S_{i_1}S_{i_2}...S_{i_n}$.

It is called *latin multiply* an operation defined by the set of *words* from an *alphabet* noted "x_L" so:

$$s\, s_{i_1}s_{i_2}...s_i \times_L s_{j_1}s_{j_2}...s_{j_m} = s_{i_1}s_{i_2}...s_{i_n}s_{j_1}s_{j_2}...s_{j_m} \quad (19)$$

the product of two words is obtained by "*counteraction*" them.

The *Latin multiply* is *associative*, has a neutral element the word void, is not commutative and an element is irreversible only is the word is void.

It is called *Latin addition* a function defined on a set of words of an alphabet with values in the set of the parts set of words, noted "+*L*" as:

$$s_{i_1}s_{i_2}...s_i +_L s_{j_1}s_{j_2}...s_{j_m} = \left\{ \begin{matrix} s_{i_1}s_{i_2}...s_{i_n} \\ s_{j_1}s_{j_2}...s_{j_m} \end{matrix} \right\} \quad (20)$$

the sum of two words is the set having those two words.

The 5th step: is calculated, successive, the matrix:

$$L^2=L\times_L \tilde{L}\ ,\ L^3=L^2\times_L \tilde{L}\ ,\ ...,\ L^{k+1}=L^k\times_L \tilde{L} \quad (21)$$

Using the Latin multiply and addition operations, the alphabet being the set of nodes of the graph, where the multiply operation is easily to modified, the product of two elements of the matrix is 0,in case at least one is zero or a common node comes and is the Latin product of them , in contrary case.

From the way it was built, the matrix L^k will contain all the elementary ways of length *k*. Due to the fact that an elementary way has at most *n* nodes it results that:

- the first *n-1* powers of *L* contain all the ways from the graph;
- there are powers of *L* that are higher or equal to *n* and has all the elements equal to 0;
- matrix L^{n-1} contain all the Hamilton ways from the graph(if they exist);

Because obtaining the matrix *D*, using the method described earlier, needs a volume very high of calculation, for example: for a graph with 100 nodes, calculations will be 100x100 raised to 100 power, for the *D* matrix it can be applied with success the next algorithm:

Step one: the adjacency matrix A is built;

Step two: for each line is addition, Boolean, all the j line for those $a_{ij} = l$;

Step 3: it repeat the step two until the matrix remain the and there are no '1's.

The last matrix resulted is the matrix of the *D* ways, also named the matrix of total connection. This method, although easier, it does not lead which are those ways and for finding them, it applies, for instance, Latin multiply.

3.2 Optimization algorithms of the economic flow based on the graph theory

Considering that, the economic flows can be associated with some flows from the classic graph theory, in economic theory and practices it assumes using a set of algorithms, evolved, and developed in operation research. From these algorithms it was choose Ford Fulkerson.

The method Ford-Fulkerson solves the maximum flow problem. This method is based on three important theories, which exceed the algorithm and it is also used in other problems related to flows: residual networks, improvement ways and cuts [CORM02], [RADE02].

Ford-Fulkerson's method is iterative. It starts with a flow *f(u, v) = 0* for $u, v \in V$, like the initial flow with value 0.At every step of the iteration it enlarges the value of the flow by finding a „improvement way", which is a way along which its flow can be enlarged, so its value too. Repeat those steps until these is no improvement way found.

a) Residual network

For a transport network and a flow, it can be said that there is a residual network that consist from the arches that admit the biggest flow. According to [CORM02], if there is a transport network with the form: *G=(V, U)* with source *s* and destination *t*, and *f* is a flow in *G* and is considered a pair of peaks $u, v \in V$, the amount of additional flow which can be transported from *u* to *v*, without overcome the capacity *c(u,v),* is the residual capacity of the arch *(u,v)* defined by:

$$c_f(u,v) = c(u,v) - f(u,v) \tag{22}$$

Having a transportation network *G= (V,E)* and a *f* flow, the residual network of *G* induced by *f* is $G_f = (V, E_f)$, where:

$$E_f = \{(u,v) \in V \times V \ : \ c_f(u,v) > 0\} \tag{23}$$

Every arch of the residual network, or residual arch, admits a strictly positive grow of the flow.

It can be observed that *(u, v)* can be an arch in E_f even though it is not an arch in *E*, is observed that $E_f \not\subset E$ .This kind of arch *(u,v)* can appear in G_f only if $(v,u) \in E$ and if there is a positive flow from *v* to *u*. Because the flow *f(u,v)* from *u* to *v* is negative, $c_f(u,v) = c(u,v) - f(u,v)$ is positive and $(u,v) \in E_f$ .Because the arch *(u,v)* can appear in the residual network only if at least one of the arches *(u,v)* and *(v,u)* appear in the original network ,we have $|E_f| \le 2|E|$ .It can be observed that the residual network G_f is a transport network with capacity function c_f.

b) Improvement ways

According to [CORM02],"having a transport network *G= (V,E)* and a flow *f*, a improvement way *p* is a simple way from *s* to *t* in the residual network G_f. After the definition of the residual network, every arch *(u,v)* on an improvement way admits a additional positive flow ,without breaking the restriction of capacity."

Residual capacity of *p* is the maximum capacity of the flow that can be transported along the improvement way *p*, given by the formula:

$$c_f(p) = \min\{c_f(u,v) \quad : \quad (u,v) \quad is \ the \ way \ \ p\} \tag{24}$$

c) Cuts in the transport network

Ford-Fulkerson's method grows the flow repeatedly, along the improvement ways, until it gets to a maximum flow. The theorem of the maximum flow-minimum cut, demonstrates the fact that a flow is maximum, if and only if in the residual network doesn't exist ways of improvement.

A cut *(S,T)* of a transport network *G=(V,E)* is a partition of the *V* set in the *S* and *T=V-S* set, so $s \in S$ and $t \in T$ .If *f* is a flow ,then the cut flow *(S,T)* is defined equal to *f(S,T)*.The capacity of the cut *(S,T)* is *c(S,T)*. A minimum cut is the cut with the lowest capacity from the whole network cuts.

d) Ford-Fulkerson algorithm

In every iteration a Ford-Fulkerson method is searching for a randomly improvement way *p* and it is growing the flow *f* along way *p* at residual capacity $c_f(p)$. Implementation of the method allows the calculation of the maximum flow in the *G= (V,E)* graph, updating the flow *f[u,v]* between any other two peaks which are bounded thru an arch. If *u* and *v* are not bounded thru an arch in any other direction, it is suppose that *f[u,v]=0*. The value of the capacity of the peaks *u* and *v* is given by the function *c(u,v)* computable in a constant time , *c(u,v)=0* if $(u,v) \notin E$.

Schematization of the algorithm Ford-Fulkerson *(G,s,t):*

1: **for** every arch $(u,v) \in E[G]$ **execute**

2: $f[u,v] \leftarrow 0$

3: $f[v,u] \leftarrow 0$

4: **as long** there is a way p from s to t in the residual network G_f **execute**

5: $c_f(p) \leftarrow \min\{c_f(u,v) : \ (u,v) \ is \ on \ \ p \ \ way\}$

6: ***for every*** arch (u,v) din p ***execute***

7: $f[u,v] \leftarrow f[u,v] + c_f(p)$

8: $f[v,u] \leftarrow -f[v,u]$

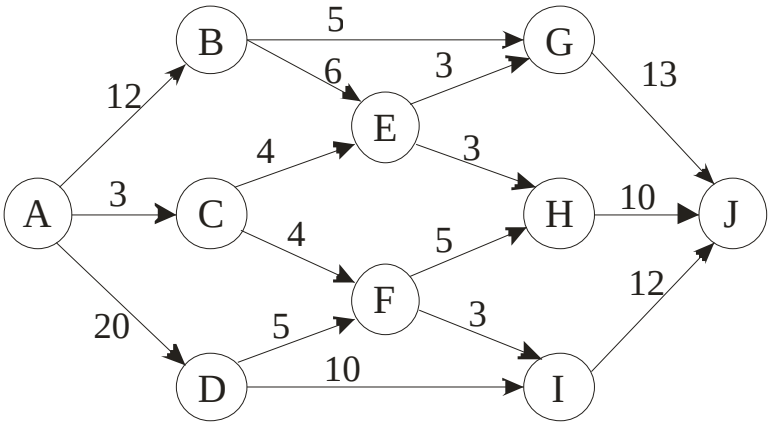
Lines between 1 and 3 initialize the flow *i* with value equal to 0. The cycle "as long" from the 4-8 lines finds, by turn, an improvement way p in G_f and increases the *f* flow along p with the value of the residual capacity $C_f(p)$. When there is no improvement way then f is at a maximum flow.

The execution time of the Ford-Fulkerson algorithm depends on the way of determination of the improvement way *p*. If the chosen way is wrong it can happen that the algorithm does not stop: value of the flow increases successive, but doesn't converge to its maximum value.

The execution time of the Ford-Fulkerson algorithm is given by $O(E \mid f^* \mid)$, where f* is the maximum flow resulted from the algorithm. The execution time for line 1 to 3 is $\Theta(E)$.Lines 4-8 executes at most | *f** |, because the value of the flow increases at every step with at least a unit.

4. IMPLEMENTATION OF OPTIMIZATION ALGORITHMS FOR A TRANSPORT PROBLEM

A transport company has 35 trucks which must move in point J. Displacement of the 35 trucks from one place to another is done in stages, so in the first step is to get as many of them in point J. In their way, the trucks have to make one more stop in one of the other intermediate point B, C, D, E, F, G,H ,I, J . Reception conditions, supply and so on, are to be a limitation of routes used, existing capacities are listed on the network arches.



The objectives are to determine the optimal transport plan so that, at this stage a large number of trucks could go toward point J.

The problem of maximum flow crossing transport network, has the following mathematical form, using linear programming in order to fix it:

$$\begin{cases} z_{\max} = \varphi_{\max} \quad \hat{i}n \quad conditiile \\ 0 \le \varphi'_{ij} \le c'_{ij} \quad (i, j = \overline{0, n+1}); \\ \sum_{j=0}^{n+1} \varphi'_{ij} = \sum_{j=0}^{n+1} \varphi'_{ij} \quad (i = \overline{1, n}); \\ \sum_{j=0}^{m+1} \varphi'_{j0} = \sum_{j=0}^{n+1} \varphi'_{0j} - \varphi \\ \sum_{j=0}^{n+1} \varphi'_{j,n+1} = \sum_{j=0}^{n+1} \varphi'_{n+1} + \varphi \end{cases}$$

Where: $u = (x_i, y_j)$ is the arch, φ is the flow value and $\varphi'_{(u)} = \varphi'_{ij}$ is arch flow.

Automaton corresponding to algorithm is $M = (Q, \Sigma, \delta, q_0, F)$ and has the following values

$Q = \{A, B, C, D, E, F, G, H, I, J\}$, F={J}, alphabet input is given by $\Sigma = \{12,3,20,5,6,4,4,5,10,3,3,5,3,13,10,12\}$, passing functions are defined in the following manner:

$\delta(A,12) = B$ $\quad$ $\delta(B,5) = G$ $\quad$ $\delta(C,4) = E$

$\delta(A,3) = C$ $\quad$ $\delta(B,6) = E$ $\quad$ $\delta(C,4) = F$

$\delta(A,20) = D$

$\delta(D,5) = F$ $\quad$ $\delta(E,3) = G$ $\quad$ $\delta(F,5) = H$

$\delta(D,10) = I$ $\quad$ $\delta(E,3) = H$ $\quad$ $\delta(F,3) = I$

$\delta(G,13) = J$ $\quad$ $\delta(H,10) = J$ $\quad$ $\delta(I,12) = J$

Regular grammar $G = (V_N, V_T, S, P)$ where $V_N = \{A, B, C, D, E, F, G, H, I, J\}$, $V_T = \{12,3,20,5,6,4,4,5,10,3,3,5,3,13,10,12\}$, with the set of rules for generating:

1) $A \to 12B$ 2) $A \to 3C$ 3) $A \to 20D$
4) $B \to 6E$ 5) $B \to 5G$
6) $C \to 4E$ 7) $C \to 4F$
8) $D \to 5F$ 9) $D \to 10I$
10) $E \to 3G$ 11) $E \to 3H$
12) $F \to 5H$ 13) $F \to 3I$
14) $G \to 13J$
15) $H \to 10J$
16) $I \to 12J$
17) $J \to \lambda$

Possible derivations, based on the grammar are:

$A \to^{(1)} 12 \cdot B \to^{(5)} 12 \cdot 5G \to 12 \cdot 5 \cdot 13J \to^{(17)} 12 \cdot 5 \cdot 13$
$A \to^{(1)} 12B \to^{(4)} 12 \cdot 6E \to^{(10)} 12 \cdot 6 \cdot 3G \to^{(14)} 12 \cdot 6 \cdot 3 \cdot 13J \to^{(17)} 12 \cdot 6 \cdot 3 \cdot 13$
$A \to^{(1)} 12B \to^{(4)} 12 \cdot 6E \to^{(11)} 12 \cdot 6 \cdot 3H \to^{(15)} 12 \cdot 6 \cdot 3 \cdot 10J \to^{(17)} 12 \cdot 6 \cdot 3 \cdot 10$
$A \to^{(2)} 3C \to^{(6)} 3 \cdot 4E \to^{(10)} 3 \cdot 4 \cdot 3G \to^{(14)} 3 \cdot 4 \cdot 3 \cdot 13J \to^{(17)} 3 \cdot 4 \cdot 3 \cdot 13$
$A \to^{(2)} 3C \to^{(6)} 3 \cdot 4E \to^{(11)} 3 \cdot 4 \cdot 3H \to^{(15)} 3 \cdot 4 \cdot 3 \cdot 10J \to^{(17)} 3 \cdot 4 \cdot 3 \cdot 10$
$A \to^{(2)} 3C \to^{(7)} 3 \cdot 4F \to^{(12)} 3 \cdot 4 \cdot 5H \to^{(15)} 3 \cdot 4 \cdot 5 \cdot 10J \to^{(17)} 3 \cdot 4 \cdot 5 \cdot 10$
$A \to^{(2)} 3C \to^{(7)} 3 \cdot 4F \to^{(13)} 3 \cdot 4 \cdot 3I \to^{(16)} 3 \cdot 4 \cdot 3 \cdot 12J \to^{(17)} 3 \cdot 4 \cdot 3 \cdot 12$
$A \to^{(3)} 20D \to^{(8)} 20 \cdot 5F \to^{(12)} 20 \cdot 5 \cdot 5H \to^{(15)} 20 \cdot 5 \cdot 5 \cdot 10J \to^{(17)} 20 \cdot 5 \cdot 5 \cdot 10$
$A \to^{(3)} 20D \to^{(8)} 20 \cdot 5F \to^{(13)} 20 \cdot 5 \cdot 3I \to^{(16)} 20 \cdot 5 \cdot 3 \cdot 12J \to^{(17)} 20 \cdot 5 \cdot 3 \cdot 12$
$A \to^{(3)} 20D \to^{(9)} 20 \cdot 10I \to^{(16)} 20 \cdot 10 \cdot 12J \to^{(17)} 20 \cdot 10 \cdot 12$

The language generated is L={$12 \cdot 5 \cdot 13$, $12 \cdot 6 \cdot 3 \cdot 13$, $12 \cdot 6 \cdot 3 \cdot 10$, $3 \cdot 4 \cdot 3 \cdot 13$, $3 \cdot 4 \cdot 3 \cdot 10$, $3 \cdot 4 \cdot 5 \cdot 10$, $3 \cdot 4 \cdot 3 \cdot 12$, $20 \cdot 5 \cdot 5 \cdot 10$, $20 \cdot 5 \cdot 3 \cdot 12$, $20 \cdot 10 \cdot 12$}

In the case where for each sub graph defined by the grammar derivations, the maximum flow is established, there is the whole graph value of 41 trucks driving from point A to reach the point J. This is not correct because there are duplicates of the minimum flows on certain routes, which must be removed. Solving the problem with the help of Ford-Fulkerson algorithm, the maximum flow starting from point A, and arrives at the point J, is 28 trucks.

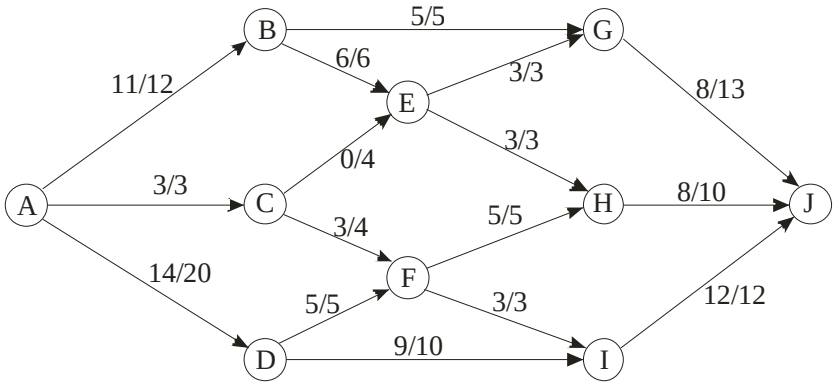


Figure 1. Solving problem with Ford-Fulkerson algorithm

It is observed that the difference between the two values is given by the carrying capacity of the point G, H and i. these values are:

○ for the G-spot maximum capacity supported is 8, and from the grammar we obtain 11

○ for the H-spot maximum capacity supported is 8, and from the grammar we obtain 14

○ for the I-spot maximum capacity supported is 12, and from the grammar, we obtain 16

The problem arises from section I for which, through grammar obtained above, there are the values given by sub graphs: ACFIJ, ADFIJ and ADIJ, which have a minimum capacity of 2, 3 and 10. Point 1 may have the maximum flow of just 12, according to problem solving using algorithm, resulting one from the two sub graphs (ACFIJ, ADFIJ) has in fact the minimum capacity of 2, and the other is doubled as value.

To remove the duplicate values that are not found, there's no method to bring a plus and to lead to a result equal to that which is found by the method obtained by Ford Fulkerson.

In the case of a graph with a small number of nodes, this is very easy to apply.

5. CONCLUSIONS

In the paper, we simplified the problem specifying economic processes so that simple notations as: a, b, c, d, signifies actions with well-defined timescales, and pairs of actions are represented by a string of characters so that will no longer be any other restrictions.

There were treated formal logic elements, and has been described as part of mathematical logic in which logical variables are sentences.

Seeing the graph like an image of system, the nodes represent the components of the system, then an immediately interpretation of an arch (x_i, x_j) it is the one specifying that component x_i is the one which influences directly the component x_j. Using this interpretation there are presented two algorithms : the algorithm in which can be found all the ways on the graph, orientated to a finite number of nodes, as well the construction algorithm of the Latin matrix in which the alphabet represent the set of the nodes of the graph.

All roads in the merged graph are decomposed into elementary roads, this being followed in practice specification economic systems. Carrying out the decomposition in the basic road is performed by using the Latin matrix.

It was presented the Ford-Fulkerson algorithm and how to solve a problem with this algorithm. The problem, initially, is resolved with the help of formal languages and grammar and language are identified. The result thus obtained was compared with the problem solved by the Ford-Fulkerson algorithm.

REFERENCES

Atanasiu A., **Limbaje formale și automate**, Editura Infodata, Cluj, 2007;

Atanasiu I., Raicu D., Sion R., Mocanu I., **Limbaje formale și Automate**, Îndrumar pentru aplicații, 2002

Bocu D., Bocu R., **Modelare obiect orientată cu UML**, Editura Albastră, Cluj Napoca, 2006

Cocan M., Pop B., **Logică computațională**, Editura Albastră, Cluj Napoca, 2006

Cormen H. Thomas, Charles E. Leiserson, Rivest R. Roland, **Introducere în algoritmi**, Editura Agora, 2002

Creanga I, Reischer C., Simovici D., **Introducere algebrică în informatică**, Editura Junimea, Iași, 2004

Jalobeanu C, Marinescu D, **Bazele teoriei calculului limbajelor formale și automate**, Editura Albastră, Cluj Napoca, 2006

Minuț P., Kupan A. P., **Cercetări operaționale**, Târgu Mureș, Editura „Dimitrie Cantemir", 2001

Minuț P., **Matematici aplicate în economie**, Târgu Mureș, Editura „Dimitrie Cantemir", 2002

Rădescu N., Rădescu E., **Probleme de teoria grafurilor**, Editura Scrisul Românesc, 2002

1

[ATAN07]	Atanasiu A., **Limbaje formale și automate**, Editura Infodata, Cluj, 2007
[ATAN02]	Atanasiu I., Raicu D., Sion R., Mocanu I., **Limbaje formale și Automate**, Îndrumar pentru aplicații, 2002
[BOCU06]	Bocu D., Bocu R., **Modelare obiect orientată cu UML**, Editura Albastră, Cluj Napoca, 2006
[COCA06]	Cocan M., Pop B., **Logică computațională**, Editura Albastră, Cluj Napoca, 2006
[CORM02]	Cormen H. Thomas, Charles E. Leiserson, Rivest R. Roland, **Introducere în algoritmi**, Editura Agora, 2002
[CREA04]	Creanga I, Reischer C., Simovici D., **Introducere algebrică în informatică**, Editura Junimea, Iași, 2004
[JALO06]	Jalobeanu C, Marinescu D, **Bazele teoriei calculului limbajelor formale și automate**, Editura Albastră, Cluj Napoca, 2006
[MINU01]	Minuț P., Kupan A. P., **Cercetări operaționale**, Târgu Mureș, Editura „Dimitrie Cantemir", 2001
[MINU02]	Minuț P., **Matematici aplicate în economie**, Târgu Mureș, Editura „Dimitrie Cantemir", 2002
[RADE02]	Rădescu N., Rădescu E., **Probleme de teoria grafurilor**, Editura Scrisul Românesc, 2002