ALGORYTMY

Vol. X • No. 18 • 1973

INSTYTUT MASZYN MATEMATYCZNYCH

A L G O R Y T M Y

Vol. X. N° 18 1973

Adres Redakcji: Warszawa, ul.Krzywickiego 34, tel.28-37-29

Druk IMM z. 126/73 n. 550 egz. GP-II-1416/68.

TREŚĆ
СОДЕРЖАНИЕ
CONTENTS

ALGORYTMY, X, N° 18, 1973, 5-20 681.322.06:800.92:801.54

MACHINE-LIKE LANGUAGES AND PROCESSES

Ludwik CZAJA

Uniwersytet Warszawski
Warszawa, PKiN

Received 14.11.1972

A semantic model for lower-level programming languages has been constructed. The class of processes describable by means of this model-language has been compared with that examined in [5].

INTRODUCTION

Semantic properties of programming languages are being investigated at various generality levels. It depends on what is assumed about their instructions. Some authors ([2], [3], [4], [6] among others) provide their model-programming-languages with fixed operational statements. These are, for instance, assignment statements, conditional jump statements, loop statements etc. They have their fixed meanings regardless of particular realization of the language*) . However, the meanings of function symbols (which in fixed realization come about to be the names of operations in a value set) depend on realization. The problems of halting, correctness and equivalence are being examined for programs (in papers mentioned) on such a generality level. When dealing with the most general semantic properties, one has to construct a model-language free of statements to be singled out as in the above said. A model-language is, in the case, the language of general computer of

*) Roughly, by "realization of the language" we understand a (non-void) set, the elements of which are values of language's variables and a mapping which assigns to each functional symbol (functor) of the language, certain operation in the set.

inner control. In other words, a scheme, permitting to get any concrete programming language, by means of specifying instruction meanings.

In this paper, we confine ourselves to (a) construct a language (free of fixed-meaning statements) which would model "lower level" programming languages, i.e. those close to machine or assembly languages, (b) compare the processes describable in this language, with those investigated in [5]. This model-language we call L_o. Other semantic properties of L_o, likewise model-language for "higher level" languages (as ALGOL, FORTRAN, etc.) and its semantic properties have been examined in [1].

The instruction of L_o we define syntactically, as a string of the form of elementary term (in Polish notation). However, we interprete such terms unlike usually. That is why we say "instruction" instead of "term" and "instruction symbol" instead of "functor". The program, however, we define semantically, as an object dependent on interpretation of the language. In this, we base on the observation that in real machine languages, any program (with its data) may be wholly characterized by valuation of memory registers in the set of instruction codes and by indicating registers containing coded instructions of program involved. Thus, by "program" we shall understand a pair $< A, v >$, where A is a subset of set V of all variables, v is a valuation of V in the set of instruction codes. Therefore, from a formal standpoint, programs are not expressions from L_o.

1. BASIC LANGUAGE L_o

Instructions of L_o are built of the following symbols:

(i) Variables, denoted $x, y, \ldots$ may be with indices. V is their set.

(ii) Instruction symbols, denoted $\alpha, \beta, \ldots \varphi, \delta, \ldots$ may be with indices. Φ_0 is their set. As usually, Φ_0 breaks up into disjoint subsets $\Phi_{00}, \Phi_{01}, \Phi_{02}, \ldots$ of 0-argument, 1-argument, 2-argument etc. instruction symbols.

Definition 1

The set I_0' of instructions with arguments is the set of all strings of the form $\varphi x_1 x_2 \ldots x_k$, where $\varphi \in \Phi_{ok}$ $x_i \in V$, $i = 1, 2, \ldots k$, $k > 0$. The set I_0 of instructions we define as the union: $I_0 = \Phi_{00} \cup I_0'$. The elements from I_0 we call instructions of L_0 or basic instructions.

Definition 2

Out of all variables we select one, which will play special part in further considerations. We call it instruction counter and denote by q. Let J be a non-empty set. Any function $v: V \longrightarrow J \cup V - \{q\}$, satisfying the condition $v(x) \in V - \{q\}$ iff $x = q$, we call a valuation of V in the set $J \cup V - \{q\}$. By W we denote the set of all valuations.

Example 1

Let N be the set of all non-negative integers. If $V = \{q\} \cup N$, $J = N$, then $V - \{q\}$ may be regarded as a set of memory addresses, $v(x)$ - as the contents of x. A valuation is a memory state.

Example 2

Let $\mathcal{A}$ be a finite alphabet and let $\mathcal{A}^*$ be the set of all strings over $\mathcal{A}$ (with empty string). If we take $J = \mathcal{A}^*$ (and define suitable instructions, e.g. example 4), we will get a language for texts processing. A valuation is an instantaneous description of an attachment of strings to variables.

Definition 3

We say, we have an interpretation $\mathcal{I}$ of the language L_o iff:

(i) there is given a partial and one-to-one function $T: J \longrightarrow I_o$ called coding,

(ii) with each symbol $\varphi \in \dot{\Phi}_{ok}$ for $k > 0$, there is associated a partial function $\varphi_{\mathcal{I}} : V^k \longrightarrow W^W$, where V^k is the Cartesian product of k factors $V \times V \times \ldots V$ and W^W is the set of all functions $W \longrightarrow W$; with each symbol $\varphi \longrightarrow \dot{\Phi}_{oo}$ there is associated a fixed function $\varphi_{\mathcal{I}} : W \longrightarrow W$.

Function $\varphi_{\mathcal{I}}$ is said to be an interpretation of symbol φ. Function $\varphi_{\mathcal{I}}(x_1, x_2, \ldots x_k) : W \longrightarrow W$ we denote by $(\varphi\, x_1 x_2 \ldots x_k)_{\mathcal{I}}$ and call interpretation or action of instruction $\varphi\, x_1 x_2 \ldots x_k$.

It is clear that the interpretation of an argument-free symbol is identical with interpretation of instruction built up of this symbol.

Reason of definition 3

Point (i) is obvious if real computer coding is taken into account. Function T is partial to mirror the fact that not every contents of memory cell must be a meaningful machine instruction code.

Point (ii). In programming languages, an action of instruction alters a valuation, therefore it must be defined as function $W \longrightarrow W$. According to definition 1, by means of given k-argument $(k > 0)$ instruction symbol, we create many instructions, writing out k-tuples of variables past the symbol. These different instructions ought to act differently (in general), thus, the instruction action has to depend on k-tuple of variables in the instruction. Therefore, k-argument instruction

symbol should be understood as a name of function which associates mappings $W \to W$ with k-tuples of variables, i.e. as a name of function $V^k \longrightarrow W^W$. By this making out of instruction symbols, instructions themselves can be understood as names of functions $W \to W$. In general, since, in practice, instruction $\varphi x_1 x_2 \dots x_k$ does not act for all k-tuples $x_1 x_2 \dots x_k$, the interpretation of instruction symbol must be a partial function. Argument-free instruction symbols we understand as names of selected elements from W^W.

Example 3

Let $J = N$ and let us take the partial algebra:
$\mathcal{O}\mathcal{l} = \{J \cup V - \{q\}, s, \dot{-}, \odot, \oplus\}$, where s is the successor, $\dot{-}$ is the substraction in N, and partial operations $\odot$ $\oplus$ are defined as follows: $0 \odot x = 0$, $1 \odot x = 0$ $\oplus$ $x = x \oplus 0 = x$, for all $x \in V - \{q\}$ (0 and 1 are integers).
Let $\Phi_0 = \Phi_{02} \cup \Phi_{03}$ where $\Phi_{02} = \{\alpha\}$, $\Phi_{03} = \{\beta, \gamma, \delta\}$

As coding T an arbitrary numbering of instructions can be taken.

Interpretations of instruction symbols we define by means of operations in $\mathcal{O}\mathcal{l}$:

$$\alpha_2(x,y)\ (v) = (v|V - \{x,q\}) \cup \{\langle x, s(v(x))\rangle\ ,\ \langle q,y\rangle\}$$
$$\beta_2(x,y,z)\ (v) = (v|V - \{x,q\}) \cup \{\langle x, v(x) \dot{-} v(y)\rangle\ ,\langle q,z\rangle\}$$
$$\gamma_2(x,y,z)\ (v) = (v|V - \{x,q\}) \cup \{\langle x, v(y)\rangle\ ,\ \langle q,z\rangle\}$$
$$\delta_3(x,y,z)\ (v) = (v|V - \{q\}) \cup \{\langle q, ((1 \dot{-} v(x)) \odot y) \oplus ((1 \dot{-} (1 \dot{-} v(x))) \odot z)\rangle\}$$

In the above notation, $v|V - \{\dots\}$ denotes function v restricted to set $V - \{\dots\}$.

It is clear that αxy is the addition of unity, βxyz is the subtraction, γxyz is the assignment of integer $v(y)$ to x, δxyz is the conditional jump to y if $v(x) = 0$ and to z otherwise. Every computable function can be programmed in this interpretation (demonstration in [1]).

Example 4

Let $J = \mathcal{A}^*$, where $\mathcal{A}$ is a finite alphabet and let us take the algebra:

$\mathfrak{A} = \{\mathcal{A}^*, \cdot, \text{head}, \text{tail}\}$ where: $\cdot$ is the concatenation,

$$\text{head}(s) = \begin{cases} \Lambda \text{ (empty)} & \text{if } s = \Lambda \\ a_1 & \text{if } s = a_1 \dots a_p, \text{ where } a_i \in \mathcal{A}, \end{cases}$$

$i = 1, \dots p$, and tail is defined by equality:
$s = \text{head}(s) \cdot \text{tail}(s)$.

Let $\Phi_0 = \Phi_{03} \cup \Phi_{04}$ where $\Phi_{03} = \{\varrho, \delta\}$, $\Phi_{04} = \{\omega\}$. A coding T can be obtained, e.g. by a superposition of mappings $\mathcal{A}^* \to N$ and $N \to I_0$, which may be easily defined, having sets $\mathcal{A}$ and V. Interpretations of instruction symbols we define by means of operations in $\mathfrak{A}$ and comparison of strings from $\mathcal{A}^*$:

$$\varrho_{\mathfrak{I}}(x,y,z)(v) = (v \mid V - \{x,y,q\}) \cup \{\langle x, \text{head}[v(y)]\rangle, \langle y, \text{tail}[v(y)]\rangle, \langle q,z\rangle\}$$

$$\delta_{\mathfrak{I}}(x,y,z)(v) = (v \mid V - \{x,y,q\}) \cup \{\langle x, v(x).v(y)\rangle, \langle y, \Lambda\rangle, \langle q,z\rangle\}$$

$$\omega_{\mathfrak{I}}(x,y,z,r)(v) = (v \mid V - \{q\}) \cup \{\langle q, \underline{\text{if}}\ v(x) = v(y)\ \underline{\text{then}}\ z\ \underline{\text{else}}\ r\rangle\}$$

It is clear that ϱxyz is the truncation of the first letter out of string $v(y)$, δxyz is the concatenation, $\omega xyzr$ is the conditional jump to z if strings $v(x)$ and $v(y)$ are identical and to r otherwise. In point 2 of this paper an example of program (in this interpretation) translating arithmetic expressions in constructed.

2. PROGRAM AND ITS AUTOTRANSFORMATION

Let A be a set of variables: $A \subset V$ and let v be a valuation: $v \in W$. We fix an interpretation $\mathfrak{I}$.

Definition 4

The program in state v with base A is a pair $P = \langle A, v \rangle$. We will also say "program in interpretation $\mathfrak{I}$ " if there is a need to mark in which interpretation it is.

Reason of definition 4

Restriction of function v to set A is the set of pairs: $v|A = \{ \langle l, a \rangle : l \in A, \ a \in J \}$, not containing two pairs with the same first and different second elements. Thus, the first element of a pair $\langle l,a \rangle$, (predecessor) may be thought of as labelling the instruction (its code, more precisely) appearing as the second element (successor) of the pair. Therefore, P represents a program (with its data) by common understanding of the term "program".

Definition 5

For program $P = \langle A, v \rangle$ and interpretation $\mathfrak{I}$ we define two sequences $\{l_n\}, \{v_n\}$, $n = 0, 1, 2, \ldots$ as follows:

$v_0 = v$

$l_n = v_n(q)$, if $v_n(q) \in A$ (otherwise l_n does not exist)

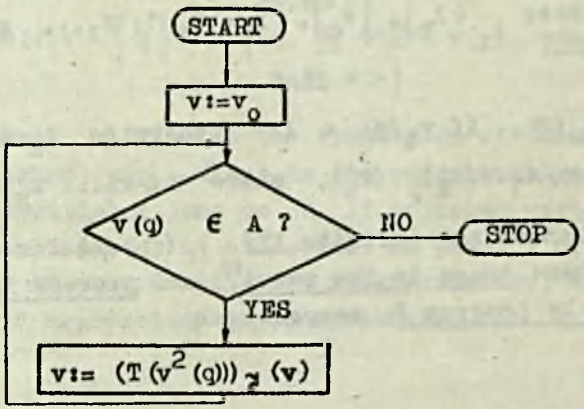
$v_{n+1} = (\varrho x_1 \ldots x_k)_{\mathfrak{I}} (v_n)$, where $\varrho x_1 \ldots x_k = T(v_n(l_n))$

These sequences are infinite iff $v_n(q) \in A$ for all n.
We call them: trace in the set $A^{*)}$ and process in the set W, performed by program P, respectively

*) see analogous notion in [2].

Note

The above definition of program and process gives an account of tendency[*)] to see the program and its data as a whole, subject to self-transformations, according to rules coded up in itself. A sequence $\langle A, v_0\rangle$, $\langle A, v_1\rangle$, $\langle A, v_2\rangle$,... is a sequence of instantaneous descriptions of program $\langle A, v\rangle$ during its run. This sequence can be named "autotransformation" of program $\langle A, v\rangle$. Base A in an autotransformation is fixed. In [1] autotransformations with altering bases have been examined, which was the ground for construction of model-language for higher-level programming languages. It follows directly from definition 5, that the trace in A and the process in W are evaluated uniquely, and that if functions v, $\varphi_{\mathcal{A}}$, T, are (by suitable assumptions) computable and so is the set A, then consecutive elements $l_0, l_1, l_2,\ldots$ and $v_0, v_1, v_2,\ldots$ are evaluated effectively. Actions being performed during autotransformation of program $P = \langle A, v_0\rangle$ and their order, may be pictured by means of the flowchart:



*) see, for instance an excerpt from [7] (page 149) : "... this approach to the study of programming languages is a relatively new one. Programming-language theory has, in the past, tended to emphasize the study of source languages and the study of translation of source languages into an initial target-language representation for purposes of execution. The present approach takes the initial representation for purposes of execution as the starting point and studies the sequence of run-time representation during execution of the function. The representation of the function at a given point during execution is referred to as an instantaneous description. The process of function evaluation may be characterized by a sequence of instantaneous descriptions".

Here, the symbol ":=" means - informally - assignment of a value to "functional variable", taking its values from the set W, whereas $T(v^2(q))$ is an instruction being the (code of) value of "individual variable" $v(q)$.

Example

Consider interpretation constructed in example 4 from point 1 of this paper and take $\mathcal{A} = \{I, +, \times, (,), ;\}$. We construct the program, which transforms arithmetic expressions (slightly simplified, for built of symbols from $\mathcal{A}$; I stands for arguments) in conventional notation into their equivalents in reverse Polish. For brevity, the pairs $\langle x, v(x)\rangle \in v$ we write in the form $x: v(x)$. Also, we omit the symbol T (coding function), that is, for instance, instead of $\langle e_1, T^{-1}(\varphi xye_2)\rangle$, we put $e_1: \varphi xye_2$. The program is the pair $\langle A, v\rangle$, where $A = \{e_1, e_2, \dots e_{17}\}$, and valuation v is the following set of pairs:

$q: e_1$
$e_1: \varphi\, xye_2$
$e_2: \omega\, xp_1e_3e_4$
$e_3: \delta\, zxe_1$
$e_4: \varphi\ rse_5$
$e_5: \omega rp_0e_9e_6$
$e_6: \omega rp_2e_{11}e_7$
$e_7: \omega rp_3e_{13}e_8$
$e_8: \omega rp_4e_{10}e_{18}$
$e_9: \omega xp_6e_{18}e_{15}$
$e_{10}: \omega\, xp_5e_1e_{15}$
$e_{11}: \omega\, xp_3e_{15}e_{12}$
$e_{12}: \omega xp_4e_{15}e_{14}$
$e_{13}: \omega xp_4e_{15}e_{14}$
$e_{14}: \delta\ zre_4$
$e_{15}: \delta\, rse_{16}$
$e_{16}: \delta\, xre_{17}$
$e_{17}: \delta\, sxe_1$
$e_{18}: \Lambda$

$p_0: \Lambda$
$p_1: I$
$p_2: +$
$p_3: \times$
$p_4: ($
$p_5:)$
$p_6: ;$
$y:$ < conventional expression >
$z: \Lambda$
$s: \Lambda$
$x: \Lambda$
$r: \Lambda$

Comment: y is an input, having an expression to be translated as its value. Consecutive characters of this expression are being out out and placed in "register" x (i.e. they are being made values of x) from which are being passed to the output z. "I" passes, by this, directly, other characters through scanning of the push-down (stack) s. This scan consists in comparing the first character in stack ("top") with the value of x. For this purpose, the top-character of the value of s, is being out out and placed in "register" r. The program terminates, when $v_n(y) = ;$. Stop-point is the variable e_{18}. There is no check of syntactic correctness, but it can be easily introduced.

End of example

3. PROCESSES

Now, we shall compare processes performed by programs of L_o language, with those considered in [5]. Let $\pi : W \to W$ be a partial function and let $v \in W$. According to [5], the sequence $\{v_n\}$, defined as $v_0 = v$, $v_{n+1} = \pi(v_n)$, $n = 0, 1, 2...$, is said to be a process and is infinite iff for every n, the value $\pi(v_n)$ is defined. We say that function π with state v performs the process $\{v_n\}$ in set W.

Lemma 1

For any set $A \subset V$ and interpretation ? , there is a partial function $\pi_A : W \to W$, which with state $v \in W$ performs the same process in W, as the program $\langle A, v \rangle$ (for any $v \in W$).

Proof

Define function π_A:

$$\mathfrak{N}_A(u) = \begin{cases} T(u^2(q)_?\ (u), & \text{if } u(q) \in A \quad (u^2(q) \text{ means } u(u(q))) \\ \text{undefined}, & \text{if } u(q) \notin A \end{cases}$$

By easy induction it can be seen that function $\mathfrak{N}_A$ with state v performs the same process in W as the program $\langle A, v \rangle$.

Lemma 2

For any partial function $\pi : W \to W$ and valuation $v \in W$, there is such an interpretation $\mathfrak{I}$ and a set $A_\pi \subset V$, that π with state v performs the same process in W as the program $\langle A_\pi, v \rangle$ (in interpretation $\mathfrak{I}$) .

Proof

For given π and v the process $\{v_n\}$ in W performed by them is determined. Define set A_π :

$$A_\pi = \{x : x = v_n(q) \text{ and } \pi(v_n) \text{ is defined } (n = 0, 1, 2, \ldots)\}.$$

Define interpretation $\mathfrak{I}$:

$$\Phi_0 = \Phi_{00} = \{\varphi\}, \qquad T(a) = \varphi \qquad \text{for any } a \in J$$

(J is arbitrary, non-void set, T is an arbitrary coding function),

$$\varphi_{\mathfrak{I}}(u) = \begin{cases} \pi(u), & \text{if } \pi(u) \text{ is defined} \\ u & \text{in the opposite case} \end{cases}$$

It is seen (by induction), that $\{v_n\}$ is also the process performed by the program $\langle A_\pi, v \rangle$.

Lemma 3

If a partial function $\pi : W \to W$ with a state $v \in W$ performs infinite process, then the following conditions are equivalent:

(i) there exist such an interpretation $\mathfrak{I}$ and a finite set $A_\pi \subset V$, that π with state v performs the same process in W, as the program $< A_\pi, v >$,

(ii) the sequence $\{v_n(q)\}$ (where $\{v_n\}$ is the process in W performed by π and v) contains no sub-sequence composed of different elements.

Proof

(i) $\Rightarrow$ (ii). If (ii) does not hold, then there is an infinite sub-sequence $\{v_{k_n}(q)\}$ composed of different elements. Any base A_π of program $< A_\pi, v>$ must contain all elements of $\{v_n(q)\}$, thus, all elements of $\{v_{k_n}(q)\}$ too. Therefore, A_π cannot be finite.

(ii) $\Rightarrow$ (i). Since $\{v_n\}$ is infinite, then the set A_π from lemma 2 is identical with the set $\{x: x = v_n(q), \text{ for all } n\}$. The infinite sequence $\{v_n(q)\}$ is composed of finite number of different elements, thus, A_π is finite. Our statement follows from lemma 2.

In practice, of course, every program has a finite base. It follows from lemmas 1 and 3 that for every interpretation and program of a finite base there is a relevant process in the sense of [5], but not conversely. However, if we are not interested in traces in bases, then, in a sense, a converse property holds. An account of it gives

Lemma 4

Let U be the set of all restrictions of valuations $v \in W$ to the set $V' = V - \{q\}$. Let π be a partial function $U \to U$ and let $u_0 \in U$. There is such an interpretation $\mathfrak{I}$ and a finite set $A_\pi \subset V$, that $u_n = v_n \mid V'$, $n = 0, 1, 2, \ldots$ where $\{u_n\}$ is the process in U, performed by π with state u_0 and $\{v_n\}$ is the process in W performed by the program $< A, v_0 >$.

Proof

Let y, z be some different elements from V'. Define interpretation $\mathfrak{I}$:

$$\mathfrak{I}_0 = \mathfrak{I}_{00} = \{\varphi\}, \qquad T(a) = \varphi \qquad \text{(see proof of lemma 2)}$$

$$\varphi_{\mathfrak{I}}(v)(t) = \begin{cases} \pi(v|V')(t) & \text{if } t\neq q \text{ and } \pi(v|V') \text{ defined} \\ y & \text{if } t=q \text{ and } \pi(v|V') \text{ defined} \\ z & \text{if } t=q \text{ and } \pi(v|V') \text{ undefined} \\ v(t) & \text{if } t\neq q \text{ and } \pi(v|V') \text{ undefined} \end{cases}$$

By induction we make sure that the program $\langle A_\pi, v_0 \rangle$, where $A_\pi = \{y\}$,

$$v_0(t) = \begin{cases} u_0(t) & \text{if } t\neq q \text{ and } \pi(u_0) \text{ defined} \\ y & \text{if } t=q \text{ and } \pi(u_0) \text{ defined} \\ z & \text{if } t=q \text{ and } \pi(u_0) \text{ undefined} \\ u_0(t) & \text{if } t\neq q \text{ and } \pi(u_0) \text{ undefined} \end{cases}$$

performs such a process $v_0, v_1, v_2, \ldots$, that $u_n = v_n|V'$ for all n. This holds for n = 0 on account of definition of v_0. Suppose it holds for n = k, i.e. suppose that:

$$u_0 = v_0|V', \quad u_1 = v_1|V', \quad \ldots \quad u_k = v_k|V'.$$

Evaluating v_{k+1}:

$$v_{k+1}(t) = \varphi_{\mathfrak{I}}(v_k)(t) = \begin{cases} \pi(v_k|V')(t) & \text{if } t\neq q \text{ and } \pi(v_k|V') \text{ defined} \\ y & \text{if } t=q \text{ and } \pi(v_k|V') \text{ defined} \\ z & \text{if } t=q \text{ and } \pi(v_k|V') \text{ undefined} \\ v_k(t) & \text{if } t\neq q \text{ and } \pi(v_k|V') \text{ undefined} \end{cases}$$

we see that:

If u_k is the last element of process $u_0, u_1, \ldots$, then $\pi(u_k)$ is undefined, thus:

$$v_{k+1}(t) = \varphi_{\mathfrak{I}}(v_k)(t) = \begin{cases} z & \text{if } t=q \\ v_k(t) & \text{if } t\neq q \end{cases}$$

Therefore, v_k is the last element of process $v_0, v_1, \ldots$.

If u_k is not the last element of process $u_0, u_1, \ldots$, then $\pi(u_k)$ is defined, thus:

$$v_{k+1}(t) = \varphi_\eta(v_k)(t) = \begin{cases} \pi(v_k|V')(t) = \pi(u_k)(t) = u_{k+1}(t) & \text{if } t \neq q \\ y & \text{if } t = q \end{cases}$$

Therefore $v_{k+1}|V' = u_{k+1}$ what ends the proof.

To express a conclusion from lemmas 1–4, we introduce the following terminology.

Processes in set W, performed by functions: $\pi : W \to W$, we call π-processes in W. Processes in set W, performed by programs having infinite bases, we call L_0^{fin}-processes in W. Two sequences $\{v_n\}$, $\{w_n\}$ in set W, satisfying:

$v_n|V-\{q\} = w_n|V-\{q\}$ for all n, we call equal up to q.
From lemmas 1–4 we get:

Theorem

Every L_0^{fin}-process in W is equal to certain π-process in W, but not conversely.

Every L_0^{fin}-process in W is equal up to q to certain π-process in W and conversely.

Proof

The first proposition of the theorem, follows immediately from lemmas 1 and 3. The second we demonstrate.

Lemma 1 implies that every L_0^{fin}-process in W is equal up to q to certain π-process in W. Conversely, let $\{v_n\}$ be a π-process in W and let U be the set of all restrictions of valuations $v \in W$ to the set $V' = V = V-\{q\}$. Let π' be the function $\pi : U \to U$ defined as: $\pi'(v|V') = \pi(v)|V'$ for all $v \in W$.

It is evident that π'-process in U performed by π' and $u_0 = v_0|V'$ has the property $u_n = v_n|V'$ for all n. For π' there is such a program, performing L_0^{fin}-process $\{w_n\}$ in W, that $u_n = w_n|V'$ for all n (it follows from lemma 4). Thereby, $w_n|V' = v_n|V'$, that is sequences $\{w_n\}$, $\{v_n\}$ in W are equal up to q, what ends the proof.

References

[1] CZAJA L.: Niektóre własności semantyczne języków programowania. Zastosowanie do konstrukcji generatora symbolicznych języków maszynowych, Uniwersytet Warszawski, 1972 (doctors thesis).

[2] ENGELER E.: Algorithmic Properties of Structures, Math. Syst. Theory, 1, 1967.

[3] FLOYD R.W.: Assigning Meanings to Programs. Mathematical Aspects of Computer Science, Proc. of Symposia in Applied Math., vol. XIX, 1967.

[4] MANNA Z., PNUELI A.: Formalization of Properties of Functional Programs, Journal of the ACM, vol. 17 n. 3, July 1970.

[5] PAWLAK Z.: Maszyny Programowane, Algorytmy, V, No. 10, 1969.

[6] SALWICKI A.: Formalized Algorithmic Languages, Bull. de l'Académie Polonaise des Sciences, math. astr. et phys., vol. XVIII, No. 5, 1970.

[7] WEGNER P.: Programming Languages Information Structures and Machine Organization, McGraw-Hill Book Company, 1968.

MASZYNOWO UKIERUNKOWANE JEZYKI I PROCESY

Streszczenie

Skonstruowano semantyczny model dla języków programowania niższego poziomu. Klasę procesów opisywalnych za pomocą języka tego modelu porównano z klasą procesów rozpatrywanych w [5] .

МАШИННО НАПРАВЛЁННЫЕ ЯЗЫКИ И ПРОЦЕССЫ

Резюме

Построена семантическая модель для языков программирования низшего уровня. Класс процессов описываемых при помощи языка этой модели сравнен с классом роцессов рассматриваемых в [5].

ALGORYTMY, X, N° 18, 1973, 21-30

517.5:519.21

A RANDOMIZED FINITE-DIFFERENTIAL ESTIMATOR OF THE GRADIENT

Ryszard ZIELIŃSKI
Instytut Matematyczny
Polskiej Akademii Nauk
Warszawa, ul. Śniadeckich 8

Received 10.10.1972

Estimation of the derivative of a function f at the point x by $(f(x+h) - f(x))/h$ yields the error $O(h)$; the estimation by $(f(x+h) - f(x-h))/2h$ yields the error $O(h^2)$. The former method required computation of k+1 values of f, the latter 2k values, provided f is a function of k variables. In the note a random method is presented which requires k+1 points and yields the expected error $O(h^2)$.

1. INTRODUCTION

This paper deals with estimation of the gradient $g = (g_1, g_2, \ldots, g_k)$ of a given function F at the point $X = (x_1, x_2, \ldots, x_k)$ without calculation of the derivatives of the function. It is assumed that all partial derivatives up to third order exist and are finite. Let h be a given number. In the simplest case the gradient g is estimated by differences $\frac{1}{h}(F(x_1,\ldots, x_{i-1}, x_i + h, x_{i+1},\ldots, x_k) - F(x_1,\ldots, x_{i-1}, x_i, x_{i+1},\ldots, x_k))$ but it is well-known that the error of such estimation, i.e.

$$\frac{F(x_1,\ldots, x_i+h, \ldots, x_k) - F(x_1,\ldots, x_i,\ldots, x_k)}{h} - g_i$$

is $O(h)$. The error of the estimator

$$\frac{F(x_1,\ldots, x_i + \frac{h}{2}, \ldots, x_k) - F(x_1,\ldots, x_i - \frac{h}{2}, \ldots, x_k)}{h}$$

is $O(h^2)$ but for such estimation the values of function F at $2k$ points are needed whereas the first estimator required only $k+1$ points. In this note two randomized estimators of the gradient are proposed which need only $k+1$ points and yield the expected error $O(h^2)$.

2. RESULTS

Without loss of generality we may consider the problem of estimation of the gradient at the origin $X = (0, 0,\ldots, 0)$. Assume that the function F can be written in the form

$$F(x_1, x_2,\ldots, x_k) = F(0, 0,\ldots, 0) + \sum_{i=1}^{k} x_i g_i + \tag{1}$$

$$+ \frac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} x_i x_j a_{ij} + \frac{1}{6}\sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{k} x_i x_j x_l b_{ijl}$$

where

$$g_i = \frac{\partial F}{\partial x_i}\Big|_{X = (0, 0,\ldots, 0)}, \quad a_{ij} = \frac{\partial^2 F}{\partial x_i \partial x_j}\Big|_{X=(0,0,\ldots,0)}$$

$$b_{ijl} = \frac{\partial^3 F}{\partial x_i \partial x_j \partial x_l}\Big|_{X=(\delta_1 x_1, \delta_2 x_2,\ldots, \delta_k x_k)} \quad \text{for some } \delta_1$$

$$\delta_2,\ldots, \delta_k \quad (0 \leqslant \delta_i \leqslant 1,\ i = 1, 2,\ldots, k).$$

Theorem 1. Let $\varepsilon_1, \varepsilon_2,\ldots, \varepsilon_k$ be independent random variables with $\Pr\{\varepsilon_i = 1\} = \Pr\{\varepsilon_i = -1\} = \frac{1}{2}$ for

$i = 1, 2, \ldots, k$. Let $g^{(1)} = (g_1^{(1)}, g_2^{(1)}, \ldots, g_k^{(1)})$ be the estimator of the gradient $g = (g_1, g_2, \ldots, g_k)$ defined as follows

$$g_1^{(1)} = \frac{F(\varepsilon_1 h, 0, 0, \ldots, 0) - F(0, 0, 0, \ldots, 0)}{\varepsilon_1 h}$$

$$g_2^{(1)} = \frac{F(0, \varepsilon_2 h, 0, \ldots, 0) - F(0, 0, 0, \ldots, 0)}{\varepsilon_2 h} \qquad (2)$$

$$\ldots$$

$$g_k^{(1)} = \frac{F(0, 0, 0, \ldots, \varepsilon_k h) - F(0, 0, 0, \ldots, 0)}{\varepsilon_k h}$$

Then

$$E(g_i^{(1)} - g_i) = \frac{h^2}{6} b_{iii} \qquad (3)$$

and

$$E \| g^{(1)} - g \|^2 = \frac{h^2}{4} \sum_{i=1}^{k} a_{ii} + O(h^4) \qquad (4)$$

where, as usual, $\| g^{(1)} - g \|^2 = \sum_{i=1}^{k} (g_i^{(1)} - g_i)^2$.

An appropriate algorithm may be formulated in the following way:

1. Sample the random variables $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_k$;

2. Calculate the values of the function F at the points $(0, 0, 0, \ldots, 0)$, $(\varepsilon_1 h, 0, 0, \ldots, 0)$, $(0, \varepsilon_2 h, 0, \ldots, 0)$, ..., $(0, 0, 0, \ldots, \varepsilon_k h)$;

3. Estimate the gradient g by the formulas (2).

It is obvious that the bias (3) of the estimator $g^{(1)}$ as well as the "variance" (4) of this estimator depend on the

orientation of the coordinate system and change with its rotation. We shall construct an estimator whose bias and variance do not depend on the rotation of the coordinate system.

Let $X_0, X_1, X_2, \ldots, X_k$, where $X_i = (x_{1i}, x_{2i}, \ldots, x_{ki})$, $\|X_i\| = h$, be vertices of a regular simplex centered at the origin. Let Ω be the rotation group. Denote by $X_0(\omega)$, $X_1(\omega)$, $\ldots$, $X_k(\omega)$ the vertices of the simplex obtained from the simplex $X_0, X_1, \ldots, X_k$ by the rotation $\omega \in \Omega$. Put $F_i = F(X_i)$ and $F_i(\omega) = F(X_i(\omega))$. Define a new estimator $g^{(2)} = (g_1^{(2)}(\omega), g_2^{(2)}(\omega), \ldots, g_k^{(2)}(\omega))$ as a direction of the hyperplane containing points $(F_i(\omega), x_{1i}(\omega), x_{2i}(\omega), \ldots, x_{ki}(\omega))$, $i = 0,1,2,\ldots, k$:

$$g_i^{(2)} = \frac{k}{k+1} h^{-2} \sum_{j=0}^{k} x_{ij}(\omega) F_j(\omega) \tag{5}$$

The last formula will be justified in section 3.

Theorem 2. Let P be the uniform distribution on Ω. Let $g^{(2)}$ be the estimator of the gradient defined by (5). Then for every $i = 1, 2, \ldots, k$

$$E_P (g_i^{(2)} - g_i) = \frac{h^2}{6(k+2)} \sum_{j=1}^{k} (b_{ijj} + b_{jij} + b_{jji}) \tag{6}$$

and

$$E_P \| g^{(2)} - g \|^2 =$$

$$= \frac{kh^2}{4(k+1)(k+2)} \sum_{i=1}^{k} \sum_{j=1}^{k} (a_{ii}a_{jj} + 2a_{ij}^2) + O(h^4) \tag{7}$$

where E_P is the expectation with respect to distribution P.

An appropriate algorithm may be formulated in the following way:

1. Sample $\omega \in \Omega$ according to the uniform distribution on Ω;

2. Construct the simplex $X_0(\omega)$, $X_1(\omega)$, ..., $X_k(\omega)$;

3. Calculate the values of the function F at the points $X_j(\omega)$, $j = 0,1,\ldots,k$;

4. Estimate the gradient g by the formula (5).

The problem of numerical sampling of random oriented simplex (points 1 and 2 of the above algorithm) is considered by the author in a separate note.

3. JUSTIFICATION OF THE FORMULA (5)

If $X_0, X_1, X_2, \ldots, X_k$ are vertices of a regular simplex centered at the origin and $\|X_j\| = h$, then every inner product $(X_i, X_j) = -h^2/k$ for $i \neq j$. Consider the matrix

$$X = \begin{bmatrix} \frac{h}{\sqrt{k}} & \frac{h}{\sqrt{k}} & \frac{h}{\sqrt{k}} & \cdots & \frac{h}{\sqrt{k}} \\ x_{10} & x_{11} & x_{12} & \cdots & x_{1k} \\ x_{20} & x_{21} & x_{22} & \cdots & x_{2k} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{k0} & x_{k1} & x_{k2} & \cdots & x_{kk} \end{bmatrix} \tag{8}$$

Notice that

$$XX^T = X^T X = \frac{k+1}{k} h^2 I_{k+1} \tag{9}$$

where I_{k+1} is the unit matrix of rank k+1. The hyperplane

$$y = \frac{\sqrt{k}}{h} a_0 + a_1 x_1 + \ldots + a_k x_k \tag{10}$$

will contain the points $(F_i, x_{1i}, x_{2i}, \ldots, x_{ki})$, $i = 0,1,2,\ldots,k$ provided $a = (a_0, a_1, a_2, \ldots, a_k)$ is a solution of the equation

$$X^{\top} a = F \tag{11}$$

where $F = (F_0, F_1, F_2, \ldots, F_k)$.

Taking into account (9), we obtain

$$a = \frac{k}{k+1}\, h^{-2}\, XF \tag{12}$$

This gives the formula (5) for $g^{(1)} = (a_1, a_2, \ldots, a_k)$.

4. PROOFS OF THE THEOREMS 1 AND 2

Theorem 1 is obvious; to prove it consider the expressions like

$$F(\varepsilon_1 h, 0, \ldots, 0) = F(0, 0, \ldots, 0) + \varepsilon_1 h g_1 + \frac{(\varepsilon_1 h)^2}{2} a_{11} + \frac{(\varepsilon_1 h)^3}{6} b_{111}$$

and calculate $E(g_i^{(1)} - g_i)$ and $E\|g^{(1)} - g\|^2$ with respect to the distribution $\Pr\{\varepsilon_1 = 1\} = \Pr\{\varepsilon_1 = -1\} = \frac{1}{2}$.

To prove theorem 2 the following fact will be used. If $\xi_1(\omega), \xi_2(\omega), \ldots, \xi_k(\omega)$ are coordinates of a vector $\xi(\omega)$, $\|\xi(\omega)\| = 1$, P is the uniform distribution on Ω and $\alpha_1, \alpha_2, \ldots, \alpha_s$ are positive integers, then

$$E_P\, \xi_{i_1}^{\alpha_1}(\omega)\, \xi_{i_2}^{\alpha_2}(\omega) \ldots \xi_{i_s}^{\alpha_s}(\omega) = 0 \tag{13}$$

if at least one of $\alpha_1, \alpha_2, \ldots, \alpha_s$ is odd and numbers $i_1, i_2, \ldots, i_s$ are different. If numbers j, p, q are different, then

$$E_P \xi_j^2(\omega)\, \xi_p^2(\omega) = \frac{1}{k(k+2)}$$

$$E_P \xi_j^4(\omega) = \frac{3}{k(k+2)}$$

$$E_P \xi_j^2(\omega)\, \xi_p^2(\omega)\, \xi_q^2(\omega) = \frac{1}{k(k+2)(k+4)} \qquad (14)$$

$$E_P \xi_j^2(\omega)\, \xi_p^4(\omega) = \frac{3}{k(k+2)(k+4)}$$

$$E_P \xi_j^6(\omega) = \frac{15}{k(k+2)(k+4)}$$

The above formulas may be obtained by integration; it is convenient to perform the integration in new variables $\varphi_1, \varphi_2, \ldots, \varphi_{k-1}$ constructed by diffeomorphism

$$\xi_1 = \cos\varphi_1 \cos\varphi_2 \ldots \cos\varphi_{k-1}$$
$$\xi_2 = \sin\varphi_1 \cos\varphi_2 \ldots \cos\varphi_{k-1}$$
$$\xi_3 = \sin\varphi_2 \ldots \cos\varphi_{k-1}$$
$$\ldots$$
$$\xi_k = \sin\varphi_{k-1}$$

Write formula (5) using the expansion (1):

$$g_s^{(2)} = \frac{k}{k+1} h^{-2} \sum_{j=0}^{k} x_{sj}(\omega) \left[\sum_{i=0}^{k} x_{ij}(\omega)\, g_i + \right.$$
$$+ \frac{1}{2} \sum_{i=1}^{k} \sum_{p=1}^{k} x_{ij}(\omega)\, x_{pj}(\omega)\, a_{ip} +$$
$$\left. + \frac{1}{6} \sum_{i=1}^{k} \sum_{p=1}^{k} \sum_{q=1}^{k} x_{ij}(\omega)\, x_{pj}(\omega)\, x_{qj}(\omega)\, b_{ipq} \right]$$

where $x_{0j} = \frac{h}{\sqrt{k}}$ and $g_0 = F(0, 0, \ldots, 0)$. This may be written in the form

$$g_s^{(2)} = \frac{k}{k+1} h^{-2} \sum_{i=0}^{k} (XX^T)_{is} g_i +$$

$$+ \frac{kh}{2(k+1)} \sum_{j=0}^{k} \sum_{i=1}^{k} \sum_{p=1}^{k} \xi_{sj}(\omega) \xi_{ij}(\omega) \xi_{pj}(\omega) a_{ip} +$$

$$+ \frac{kh^2}{6(k+1)} \sum_{j=0}^{k} \sum_{i=1}^{k} \sum_{p=1}^{k} \sum_{q=1}^{k} \xi_{sj}(\omega) \xi_{ij}(\omega) \xi_{pj}(\omega) \xi_{qj}(\omega) b_{ipq}$$

where $(XX^T)_{is}$ is the (i,s)-th element of the matrix XX^T and $\xi_{ij}(\omega) = x_{ij}(\omega)/h$; thus $|\xi_j| = 1$. Using (9) we have

$$\frac{k}{k+1} h^{-2} \sum_{i=0}^{k} (XX^T)_{is} g_i = g_s$$

Because of (13)

$$E_p \xi_{sj}(\omega) \xi_{ij}(\omega) \xi_{pj}(\omega) = 0$$

for all triplets (s,i,p). Using (14) and (13) we obtain

$$E_p \sum_{i=1}^{k} \sum_{p=1}^{k} \sum_{q=1}^{k} \xi_{sj}(\omega) \xi_{ij}(\omega) \xi_{pj}(\omega) \xi_{qj}(\omega) b_{ipq} =$$

$$= \frac{1}{k(k+2)} \sum_{p=1}^{k} (b_{spp} + b_{psp} + b_{pps})$$

Consequently,

$$E_p g_s^{(2)} = g_s + \frac{h^2}{6(k+2)} \sum_{p=1}^{k} (b_{spp} + b_{psp} + b_{pps})$$

which is the first part of theorem 2. The second part of this theorem is proved by squaring the difference

$$g_s^{(2)} - g_s = \frac{kh}{2(k+1)} \sum_{j=0}^{k} \sum_{i=1}^{k} \sum_{p=1}^{k} \xi_{sj}^{(\omega)} \xi_{ij}^{(\omega)} \xi_{pj}^{(\omega)} a_{ip} +$$

$$+ \frac{kh^2}{6(k+1)} \sum_{j=0}^{k} \sum_{i=1}^{k} \sum_{p=1}^{k} \sum_{q=1}^{k} \xi_{sj}^{(\omega)} \xi_{ij}^{(\omega)} \xi_{pj}^{(\omega)} \xi_{qj}^{(\omega)} b_{ipq}$$

and calculating the expectation using formulas (13) and (14).

O PEWNEJ STOCHASTYCZNEJ SKOŃCZENIE-RÓŻNICOWEJ METODZIE SZACOWANIA GRADIENTU FUNKCJI

Streszczenie

Oszacowanie pochodnej funkcji f w punkcie x za pomocą ilorazu $(f(x+h) - f(x))/h$ daje błąd $O(h)$; oszacowanie $(f(x+h) - f(x-h))/2h$ daje błąd $O(h^2)$. W przypadku szacowania gradientu funkcji k zmiennych pierwsza metoda wymaga obliczenia wartości funkcji f w $k+1$ punktach, druga w $2k$ punktach. W pracy przedstawiono pewną metodę losowania $k+1$ punktów tak, żeby oczekiwany błąd był $O(h^2)$.

ОБ ОДНОМ РАНДОМИЗИРОВАННОМ МЕТОДЕ ОЦЕНКИ ГРАДИЕНТА ФУНКЦИИ ПРИ ПОМОЩИ КОНЕЧНЫХ РАЗНОСТЕЙ

Резюме

Оценка производной функции f в точке x при помощи $(f(x+h) - f(x))/h$ имеет ошибку $O(h)$; оценка $(f(x+h) - f(x-h))/2h$ имеет ошибку $O(h^2)$. В случае функции k переменных первая оценка требует вычисления значении функции f в $k+1$ точках, вторая - в $2k$ точках. В статье предлагается рандомизированная оценка которая требует вычисления значении функции в $k+1$ точках и имеет ожидаемую ошибку порядка $O(h^2)$.

Warszawa, ul. Krzywickiego 34
A.M.

ALGORYTMY, X, No 18, 1973, 31-61 518.6:517.512.2

DFT

DYSKRETNA TRANSFORMACJA FOURIERA I SZYBKIE ALGORYTMY OBLICZENIOWE ANALIZY WIDMOWEJ PRZY UŻYCIU MASZYNY CYFROWEJ

Andrzej JABŁOŃSKI

Instytut Cybernetyki Technicznej
Politechniki Gdańskiej

Pracę złożono 20.08.1971

W artykule przedstawiono ideę działania szybkich algorytmów dokonywania dyskretnej transformacji Fouriera skończonych ciągów liczbowych. Podano związek DFT z przekształceniem całkowym Fouriera oraz wyszczególniono niektóre z jej własności. Opisane zostały dwie tzw. kanoniczne formy algorytmu szybkiej transformacji Fouriera używane w stosunku do ciągów, których liczba elementów jest naturalną potęgą liczby dwa. Zwrócono uwagę na błędy pojawiające się przy stosowaniu dyskretnej transformacji Fouriera w zastępstwie przekształcenia ciągłego i podano metody ich redukcji. Na przykładzie splotu i funkcji autokorelacji pokazano, jak szybka transformacja Fouriera wykorzystywana może być do przyspieszenia obliczeń innych przekształceń funkcji.

Ważniejsze oznaczenia

$x(t)$ - ciągła funkcja czasu

$a(f)$ - " " częstotliwości

$x_p(t) = \sum_{s=-\infty}^{+\infty} x(t+s\cdot T)$ - OSFN (okresowa suma funkcji nieokresowych) o okresie T dla funkcji $x(t)$

$a_p(f) = \sum_{l=-\infty}^{+\infty} a(f+l\cdot F)$ - OSFN dla funkcji $a(f)$ o okresie F

$X(k \cdot \Delta t)$ - ciąg wartości funkcji $x(t)$ w punktach $t = k\cdot\Delta t$ $k = 0, \pm 1, \pm 2, \ldots$

$A\ (n \cdot \Delta f)$ - ciąg wartości funkcji a (f) w punktach $f = n \cdot \Delta t$ $n = 0, \pm 1, \pm 2, \ldots$

$$X_p(k \cdot \Delta t) = X_k = \sum_{s=-\infty}^{+\infty} X\left[(k+s \cdot N)\ \Delta t\right]$$ - OSFN dla ciągu $X(k \cdot \Delta t)$ o okresie N

$$A_p(n \cdot \Delta f) = A_n = \sum_{l=-\infty}^{+\infty} A\left[(n+l \cdot N)\ \Delta f\right]$$ - OSFN dla ciągu $A\ (n \cdot \Delta f)$ o okresie N

1. WPROWADZENIE

W wielu dziedzinach działalności ludzkiej spotykamy się z problemem analizy wielkości zmiennych w czasie. Wielkości te mają najczęściej postać funkcji czasu, przebiegu elektrycznego, bądź też sekwencji liczb reprezentujących pomiary zmiennej.

Analiza ta polega najczęściej na wyłowieniu jakościowych i ilościowych cech zmian wielkości badanej. Jedną z bardzo rozpowszechnionych, a do niedawna jedyną z praktycznie stosowanych metod, jest analiza korelacyjna. Polega ona na określeniu cech dynamicznych źródła wielkości zmiennej, na podstawie kształtu funkcji autokorelacji przebiegu tej zmiennej.

Inną, z wielu względów wygodniejszą, metodą jest analiza spektralna [1], [2].

Analiza widmowa sygnałów elektrycznych dokonywana była początkowo wyłącznie metodą analogową, przy użyciu zespołu odpowiednio dostrojonych filtrów. Jest to metoda szybka, ale mało dokładna, pozwalająca na analizowanie jedynie sygnałów o wąskich pasmach i to w stosunkowo niewielu punktach osi częstotliwości. Mankamentów tych nie posiada metoda cyfrowa, bazująca na dyskretnej transformacji Fouriera, w skrócie zwanej DFT (Discrete Fourier Transform) i wykorzystująca maszynę cyfrową. Dokładność może być w tym przypadku dowolnie zwiększana, wymagane jest jednak wykonanie wielu czasochłonnych obliczeń.

Przez długi czas, z winy niewielkiej szybkości działania istniejących maszyn cyfrowych, jak i z braku efektywnych algorytmów obliczeniowych, analiza widmowa z zastosowaniem maszyny cyfrowej zajmowała tyle czasu, że nie można było mówić o zastosowaniu jej do celów praktycznych.

Jak wiadomo, dla celów analizy widmowej metodą cyfrową sygnał poddany musi być uprzednio konwersji analogowo-cyfrowej. Nie występuje przy tym strata informacji, jeśli próbkowanie jest dostatecznie gęste, co z grubsza gwarantuje spełnienie twierdzenia o próbkowaniu Kotielnikowa-Shannona.

Dla spotykanych w praktyce sygnałów, ilość próbek sekwencji podlegającej analizie była tak duża, że obliczenie wszystkich współczynników rozkładu Fouriera metodą bezpośrednią okazywało się niemożliwe.

Punktem zwrotnym w rozwoju cyfrowej metody analizy widmowej, stało się opracowanie w 1965 r. algorytmów tzw. szybkiej transformacji Fouriera popularnie zwanej FFT (Fast Fourier Transform). Dzięki nim czas potrzebny na cyfrową analizę widmową sekwencji próbek, przy użyciu dyskretnej transformacji Fouriera (dokładnie omówionej w dalszej części), został dla sekwencji o dużych długościach skrócony o kilka rzędów wielkości.

Spowodowało to istny przewrót w analizie przebiegów czasowych. Możliwe stało się nie tylko obliczenie wszystkich współczynników Fouriera, długiej sekwencji próbek, ale analiza widmowa stała się najkrótszą drogą wykonywania wszelkich innych rodzajów obróbki sygnałów, takich jak obliczanie funkcji autokorelacji lub korelacji skrośnych, splotów, dokonywania filtracji a nawet transformacji Z. Analizę widm wielowymiarowych zastosowano m.in. do rozpoznawania obrazów.

Ze względu na uniwersalną postać danych wejściowych FFT może być stosowana z równym powodzeniem dla analizy zmienności wskaźników ekonomicznych, rozpoznawania dźwięków mowy, analizy ech sejsmicznych jak i systemów optycznych.

Nastąpił szybki rozwój prac w trzech kierunkach: badań teoretycznych, metod wykorzystania analizy widmowej w dziedzinach dla niej nowych, jak i w dziedzinie konstrukcji maszyn cyfrowych specjalistycznych, które w chwili obecnej pozwalają na analizę np. sygnałów radiowych z niewiarygodną szybkością 2 000 000 próbek na sekundę, dając przy tym widma opisane przy pomocy N = 1024 próbek [10].

Podobnymi osiągnięciami nauka może poszczycić się i w pozostałych dwóch dziedzinach.

W Polsce szybkie przekształcenie Fouriera zdobywa dopiero popularność. W Ośrodku Gdańskim badania nad FFT i możliwościami jej stosowania prowadzone są od 1969 r. Prace prowadzone w Instytucie Cybernetyki Technicznej Politechniki Gdańskiej dały wiele cennych rezultatów. Zastosowanie FFT do celów analizy sygnałów radiowych, biomedycznych, radiolokacyjnych, analizy obwodów elektrycznych czy też szybkiego obliczania transformacji Z, dyskretnej transformacji Laplace'a czy transformacji Hilberta potwierdziło jej olbrzymią przydatność.

Chciałbym też przy tej okazji gorąco podziękować dr A. Saynagi i dr H. Takashima z Canon Research and Development Laboratory jak również doc. dr Z. Bogusiowi z Instytutu Cybernetyki Technicznej Politechniki Gdańskiej, z których wydatnej pomocy korzystałem.

2. DYSKRETNA TRANSFORMACJA FOURIERA

Pod pojęciem analizy widmowej sygnału rozumiemy zwykle poddanie go całkowemu przekształceniu Fouriera, lub, jeżeli mamy do czynienia z sygnałem okresowym, rozłożenie go na szereg składowych harmonicznych Fouriera. Oba te rodzaje przekształceń są szeroko znane i stosowane, nie dają się jednak wykonać przy użyciu maszyny cyfrowej, która może wykonywać obliczenia jedynie na ciągach liczb.

Do tego celu doskonale z kolei nadaje się, znane od stosunkowo dawna, tzw. dyskretne przekształcenie Fouriera, w skrócie zwane DFT (Discrete Fourier Transform). Przy spełnieniu całego szeregu założeń, DFT może służyć jako przybliżona metoda obliczania wartości transformat ciągłych w dyskretnych punktach.

Aby więc móc poprawnie stosować DFT w zastępstwie przekształceń ciągłych, trzeba znać związki istniejące między przekształceniem dyskretnym a całką i szeregiem Fouriera.

Para związków całkowego przekształcenia Fouriera ma dla sygnału ciągłego $x(t)$ postać

$$x(t) = \int_{-\infty}^{+\infty} a(f)\, e^{i2\pi ft}\, df$$
$$a(f) = \int_{-\infty}^{+\infty} x(t)\, e^{-i2\pi ft}\, dt \qquad (2.1)$$

gdzie $a(f)$ ma sens widma sygnału ciągłego $x(t)$. Jeśli sygnał $x(t)$ poddać dyskretyzacji w momentach $t = k \cdot \Delta t$, $k = 0, \pm 1, \pm 2\ldots$, to dla otrzymanego ciągu dyskretnych wartości słuszne jest, że

$$X(k \cdot \Delta t) = \int_{-\infty}^{+\infty} a(f) e^{i2\pi fk \cdot \Delta t}\, df = \sum_{l=-\infty}^{+\infty} \int_{l \cdot F}^{(l+1)F} a(f) e^{i \frac{2\pi fk}{F}}\, df$$

gdzie $F = \frac{1}{\Delta t}$ jest częstotliwością próbkowania. Ze względu na periodyczność funkcji e^{ix} można napisać

$$X(k \cdot \Delta t) = \int_{0}^{F} \sum_{l=-\infty}^{+\infty} a(f + l \cdot F)\, e^{i \frac{2\pi fk}{F}}\, df = \int_{0}^{F} a_p(f) e^{i \frac{2\pi fk}{F}}\, df \qquad (2.2)$$

gdzie dla $a_p(f) = \sum_{l=-\infty}^{+\infty} a(f + l \cdot F)$ można przyjąć nazwę: okresowa suma funkcji nieokresowych (w skrócie OSFN).

Jeśli dla obliczenia całki

$$\int_0^F a_p(f)\, e^{i\frac{2\pi fk}{F}}\, df$$

zastosować metodę trapezów, to otrzymamy przekształcenie

$$X(k.\Delta t) \cong \Delta f \sum_{n=0}^{N-1} A_p(n.\Delta f)\, e^{i\frac{2\pi nk}{N}} = X_p\,(k.\Delta t) \qquad (2.3)$$

gdzie $$A_p\,(n.\Delta f) = \sum_{l=-\infty}^{+\infty} A\,[(n + l\,.\,N)\Delta f] \qquad (2.4)$$

$$X_p\,(k\,.\Delta t) = \sum_{s=-\infty}^{+\infty} X\,[(n + s\,.\,N)\;\Delta t)] \qquad (2.5)$$

(patrz wykaz ważniejszych oznaczeń).

Jako funkcja okresowa o okresie F, $a_p(f)$ może być z kolei rozłożona na szereg Fouriera w przedziale $< 0, F)$. Jeśli dodatkowo poddać ją dyskretyzacji, to otrzymamy w sumie parę zależności

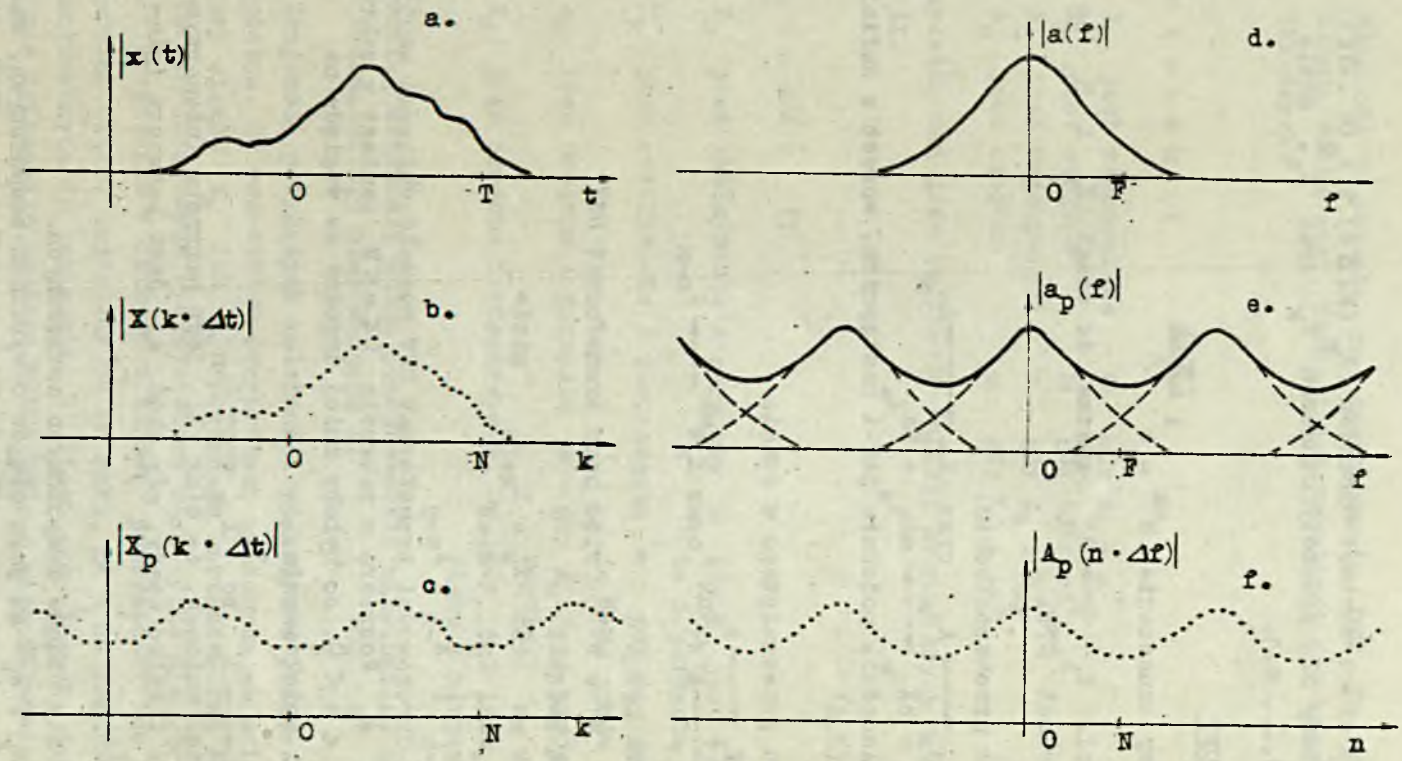
$$X_p\,(k\,.\Delta t) = \sum_{n=0}^{N-1} A_p\,(n\,.\Delta f)\, e^{i\frac{2\pi nk}{N}}$$

$$A_p\,(n\,.\Delta f) = \frac{1}{N} \sum_{k=0}^{N-1} X_p\,(k\,.\Delta t)\, e^{-i\frac{2\pi nk}{N}} \qquad (2.6)$$

$$n, k = 0, 1, 2, \ldots N-1$$

gdzie $X_p(k.\Delta t)$ i $A_p\,(n\,.\Delta f)$ mają sens określony przez (2.4) i (2.5).

Ta sama para przekształceń może być również łatwo uzyskana z rozkładu ciągu $X_p\,(k.\Delta t)$ na szereg Fouriera.



Rys. 2.1. Widma sygnałów

a - sygnał ciągły, b - sygnał spróbkowany, c - sygnał o widmie periodycznym dyskretnym, d - widmo sygnału ciągłego, e - widmo sygnału dyskretnego, f - widmo sygnału dyskretnego po spróbkowaniu

W dalszym ciągu zamiast oznaczeń X_p $(k.\Delta t)$ i $A_p(n.\Delta f)$ używać będziemy dla prostoty oznaczeń X_k oraz A_n, gdzie $k, n = 0, 1, \ldots N-1$.

Własności DFT

Wprowadźmy oznaczenie $W^{nk} = e^{i\frac{2\pi nk}{N}}$

Niech zapis $Z_k \longleftrightarrow B_n$ oznacza, że ciągi Z_k i B_n są parą transformat DFT.

1. Liniowość przekształcenia.
   Jeśli $X_{1_k} \longleftrightarrow A_{1_n}$ oraz $X_{2_k} \longleftrightarrow A_{2_n}$
   to $aX_{1_k} + bX_{2_k} \longleftrightarrow aA_{1_n} + bA_{2_n}$
   Dowód własności, podobnie jak i następnych, wprost z definicji DFT (2.6)

2. DFT ciągu przesuniętego w czasie
   Jeśli $X_k \longleftrightarrow A_n$
   to $X_{k-l} \longleftrightarrow A_n W^{\mp nl}$ oraz $X_k W^{mk} \longleftrightarrow A_{n-m}$

3. Okresowość DFT
   Ponieważ $W^n = W^{n+N}$ więc pary transformat DFT spełniają związki
   $A_n = A_n + mN$ oraz $X_k = X_{k+lN}$ gdzie
   $l, m = 0, \pm 1, \pm 2 \ldots$ oraz
   $X_{-k} = X_{N-k}$ i $A_{-n} = A_{N-n}$
   Własność okresowości transformat DFT pozwala obliczać współczynniki A_n Fouriera w zakresie $0 \leqslant n < N$ zamiast w zakresie $-\frac{N}{2} \leqslant n < \frac{N}{2}$, co byłoby mniej dogodne ze względu na trudności indeksowania.

4. Jeśli $X_k \longleftrightarrow A_n$ to $X_{-k} \longleftrightarrow A_{-n}$
   Wynika stąd wniosek, że ciąg X_k jest parzysty (nieparzysty) wtedy i tylko wtedy, gdy ciąg A_n jest parzysty (nieparzysty).

5. Transformaty ciągów zespolonych sprzężonych.
   Jeśli X_k i X_k^* są parą ciągów o wyrazach zespolonych, wzajemnie sprzężonych, to $X_k^* \longleftrightarrow A_{-n}^*$ oraz $X_{-k}^* \longleftrightarrow A_n^*$.

Własność ta pozwala na zastosowanie DFT do obliczenia widma mocy, funkcji autokorelacji czy korelacji skrośnych.

W n i o s e k I

1. X_k jest rzeczywiste ≡ gdy $A_n = A^*_{-n}$
2. X_k jest urojone ≡ gdy $A_n = -A^*_{-n}$
3. A_n jest rzeczywiste ≡ gdy $X_k = X^*_{-k}$
4. A_n jest urojone ≡ gdy $X_k = -X^*_{-k}$

Przydatna też jest znajomość relacji stanowiących treść wniosku II.

W n i o s e k II

1. X_k jest rzeczywiste i parzyste ≡ gdy A_n jest rzeczywiste i parzyste
2. X_k jest rzeczywiste i nieparzyste ≡ gdy A_n jest urojone i parzyste
3. X_k jest urojone i parzyste ≡ gdy A_n jest urojone i parzyste
4. X_k jest urojone i nieparzyste ≡ gdy A_n jest rzeczywiste i nieparzyste.

Dowód tej własności można przeprowadzić natychmiast na podstawie własności 3,5 oraz wniosku I.

Znajomość powyższych zależności może być w praktyce bardzo przydatna. We wszystkich przypadkach opisanych we wniosku II wyrazy ciągów X_k lub A_n posiadają wartość różną od zera tylko dla składowych rzeczywistych bądź tylko dla urojonych. Jeśli uwzględnić dodatkowo własność parzystości lub nieparzystości ciągu, dojdziemy do wniosku, że z 2N komórek pamięci potrzebnych do zapisania N liczb zespolonych, istotną dla nas informację zawiera tylko N/2, a więc zaledwie czwarta część. Nasuwa się więc możliwość wykorzystania tego nadmiaru

poprzez zastosowanie odpowiednio zbudowanego algorytmu, zwanego algorytmem "podwójnego upakowania", pozwalającego na analizowanie przy tej samej zajętości pamięci ciągów dwukrotnie dłuższych, dzięki czemu zwiększyć można efektywność obliczeń FFT.

3. SZYBKA TRANSFORMACJA FOURIERA (FFT)

Jak już wspomniano, DFT znane było stosunkowo dawno. Ponieważ, jak wynika ze związku (2.6) dokonanie transformacji ciągu o N elementach wymaga wykonania N^2 zespolonych dodawań i mnożeń, o praktycznym stosowaniu DFT nie mogło być mowy.

Pierwszy algorytm FFT (Fast Fourier Transform) opracowany został przez Jamesa Cooley'a z IBM Watson Research Center, na podstawie sugestii J.W. Tukey'a i przy współpracy Richarda L. Garwina w 1965 r. Bliższych informacji na temat okoliczności w jakich doszło do odkrycia FFT szukać można w [3], [5].

Nie wgłębiając się jednak w historię rozwoju badań nad FFT przejdę do omówienia jej istoty.

Szybka transformacja Fouriera jest algorytmem iteracyjnym, pozwalającym na obliczenie transformat DFT określonych związkiem (2.6) za pomocą mniejszej niż N^2 liczby mnożeń zespolonych. Redukcja liczby obliczeń jest możliwa, jeżeli N nie jest liczbą pierwszą.

Algorytm zmniejsza liczbę obliczeń, gdy

$$N = r_1, r_2 \cdots r_t \tag{3.1}$$

przy czym korzystne jest aby N było liczbą maksymalnie złożoną, a więc aby t było możliwie duże.

Jeśli $r_1 = r_2 = \ldots = r_t = 2$, czyli $N = 2^n$, to (3.2)
liczba działań zespolonych zostaje zredukowana do $2N \log_2 N$.

Stosunek liczby działań wymaganych przez FFT do liczby działań metody bezpośredniej

$$\frac{2N \log_2 N}{N^2} = \frac{2 \log_2 N}{N} \quad \underset{N \to \infty}{\longrightarrow} 0 \qquad (3.3)$$

a więc stosowanie FFT jest szczególnie korzystne dla ciągów o dużej liczbie próbek.

Tabela 3.1

Porównanie czasów realizacji analizy widmowej w EMC przy użyciu FFT oraz metodą bezpośrednią dla ciągów o różnej długości

Liczba próbek	Wymiary macierzy	Metoda bezpośrednia	FFT
4 096	64 . 64	8 min	3 s
8 192	256 . 64	30 min	5 s
65 536	256 . 256	30 godz	1 min
262 144	512 . 512	20 dni	5 min
1 048 576	1024 . 1024	1 rok	20 min

N = 1024 B = 1 μs (B - cykl pamięci EMC)

Nie bez znaczenia jest również fakt, że podobnej redukcji ulega błąd zaokrągleń iloczynów, ze względu na mniejszą liczbę mnożeń zespolonych występujących przy obliczaniu FFT.

Redukcję czasów obliczeń o dalszych kilka rzędów wielkości dało w ostatnich latach stosowanie specjalistycznych maszyn cyfrowych, tzw. analizatorów cyfrowych, przeznaczonych wyłącznie do analizy widmowej z użyciem algorytmów FFT.

Tabela 3.2

Porównanie czasów analizy widmowej przy użyciu UMC oraz przy użyciu różnych typów analizatorów cyfrowych

Rodzaj organizacji	Liczba arytmometrów	Czas trwania obliczeń	Przepustowość w próbkach/s
Uniwersalna maszyna cyfrowa	1	500 000 "	2 000
Analizator sekwencyjny	1	5 000	200 000
Analizator kaskadowy	10	1 000	1 000 000
Analizator iteracyjno-równoległy	512	10	100 000 000
Analizator równoległo-równoległy	5120	1	1 000 000 000

B = 50 μs (IBM 7094)

Istnieje wiele algorytmów FFT. Najprostsze z nich odnoszą się do przypadku (3.2), tzn. gdy liczba próbek jest potęgą liczby 2. Spośród tych z kolei można wyróżnić trzy zasadnicze formy kanoniczne FFT. Z wielu spotykanych opisów idei FFT wybrałem najbardziej poglądowy [4], [6], [7], [8], [9].

Grupowanie w dziedzinie czasu

Załóżmy, że dany jest ciąg dyskretnych wartości X_k, $k = 0, 1, 2 \ldots N-1$, gdzie N - liczba parzysta. Grupując osobno wyrazy o indeksach parzystych i osobno o nieparzystych, uzyskamy dwa ciągi o połowę krótsze, mianowicie

$$Y_k = X_{2k} \quad \text{oraz} \quad Z_k = X_{2k+1}$$
$$k = 0, 1, 2, \ldots \frac{N}{2} - 1 \tag{3.4}$$

Każdy z ciągów poddać można transformacji DFT otrzymując odpowiednio

$$B_n = \sum_{k=0}^{\frac{N}{2}-1} Y_k e^{-i\frac{4\pi nk}{N}}$$
$$C_n = \sum_{k=0}^{\frac{N}{2}-1} Z_k e^{-i\frac{4\pi nk}{N}} \tag{3.5}$$

gdzie $n = 0, 1, 2 \ldots \frac{N}{2} - 1$.

Transformata ciągu X_k wyrażona może być poprzez transformaty ciągów Y_k i Z_k, a mianowicie

$$A_n = \sum_{k=0}^{\frac{N}{2}-1} \left\{ Y_k W^{-2nk} + Z_k W^{-n(2k+1)} \right\} =$$
$$= \sum_{k=0}^{\frac{N}{2}-1} Y_k W^{-2nk} + W^{-n} \sum_{k=0}^{\frac{N}{2}-1} Z_k W^{-2nk} \tag{3.6}$$

Korzystając z (3.5) możemy powyższe zapisać w postaci

$$A_n = B_n + W^{-n} C_n$$
$$n = 0, 1, 2, \ldots \frac{N}{2} - 1 \tag{3.7}$$

Pozostałe $\frac{N}{2}$ wyrazów ciągu A_n obliczyć można korzystając z periodyczności zarówno ciągów B_n i C_n z okresem $\frac{N}{2}$, jak i z periodyczności funkcji W^{-n} z okresem dwa razy większym.

W sumie więc

$$A_{n+\frac{N}{2}} = B_n + W^{-(n+\frac{N}{2})} \cdot C_n = B_n - W^{-n} \cdot C_n \qquad (3.8)$$

$$n = 0, 1, 2, \ldots \frac{N}{2} - 1$$

Otrzymaliśmy parę związków pozwalających na obliczenie transformaty ciągu X_k, na podstawie transformat ciągów o połowę krótszych

$$\begin{aligned} A_n &= B_n + W^{-n} C_n \\ A_{n+\frac{N}{2}} &= B_n - W^{-n} C_n \end{aligned} \qquad n = 0, 1, 2, \ldots \frac{N}{2} - 1 \qquad (3.9)$$

Obliczenie B_n lub C_n wymaga już tylko $\left(\frac{N}{2}\right)^2$ działań. Jeżeli $\frac{N}{2}$ jest nadal liczbą parzystą, opisaną wyżej procedurę można powtórzyć w stosunku do ciągów Y_k oraz Z_k, tak by ich transformaty liczyć na podstawie transformat ciągów o długości $\frac{N}{4}$.

Jeżeli więc $N = 2^n$, to postępowanie takie może być powtórzone n razy, w wyniku czego ciąg X_k $k = 0, 1, \ldots N-1$ podzielony zostanie na N ciągów jednoelementowych.

Ponieważ zgodnie z (2.6) widmo ciągu jednoelementowego x^l

$$\sum_{k=0}^{0} X^l e^{-i2\pi kn} = X^l \qquad l = 0, 1, 2, \ldots N-1 \qquad (3.10)$$

składa się tylko z jednego elementu i to równego jedynemu wyrazowi ciągu, ciągi jednoelementowe X^l uznać można za swoje transformaty DFT.

Tak więc obliczenia DFT sprowadzone zostały wyłącznie do dodawań i mnożeń służących obliczaniu kombinacji liniowych krótszych transformat pośrednich.

Opisany wyżej sposób postępowania ukazany został na rys. 3.1 dla przypadku

$$N = 2^3 = 8$$

Jak widać z rys. 3.1.c, liczba dodawań, równa liczbie węzłów sieci, wynosi 8 x 3 (ogólnie $N.\log_2 N$), a liczba mnożeń, reprezentowanych przez strzałki, wynosi 2 x 8 x 3 (ogólnie $2N\log_2 N$).

Z(3.9) widać jednak, że połowa mnożeń jest zbędna, ponieważ współczynnik przy wyrazach B_n wynosi 1. Połowa pozostałych mnożeń może być wyeliminowana z uwagi na okresowość funkcji W^n, co również wynika z (3.9). Niezbędne jest więc wykonanie $\frac{1}{2} N \log_2 N$ dodawań zespolonych oraz tylu odejmowań i mnożeń.

Dużą zaletą zilustrowanej na rys. 3.1.c procedury jest możliwość wykonywania tzw. "in place computation", tzn. wykonywanie obliczeń w taki sposób, że wszelkie wyniki pośrednie, jak również końcowe, umieszczane są w tych samych komórkach, z których pochodzą użyte do nich dane. Odbywa się to bez konieczności zajmowania dodatkowych komórek, przepisywania i tym podobnych zabiegów. Natomiast wadą algorytmu jest konieczność zmiany porządku wyrazów transformowanego ciągu w sposób widoczny na rys. 3.1.c.

Grupowanie w dziedzinie częstotliwości

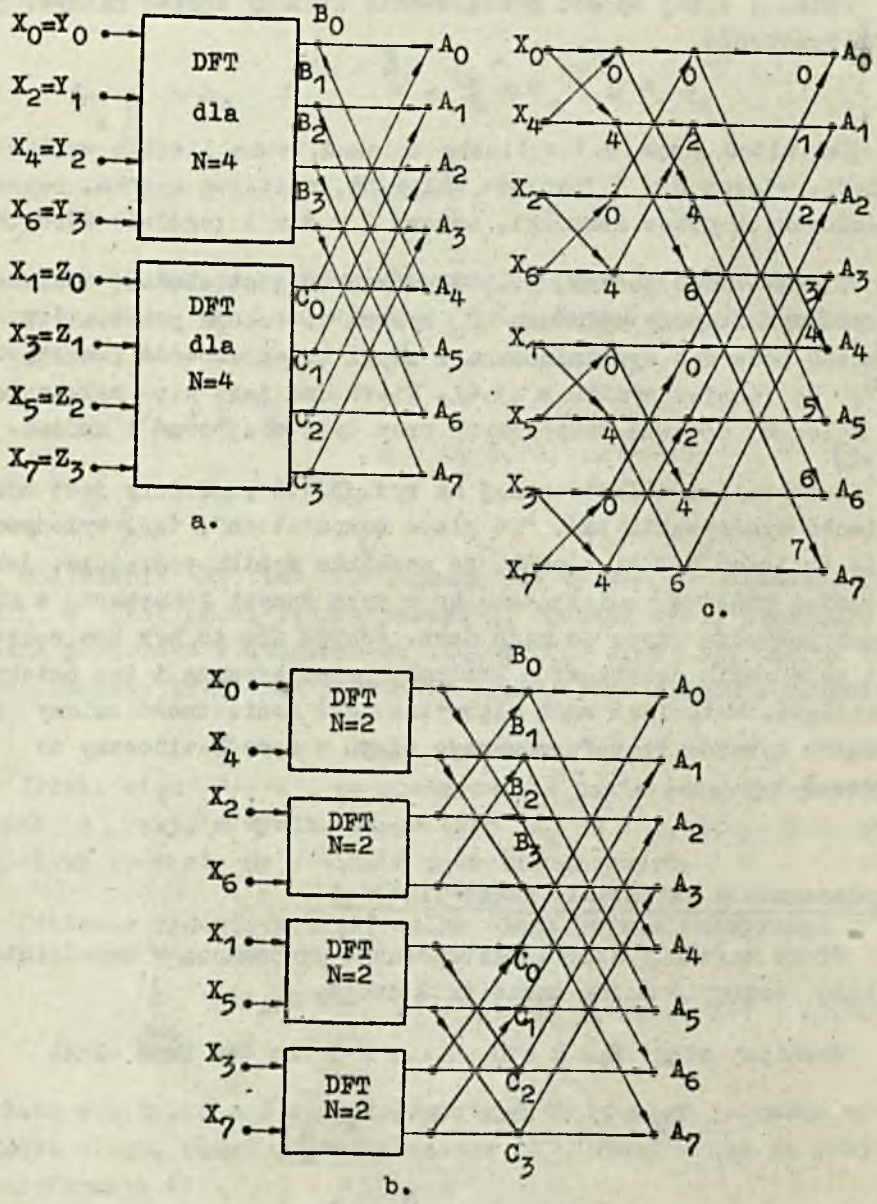
Efekt uzyskany dzięki zastosowaniu grupowania w dziedzinie czasu, osiągnąć można także inną drogą.

Rozbijmy ciąg X_k, $k = 0, 1, \ldots N-1$ na dwa inne ciągi

$$Y_k = X_k \quad \text{oraz} \quad Z_k = X_{k+\frac{N}{2}} \tag{3.11}$$

$$k = 0, 1, \ldots \frac{N}{2} - 1$$

Jak łatwo można sprawdzić, wyrazy A_n o indeksach parzystych uzyskamy poddając DFT ciąg $Y_k + Z_k$, natomiast wyrazy o indeksach nieparzystych transformując ciąg



Rys. 3.1. a, b, c – kolejne fazy grupowania w dziedzinie czasu. Węzły oznaczają dodawanie lub odejmowanie zgodnie z (3.9), strzałki reprezentują mnożenie przez W^n, przy czym wskaźnik n umieszczony został na rys. c obok odpowiadającej mu niżej położonej strzałki

$$(Y_k - Z_k)\ W^{-k}, \quad k = 0, 1 \ldots \frac{N}{2} - 1$$

Metodę grupowania w dziedzinie częstotliwości ilustruje dokładnie rys. 3.2.

Uzyskany tą drogą algorytm posiada podobne własności jak poprzedni. Jedyna w zasadzie różnica polega na tym, że dane wejściowe nie wymagają zmiany porządku, ale za to wyniki uzyskiwane są w porządku "odwrotnych numerów". Jeżeli w sieci z rys. 3.1.c. dokonać uporządkowania wyrazów ciągu wejściowego według ich kolejności, przenosząc jednocześnie wszystkie węzły leżące na tym samym poziomie, to uzyskamy w ten sposób sieć z rys.3.2.c. Tak więc oba te algorytmy są w zasadzie równoważne.

Można też zbudować algorytmy dające wyniki w normalnym porządku i nie wymagając przy tym zmiany kolejności wyrazów wejściowych. Takie jednak algorytmy nie dają już możliwości dokonywania obliczeń "in place", a więc wymagają dodatkowej pamięci do przechowywania wyników pośrednich.

Dwa takie algorytmy zilustrowane zostały przez sieci a i b z rys. 3.3. Podobnie jak poprzednio można tu dokonać przekształcenia jednej sieci w drugą w sposób opisany powyżej.

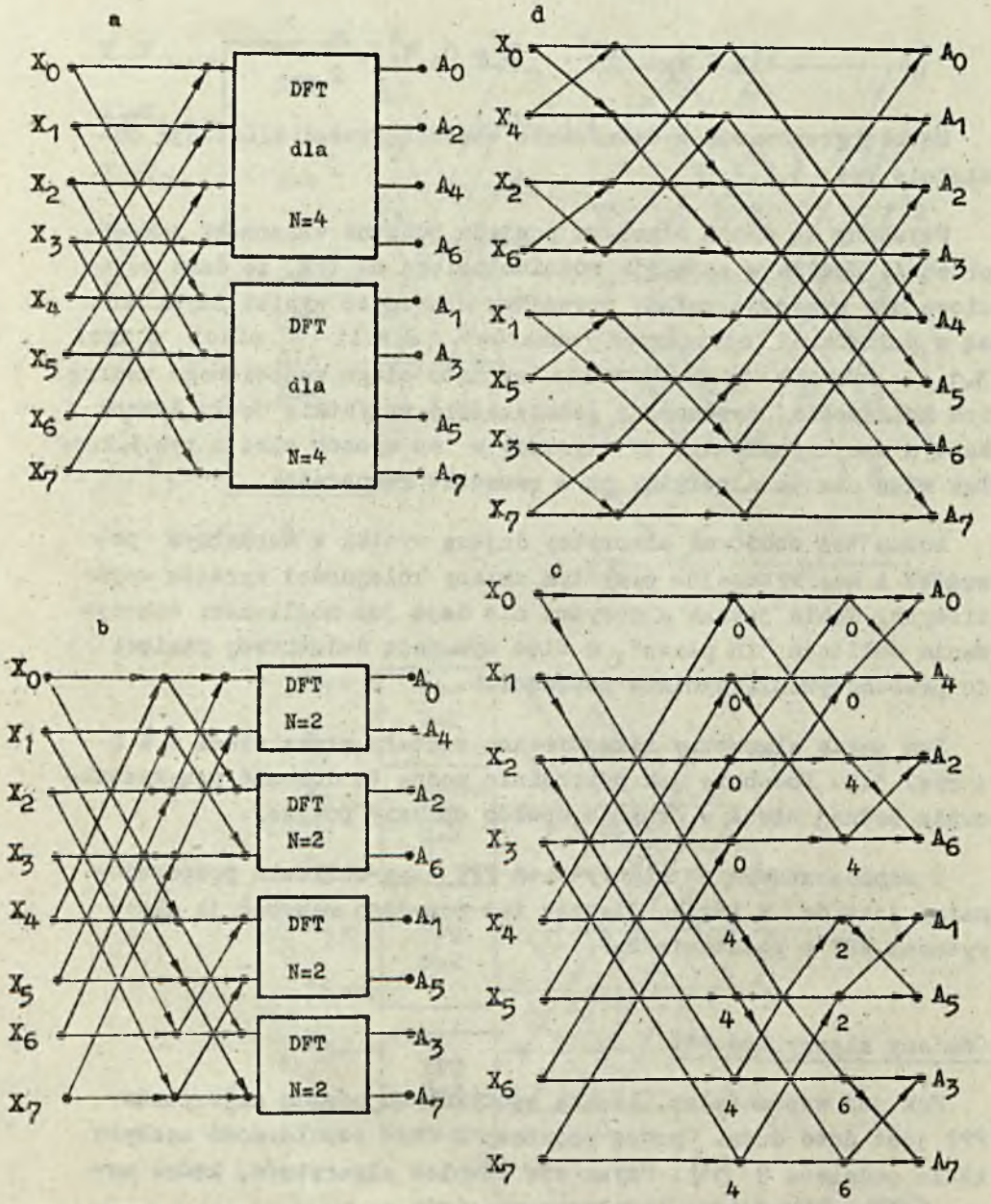
W zaprezentowanych algorytmach FFT czas obliczeń proporcjonalny jest do $N \log_2 N$, dlatego też przyjęto nazywać je algorytmami FFT o podstawie 2.

Odmiany algorytmów FFT

Jak już wspomniałem, liczba spotykanych odmian algorytmów FFT jest dość duża. Oprócz podstawy 2 dużą popularność zdobyła także podstawa 8 [11]. Używa się również algorytmów, które mogą być stosowane dla dowolnego m, gdzie

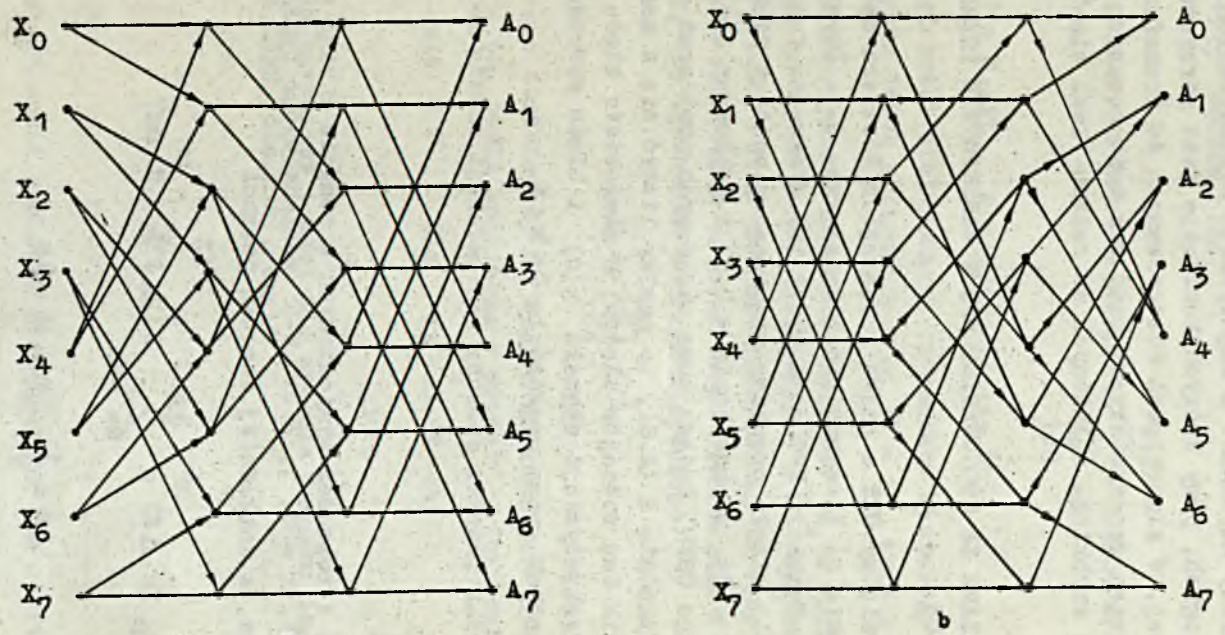
$$N = m^n \qquad m - \text{liczba naturalna}, \qquad (3.12)$$

a nawet algorytmy dla podstaw mieszanych [8], [7].



Rys. 3.2.

a, b, c - kolejne fazy grupowania w dziedzinie częstotliwości (oznaczenia jak na rys. 3.1.), d - sieć uzyskana przez porządkowanie wyników z jednoczesnym przenoszeniem wszystkich węzłów leżących na tym samym poziomie co przenoszony wynik końcowy



Rys. 3.3. a, b – dwa warianty algorytmów FFT, nie wymagające sortowania danych wejściowych ani wyników. Jak widać z rysunku, algorytmy nie dają możliwości wykonywania tzw. "in place computation", tzn. wymagają przynajmniej jednej tablicy dodatkowej dla przechowywania wyników pośrednich. Wykładniki odpowiednich czynników W_N^m zostały dla większej przejrzystości pominięte

Duży wpływ na postać algorytmu ma także cel jakiemu ma on służyć. Wystarczy wspomnieć o algorytmach dostosowanych do ciągów bardzo długich, przy których konieczne jest korzystanie z pamięci masowej, o algorytmach dostosowanych do transformacji ciągów o wyrazach rzeczywistych, bądź też dostosowanych do obliczeń bardzo szybkich, zbliżonych do reżimu "real time".

4. BŁĘDY ZWIĄZANE ZE STOSOWANIEM DFT W ZASTĘPSTWIE PRZEKSZTAŁCEŃ CIĄGŁYCH

Związek pomiędzy DFT a całką i szeregiem Fouriera omówiony został w punkcie 2. Ilustruje go również rys. 2.1. Oryginałami w sensie transformacji DFT mogą być tylko dyskretne funkcje okresowe. Jeśli ciągły przebieg nieokresowy poddamy dyskretyzacji dla $t = k \cdot \Delta t$, w przedziale $< 0,T)$ i posłużymy się w dalszym ciągu jego OSFN, będącą sumą okresowych powtórzeń z przedziału $< 0,T)$ zgodnie z (2.6), to musimy liczyć się z możliwością wystąpienia dwu rodzajów błędów, a mianowicie błędu będącego wynikiem dyskretyzacji sygnału $x(t)$ i błędu spowodowanego pominięciem próbek spoza przedziału $0 \leq k < N$.

Błędy te, jak również sposoby ich redukcji omówię kolejno.

Dyskretyzacja

Jak wynika z rozważań wymienionych w punkcie 2 ciąg dyskretny $X(k.\Delta t)$, $k = 0, 1, 2 \ldots N-1$ posiada widmo będące ciągiem okresowym, a dokładniej ciągiem postaci

$$A_p(n.\Delta f) = \sum_{s=-\infty}^{+\infty} A\left[(n + s \cdot N)\Delta f\right] \qquad (4.1)$$

$$\Delta f = \frac{1}{T} \qquad T = N \cdot \Delta t$$

jest to więc ciąg odpowiadający OSFN dla widma funkcji ciągłej $x(t)$ o okresie równej ilości próbek N, pobranych w czasie $< 0,T)$.

Jeśli przy zadanym Δt liczba N zostanie wybrana zbyt mała, to efektem sumowania okresowych powtórzeń widma $a(f)$ zgodnie z (4.1) będzie błąd

$$A_r(n.\Delta f) = A_p(n.\Delta f) - A(n.\Delta f) = \sum_{\substack{l=-\infty \\ l\neq 0}}^{+\infty} A(n.\Delta f + l.N.\Delta f) \qquad (4.2)$$

gdzie $F = N.\Delta f \qquad n = 0, 1, 2, \ldots N-1$

W większości praktycznych przypadków wystarczy uwzględnić wpływ tylko dwóch składników sumy ($l = -1$ oraz $l = +1$).

W takim przypadku największa wartość $A_r(n.\Delta f)$, przypadająca, jak to widać z rys. 2.1.e dla $n = \frac{N}{2}$ wyniesie $0.5\, A_p\left(\frac{N}{2}.\Delta f\right)$. Przy zadanej maksymalnej wartości A_r równej δ, należy tak dobrać częstotliwość próbkowania $F = \frac{1}{\Delta}$ $t = N.\Delta f$, aby spełniona została nierówność

$$0.5\, Ap\left(\frac{F}{2}\right) < \delta \qquad (4.3)$$

Znacznie trudniejszy do wyeliminowania jest drugi błąd. Nie występuje on, jeśli analizowany sygnał $x(t)$ posiada własność

$$x(t) \equiv 0 \quad \text{dla} \quad t \notin \langle \Delta, T-\Delta \rangle \qquad (4.4)$$
$$\Delta > 0$$

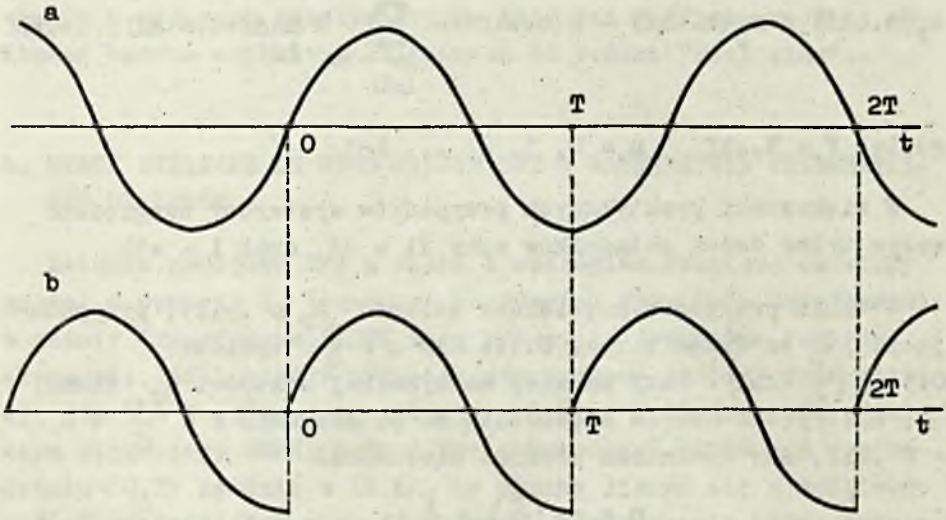
Jeżeli natomiast $x(t)$ nie spełnia warunku (4.4), funkcja okresowa OSFN dla $x(t)$, czyli

$$x_p(t) = \sum_{l=-\infty}^{+\infty} x(t + l.T) \qquad (4.5)$$

będzie w ogólności posiadać nieciągłości lub ostrza w punktach

$$t = l.T \qquad l = 0, \pm 1, \pm 2 \ldots$$

Spowoduje to znaczne zniekształcenie widma, w postaci wzrostu udziału wyższych harmonicznych. Efekt ten nazywany jest zwykle rozpływem lub przeciekiem widma (tzw. leakage effect) [13].



Rys. 4.1. Zniekształcenia sygnału wynikające ze złego doboru okresu T

a - sygnał pierwotny, b - sygnał odpowiadający uzyskanemu poprzez DFT widmu

W praktyce analizowane sygnały mają czas trwania wielokrotnie dłuższy od okresu analizy, trzeba się więc liczyć z możliwością wystąpienia "przecieku". Jest on oczywiście mniej niebezpieczny, jeżeli sygnał jest mało skorelowany, niż wówczas, gdy sygnał jest korelowany mocno. W tym ostatnim przypadku wskazana jest tzw. modyfikacja liniowa widma - zabieg pozwalający na wydatne zmniejszenie "przecieku".

W dziedzinie czasu, modyfikacji takiej odpowiada wymnożenie sygnału przez odpowiednio dobraną funkcję noszącą nazwę okna modyfikacji liniowej ("data widnow").

Oknem może być funkcja w(t) posiadająca następujące cechy:
1. osiąga zero na końcach przedziału analizy,
2. wzrasta łagodnie w miarę przesuwania się ku środkowi przedziału,

3. spełnia równość

$$\frac{1}{T}\int_0^T w^2(t)\,dt = 1$$

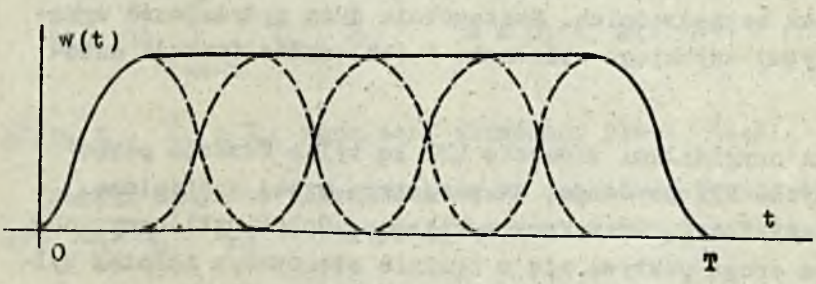
4. jej ciągła transformata Fouriera, a więc widmo V(f) spełnia nierówność

$$\frac{d\,[\lg|V(f)|]}{d\,(\lg f)} < -0.05$$

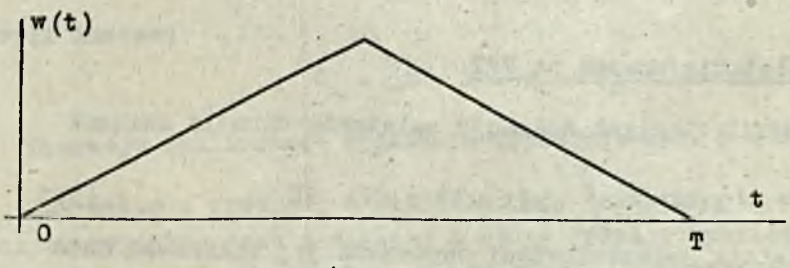
dla $f > \varepsilon$, gdzie ε odpowiednio mała stała dodatnia.

Przykłady takich okien pokazują rys. 4.2, 4.3, 4.4.

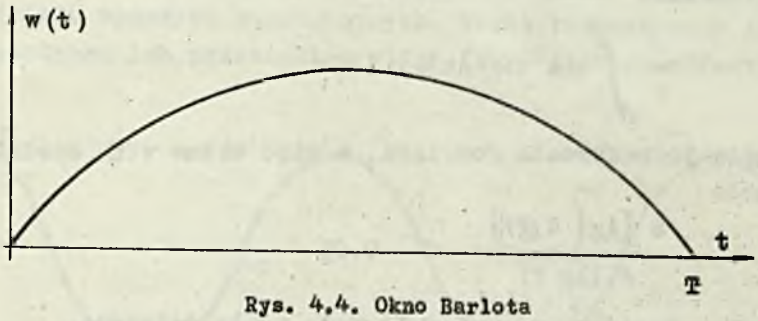
Należy dodać, że w przypadku sygnałów niezdeterminowanych, słabo skorelowanych modyfikacja liniowa ustępuje tzw. modyfikacji kwadratowej, wykonywanej nie na widmie sygnału, a na jego periodogramie (widmie mocy) [15], [16].



Rys. 4.2. Uogólnione okno kosinusoidalne rzędu piątego



Rys. 4.3. Okno Partzena



Rys. 4.4. Okno Barlota

5. SZYBKIE ALGORYTMY OPARTE NA FFT

Jak wspomniano wcześniej, FFT okazała się nie tylko najszybszą metodą obliczania widma ciągu dyskretnego, ale nawet obliczanie innych charakterystyk ciągów, takich jak funkcje autokorelacji, korelacji skośnych czy splotu, a nawet transformacji Z czy Hilberta, drogą niejako okrężną, bo na podstawie widma uzyskiwanego dzięki FFT, okazało się szybsze niż na podstawie obliczeń bezpośrednich. Szczególnie dużą przydatność wykazały algorytmy szybkiego splatania i obliczania funkcji autokorelacji.

Ponieważ oryginałami w sensie DFT są tylko funkcje periodyczne, użycie FFT powoduje, że wszystkie wyżej wymienione charakterystyki mają charakter cykliczny. Splot cykliczny, obliczany tą drogą pokrywa się z ogólnie stosowanym splotem tylko przy zastosowaniu pewnych dodatkowych zabiegów.

Zilustrować można to na przykładzie cyklicznego splotu i cyklicznej funkcji autokorelacji.

Szybkie splatanie oparte na FFT

Dla funkcji ciągłych operację splatania określa związek

$$z(t) = \int_{-\infty}^{t} y(t-\tau)\, x(\tau)\, d\tau \qquad (5.1$$

W dziedzinie częstotliwości odpowiada jej zależność dużo prostsza, a mianowicie

$$c(f) = b(f) \cdot a(f) \tag{5.2}$$

gdzie a(f), b(f) i c(f), to widma x(t), y(t) i z(t) odpowiednio.

Interesuje nas określenie podobnej pary zależności dla ciągów dyskretnych. W tym celu ograniczymy się do sygnałów o skończonym czasie trwania T

$x(t) \equiv y(t) \equiv 0$ dla $t \notin <0,T)$. Po dyskretyzacji funkcji x(t), y(t) i z(t) dla $t = k.\Delta t$, $k = 0, 1, 2 \ldots N-1$ $\Delta t = \frac{T}{N}$ związek (5.1) przyjmuje postać

$$Z(k.\Delta t) = \Delta t \sum_{l=k-N}^{k} Y\left[(k-l)\Delta t\right] X(l.\Delta t) \tag{5.3}$$

Jeśli zastąpić powyższe ciągi dyskretne przez ich OSFN, to otrzymamy związek

$$Z_k = \Delta t \sum_{l=k-N}^{k} Y_{k-l} X_l \qquad k = 0, 1, 2 \ldots N-1 \tag{5.4}$$

gdzie Z_k, Y_k i X_k mają sens określony przez (2.5).

Jeżeli ciągi te potraktować jako transformaty odwrotne ciągów C_n, B_n i A_n, to otrzymamy związek analogiczny do (5.2).

$$Z_k = \Delta t \sum_{m=0}^{N-1} N \cdot A_m \cdot B_m W^{mk}$$

$$k = 0, 1, \ldots N-1$$

czyli inaczej

$$C_n = \Delta t \cdot N \cdot A_n \cdot B_n \tag{5.5}$$

Operację cyklicznego splatania ilustruje rys. 5.1 a b c.

Jak widać z rysunku, w przypadku gdy czas trwania obu splatanych sygnałów jest jednakowy i równy T, splot cykliczny wcale nie odpowiada splotowi w sensie (5.3), jest więc zupełnie

nieprzydatny do celów praktycznych. Trudność tę można jednak łatwo usunąć wydłużając oba splatane ciągi, przez dodanie na końcu każdego z nich liczby próbek zerowych odpowiadającej długości drugiego ze splatanych ciągów.

Ilustruje to dokładnie rys. 5.1 d i e.

Szybkie obliczanie funkcji autokorelacji

Sytuacja przedstawia się w tym przypadku podobnie. Dla funkcji ergodycznej jej funkcja autokorelacji obliczona być może na podstawie zależności przybliżonej

$$g(\tau) \cong \frac{1}{T} \int_0^T x(t+\tau)\, x(t)\, dt \qquad (5.6)$$

gdzie T musi być dostatecznie duże.

Jeśli $x(t) \equiv 0$ dla $t \notin \langle 0,T)$, to po dyskretyzacji w momentach $t = k \cdot \Delta t$ otrzymamy

$$G(l \cdot \Delta t) = \frac{1}{N} \sum_{k=0}^{2N-1} X[(k+l)\Delta t]\, X(k \cdot \Delta t) \qquad (5.7)$$

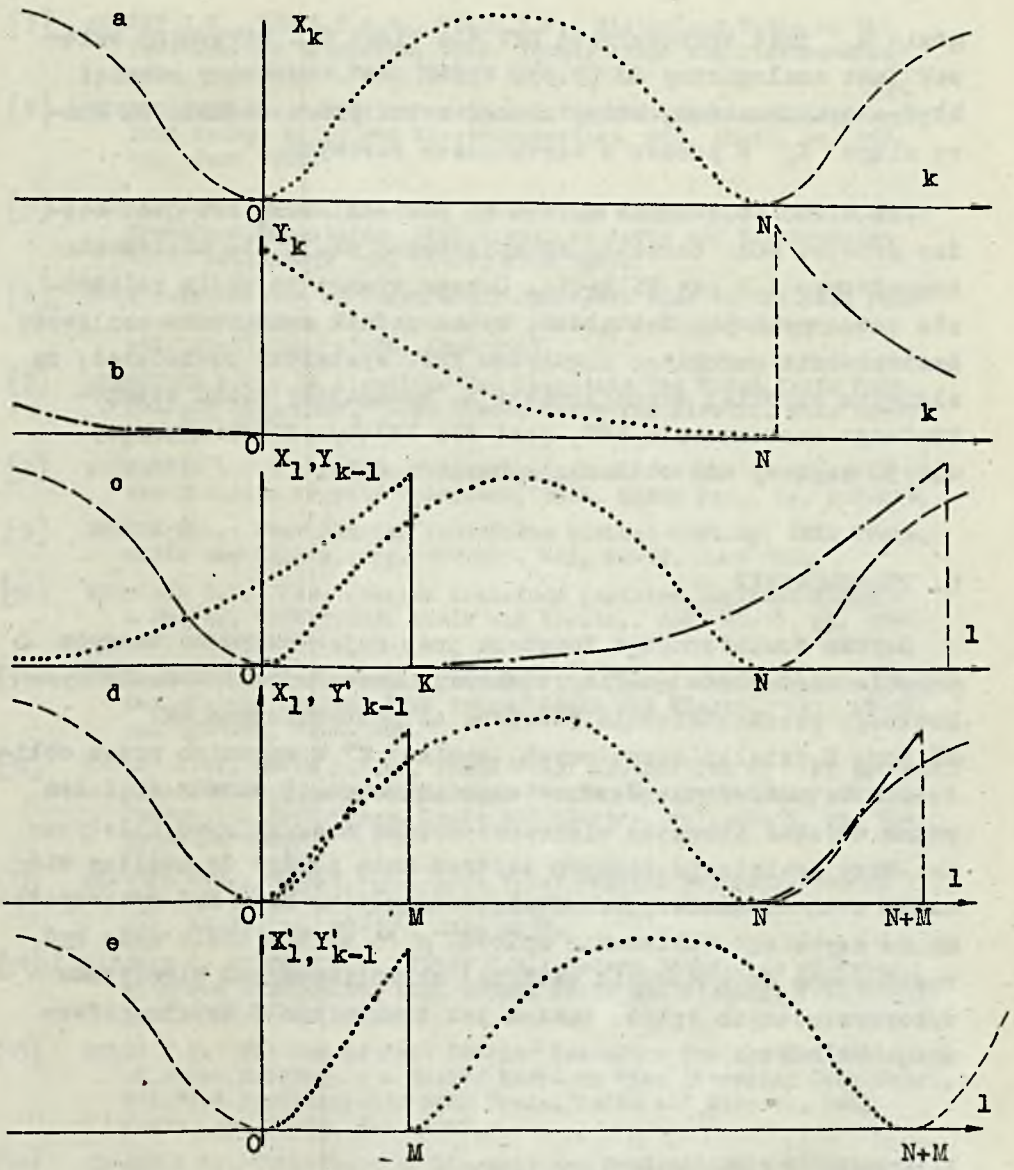
Jeśli powyższe ciągi zastąpić odpowiadającymi im oryginałami DFT, to otrzymamy zależność

$$G_l = \frac{1}{N} \sum_{k=0}^{N-1} X_{k+l}\, X_k \qquad (5.8)$$

Stosując postępowanie podobne jak w przypadku splotu otrzymamy interesującą nas zależność

$$G_l = \sum_{n=0}^{N-1} A_n\, A_{-n}\, W^{nl} \quad \text{lub też}$$

$$R_n = A_n\, A_{-n} \qquad (5.9)$$



Rys. 5.1. Obliczanie splotu cyklicznego

a, b - splatane ciągi, c - obliczanie splotu cyklicznego na podstawie OSFN tych ciągów, d - splatanie ciągów różnej długości; ciąg krótszy uzupełniono N-M zerami; uzyskany splot cykliczny pokrywa się z (5.3) dla $M < k < N$, e - splatane ciągi uzupełniono zerami do długości M+N - wynik pokrywa się z (5.3) dla $M < k < M+N$. Linią przerywaną zaznaczono okresowe powtórzenia splatanych ciągów

gdzie R_n jest transformatą DFT dla ciągu G_1. Otrzymany związek jest analogiczny do (5.5). Wynik jest obarczony również błędem cykliczności, który usunąć można przez dodanie na końcu ciągu X_k N próbek o wartościach zerowych.

Jak widać obliczanie splotu za pośrednictwem FFT jest bardzo proste. Dużo bardziej skomplikowane stają się obliczenia transformacji Z czy Hilberta. Czasem wymaga to wielu całkowicie sztucznych przekształceń, które jednak wynagradza możliwość zastosowania szybkiego algorytmu FFT. Wystarczy powiedzieć, że algorytm szybkiej transformacji Z, wymagający m.in. czterokrotnego zastosowania FFT, jest dla liczby próbek większej od 50 szybszy niż obliczenia bezpośrednie.

6. PODSUMOWANIE

Szybka transformacja Fouriera jest najszybszym ze znanych obecnie algorytmów analizy widmowej. Pozwala na dokonanie dyskretnego przekształcenia Fouriera za pośrednictwem ok. $2N \log_2 N$ działań zespolonych, zamiast N^2 wymaganych przez obliczenia bezpośrednie. Jeszcze większą redukcję czasów obliczeń można uzyskać stosując wieloprocesorowe maszyny specjalistyczne. Przy spełnieniu pewnych założeń może służyć do analizy widmowej w zastępstwie przekształceń ciągłych. Może być wykorzystana do szybkiego obliczania splotu, a co za tym idzie może być zastosowana do budowania bardziej skomplikowanych algorytmów wykorzystujących splot, takich jak transmisja Z czy transformacja Hilberta.

Literatura

[1] GOLDSTEIN R., KENDALL W.B.: Low Data Rate Telemetry, Proc. Am. Astronautical Soc.Symp. on Unmanned Exploration of the Solar System, pp. 501-516, February 1965.

[2] FERGUSON J.J.: Communication at Low Data Rateoscillator Models and Corresponding Optimal Receivers, IEEE. Trans. Communication Technology /Concise Papers/, vol. COM-16, pp. 629-631, August 1968.

[3] COOLEY J.W., LEWIS P.A.W., WELCH P.P.: Historical Notes on the Fast Fourier Transform, IEEE. Trans. Audio and Electroacoustics, vol. AU-16, pp. 79-85, June 1967.

[4] THEILHEIMER F.: A Matrix Version of the Fast Fourier Transform, IEEE Trans. Audio and Electroacoustics, vol. AU-17, pp. 158-162, June 1969.

[5] COOLEY J.W. i inni: The 1968 Arden House Workshop on Fast Fourier Transform Processing, IEEE Trans. on Audio and Electroacoustics, vol. AU-17, pp. 66-77, June 1969.

[6] G-AE Subcommittee on Measurement Concepts. What is the Fast Fourier Transform, IEEE Trans. on Audio and Electroacoustics, vol. AU-15, pp. 45-56, June 1967.

[7] SINGLETON R.C.: An Algorithm for Computing the Mixed Radix Fast Fourier Transform, IEEE Trans. Audio and Electr., vol. AU-17, pp. 93-104, June 1969.

[8] BLUESTEIN L.I.: A Linear Filtering Approach to the Computation of the Discrete Fourier Transform, 1968. NEREM Rec., pp. 218-219.

[9] UHRICH M.L.: Fast Fourier Transforms Without Sorting, IEEE Trans. Audio and Electr., pp. 170-173, vol. AU-17, June 1969.

[10] BERGLAND G.D.: Fast Fourier Transform Hardware Implementation - A Survey, IEEE Trans. Audio and Electr., vol. AU-17, pp. 109-120, June 1969.

[11] BERGLAND G.D.: Radix-Eight Fast Fourier Transform Subroutine for Real-Valued Series, IEEE Trans. Audio and Electr., vol. AU-17, pp. 138-145, June 1969.

[12] COOLEY J.W., LEWIS P.A.W., WELCH P.D.: Application of FFT to Computation of Fourier Integrals, Fourier Series and Convolution Integrals, IEEE Trans. Audio and Electr., vol. AU-15, pp. 79-89, June 1967.

[13] MALLING G.G., MORREY W.T., LANG W.W.: Digital Determination of Third-Octave and Full-Octave Spectra of Acoustical Noise, IEEE, vol. AU-15, pp. 98-104, June 1967.

[14] BINGHAM C., GODFREY M.D., TUKEY J.W.: Modern Techniques of Power Spectrum Estimation, IEEE Trans. Audio and Electr., vol. AU-15, pp. 56-66, June 1967.

[15] WELCH P.D.: The Use of Fast Fourier Transform for the Estimation of Power Spectrum - A Method Based on Time Averaging Over Short, Modified Periodograms, IEEE Trans. Audio and Electr., vol. AU-15, pp. 70-74, June 1967.

[16] SLOANE E.A.: Comparison of Linearly and Quadratically Modified Spectral Estimates of Gaussian Signals, IEEE Trans. Audio and Electr., vol. AU-17, pp. 133-138, June 1969.

[17] COOLEY J.W., LEWIS P.A.W.; WELCH P.P.: The Finite Fourier Transform, IEEE Trans. Audio and Electr., vol. AU-17, pp. 77-86, June 1969.

ДИСКРЕТНОЕ ПРЕОБРАЗОВАНИЕ ФУРЬЕ И БЫСТРЫЕ АЛГОРИТМЫ СПЕКТРАЛЬНОГО АНАЛИЗА С ПРИМЕНЕНИЕМ ЦИФРОВОЙ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ

Резюме

Быстрое развитие цифровых вычислительных машин и разработка новых алгоритмов вычисления позволили заступить несовершенные аналоговые методы спектрального анализа цифровыми методами.

Алгоритм позволяющий значительно сократить время вычисления, так называемого Дискретного Преобразования Фурье, изобретенный Кулейем в 1965 году, ввёл новые качества в области вычислительных методов спектрального анализа функции (в смысле Фурье) , вычисления разных трансформат и характеристик функции, а также в области конструкции специализованных вычислительных цифровых машин, предназначенны главным образом - или исключительно - для вычислении по этим алгоритмам.

Дискретным преобразованием Фурье называется здесь вид интегрального преобразования Фурье, приспособленный для преобразования периодических числовых последовательностей.

Вычисления спектра последовательности с периодом N элементов требует, на основе определяющей формулы, выполнения N^2 комплексных умножений и сложений.

Итеративные алгоритмы, так называемого Быстрого преобразования Фурье, позваляют значительно уменьшит порядок требуемого числа операций для больших N , принимая что N является достаточно сложным числом. Для $N=2^m$, где m целое положительное число, применение Быстрого Преобразования Фурье позволяет уменьшить число комплексных умножений и сложений с N^2 на $N \cdot {}^{10}g2N$.

Посколько как оригиналы Дискретного Преобразования Фурье, так и полученные преобразователи являются периодическими числовыми последовательностями, применение быстрого преобразования Фурье к вычислению преобразований кусков непериодических непрерывных функций, дает результаты, на которые наложены ошибки двух видов.

Увеличивая частату пробования анализуемой функции, или применяя модификацию спектра можно сократить размер этих ошибок. Так называемая линейная модификация состоящая в предварительном умножении значения анализованной функции на значение модифицирующей функции (так называемые окна линейной модификации) , применяется по отношению к сильно коррегированным функциям. Для шумоподобных (слабо коррегированных) функций применяется так называемая квадратная модификация.

Блогодаря прибыли времени, являющейся результатом применения алгоритмов Быстрого Преобразования Фурье к вычислению спектра функции, вычисление для примера сплетения двух функций как обратного преобразования произведения их спектров требует значительно меньше времени для непосредственных вычислений. Подобную технику можно применят к вычислении функций автокорреляции, преобразования z , Гильберта и других.

DISCRETE FOURIER TRANSFORMATION AND COMPUTER AIDED FAST COMPUTATION ALGORITHMS OF SPECTRAL ANALYSIS

Summary

Due to computer fast development and new computation algorithms, imperfect analogue methods of spectral analysis could be replaced by digital methods.

Digital Fourier Transformation algorithm given by J.W. Cooley in 1965, permitted to shorten significantly the computation time, and accomplished a revolution in the field of function spectral anylysis computation methods, in the Fourier sense, in computing various transforms and characteristics of functions and also in the field of special-purpose computer construction destined mainly or exclusively for these algorithms realization.

The form of an integral Fourier transformation adapted to transform periodical number sequences is called Discrete Fourier Transformation. Computation of a sequence spectrum with a period of N elements based on a defining formula requires N^2 compound multiplications and additions.

Iteration algorithms - so called Fast Fourier Transformation - permit for big N to reduce the required number of operations by a few orders of magnitude N being assumed a sufficiently complex number. If N is a natural exponent of number two, then the use of Fast Fourier Transformation permits to reduce the number of compound multiplications and additions from N^2 to $N.\log 2N$.

As both originals of Discrete Fourier Transformation and transforms obtained are periodical number sequences, the use of Fast Fourier Transformation to compute transforms of fragments of continuous nonperiodical functions provide the results that bear two kinds of errors.

These errors can be reduced either by augmenting the frequency of the analyzed function samplying or by using spectrum modification. The so called line modification, consisting of a preliminary multiplication of the analyzed function value by the modifying function value (so called "window" of line modification), is used in relation to strough correlated functions. For weakly correlated functions so called square modification is used.

Due to the gain of time resulting from the use of Fourier Fast Transformation algorithms to compute the spectrum function the computation for example of an entanglement of two functions as an inverse transform of their spectrum product, is much less time consuming than direct computation. A similar techniqus can be used to compute functions of autocorrelation, Z transformation, Hilbert's and others.

ALGORYTMY, X, N° 18, 1973, 63-78

681.322.004.14:518.9

МОДЕЛИРОВАНИЕ ИГРЫ НА ВЫЧИСЛИТЕЛЬНОЙ МАШИНЕ

Герард ЗЕЛИНСКИЙ

Instytut Matematyki
Politechniki Warszawskiej
Warszawa, Pl.Jedności Robotniczej

Поступило 22.05.1972

Описано моделирование игры на вычислительной машине, реализованное на малой машине Урал-1. Моделированная игра является игрой для 4 человек с неполной информацией с 2 коалициями определёнными случайно. Подана характеристика машинных техник моделирования, описаны правила игры, а также проведён анализ игры на основе теории игр. Рассмотрена машинная программа моделирования и указана возможность использования техники моделирования на цифровых вычислительных машинах.

В последнее время все чаще на универсальных цифровых машинах ставятся задачи, принадлежащие к классу невычислительных задач. Эти задачи обыкновенно решаются путем машинной симулировки.

Машинная симулировка бывает различная. Существенно отметить, что основным принципом симулировки принимается или подражание течению симулированного процесса, или получение результатов аналогичных результатам симулированного процесса. В первом случае мы будем называть симулировку моделированием. Конечно всякое моделирование удовлетворяет второму принципу симулировки,

но не всякая симулировка является моделированием. Пример симулировки - не моделирования - описывает [10].

Настоящая статья содержит описание моделирования карточной игры Васька. Игру эту автор предпочел по нескольким причинам: хорошее знакомство автора с игрой, принадлежность ее к классу игр с неполной информацией, характерность Васьки для многих игр.

1. ПРАВИЛА ИГРЫ

Игру проводят 4 игрока с помощью 16 карт: десяток, тузов, валетов и дам. Валеты и дамы принимаются единственными и постоянными козырями в игре, упорядоченными по старшинству следующим образом - начиная со старшей: D♣, D♠, D♡, D♢, W♣, W♠, W♡, W♢. Игра по масти обязательна лишь в отношении к некозырям, причем старшим является здесь туз.

Каждый из игроков получает до игры случайным образом по 4 карты. Игру открывает следующий по часовой стрелке за сдающим карты. Дальше ходят остальные игроки в порядке часовой стрелки. Игра состоит из 4 раундов, в каждом из которых совершают ход поочередные игроки. Первый раунд начинает открывающий игру, следующие раунды открывают получившие предыдущие взятки.

Ходы игроков должны удовлетворять следующим правилам. Первый ход в раунде произвольный. Каждый следующий ход раунда зависит от предыдущих таким образом. (к) Если раунд открыт козырем, то игрок делающий ход должен: 1^{o} ложить козыря выше всех предыдущих в раунде; 2^{o} если к1 невозможно, играть произвольным козырем; 3^{o} в случае отсутствия козырей допускается

ход любым некозырем. (н) Если раунд открыт некозырем, то игрок делающий ход должен: 1^{0} играть по масти; 2^{0} если н1 невозможно, играть согласно к1, к2; 3^{0} если н1, н2 невозможно, играть любым некозырем.

Взятку берет тот, который положил козыря выше карт остальных игроков. Если в раунде нет козырей, то взятку берет игрок, у которого был туз масти хода открывающего раунд. Получить взятку, открывая раунд десяткой, можно лишь в случае, когда в раунде нет туза масти открывающей десятки и нет козырей.

Результат игры (выигрыш, проигрыш) определяется по следующим правилам. В зависимости от того, кому попали путем случайной сдачи 2 старшие карты игры, игроки разделяются на 2 коалиции - Старых (получивших D♣,D♠)и Молодых (остальные игроки). Может случиться, что коалиция Старых одноличная. Количесвто очков, набранных коалициями во взятках, решает о победе. Очки подсчитываются согласно нижеследующему:

11 - туз, 10 - десятка, 3 - дама, 2 - валет.

Старые побеждают набрав более половины (62) очков; в противном случае выигрывают Молодые. Таблица платежей для нашей игры следующая:

Очки		Платежи		
Молодых	Старых	Молодых	2 Старых	1 Старого
0	104	3x	-3x	3(-3x)
8 ÷ 25	96 ÷ 79	2x	-2x	3(-2x)
26 ÷ 51	78 ÷ 53	x	-x	3(-x)
52 ÷ 77	52 ÷ 27	-2x	2x	3(2x)
78 ÷ 94	26 ÷ 10	-4x	4x	3(4x)

где x есть основная ставка игры.

2. ТЕОРЕТИЧЕСКИЙ АНАЛИЗ ИГРЫ

Основные теоретико-игровые понятия приняты здесь в изложении [8].

Дерево нашей игры конечное. Выделенный узел принадлежит случайности. Ходов в игре 17 - первый ход случайный, остальные личные. Если разбить ходы на классы указывающие чей ход, то первый ход принадлежит 1 классу, остальные распределяются по 4 классам. Баська - игра с неполной информацией. Потому среди информационных множеств существенно найдутся множества, содержащие более чем 1 узел. Множество исходов содержит 5 элементов.

Вопрос коалиций особенно сложен в Баське, ибо принадлежит она к классу игр, в которых нет сообщений до игры. В течении игры наступают коалиционные перемены. С момента полного выявления коалиций одна платежная функция однозначно определяет остальные. Игра превращается тогда в игру 2 лиц с нулевой суммой, если предположить, что игроки выберут из нескольких возможностей всегда лучшую для своей коалиции [3]. Баська полностью соответствует определению коалиционной игры данному Воробьевым [4].

Игра принадлежит к числу несимметричных задач теории игр [1]. Если представить Баську в нормальной форме, то окажется она игрой в нормальной форме с запрещенными ситуациями [5]. В силу системы правил игры множество запрещенных ситуаций довольно большое.

А теперь представим конструктивное теоретико-игровое решение Баськи. Для действительного числа x через $[x]$ обозначим целое число, удовлетворяющее неравенству $x-1 < [x] \leqslant x$.

Пусть к обозначает номер хода игры, который мы совершаем (к = 2, 3 ..., 17). Мы будем рассматривать лишь случаи, когда к < 14, ибо 4 последние хода однозначно определены предыдущими. До хода к, который мы делаем, в игре использовано уже (k-2) карты, у нас в руке имеется [(21-k)/4] карт. В наиболее общем случае - когда (k-1) начальных ходов игры не содержит информации о картах имеющихся у игроков после их совершения - неизвестно нам расположение $\{16 - (k-2) - [(21-k)/4]\}$ карт. С одинаковой вероятностью допускаем каждое из возможных разложений карт. Число возможных разложений можно уменьшить к числу существенно различных разложений, а иногда еще более уменьшить за счет информации, содержащейся в (k-1) начальных ходах. При таком подходе в каждом из допускаемых вариантов (вариантов меньше чем $(16 - (k-2) - [(21-k)/4])!$) коалиции являются однозначно определенными и постоянными для каждого варианта. В связи с тем, мы можем рассматривать поочереди все варианты как игры 2 лиц с нулевой суммой. Чистыми стратегиями игрока - коалиции принимаем всевозможные последовательности карт имеющихся в распоряжении коалиции. Конечно, если в коалиции 2 или 3 игрока,то согласно правилам игры их карты должны чередоваться соответствующим образом. Для таких наборов стратегий нашей коалиции и коалиции-противника строим матрицу игры (платежную матрицу). Если ситуация, которая получается в результате выбора нами i-стратегии и выбора противником j-стратегии несовместима с правилами игры, то на пересечении i-строки и j-столбца матрицы игры оставим пустое место (предполагаем соблюдение правил игры игроками). Если правила игры допускают такую ситуацию - пару стратегий, то, подсчитав сколько очков мы тогда получим, выпол-

няем соответствующее место в матрице согласно таблице платежей. Заполнив матрицу, мы ищем цену игры для данной матрицы (она всегда существует, так как Баська, игра с конечным числом стратегий для каждого игрока). В зависимости от природы матрицы находим цену игры либо для пары чистых, либо для пары смешанных стратегий. Значение оптимального в среднем выигрыша мы иногда не в состоянии найти точно - в силу затруднений, связанных с большим рангом матрицы, со сведением такого рода матрицы к ее существенной части. Тогда применяются приближенные методы нахождения смешанных стратегий [1]. Найденные цены игр для возможных разложений карт сравниваем, учитывая вероятность разложений, т.е. вычисляем математические ожидания для имеющихся у нас альтернатив при совершении k-того хода. Альтернатив столько, сколько у нас карт, допускаемых правилами игры к ходу. Лучший ход это ход картой, для которой математическое ожидание достигает минимума.

Отметим одно обстоятельство. Мы знаем, как решать Баську, но, к сожалению, мы не в состоянии на практике применить наш способ решения. Суть дела в том, что возможны в игре ситуации (когда открываем игры), в которых пришлось бы строить очень много больших матриц, находить для них цену игры и сравнивать цены игры для выбора лучшей из альтернатив. А с такой работой даже при помощи небольшой вычислительной машины не справиться. Прежде всего не хватит памяти машины, так как для открывающего игру надо построить $12!/(4!)^3$ матриц (больше чем 10^4), причем количество элементов в матрице может достигать числа $(4!)^4$ (больше чем 10^5).

3. ПРИНЦИПЫ МАШИННОГО РЕШЕНИЯ ИГРЫ

Из-за трудностей теоретико-игрового решения игры на небольшой вычислительной машине автор принял другой принцип программного решения игры: возможно точно передать машине поведение опытного игрока. Таким образом поставлена была задача моделирования Васьки на УЦМ Урал-1 [7].

Программа моделирования была написана на машинном языке. Занимала 1116 (десятичное число) ячеек памяти. Работала в нескольких вариантах. Варианты определялись по ключам. Основные варианты были: сдача карт машиной (ключ 3 включен), сдача карт вне машины (ключ 3 выключен), игра за 4 игроков (ключ 4 включен), игра за 1 игрока (ключ 4 выключен), осторожное открытие игры Молодым (ключ 5 включен), агрессивное открытие игры Молодым (ключ 5 выключен), осторожное открытие раунда Старым (ключ 6 включен), агрессивное открытие раунда Старым (ключ 6 выключен). Программа была построена по следующей блок-схеме:

В блок-схеме приняты обозначения следующие:

G_m: игрок-машина;

G_{i_j}, G_{k_j}: игроки заменяемые машиной;

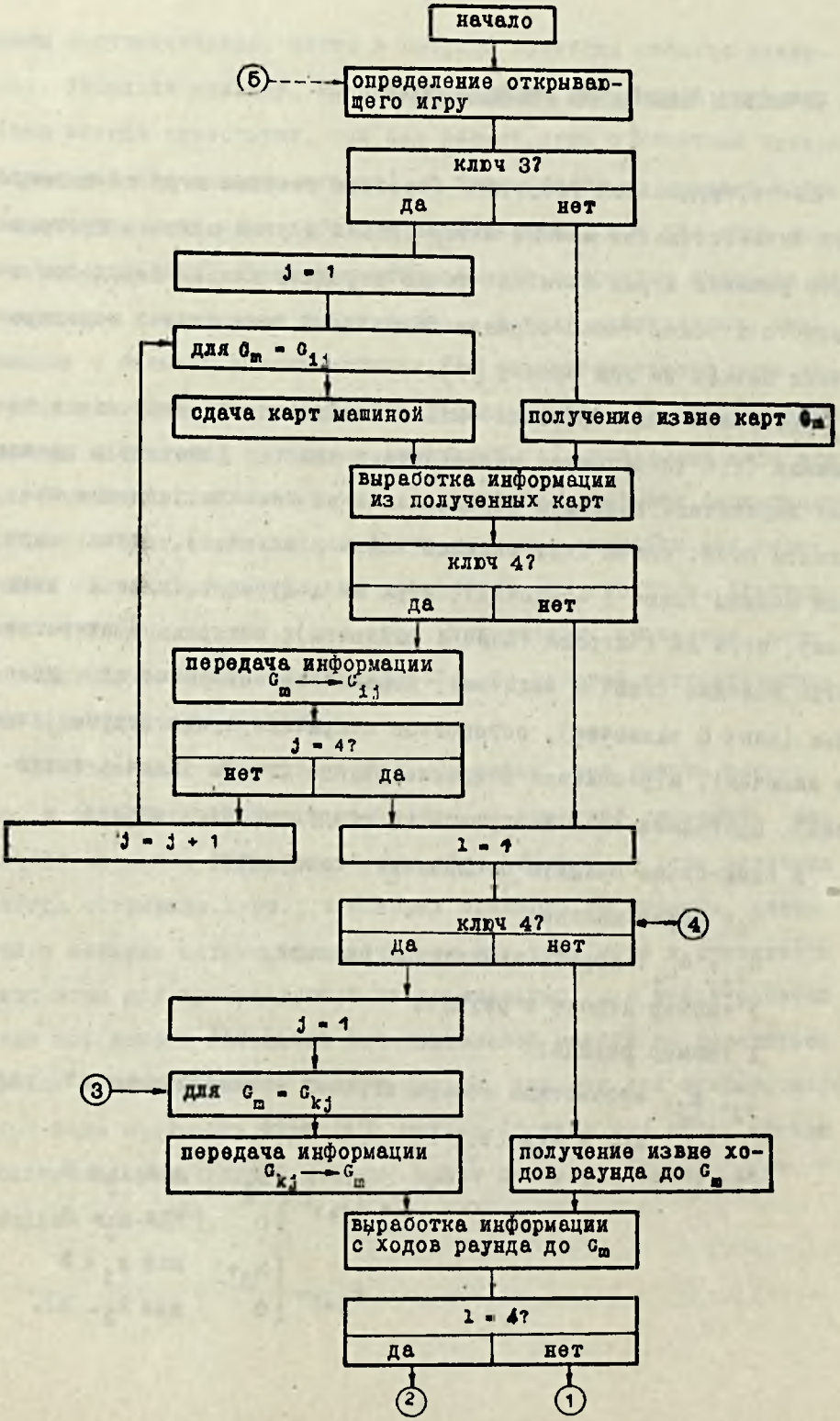
j: номер игрока в раунде;

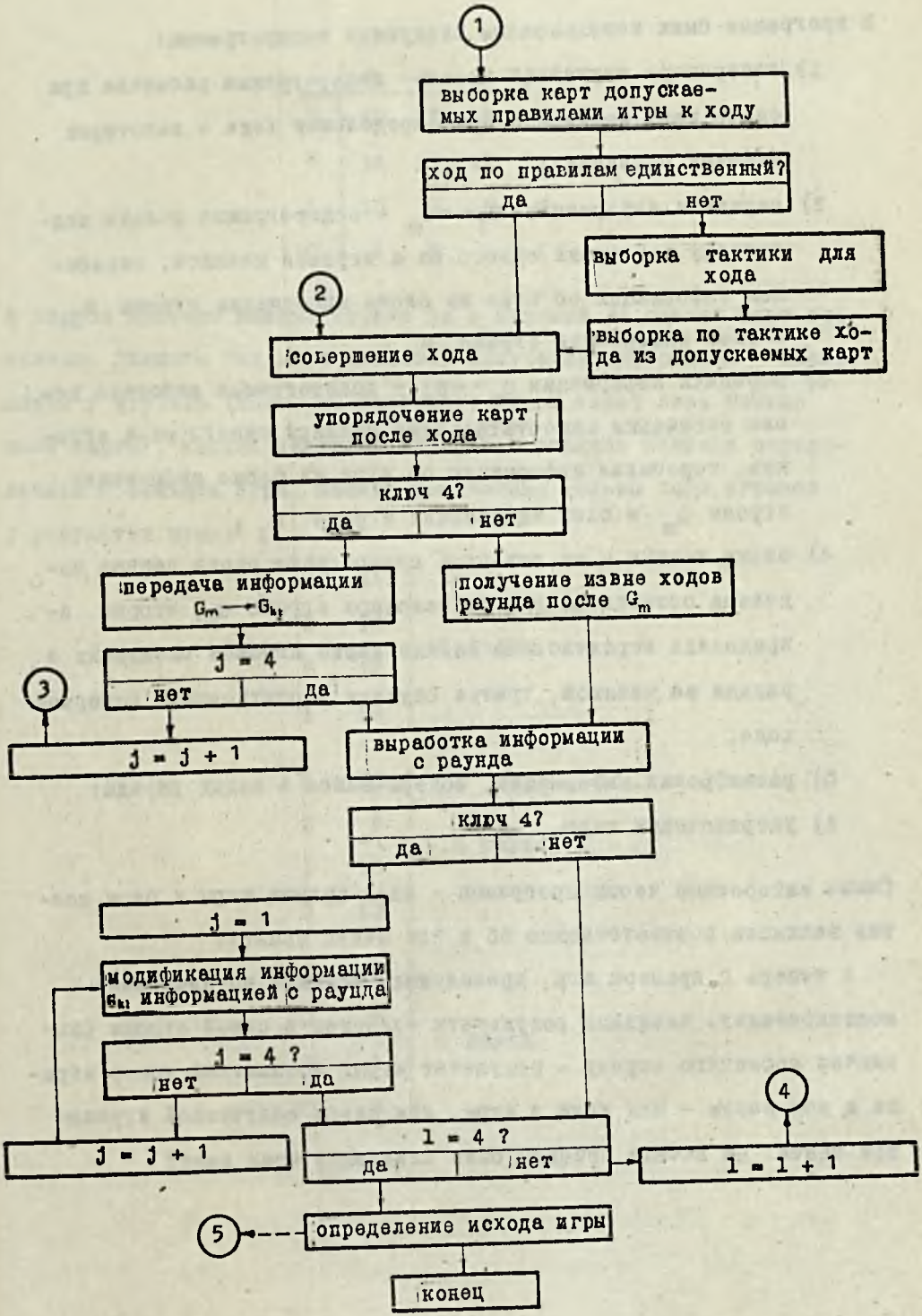
l: номер раунда;

i_j, k_j: абсолютные номера игроков - независимые от раундов и игр (i_j, k_j = 0, 1, 2, 3;

$$i_{j+1} = \begin{cases} i_j+1 & \text{для } i_j < 3 \\ 0 & \text{для } i_j = 3 \end{cases}$$

$$k_{j+1} = \begin{cases} k_j+1 & \text{для } k_j < 3 \\ 0 & \text{для } k_j = 3). \end{cases}$$


начало
(5)
определение открывающего игру
ключ 3?
да
нет
j = 1
для $G_m = G_{ij}$
сдача карт машиной
получение извне карт G_m
выработка информации из полученных карт
ключ 4?
да
нет
передача информации $G_m \rightarrow G_{ij}$
j = 4?
нет
да
j = j + 1
i = 1
ключ 4?
(4)
да
нет
j = 1
(3)
для $G_m = G_{kj}$
передача информации $G_{kj} \rightarrow G_m$
получение извне ходов раунда до G_m
выработка информации с ходов раунда до G_m
i = 4?
да
нет
(2)
(1)


1
выборка карт допускаемых правилами игры к ходу
ход по правилам единственный?
да
нет
выборка тактики для хода
2
выборка по тактике хода из допускаемых карт
совершение хода
упорядочение карт после хода
ключ 4?
да
нет
передача информации $G_m \rightarrow G_{k_i}$
получение извне ходов раунда после G_m
j = 4
нет
да
3
j = j + 1
выработка информации с раунда
ключ 4?
да
нет
j = 1
модификация информации G_{k_i} информацией с раунда
j = 4 ?
нет
да
j = j + 1
l = 4 ?
да
нет
4
l = l + 1
5
определение исхода игры
конец

В программе были использованы следующие подпрограммы:

1) генератора случайных чисел - подпрограмма работала при сдаче карт машиной и при определении хода в некоторых игровых ситуациях;
2) передачи информации $G_j \longrightarrow G_m$ - подпрограмма делала подготовку замещения одного из 4 игроков машиной, пересылая информацию об игре из блока информации игрока G_j в блок информации игрока G_m;
3) передачи информации $G_m \longrightarrow G_j$ - подпрограмма работала после окончания заместительства машиной одного из 4 игроков, пересылая информацию об игре из блока информации игрока G_m в блок информации игрока G_j;
4) блока тактик - из основных подпрограмм блока первая выделяла остающиеся в игре некозыри игрока G_m, вторая определяла вероятностно высшие карты игроков следующих в раунде за машиной, третья служила вероятностной выборкой хода;
5) расшифровки информации, содержащейся в ходах раунда;
6) упорядочения карт.

Самые интересные части программы - блок правил игры и блок тактик занимали соответственно 55 и 319 ячеек памяти.

А теперь 2 примера игр, проведенных машиной по программе моделирования. Машинные результаты содержат в одной строке (исключая последнюю строку - результат игры) абсолютный номер игрока и код карты - или хода в игре, или карты полученной игроком при сдаче. На печати приняты были следующие коды карт:

	♣	♠	♡	◇
D	20	17	16	15
W	14	13	12	11
A	10	7	6	5
10	4	3	2	1

В первом примере машина играла за 4 игроков со сдачей карт вне машины. Включен был ключ 4. После получения информации - какие карты у игроков (согласно принципу: игрок знает лишь только свои карты), машина замещала игроков в порядке очереди определенном правилами игры. Машина напечатала только ходы игроков и результат игры:

2	11	I раунд
3	14	
0	12	
1	16	
1	20	II раунд
2	6	
3	15	
0	13	
1	17	III раунд
2	7	
3	1	
0	3	

1	5
2	10
3	2
0	4

IУ раунд

11 64 } исход игры - победила коалиция Старих (игроки номер 1, 1, т.е. игрок G_1) с перевесом 64 (восьмиричное число) очков больше половины всевозможных очков.

Во втором примере машина играла за 4 игроков со сдачей карт машиной. Включены были ключи 3 и 4. Машина напечатала карты игроков, ходы игроков и результат игры:

3	1
3	5
3	11
3	15

карты игрока G_3

0	13
0	14
0	16
0	17

карты игрока G_0

1	2
1	3
1	7
1	10

карты игрока G_1

2	4
2	6
2	12
2	20

карты игрока G_2

3	1	I раунд
0	13	
1	2	
2	20	
2	4	II раунд
3	11	
0	17	
1	10	
0	14	III раунд
1	3	
2	12	
3	15	
3	5	IУ раунд
0	16	
1	7	
2	6	
02	43	исход игры - победила коалиция

Старых (игроки G_0 и G_2) с перевесом 43 очков.

4. ВОЗМОЖНЫЕ ПРИМЕНЕНИЯ МЕТОДА МАШИННОЙ СИМУЛИРОВКИ

Используя программу моделирования, автор провел опыт сравнения тактик осторожного и агрессивного открытия игры Молодым (ключ 5), повторяя реализацию программы многократно. Показалось, что тактики эти статистически равносильные. Таким-же

образом сравнено тактики осторожного и агрессивного открытия раунда Старым (ключ 6). Значит, моделируя процесс на вычислительной машине, мы сможем довольно быстро получить статистическую оценку различных вариантов процесса.

В случае невозможности оптимального решения какой-то задачи (из-за незнакомства оптимального алгоритма или из-за невозможности реализации его - что случилось автору) техника машинной симулировки служит возможностью совершенствования предлагаемых неоптимальных алгоритмов.

Интересно отметить, что, применяя симулировку отличную от моделирования, можно получить алгоритм общий совершенно различным физически процессам. Благодаря тому, можно с помощью машинной симулировки решать проблемы по алгоритмам, которых действие оправдалось на других проблемах.

Надо подчеркнуть, что машинная симулировка может оказаться надежной техникой решения проблем везде там, где отсутствуют применимые математические алгоритмы (смотри нп. [6]).

Литература

[1] БЕНЕНБЛУСТ Г.Ф.: Теория игр. Математическое просвещение, 4, 1959.

[2] БОТВИННИК М.М.: Алгоритм игры в шахматы, Наука, 1968.

[3] ВЕНТЦЕЛЬ Е.О.: Элементы теории игр, Физматгиз, 1959.

[4] ВОРОБЬЕВ Н.Н.: О коалиционных играх, Доклады АН СССР, 124, 2, 1959.

[5] ВОРОБЬЕВ Н.Н.; РОМАНОВСКИЙ И.В.: Игры с запрещенными ситуациями, Вестник ЛГУ, 2, 7, 1959.

[6] ЗАРИПОВ Р.Х.: Кибернетика и музыка, Наука, 1971.

[7] ЗЕЛИНСКИЙ Г.Б.: Моделирование игровых ситуаций и решение игровых проблем на УЦМ Урал-1, Дипломная работа, ЛГУ, 1963.

[8] ЛЬЮС Р.Д., РАЙФА Х.: Игры и решения, ИИЛ, 1961.

[9] ПЕРВИН Ю.А.; Об алгоритмизации и программировании игры в домино, Проблемы кибернетики, 3, 1960.

[10] ZIELIŃSKI G.: Algorytmizacja procesu organizowania punktów dyskretnej przestrzeni dźwiękowej, Praca doktorska, PW, 1970.

MODELOWANIE GRY NA MASZYNIE LICZĄCEJ

Streszczenie

Praca opisuje modelowanie gry na maszynie liczącej. Modelowanie zrealizowane zostało na małej maszynie Ural-1. Gra towarzyska, którą zamodelowano, jest grą 4 osób, z niepełną informacją, o 2 koalicjach wyznaczanych losowo.

Na wstępie podano charakterystykę maszynowych technik modelowania i symulacji. W rozdziale 1 opisano reguły gry. Rozdział 2 stanowi analizę gry na podstawie teorii gier. Maszynowy program modelowania omówiono w rozdziale 3, podając zarówno schemat programu, jak i przykłady wyników maszynowych z objaśnieniami. W rozdziale 4 wskazano na możliwości wykorzystania techniki komputerowej symulacji.

COMPUTER MODELLING OF A GAME

Summary

The paper deals with computer modelling of a game. The modelled game is a 4-person game with non-complete information and with two random coalitions. The game rules are given in the chapter 1. The chapter 2 is devoted to a game-theoretical analysis of the considered game. A modelling computer program and some examples of computer results are discussed in the chapter 3. The chapter 4 contains a review of computer simulation applications.

ALGORYTMY, X, N°18, 1973, 79-103

681.322.004.14:51.004.14:
378:371.214

PROBLEMY AUTOMATYCZNEGO UKŁADANIA ROZKŁADÓW ZAJĘĆ DLA SZKÓŁ WYŻSZYCH

Marek KUBALE
Instytut Informatyki
Politechniki Gdańskiej

Pracę złożono 5.07.1972

Przedmiotem pracy jest analiza modelu matematycznego rozkładu zajęć dydaktycznych uczelni wyższej oraz przegląd metod algorytmizacji układania rozkładów zajęć. Rozważania ogólne poparte są przykładem systemu programów zrealizowanych w Instytucie Informatyki Politechniki Gdańskiej.

WSTĘP

Dotychczasowe prymitywne i pracochłonne metody sporządzania rozkładów zajęć absorbują dużą ilość czasu pracowników naukowych. Dość wspomnieć, że na Politechnice Gdańskiej układa się rocznie kilkadziesiąt planów zajęć, zaś jeden tylko plan wymaga około miesiąca pracy jednej osoby. Powyższe czynniki powodują, że w wielu krajach prowadzi się obecnie badania nad zastosowaniem maszyn matematycznych do tej pracy. Dwa zadania są przedmiotem tych badań:

1. układanie rozkładów zajęć dla szkół
2. układanie rozkładów zajęć dla uczelni.

Zadanie pierwsze jest w zasadzie prostsze od zadania drugiego, któremu poświęcona jest niniejsza praca. Stanowi ona wynik dwuletnich doświadczeń autora nad wprowadzeniem systemu programów na Wydziale Elektroniki Politechniki Gdańskiej. Pra-

ca składa się z dwóch części. W pierwszej przedstawiono problem układania planów zajęć w przypadku ogólnym oraz dokonano przeglądu teoretycznych metod układania rozkładów zajęć. W drugiej części opisano system programów dla maszyny ZAM 41α oraz przedstawiono wyniki jego działania.

1. MODEL MATEMATYCZNY ROZKŁADU ZAJĘĆ

W punkcie tym przedstawimy zaproponowany przez Gotlieba [1] model boolowski rozkładu zajęć adaptując go do potrzeb wydziału szkoły wyższej.

1.1. Założenia początkowe

Przystępując do układania rozkładu zajęć dla wydziału wyższej uczelni mamy daną uporządkowaną piątkę $< W,G,H,S,P >$, gdzie:

$W = \{w_i\}$	$i = 1, \ldots, w$	zbiór wykładowców
$G = \{g_j\}$	$j = 1, \ldots, g$	zbiór grup
$H = \{h_k\}$	$k = 1, \ldots, h$	zbiór kolejnych godzin
$S = \{s_l\}$	$l = 1, \ldots, s$	zbiór sal
$P = \{p_r\}$	$r = 1, \ldots, p$	zbiór przedmiotów

oraz zbiór założeń w postaci pary macierzy $< Q_o, D_o >$ o elementach będących liczbami całkowitymi nieujemnymi:

$Q_o = [q_{jr}]$	$j = 1, \ldots, g;\ r = 1, \ldots, p$	programu studiów
$D_o = [d_{ir}]$	$i = 1, \ldots, w;\ r = 1, \ldots, p$	projektu dydaktycznego

Elementy q_{jr} przyjmują wartość 1, jeżeli grupa g_j ma mieć zajęcia z przedmiotu p_r lub 0, jeżeli program studiów nie przewiduje zajęć dla g_j z przedmiotu p_r. Elementy d_{ir} informują z iloma grupami wykładowca w_i ma mieć zajęcia z przedmiotu p_r. Oczywiście pomiędzy macierzami Q_o i D_o musi

zachodzić związek taki, że zapotrzebowanie na wykładowców danego przedmiotu musi odpowiadać liczbie grup, z którymi prowadzący mają te zajęcia poprowadzić, czyli

$$\sum_{i=1}^{w} d_{ir} = \sum_{j=1}^{g} q_{jr}$$

Ponadto, prócz powyższych zbiorów mamy daną funkcję n: P⟶N (zbiór liczb naturalnych), której wartość $n(p_r) = n_r$ określa ile kolejnych godzin lekcyjnych ma trwać p_r.

1.2. Ograniczenia

Jak wiadomo niektóre sale są zbyt małe, aby można prowadzić w nich zajęcia z pewnymi grupami, a ponadto niektórzy wykładowcy niechętnie godza się na zajęcia w pewnych salach, np. daleko położonych. Zajęcia z przedmiotów laboratoryjnych mogą odbywać się tylko w laboratoriach. Powyższe czynniki powodują, że problem sal jest problemem kluczowym w warunkach szkoły wyższej.

Powyższe ograniczenia dobrze opisują macierze trójwskaźnikowe: $M = [m_{jlr}]$ $j = 1,\ldots,g$; $l = 1,\ldots,s$; $r = 1, \ldots, p$, której elementy przyjmują wartość 1, jeżeli zajęcia z p_r dla g_j mogą odbywać się w s_l, zaś 0 w przeciwnym przypadku oraz macierz $N = [n_{ilr}]$ $i = 1, \ldots, w$; $l = 1, \ldots, s$; $r = 1, \ldots, p$, której elementy przyjmują wartość 1, jeżeli w opinii w_i sala s_l jest odpowiednia na zajęcia z p_r, zaś 0 w przypadku przeciwnym.

Poniżej wprowadzimy pojęcia macierzy rozporządzalności zbiorów W, G i S.

Definicja 1

Macierz $A = [a_{ik}]$ $i = 1, \ldots, w$; $k = 1, \ldots, h$ nazywać będziemy macierzą rozporządzalności wykładowców wtedy i tylko wtedy, gdy

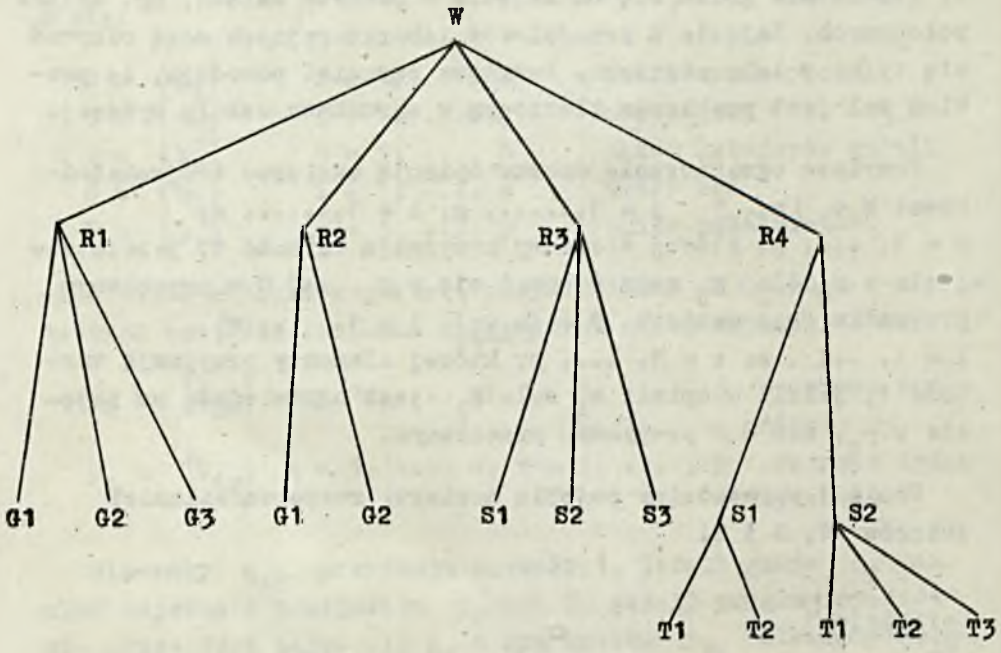
$$a_{ik} = \begin{cases} 1, \text{ jeżeli } w_i \text{ jest wolny w } h_k, \\ 0, \text{ jeżeli } w_i \text{ jest zajęty w } h_k. \end{cases}$$

Analogicznie definiujemy macierz rozporządzalności grup $B = [b_{jk}]$ $j = 1, \ldots, g$; $k = 1, \ldots, h$ oraz macierz rozporządzalności sal $C = [c_{lk}]$ $l = 1, \ldots, s$; $k = 1, \ldots, h$.

1.3. Drzewo wydziału

Jak wiemy wydział wyższej uczelni składa się z wielu grup, przy czym niektóre z nich mogą mieć pewne zajęcia wspólne. Oznacza to, że zbiór G stanowi półstrukturę górną, której w sposób jednoznaczny można przyporządkować drzewo wydziału.

Za Kreczmarem [4] podamy notację nawiasową drzewa wydziału.



Rys. 1. Przykład drzewa wydziału

Oznaczenia: W - wydział, R - rok, G - grupa dziekańska, S - specjalność, T - specjalizacja

Będziemy mówić, że słowo: W(R1(G1, G2, G3) R2(G1, G2) R3(S1, S2, S3)R4(S1 (T1, T2)S2(T1, T2, T3))) jest notacją nawiasową drzewa wydziału.

Drzewo wydziału możemy opisać również relacją dwuczłonową $\varrho \subset G \times G$, określoną następująco:

$$g_i \varrho g_j \Longleftrightarrow (g_i \subset g_j) \vee (g_j \subset g_i)$$

co oznacza, że dwie grupy g_i i g_j (traktowane tutaj jako podzbiory G) są w relacji ϱ wtedy i tylko wtedy, gdy g_i jest podgrupą g_j np. specjalizacją danej specjalności lub też g_j jest podgrupą g_i. Dla dalszych celów określimy zbiór $G^{j1} = \{j: g_j \varrho g_{j1}\}$ jako zbiór tych grup, które nie mogą mieć jednocześnie zajęć z g_{j1}.

1.4. Definicja dopuszczalnego rozkładu zajęć

Definicja 2

Rozkładem zajęć będziemy nazywali czterowskaźnikową macierz bcolowską $R = [r_{ijkl}]$ $i = 1, \ldots, w$; $j = 1, \ldots, g$; $k = 1, \ldots, h$; $l = 1, \ldots, s$, gdzie

$$r_{ijkl} = \begin{cases} 1, \text{ jeżeli } w_i \text{ ma zajęcia z } g_j \text{ o godz. } h_k \text{ w } s_l, \\ 0 \text{ w pozostałych przypadkach.} \end{cases}$$

Jednakże tak zdefiniowany rozkład zajęć dopuszcza możliwość kolizji.

Definicja 3

Dopuszczalnym rozkładem zajęć będziemy nazywali rozkład zajęć, który spełnia następujące warunki:

$$\sum_{i=1}^{g} \sum_{l=1}^{s} r_{ijkl} \leqslant 1 \qquad (1)$$

$$\sum_{i=1}^{w} \sum_{j=1}^{g} r_{ijkl} \leqslant 1 \qquad (2)$$

$$\sum_{i=1}^{w} \sum_{l=1}^{s} \sum_{j \in G^{j_1}} r_{ijkl} \leqslant 1 \text{ dla } j_1 = 1, \ldots, q \qquad (3)$$

$$\sum_{i=1}^{w} \sum_{k=1}^{h} \sum_{l=1}^{s} r_{ijkl} = \sum_{r=1}^{p} n_r \cdot d_{jr} \qquad (4)$$

$$\sum_{j=1}^{g} \sum_{k=1}^{h} \sum_{l=1}^{s} r_{ijkl} = \sum_{r=1}^{p} n_r \cdot d_{ir} \qquad (5)$$

$$\text{jeżeli } r_{ijkl} < r_{ijk+1l} \text{ to } \sum_{k_1=k}^{k+n_r} r_{ijk_1l} = n_r \qquad (6)$$

$$r_{ijkl} \leqslant m_{jlr} \cdot n_{ilr} \qquad (7)$$

$$r_{ijkl} \leqslant a_{ik} \cdot b_{jk} \cdot c_{lk} \qquad (8)$$

dla każdego $i = 1, \ldots, w$; $j = 1, \ldots, g$;
$k = 1, \ldots, h$; $l = 1, \ldots, s$.

Jest oczywiste, że w_i w h_k może mieć zajęcia tylko z jedną grupą i tylko w jednej sali. Oznacza to, że suma elementów rozkładu zajęć po wszystkich j oraz l winna być równa 0 lub 1, co zapisujemy używając symbolu $\leqslant 1$. Analogiczne rozumowanie przeprowadzamy dla s_l otrzymując warunek (2). Warunek (3) jest rozszerzeniem warunków (1) i (2) na warunek, aby żadna podgrupa nie miała zajęć równocześnie z g_j. Warunki (4) i (5) mówią, że godzinowe obciążenia g_j oraz w_i winny być zgodne z $< Q_o, D_o >$. Warunek (6) jest warunkiem ciągłości zajęć. Warunki (7) i (8) mówią, że zajęcia możemy planować tylko w tych salach oraz terminach, które są wolne i odpowiadają zarówno g_j jak i w_i.

Jak wynika z definicji 3 rozkład zajęć jest macierzą boolowską, którą mamy zbudować wychodząc ze zbioru założeń

$<Q_o, D_o>$, funkcji n, drzewa wydziału oraz ograniczeń czasowych A, B, C i pomieszczeniowych M, N. Rozkład dopuszczalny będziemy konstruowali poczynając od macierzy R_o, której wszystkie elementy są zerami, a następnie wypełniając ją jedynkami w taki sposób, aby ostatecznie macierz R spełniała warunki rozkładu dopuszczalnego. Po wstawieniu serii n_r jedynek do macierzy R musimy zmniejszyć odpowiedni element q_{jr} macierzy Q i d_{ir} macierzy D o jeden. W ten sposób otrzymujemy ciąg R_o, R_1,....., R_t,..... rozkładów zajęć, który winien w skończonej liczbie kroków dojść do rozkładu dopuszczalnego, oraz ciąg $<Q_o, D_o>$, $<Q_1, D_1>$,..., $< Q_t, D_t >$,... macierzy założeń, które winny dążyć do macierzy zerowych. Na ogół pierwsze przybliżenie R_1 będzie narzucone z góry, ponieważ zajęcia z pewnych przedmiotów będą musiały odbywać się w określonych godzinach i salach.

Po wprowadzeniu pewnych pojęć pomocniczych można podać nierówności jakie musi spełniać macierz R_t, aby mógł istnieć rozkład dopuszczalny. Nierówności te, analogiczne do nierówności Halla [3], są warunkami koniecznymi istnienia rozkładu dopuszczalnego w sensie definicji 3.

1.5. Przegląd metod układania rozkładów zajęć

Jak wspominaliśmy w p. 1.4 rozkład zajęć będziemy konstruowali poczynając od macierzy R_o lub R_1, a następnie wypełniając ją jedynkami tak, aby ostatecznie spełniała warunki (1) ÷ (8). W zależności od sposobu wypełniania macierzy R jedynkami wyróżniamy metody deterministyczne i probabilistyczne układania rozkładów zajęć.

Na wstępie przez $Z = \{z_m\}$ $m = 1,....., z$ oznaczymy zbiór zajęć; każde zajęcie interpretować będziemy jako uporządkowaną trójkę $<w_i, g_j, p_r>$, przy czym spełnione muszą być warunki:

$$<w_i, g_j, p_r> \in Z \Rightarrow (d_{ir} \cdot q_{jr} > 0)$$

$$<w_i, g_j, p_r> \neq <w_a, g_b, p_c> \Rightarrow (j \neq b) \vee (r \neq c)$$

1.5.1. Metody deterministyczne

Metoda heurystyczna

Metoda ta [4, 6, 12, 13, 17] polega na wypełnianiu początkowej macierzy R_0 lub R_1 kolejno według pewnych kryteriów. W tym celu w każdym kroku algorytmu dla każdego niezaplanowanego zajęcia z_m oblicza się wielkości P/N, P - N itp. mające sens swobody, gdzie:

P - liczba godzin rozporządzalnych
N - liczba godzin, które winny być zaplanowane

W trakcie obliczeń staramy się zachować te wielkości możliwie jak największe. Jeżeli w kolejnym kroku algorytmu dla pewnego z_m swoboda osiągnęła wartość minimalną nie doprowadzając do rozkładu dopuszczalnego, nie oznacza to, że rozkład dopuszczalny nie istnieje. W tym przypadku należy cofnąć się do kroku poprzedniego, odtworzyć poprzedni stan macierzy R i $<Q, D>$ i próbować ułożyć rozkład zajęć poczynając od innego zajęcia, dla którego również swoboda osiąga minimum.

Metoda heurystyczna jest metodą najszybszą, a dzięki temu najszerzej stosowaną. Do jej wad należy zaliczyć możliwość uzyskania tylko jednego rozwiązania (o ile ono istnieje dla danych założeń) oraz brak pewności czy rozwiązanie takie w ogóle nie istnieje, jeżeli algorytm nie doprowadził do rozkładu dopuszczalnego.

Metoda kolorowania grafów

Powyższych wad nie posiada metoda kolorowania grafów. Polega ona na sprowadzeniu zagadnienia układania rozkładu zajęć do problemu kolorowania grafu. W tym celu przyjmuje się bardziej ogólną definicję dopuszczalnego rozkładu zajęć wprowadzając rodzinę zbiorów $F = \{z_1, z_2, \ldots, z_m, h_1, \ldots, h_h\}$ oraz relację binarną $\pi \subset F \times F$ wzajemnego wykluczania się zajęć i godzin. Graf $E = <F, \pi>$, gdzie F jest zbiorem wierzchołków, a π zbiorem krawędzi, nazywamy grafem rozkładu zajęć. Kreczmar [5] udowodnił następujące twierdzenie

Twierdzenie Kreczmara

Rozkład zajęć istnieje wtedy i tylko wtedy, gdy liczba chromatyczna grafu E jest równa liczbie elementów zbioru H (E jest h-chromatyczny).

Algorytm składa się z dwóch etapów. W pierwszym otrzymujemy wszystkie rozkłady zajęć. Etap drugi zajmuje się wyznaczaniem sal. W sposób analogiczny jak w pierwszym etapie problem ten sprowadza się do kolorowania grafu.

Metoda kolorowania grafów daje nam wszystkie rozwiązania, tym samym umożliwia eliminację rozwiązań niewygodnych. W przypadku praktycznym, gdy okaże się, że rozkład zajęć nie istnieje, pojawia się interesujący problem: jak należy przekształcić założenia zadania, aby przy minimalnej liczbie zmian uzyskać rozwiązanie. Pytanie to postawione przez Kreczmara [5] jest związane z pojęciem grafu krytycznego.

Efektywność tej metody jest ściśle związana z efektywnością algorytmów kolorowania grafu, dla których czas działania proporcjonalny jest do 2^n. W przypadku dużego wydziału algorytm wymaga odwoływania się do pamięci zewnętrznej, co bardzo niekorzystnie wpływa na czas obliczeń. Jest to istotna wada tej metody.

Metoda kombinatoryczna

Metoda ta [7] polega na wypełnianiu jedynkami początkowej macierzy R_0 zgodnie z założeniami $<Q_0, D_0>$, a następnie przesuwaniu jedynek kolejno we wszystkich możliwych kombinacjach. Liczba otrzymywanych rozwiązań zależy od tego, czy proces ten prowadzimy do momentu otrzymania pierwszego rozkładu dopuszczalnego, czy też kontynuujemy go do momentu otrzymania rozkładu początkowego.

Najbardziej znaną metodą kombinatoryczną pozwalającą na skrócenie czasu obliczeń dzięki omijaniu serii rozkładów nie-

dopuszczalnych jest metoda kombinacji zwartych [1, 2] oparta na twierdzeniu Halla [3]. Twierdzenie to podaje warunki konieczne istnienia rozkładu zajęć, ale nie określa warunków dostatecznych. Gotlieb i Csima [2] podali kontrprzykład o wymiarach 3 x 3 x 3, w którym rozkład zajęć nie istnieje mimo spełnienia warunków Halla. Czas działania metody proporcjonalny jest do 2^n. Lions [16] zaproponował modyfikację algorytmu, która daje czas proporcjonalny do n^2.

Poważną wadą metody jest mała możliwość zastosowania jej w przypadku szkoły wyższej oraz brak pewności uzyskania rozwiązania.

Programowanie liniowe

W przypadku wyższej uczelni zadanie traktuje się jako sytuację konfliktową i rozwiązuje za pomocą metod programowania liniowego [8]. Istnieją również próby zastosowania programowania całkowitoliczbowego [9]. Jak się wydaje istnieje również możliwość zastosowania programowania dwuwartościowego do rozwiązania tego problemu.

Oprócz metod wyżej wymienionych istnieją próby zastosowania innych teorii do układania harmonogramów zajęć dydaktycznych, jak np. programowania pseudoboolowskiego [14], czy też metody przepływów w sieciach [15]. Jednakże autorowi nie są znane, jak dotychczas, realizacje maszynowe tych metod w zastosowaniu do szkół wyższych, ich efektywność itp.

1.5.2. Metody probabilistyczne

Metoda "orzeł- reszka"

Metoda ta [10] polega na wypełnianiu jedynkami macierzy R w dwu krokach:

1. generowanie kolejnego elementu w sposób losowy,

2. sprawdzenie czy aktualny stan R nie jest sprzeczny z warunkami (1) ÷ (8). Jeśli jest sprzeczny, ustalony ostatnio element usuwamy i przechodzimy do punktu 1.

Metoda ta wymaga stosowania generatora liczb losowych. Jest ona mniej efektywna niż metody deterministyczne na skutek stosowania losowego błądzenia i nie znalazła szerszego zastosowania.

Metoda mieszana

Lepsze efekty daje metoda mieszana. Polega ona na otrzymaniu rozkładu dopuszczalnego w dwu etapach:

1. wygenerowanie w sposób losowy dowolnej macierzy R o określonej przez $<Q_o, D_o>$ i funkcję n liczbie elementów = 1,
2. permutowanie według pewnych kryteriów elementów macierzy R tak, by spełniała ona wszystkie warunki rozkładu dopuszczalnego.

W etapie drugim można zastosować metodę minimum sprzeczności Lazaka [11]. Algorytm ten w każdym kroku przesuwa serię jedynek odpowiadającą danemu z_m, przy czym każdemu nowemu położeniu z_m odpowiada tzw. indeks sprzeczności i_m będący sumą tych elementów, które należy usunąć, aby dane położenie z_m uczynić dopuszczalnym. Lazak proponuje w każdym kroku wybierać z_m, dla którego i_m osiągnie minimum, gdyż w ten sposób najszybciej dochodzi się do rozkładu dopuszczalnego.

2. OPIS REALIZACJI SYSTEMU PROGRAMÓW DLA EMC ZAM 41α

Poniżej podamy opis programów zrealizowanych w Instytucie Informatyki Politechniki Gdańskiej umożliwiających automatyzację układania rozkładów zajęć dla Wydziału Elektroniki Politechniki Gdańskiej. Programy napisane zostały w języku PJP dla maszyny ZAM 41α.

2.1. Opis systemu programów

W skład systemu wchodzą następujące programy:

1. wczytywanie danych wejściowych,
2. algorytm układania rozkładu zajęć,
3. testowanie wyników,
4. drukowanie wyników,
5. drukowanie listy wykładowców,
6. drukowanie listy grup,
7. drukowanie listy sal,
8. drukowanie listy przedmiotów,
9. drukowanie instrukcji,
10. korekta taśmy perforowanej.

Programy wraz z danymi zostają wczytane do pamięci bębnowej 32 k. Wyboru programu dokonuje operator, wypisując na monitorze instrukcję polecającą przesłać żądany program do pamięci operacyjnej o pojemności 12 k.

Program drukowania listy wykładowców wypisuje na drukarce wierszowej stopnie i nazwiska wykładowców oraz przypisuje im numery. Program sporządza po dwie listy dla pracowników Wydziału oraz pracowników z zewnątrz. Listy te służą jako formularze do pisania danych.

Program drukujący listę grup sporządza siedem list po jednej dla każdego instytutu i cztery dla pracowników z zewnątrz. (Na Wydziale znajdują się trzy instytuty, które kształcą studentów na trzech kursach: magisterskim, inżynierskim i wieczorowym oraz na studiach doktoranckich i podyplomowych. Rozkłady zajęć dla studiów mogą być również układane przez maszynę, o ile ich zajęcia odbywają się cyklem tygodniowym). Każda lista składa się z trzech części, po jednej na każdy kurs. Każdą jej część tworzą nazwy grup i przyporządkowane im numery w kolejności od roku pierwszego do ostatniego. Każdy kurs ma osobną numerację.

Analogicznie działają programy drukowania list przedmiotów i sal.

Program drukujący instrukcję sporządza opis czynności wykonywanych przez wykładowców w celu zakodowania danych dla maszyny.

Pliki dokumentów składające się z list wykładowców, grup, sal, przedmiotów i instrukcji są przesyłane do poszczególnych instytutów. Pracownicy wypełniają formularze zgodnie z instrukcją, umieszczając na nich informacje o zajęciach i czasie zastrzeżonym. Informacje te z kolei wypisywane są na taśmie perforowanej, która stanowi dane dla maszyny.

Po zakończeniu algorytmu uruchamiany jest program testujący wyniki. Sprawdza on czy otrzymany rozkład jest dopuszczalny. O ile tak jest, inicjuje program drukowania harmonogramu.

W dalszej części zajmiemy się wyłącznie programem realizującym algorytm układania rozkładu zajęć oraz postacią danych wejściowych i wyników.

2.2. Algorytm

Poniżej opiszemy algorytm heurystyczny, który został wybrany do realizacji zadania. Na wstępie wprowadzimy pojęcia pomocnicze.

Przez $W_r = \{i: d_{ir} > 0\}$ oznaczymy zbiór wykładowców przedmiotu p_r, przez $G_r = \{j: q_{jr} > 0\}$ zbiór grup mających mieć zajęcia z p_r, następnie przez $S_r^{ij} = \{l: n_{ilr} \cdot m_{jlr} > 0\}$ zbiór sal dogodnych dla w_i i g_j na p_r, wreszcie przez $H_k^n = \{h_k, h_{k+1}, \ldots, h_{k+n-1}\}$ interwał n kolejnych godzin, począwszy od k-tej.

Definicja 4

Wektor $Y_r^{ij} = [y_{rk}^{ij}]$ $i \in W_r$, $j \in G_r$, $k \in H$ będziemy nazywali wektorem dopuszczalnych położeń p_r dla w_i i g_j wtedy i tylko wtedy, gdy

$$y_{rk}^{ij} = \begin{cases} 1, \text{ jeżeli } w_i, g_j \text{ i odpowiadająca im } s_l \text{ są} \\ \text{wolne w } H_k^n r, \\ 0 \text{ w pozostałych przypadkach.} \end{cases}$$

Dla każdego elementu $z_m \in Z$ musimy ustalić tyle kolejnych godzin, w których w_i, g_j oraz s_l są jednocześnie wolne, ile wynosi n_r. Po znalezieniu takiego $H_k^n r$ musimy wprowadzić zmiany w macierzach rozporządzalności A, B i C, a następnie zmniejszyć d_{ir} i q_{jr} o jeden. Przypuśćmy, że mamy wprowadzić do rozkładu kolejne zajęcia. Nasuwają się następujące pytania:

1. od którego elementu z_m winniśmy tę procedurę zacząć,
2. jeżeli wybraliśmy już z_m, to którą salę i w jakim terminie należy mu przydzielić.

Definicja 5

Swobodą zajęcia z_m, $m = 1,\ldots, z$ nazwiemy liczbę całkowitą nieujemną s_m zdefiniowaną równością

$$s_m = \sum_{k=1}^{h} y_{rk}^{ij}$$

Oznaczmy przez $\bar{Z} = \{z_m : s_m = 0\}$ zbiór zajęć niezaplanowanych. Wybieramy to z_α, dla którego swoboda osiąga wartość minimalną, gdyż wybór innego z_m, $m \neq \alpha$ mógłby nie doprowadzić do rozkładu dopuszczalnego. Jeżeli $s_\alpha = 0$, to z_α usuwamy z Z wpisując do $\bar{Z}$ i próbujemy zaplanować pozostałe zajęcia.

Postarajmy się zatem ustalić termin dla z_α oraz przylić mu salę.

Kryterium wyboru terminu i sali

Jeżeli wartość s_α jest większa od pewnej wartości progowej p, wybór terminu i sali nie jest tak istotny jak w przypadku przeciwnym i zdajemy się tutaj na zasadę wyboru najmniej-

szej sali i najwcześniejszej godziny. Załóżmy zatem, że $s_\alpha \leqslant p$. Oznaczmy przez $H_r^{ij} = \{k : y_{rk}^{ij} = 1\}$ zbiór wszystkich godzin, które mogą stanowić pierwszą godzinę zajęć dla $\langle w_i, g_j, p_r \rangle$, zaś przez $Z_p = \{z_m : s_m \leqslant p\}$ zbiór wszystkich zajęć, dla których swobody nie przekraczają wartości progowej p.

Definicja 6

Swobodą układu w punkcie (k, l) dla z_i nazwiemy liczbę

$$S^i(k, l) = \prod_{\substack{m=1 \\ m \neq i}}^{z_p} \bar{s}_m$$

gdzie z_p - liczba elementów zbioru Z_p, zaś $\bar{s}_m$ wartości swobód, jakie byłyby gdyby do rozkładu wstawić z_i o h_k w s_l.

W każdym kroku wybieramy ten termin i tę skalę, dla których $S^\alpha(k, l)$ osiąga maksimum, gdyż wówczas możliwości wstawienia pozostałych zajęć zmniejszają się minimalnie.

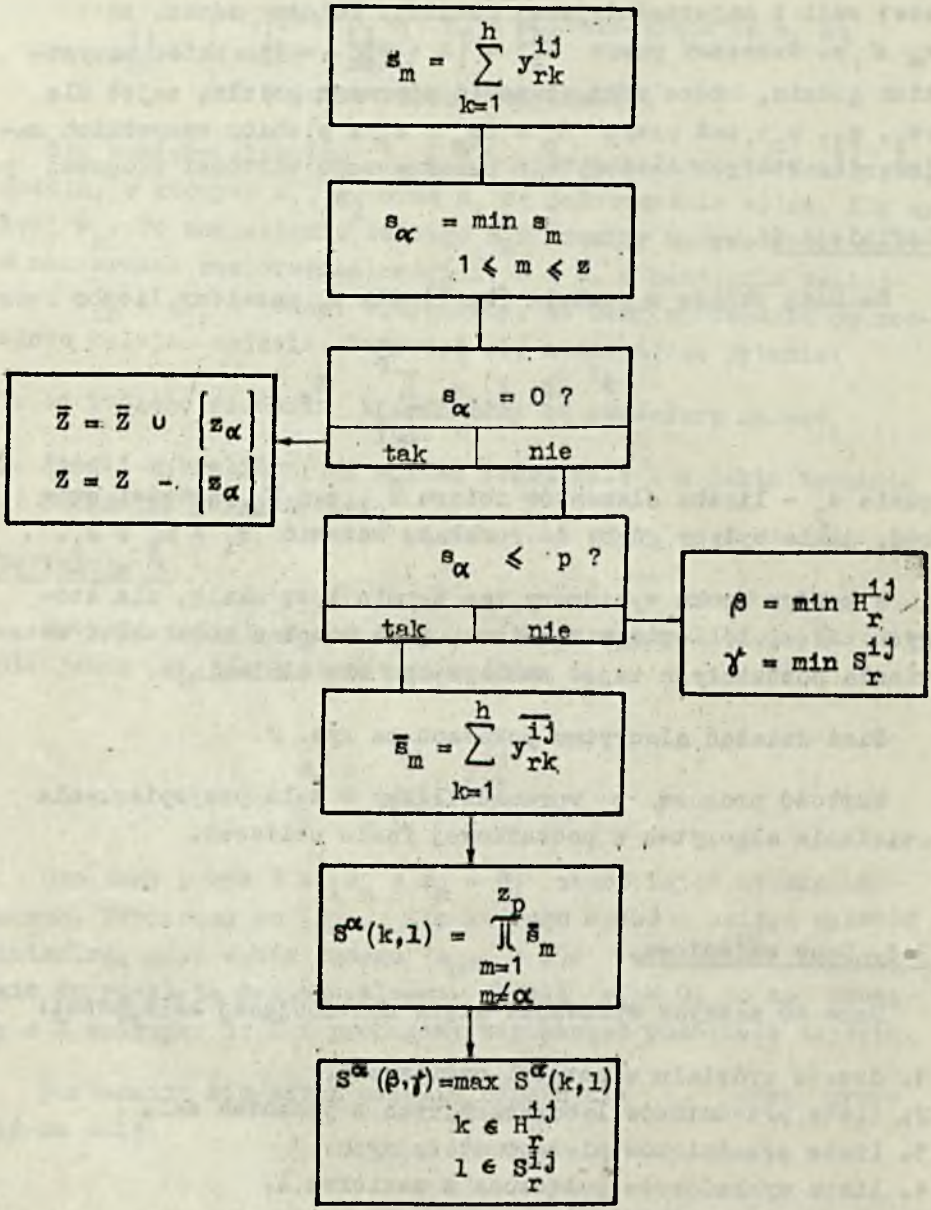
Sieć działań algorytmu pokazano na rys. 2.

Wartość progową p wprowadziliśmy w celu przyspieszenia działania algorytmu w początkowej fazie obliczeń.

2.3. Dane wejściowe

Dane do maszyny wprowadza się w następującej kolejności:

1. drzewo wydziału w notacji nawiasowej,
2. lista przedmiotów laboratoryjnych z podaniem sal,
3. lista przedmiotów nielaboratoryjnych,
4. lista wykładowców połączona z macierzą A,
5. lista grup połączona z macierzą B,
6. lista sal połączona z macierzą C,
7. numery sal uprzywilejowanych,
8. lista założeń.



Rys. 2. Sieć działań algorytmu

Oznaczenia:

β - godzina rozpoczęcia zajęcia z_α

γ - nr sali, w której będzie odbywać się z_α

$\overline{y_{rk}^{ij}}$ - wartość y_{rk}^{ij} po wprowadzeniu z_α do rozkładu

Obok każdego wykładowcy (grupy, sali) podany jest wykaz godzin zajętych.

Wczytanie najpierw drzewa wydziału w postaci trzech osobnych drzew pozwala pominąć wykaz godzin zajętych w liście nr 5, gdyż są one zwykle takie same dla całego kursu.

Umieszczenie numerów sal w liście nr 2 pozwala pominąć te informacje w liście nr 8.

Lista nr 6 składa się z dwóch części. W pierwszej podano numery sal laboratoryjnych w kolejności zgodnej z listą nr 2. W części drugiej podano sale audytoryjne w kolejności od najmniejszej do największej.

Sale wymienione w liście nr 7 są uprzywilejowane w tym sensie, że jeżeli sala taka zostanie wymieniona w liście nr 8, to musi być przydzielona na zajęcia.

Lista nr 8 stanowi wykaz zajęć prowadzonych przez poszczególnych wykładowców. Po prawej stronie numeru ewidencyjnego wykładowcy zestawia się wszystkie jego zajęcia. Każde zajęcie zamknięte jest w okrągłych nawiasach. Wyróżniamy trzy typy zajęć:

1. laboratoryjne, np. (2, 13, 7-M), co oznacza, że dany w_i ma prowadzić zajęcia dwugodzinne z p_r nr 13 dla g_j nr 7 kursu magisterskiego w sali, która została podana na liście nr 2,

2. audytoryjne, np. (1, 25, 23-I, 5), co oznacza, że dany w_i ma prowadzić godzinę zajęć z p_r nr 25 dla g_j nr 23 kursu inżynierskiego w s_l nr 5 z listy sal audytoryjnych, o ile jest ona uprzywilejowana lub s_l o numerze $\geqslant 5$,

3. narzucone, np. (2, 21, 5-W, 8, 5, 16), co oznacza, że dany w_i będzie prowadził dwie godziny zajęć z p_r nr 21 dla g_j nr 5 kursu wieczorowego w s_l nr 8, w piątek od godz. 16.00.

2.4. Składnia danych

Symbole podstawowe

<litera> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<cyfra> ::= 0|1|2|3|4|5|6|7|8|9
<ograniczn ik> ::= ,|.|-|Ω| (|)

Wyrażenia pomocnicze

<nazwa> ::= <litera>|<nazwa><litera>|<nazwa><cyfra>
<liczba całkowita bez zn.> ::= <cyfra>|<liczba całkowita bez zn.><cyfra>

<para liczb> ::= <liczba całkowita bez zn.>,<liczba całkowita bez zn.>

<ciąg liczb> ::= <para liczb>|<ciąg liczb>,<liczba całkowita bez zn.>

<przedział czasu> ::= <liczba całkowita bez zn.> - <liczba całkowita bez zn.>

<ciąg przedziałów czasu> ::= <przedział czasu>|<ciąg przedziałów czasu>,<przedział czasu>

<ciąg przedziałów czasu dnia> ::= <liczba całkowita bez zn.>(<ciąg przedziałów czasu>)

<ciąg przedziałów czasu tygodnia> ::= <ciąg przedziałów czasu dnia>|
<ciąg przedziałów czasu tygodnia>
<ciąg przedziałów czasu dnia>

<wykaz godzin zajętych> ::= 0|<ciąg przedziałów czasu tygodnia>
0

Drzewo kursu

<podział roku> ::= <liczba całkowita bez zn.>|<liczba całkowita bez zn.>(<ciąg liczb>)|<podział roku>,
<podział roku>

<ciąg lat> ::= <liczba całkowita bez zn.> (<podział roku>) |
<ciąg lat>, <liczba całkowita bez zn.>
(<podział roku>)

<drzewo kursu> ::= 0 (<ciąg lat>)

Lista przedmiotów laboratoryjnych

<lista przedmiotów laboratoryjnych> ::= <nazwa>Ω<liczba całkowita bez zn.> | <lista przedmiotów laboratoryjnych> <nazwa>Ω <liczba całkowita bez zn.>

Lista przedmiotów nielaboratoryjnych

<lista przedmiotów nielaboratoryjnych> ::= <nazwa>Ω | <lista przedmiotów nielaboratoryjnych> <nazwa>Ω

Listy wykładowców, grup i sal

<odcinek listy> ::= <nazwa>Ω<wykaz godzin zajętych>
<lista sal> ::= <odcinek listy> | <lista sal> <odcinek listy>
<lista grup> ::= <lista sal>
<lista wykładowców> ::= <lista sal>

Numery sal uprzywilejowanych

<numery sal uprzywilejowanych> ::= <ciąg liczb>

Lista założeń

<grupa> ::= <liczba całkowita bez zn.> - M | <liczba całkowita bez zn.> - I | <liczba całkowita bez zn.> - W

<zajęcie> ::= (<para liczb> ,<grupa>)| (<para liczb> , <grupa>, <liczba całkowita bez zn.>) | (<para liczb> , <grupa> , <liczba całkowita bez zn.> , <para liczb>)

<ciąg zajęć> ::= <zajęcie> | <ciąg zajęć> <zajęcie>

<odcinek listy założeń>::= <liczba całkowita bez zn.> .<ciąg zajęć> 0

<lista założeń> ::= <odcinek listy założeń>|<lista założeń> <odcinek listy założeń>

2.5. Postać wyników

Wyniki programu wyprowadza się na drukarkę wierszową w dowolnej liczbie egzemplarzy. Najpierw drukowana jest lista zajęć niezaplanowanych (o ile istnieją), następnie zajęcia zaplanowane w trzech wersjach:

1. uporządkowane według numerów ewidencyjnych listy grup w kolejności: kurs magisterski, kurs inżynierski, kurs wieczorowy,

2. uporządkowane według numerów ewidencyjnych listy wykładowców w kolejności: Instytut Informatyki, Instytut Telekomunikacji, Instytut Technologii Elektronicznej, pracownicy z zewnątrz,

3. uporządkowane według numerów ewidencyjnych listy sal w kolejności: sale laboratoryjne, sale audytoryjne.

Rozkład zajęć dla każdego wykładowcy (grupy, sali) podany jest chronologicznie. Na rys. 3 podano rezultaty działania systemu programów dla semestru zimowego 1971/72.

Egzemplarze pierwszej wersji przesyłane są do dziekanatu, zaś drugiej do poszczególnych instytutów. Egzemplarze trzeciej wersji do Działu Nauczania oraz służą jako obciążenia sal.

ROZKLAD ZAJEC DLA WYDZIALU ELEKTRONIKI P.G.

KURS MAGISTERSKI

ROK 1/M

PONIEDZ.	12-14	DOC. DR BOJARSKI	SALA CH 215	FIZYKA
SRODA	9-11	MGR BESALA	SALA 401E	MATEMATYKA
CZWARTEK	7- 9	DOC. DR SYNOWIECKI	SALA 213	FILOZOFIA MARKSISTOW.
CZWARTEK	9-11	DOC.DR DOBROWOLSKI	SALA CH 215	CHEMIA
CZWARTEK	11-12	MGR LASOTA	SALA 213	RYSUNEK TECHNICZNY
SOBOTA	9-11	MGR BESALA	SALA 462C	MATEMATYKA

GRUPA 1 ROK 1/M

PONIEDZ.	8-11	MGR SZMYTKOWSKI	SALA 415	FIZYKA-LAB.
PONIEDZ.	11-12	MGR KILIAN	SALA CH 215	RYSUNEK TECHNICZNY
SRODA	7- 9	DOC.DR SYNOWIECKI	SALA CH 112	FILOZOFIA MARKSISTOW.
SRODA	11-12	MGR RUDAK	SALA NE 3	ROSYJSKI
SRODA	12-14	MGR IDZIAK	SALA A.S.O.	WYCHOWANIE FIZYCZNE

ROZKLAD ZAJEC DLA WYDZIALU ELEKTRONIKI P.G.

INSTYTUT INFORMATYKI

PROF. DR SEIDLER

WTOREK	9-12	AUTOMATYKA ROK 5/M	SALA NE 505	TEORIA INFORMACJI

DOC.DR BARTKOWSKI

PONIEDZ.	9-11	AUTOMATYKA ROK 5/M	SALA NE 505	OBWODY FUNKCJONALNE M.C.
CZWARTEK	9-11	ROK 4/M	SALA 213	PROGRAMOWANIE M.C.
CZWARTEK	11-13	AUTOMATYKA ROK 5/M	SALA 401E	OBWODY FUNKCJONALNE M.C.

SALA 213

CZWARTEK	7- 9	ROK 1/M	DOC. DR SYNOWIECKI	FILOZOFIA MARKSISTOW.
CZWARTEK	9-11	ROK 4/M	DOC.DR BARTKOWSKI	PROGRAMOWANIE M.C.
CZWARTEK	11-12	ROK 1/M	MGR LASOTA	RYSUNEK TECHNICZNY
CZWARTEK	13-15	MIERNICTWO ROK3/I	MGR NOWAKOWSKI	PODSTAWY MIERNICTWA
PIATEK	8-10	ROK 2/M	DOC. DR PALCZEWSKI	MATEMATYKA
PIATEK	11-13	ROK 2/I	DOC. DR MATUSEWICZ	LAMPY ELEKTRONOWE
PIATEK	13-15	GRUPA 2 ROK 2/I	MGR BESALA	MATEMATYKA
SOBOTA	7- 9	ROK 5/M	DR LASKI	PROGRAMOWANIE M.C.
SOBOTA	9-12	ROK 3/M	PROF. DR GRABOWSKI	TEORIA POLA

WYKONALA MASZYNA CYFROWA ZAM 41 ALFA
INSTYTUT INFORMATYKI POLITECHNIKI GDANSKIEJ

Rys. 3. Postać wyników końcowych

2.6. Parametry systemu

Czas

Podane czasy zmierzone zostały dla następujących danych: 125 przedmiotów, 189 prowadzących, 170 grup, 51 sal, 488 założeń. Otrzymano:

1. czas czytania danych	- około 4 min.
2. czas obliczeń dla p = 10	- około 5 godz. 15 min.
3. czas drukowania wyników (2 egz.)	- około 55 min.

Podział pamięci bębnowej

1. system programów	- około 5000 słów
2. macierze A, B i C	- 3648 "
3. lista wykładowców	- 2000 "
4. lista grup	- 2500 "
5. lista sal	- 500 "
6. lista przedmiotów	- 1500 "
7. drzewo wydziału	- 400 "
8. lista założeń	- 2400 "

Na rozporządzalność w tygodniu salą (grupą, wykładowcą) przeznaczono 6 słów 24-bitowych.

Ograniczenia obowiązujące system

1. maksymalna liczba wykładowców	- 256
2. maksymalna liczba grup	- 256
3. maksymalna liczba sal laboratoryjnych	- 64
4. maksymalna liczba sal audytoryjnych	- 32
5. maksymalna liczba przedmiotów	- 256
6. maksymalna liczba przedmiotów laboratoryjnych	- 100
7. maksymalna liczba założeń	- 1200
8. maksymalny wymiar przedmiotu	- 7

3. ZAKOŃCZENIE

O możliwościach eksploatacyjnych systemu najlepiej świadczy fakt, że był on z pożytkiem wykorzystany do układania rozkładu dla Wydziału Techniki Elektronicznej Uniwersytetu w Rostocku. Program realizujący algorytm ma jednak zasadniczą wadę. Jest nią jakość rozwiązania. Jak wspominaliśmy, wybrana metoda nie pozwala na optymalizację rozkładu zajęć. Rozkład zajęć zaproponowany przez maszynę należy traktować jako pierwsze przybliżenie rozkładu zajęć, który będzie obowiązywał. Jeżeli istnieje konieczność należy dokonać drobnych korekt, ewentualnie wstawić zajęcia niezaplanowane.

Nowe metody np. kolorowanie grafów, wyprzedzają jeszcze możliwości współczesnych EMC, a poza tym nie ujawniają nierozwiązalności zadania w początkowym etapie obliczeń. Niemniej wprowadzanie coraz to szybszych maszyn pozwoli je z pożytkiem wykorzystać. Tymczasem duże usługi mogą oddać metody konwersacyjnego układania planów zajęć. Polegają one na współdziałaniu człowieka i maszyny, jego kontroli i ingerencji w proces obliczeniowy.

Na marginesie warto zwrócić uwagę na fakt, że algorytmy tego typu po wprowadzeniu drobnych zmian pozwalają na układanie harmonogramów sesji egzaminacyjnej, czy też planowanie urlopów, a nawet na projektowanie rozmieszczenia surowców w magazynach chemicznych.

Autor wyraża podziękowanie Panu doc. dr inż. Tadeuszowi Bartkowskiemu za uwagi krytyczne, które zostały uwzględnione w redakcji tej pracy.

Literatura

[1] GOTLIEB C.C.: The Construction of Class-Teacher Time-tables, Proceedings of the IFIP Congress 1962, Amsterdam 1963, p. 73.

[2] CSIMA J., GOTLIEB C.C.: Tests on a Computer Method for Constructing School Timetables, Comm. ACM, v. 7, 1964, p. 160.

[3] HALL P.: On Representatives of Subsets, J. Lond. Math. Soc., v. 10, 1935, p. 26.

[4] KRECZMAR A.: Algorytm układania rozkładu zajęć dla wydziału szkoły wyższej, Praca dyplomowa, ZON UW, maj 1967.

[5] KRECZMAR A.: Algorithm for Constructing of University Timetables and Criterion of Consistency of Requirements, praca nie publikowana.

[6] ALMOND M.: An Algorithm for Constructing University Timetables, The Comp. Jour., v. 8, No 4, Jan. 1966, p. 331.

[7] SHERMAN G.R.: Combinatorial Problem Arising from Scheduling University Classes, Jour. of the Tennesse Academy of Science, v. 38, No 3, July 1963, p. 115.

[8] HARDING R.E.: Linear Programming as an Aid in University Timetables, A Progress Report Dept. of Industrial Engineering and Graduate Schools of Business, University of Pittsburgh, May 1963.

[9] LAWRIE N.L.: An Integer Linear Programming Model of a School Timetabling Problem, The Comp. Jour., v. 12, 1969, p. 307.

[10] UNJINO K.: A Preparation Program for the Timetable Using Random Numbers, Inf. proc. Japan, v. 5, 1965, p. 8.

[11] LAZAK D.: Ein Mathematisches Modell zur Erstellung von Stundenplänen, Elektron Datenverarb, v. 8, Nr 3, 1966, p. 102.

[12] YULE A.P.: Extension to the Heuristic Algorithm for University Timetables, The Comp. Jour., v. 10, May 1967, p. 360.

[13] APPLEBY S.J., BLAKE D.V., NEWMAN E.A.: Techniques for Producing School Timetables on a Computer and their Application to other Scheduling Problems, The Comp. Jour., v. 3, No 5, 1961, p. 237.

[14] GREKO B.: School Scheduling through Capacitated Network Flow Analysis, Report Off. organ. mana, Stockholm, Sept. 1965, p. 19.

[15] HAMMER P.L., RUDEANU S.: Boolean Methods in Operations Research and Related Areas, Berlin 1968, Springer-Verlag, pp. 274-276.

[16] LIONS J.: Matrix Reduction Using the Hungarian Method for the Generation of School Timetables, Comm. ACM, v. 9, 1966, p. 349.

[17] KUBALE M.: Heurystyczny algorytm układania rozkładu zajęć dla wydziału wyższej uczelni, Zeszyty Naukowe Politechniki Gdańskiej, Elektronika (w przygotowaniu).

ПРОБЛЕМЫ СОСТАВЛЕНИЯ РАСПИСАНИЯ ЗАНЯТИЙ В ВЫСШИХ УЧЕБНЫХ ЗАВЕДЕНИЯХ С ПРИМЕНЕНИЕМ ЦИФРОВЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИН

Резюме

Предметом работы является анализ математической модели расписания дидактических занятий в высших учебных заведениях и обзор методов алгоритмизации составления расписания занятий. Общие рассуждения обоснованы на примере системы программа реализованных в Институте Информатики Гданьской Политехники.

PROBLEMS OF COMPUTER AIDED UNIVERSITY TIME-TABLE ORGANIZATION

Summary

The subject of the paper is an analysis of a mathematical model of lecture time-table in Universities and Academies and a review of algorithmization methods of lecture time-table organization. General considerations are supported by an example of a system of programs realized in the Institute of Informatics of the Gdańsk Technical University.

Cena zł 40.-