ALGORYTMY

VOL. II • NO. 4 • 1965

INSTYTUT MASZYN MATEMATYCZNYCH PAN

Od redakcji

Dotychczas w Serii ALGORYTMY ukazały się dwa tomy:

Tom I - zawierający zeszyt Nr 1 i 2
Tom II - zawierający zeszyt Nr 3 i 4
Zeszyt Nr 5 rozpocznie tom III ser. ALGORYTMY.

Notice

The series ALGORYTMY contains two volumes:

Volume I - comprising essues No 1 and 2
Volume II - comprising essues No 3 and 4
Essue No 5 will be enclosed in Volume III.

A L G O R Y T M Y

Vol. II N°4 1965

P R A C E

Instytutu Maszyn Matematycznych

P o l s k i e j A k a d e m i i N a u k

Adres redakcji: Warszawa, ul.Koszykowa 79, tel.28-37-29

WDN-Zam. 493/o/65 Nakł. 600 egz.

T R E Ś Ć
C O N T E N T S

P365/615

N U M E R I C A L A N A L Y S I S

Instytut Maszyn Matematycznych
Algorytmy N° 4

518.12:519.28

ON THE MONTE-CARLO EVALUATION OF THE EXTREMAL VALUE OF A FUNCTION

by Ryszard ZIELIŃSKI

Received May 15th 1964

An algorithm for evaluating the extremal value of a function, based on the Monte-Carlo method, is considered. Under very general assumptions the probabilistic interpretation of the solution is obtained.

1. INTRODUCTION

Let A_0 be a set in an Euclidean space and let μ be a normed measure defined over the class $\mathcal{B}$ of Borel subsets of A_0. Suppose A_0 be a set bounded and dense in itself or a finite one. Let f be a μ - measurable function over A_0 with range F. Assume that F is a well ordered set, where the ordering relation is denoted by inequality sign.

The problem is to evaluate an extremal value of the function f. Without loss of generality we shall evaluate the maximum of this function.

Classical methods are not applicable in such a general treatment. On the other hand, in many practical problems more detailed assumptions are hardly possible to be specified. This is illustrated by the following examples.

Example 1

The maximum of a real function over A_0 is to be evaluated. Suppose A_0 to be a subset in Euclidean space defined by a system of nonlinear inequalities. This is a standard non-linear programming problem. A_0 may be a non connected set.

Example 2

Over A_0 such a function f is defined that for each pair of its values f_1 and f_2 one can decide whether or not f_1 is "better" than f_2. The values of that function are, for example, pairs of numbers. Such functions appear in econometry /e.g. the value of a function is a pair (f', f''), where f' is the price of some device and f'' - the time of its exploitation/, in mathematical statistics /the probability of errors of the I and II kind as values of a certain function/ etc. The problem is to find "the best" value. The funotion f is μ - measurable if $f^{-1}(F_0) \in \beta$, where F_0 is the set of values which are better than the specified one.

There exist several algorithms for solution of some problems of this type. In the oase where A_0 is a set defined by a system of linear inequalities and f is a linear function, one can apply a linear programming algorithm, e.g. the simplex algorithm. Similarly, there exist algorithms for solution of quadratic programming problem when A_0 is a simple connected set.

It seems that the Monte-Carlo method is unique applicable in such general treatment of the problem. An algorithm based on the Monte-Carlo method will be formulated, and a probabilistic interpretation of the results as well as some generalizations of the problem will be given.

2. THE EVALUATION OF THE MAXIMUM OF A FUNCTION BY THE MONTE-CARLO METHOD

Let X be a random variable uniformly distributed over A_0 and let $\{x_i\}$, $i = 1, 2, \ldots$ be a sequence of values taken by a random variable X. Let $G_1 = f(x_1)$ and

$$G_i = \begin{cases} f(x_i), & \text{if} \quad f(x_i) > G_{i-1} \\ G_{i-1}, & \text{if} \quad f(x_i) \leq G_{i-1} \end{cases} \qquad /1/$$

The algorithm is usually formulated in the following way:

1° Construct the sequence $\{x_i\}$

2° Construct the sequence $\{G_i\}$ according to /1/. This is a nondecreasing sequence.

3° If the value $G^* = \max_i \{G_i\}$ appears sufficiently many times in the sequence $\{G_i\}$, one says that G^* is equal to the maximum of the function f.

The use of this algorithm is simple on a digital computer. It is however necessary to explain:

1' the meaning of "sufficiently many times" and

2' how far justified is the conclusion: "G^* being equal to the maximum of the function f".

The explanation is given below.

Let A_i be a set defined as:

$$A_i = \{x: f(x) > G_i\} \qquad /2/$$

and let $\mu_i = \mu(A_i)$.

The sequence $\{\mu_i\}$ is nonincreasing. If G_i is the maximum of f, then obviously $\mu_i = 0$. The converse is not always true.

It is true if f is continuous, however, this assumption will be not needed further.

Introducing $A^* = \{x: f(x) > G^*\}$, questions 1' and 2' may be replaced by the question: whether $\mu^* = \mu(A^*) = 0$ if the value G^* appeared N times in the sequence $\{G_i\}$.

We denote by $(n=m)/N$ the event which occurs when the value of f exceeds m times in the N-elements sequence a specified value G^*. Let us evaluate the credibility of the event $\mu^* < \varepsilon$. This credibility is defined in [1] as follows:

$$W\{\mu^* < \varepsilon \ ; \ (z = 0)/N\} \stackrel{df}{=} P\{(z > 0)/N+1; \ \mu^* = \varepsilon\} \qquad /3/$$

where ε is a number from the interval $(0,1)$. The left-hand side of /3/ we denote shortly by $W(\mu^* < \varepsilon)$

We obtain:

$$P\{(z > 0)/N+1; \ \mu^* = \varepsilon\} = 1-P\{(z=0)/N+1; \ \mu^* = \varepsilon\} = 1-(1-\varepsilon)^{N+1}$$

If we put *)

$$W(\mu^* < \varepsilon) = 1 - \varepsilon \qquad /4/$$

then the relation between ε and N is given by the formula:

$$(1-\varepsilon)^{N+1} = \varepsilon \qquad /5/$$

Some values of ε and N are given in the following table:

ε	0,5	0,2	0,1	0,05	0,02	0,01	0,005	0,002	0,001
N	0	7	21	57	193	457	1056	3103	6903

*) The idea of evaluating credibility $W(\mu^* < \varepsilon) = 1 - \eta$ for $\eta = \varepsilon$ is due to H. Steinhaus. He showed also, that given N, /5/ has a unique solution in $(0, 1)$.

Let E be an event which occurs when the measure of such a subset of A_0, on which f is greater than G^*, is smaller than ε. Now we can say:

(S): if in N trials the value of a function f does not exceed a specified value of G^*, the credibility of E is equal to $1 - \varepsilon$.

We can also solve the above formulated problem using a posteriori probability of the event $\mu^* < \varepsilon$ instead of its credibility.

The numbers x_i are the values of a random variable X, therefore μ_i are the values of some random variables, say M_i. Let us assume M_1 to be uniformly distributed over the interval $(0,1)$. The reason of such an assumption is that the random variable X is uniformly distributed. Because of the way in which the sequence $\{G_i\}$ and therefore $\{\mu_i\}$ are obtained, such assumption on distributions of M_i for $i > 1$ is unfounded; we shall assume that a priori distribution of M_i, $i > 1$, is a uniform one over the interval $(0, \mu_{i-1})$.

We denote by $P\{\mu^* < \varepsilon\}$ the posteriori probability of the event $\mu^* < \varepsilon$ when f does not exceed a specified value of G^* in N trials. It is easy to calculate that

$$P\{\mu^* < \varepsilon\} \geq \frac{\int_0^{\varepsilon} (1 - s)^N \, ds}{\int_0^1 (1 - s)^N \, ds} \qquad /3'/$$

As above in /4/ we put:

$$1 - \varepsilon = \frac{\int_0^{\varepsilon} (1 - s)^N \, ds}{\int_0^1 (1 - s)^N \, ds} \qquad /4'/$$

Then the relation between ε and N is once again given by /5/.

Now the following solution is obtained:

(S'): if in N trials the value of a function f does not exceed a specified value of G^*, then the probability of E is not smaller than $1 - \varepsilon$.

We shall express the statements (S) or (S') shortly as follows: the credibility /or probability/ that G^* is the maximum of a function f is equal to $1 - \varepsilon$.

Now we can modify the algorithm:

1. Choose a number ε /or N/. Then the number N /or ε / is uniquely determined by /5/.
2. Construct a sequence $x_1, x_2, \ldots, x_N$ as a simple sample from population X.
3. Find $G^* = \max_i f(x_i)$.
   Then G^* is equal to the maximum of a function f over A_0 in the sense of (S) or (S').

In the new algorithm the number N of trials is a priori specified while in the first formulation N was a random variable.

3. SOME GENERALIZATIONS OF THE PROBLEM

3.1. The assumption that A_0 is bounded can be omitted, provided that $\mu(A_0) = 1$. The range of the measure μ is then the interval $[0,1]$. Let Y be uniformly distributed over $[0,1]$. Let $\{y_i\}$ be a simple sample from the population Y. We construct a sequence $\{x_i\}$, $x_i \in A_0$ in the following way:

$$y_1 = \mu\{x: x \leq x_1\} \qquad /6/$$

The value x_1, given y_1, is not always unique but when x_1', $x_1' \neq x_1$ is the value for which /6/ also holds, then the sets $\{x: x \leq x_1\}$ and $\{x: x \leq x_1'\}$ differ on the set of μ - measure zero. However, this is not essential for the obtained results.

Similarly the assumption that A_0 is a set dense in itself may be omitted.

The original assumptions were necessary to define a random variable uniformly distributed over A_0.

3.2. The above considered methods are sometimes used as follows. Given $x_1, \ldots, x_n$, the set A_0 is divided in mutually exclusive subsets. Now one seeks for the maximum of f over such of the above mentioned subsets which contains x satisfying $f(x) = \max_i f(x_i)$. Given the posteriori probabilities /or oredibilities/ of the event $\mu^* < \varepsilon$ over each of the subset of A_0, the calculation of a posteriori probability /or oredibility/ of the event $\mu^* < \varepsilon$ over a whole set A_0 is trivial.

3.3. Let $C \subset A_0$ be a subset of elements x's with some property V. The problem is to evaluate the fraction of these elements in the population A_0 of all elements. This is equivalent to the problem of evaluating the measure of the subset C.

In the problem considered in this note, $x \in A_0$ possess the property V if $f(x) > G$. In the general case, given the function of elements with property V in N-element sample, the fraction of these elements in A_0 is to be calculated.

If, as above, the x's are independently obtained, we are faced with the known problem of estimation of a parameter of the binomial distribution. Thus the problem of evaluation of extremal value of a function by the Monte-Carlo method is being reduced to a well known statistical inference problem.

References

1. STEINHAUS H.: Probability, Credibility, Possibility, Zastosowania Matematyki, 1963:VI, 4, 341-361.

Instytut Maszyn Matematycznych
Algorytmy N° 4

518.6:519.21:621.833

O ZASTOSOWANIU METOD MONTE-CARLO DO SYNTEZY MASZYN

Jan GOLIŃSKI

Pracę złożono 9.07.1964 r.

W pracy rozpatrzono przypadek programowania polegający na znalezieniu ekstremum absolutnego w obszarze wyciętym z przestrzeni wielowymiarowej i ograniczonej hyperpowierzchniami. Zadanie z zakresu syntezy maszyn rozwiązano metodą Monte-Carlo.

1. WSTĘP

Konstruowanie stanowi jedną z najważniejszych czynności inżyniera.

Konstruowanie jest, i zapewne zawsze pozostanie, zespoleniem elemen ów sztuki i elementów metody naukowej.

Metoda powszechnie stosowana polega na przyjęciu ogólnej koncepcji, na wstępnym doborze parametrów i na sprawdzeniu warunków wytrzymałościowych, technologicznych, montażowych itd. Doświadczenie i wyczucie konstrukcyjne projektującego ma tu istotne znaczenie, szczególnie przy wstępnym doborze parametrów. Znamiennymi rysami tej metody są:

1. niewielka liczba rozpatrywanych wariantów;
2. formułowanie różnych warunków w toku projektowania i analizy wariantów.

Stan taki, pozwalający na szeroko pojętą dowolność przy konstruowaniu, zmuszał różnych autorów do poszukiwania innych metod pro-

jektowania. Dążono do takiego przeprowadzenia procesu konstrukcyjnego, który by umożliwił sensowny jego przebieg, oparty na doświadczeniu, ale sprawdzany metodami ścisłymi. Chodziło więc przede wszystkim o to, aby dane doświadczalne i wszystkie więzy wbudować z góry w schemat obliczeń. Poprawne ujęcie tego zagadnienia stwarzało dużo kłopotów, między innymi dlatego, że napotykało na trudności poprawnego sformułowania zadania i właściwego matematycznego ujęcia wpływu różnych czynników na konstrukcje. Dodać należy, że metody, które można by było zastosować do obliczeń konstrukcyjnych wymagały dużego nakładu obliczeń rachunkowych i dopiero zastosowanie nowej, maszynowej techniki liczenia zmieniło radykalnie praktyczną możliwość ich użycia.

2. DOTYCHCZASOWY STAN WIEDZY W ZAKRESIE OPTYMALIZACJI KONSTRUKCJI METODAMI PROGRAMOWANIA NIELINIOWEGO

W różnych dziedzinach techniki poszukiwano w różny sposób optymalnych parametrów układów i maszyn. Najszerzej korzystano z tego w dziedzinie automatyki. W dziedzinie konstrukcji czysto mechanicznych stosowano metody programowania liniowego w zakresie projektowania obrabiarek, linii automatycznych oraz do określania optymalnych warunków skrawania.

Metody programowania liniowego zastosowano również do dynamicznego wyrównoważenia wałów korbowych, a nieliniowe do syntezy mechanizmów itd.

W przedstawionej pracy opracowano metodę pozwalającą na uzyskanie konstrukcji bardziej oszczędnych niż dotychczasowe. Osoba kierująca obliczeniami, wolna od trudu arytmetycznego, organizuje je, modyfikuje, ocenia i decyduje o wariancie rozwiązania i jego dokładności.

Praca określa sposób poprawnego formułowania zadań konstrukcyjnych, uwypuklając funkcję kryterium, do którego dąży konstruktor. Metoda, algorytm i programy dla maszyny matematycznej umożliwiają szybkie uzyskanie rozwiązania.

3. OPIS METODY

3.1. Założenia do metody rozwiązania

Wynik rozwiązania należy przedstawić wektorem

$$X = (X_1, X_2, \ldots\ldots\ldots\ldots, X_k), \qquad /1/$$

a występujące ograniczenia w postaci

$$\varphi_i(x) \geqslant 0; \qquad (i = 1, 2, \ldots, n) \qquad /2/$$

Wreszcie należy zbudować funkcję-kryterium $f(x)$, która wyraża nasz pogląd o maszynie.

Zadanie polega na znalezieniu wektora X określonego przez /1/, dla którego wszystkie warunki typu /2/ są spełnione, a jednocześnie wartość funkcji optymizowanej $f(x)$ osiąga ekstremum absolutne. Proponowana metoda rozwiązania polega na losowym przeszukiwaniu obszaru z zapamiętywaniem kolejnego, najlepszego wyniku. W zależności od rodzaju zadania zaproponowano różne warianty postępowania, które można ująć następująco:

1. długotrwałe losowanie w ustalonej kostce n wymiarowej,
2. losowanie w kostkach n wymiarowych zmienianych i coraz ciaśniej obejmujących niewiadome ekstremum,
3. ustalenie pewnych parametrów i dalsze losowanie w kostce $n - k$ wymiarowej, gdzie k - stanowi ilość ustalonych parametrów.

Po każdym z wariantów /1,2,3/, zależnie od zadania, można, przez specjalne opracowanie statystyczne otrzymanych wyników, oszacować najlepszy, możliwy do osiągnięcia wynik, lub stosując jedną z metod programowania wypukłego dojść do ekstremum względnego, które wobec uprzednich czynności 1,2,3, będzie niemal na pewno absolutne.

3.2. Losowanie w obszarze jednospójnym /patrz sieć działań SD-1/

Każdemu rzutowi wektora X przyporządkowuje się kolejną generowaną liczbę losową z rozkładu równomiernego. Po losowaniach ma-

my wektor X opisany kolejnymi liczbami losowymi. Dla tego wektora sprawdzamy spełnienie warunków typu /2/. Przy pierwszym napotkanym niespełnionym warunku, generujemy następne wartości, powtarzając sprawdzenie. Po losowej liczbie takich prób trafiamy /przy założeniu, że obszar nie jest pusty, czyli że warunki techniczne nie były sprzeczne/ na układ wartości parametrów spełniających wszystkie warunki.

Dla tych parametrów obliczamy wartość funkcji optymizowanej, zapamiętujemy ją i generując kolejno n wartości powtarzamy opisaną procedurę.

Przy kolejnym trafieniu na układ wartości spełniającej wszystkie warunki porównujemy obliczoną wartość funkcji optymizowanej z uprzednio zapamiętaną, zachowując zawsze wynik najlepszy. Otrzymujemy zatem ciąg wartości funkcji $f(x)$, który nie rośnie przy poszukiwaniu minimum, a nie maleje przy szukaniu maksimum. Przy pewnej liczbie losowań wskazane jest zacieśnienie granic losowania. Ten fragment postępowania wymaga osobistej interwencji obliczającego, a więc decyzji o charakterze inżynierskim. Można by było bez trudu wbudować w program maszyny cyfrowej i tę decyzję, wydaje się jednak, że nie byłoby to ekonomiczne, zwłaszcza przy projektowaniu obiektu nietypowego.

W nowej zmniejszonej kostce losujemy powtórnie. Przestajemy losować wówczas, gdy wynik nie ulega już wyraźnej poprawie. Liczba losowań w różnych kostkach, coraz to lepiej obejmujących optimum, zależna jest od rozwiązywanego zadania.

W niektórych przypadkach ustala się pewne parametry po pewnej liczbie losowań i losuje się dalej w obszarze o mniejszej liczbie wymiarów.

3.3. Metoda opracowania statystycznego

Po zakończeniu losowania wyprowadzamy dane statystyczne w postaci tabelki:

PRZEDZIAŁ WARTOŚCI FUNKCJI KRYTERIUM	LICZBA TRAFIEŃ W PRZEDZIAŁ (F_{i-1}, F_i)	KUMULACYJNA LICZBA TRAFIEŃ
F_i	k(i)	K(i)
35	2	2
36	5	7
37	13	20
38	15	35
39	22	57
40	20	77
41	31	108

Jeżeli przez F_i określimy wartość funkcji optymizowanej w i-tym przedziale, wówczas w dostatecznie bliskim otoczeniu absolutnego ekstremum możemy założyć, że funkcja $F_i = f(K(i))$ jest parabolą postaci

$$F_i - F_o = A(K(i))^m \qquad /3/$$

gdzie

F_i – wartość funkcji optymizowanej odpowiadająca K (i)-tej wartości kumulacyjnej

F_o – poszukiwana, ekstremalna wartość funkcji optymizowanej

A – poszukiwany współczynnik paraboli

m – niewiadomy wykładnik paraboli.

Zaznaczyć należy, że ekstrapolację paraboliczną przyrostów skończonych tej funkcji w przedziałach podsunął mi prof.dr Jan Oderfeld. Praktyczna realizacja tej sugestii jest nieco inna, jak to niżej pokazano.

Mamy zatem funkcję $F_i = f(K(i))$, która w różnych punktach $K_1, K_2, K_3, \ldots, K_n$ przyjmuje wartości $F_1, F_2, \ldots, F_n$. Chcemy znaleźć takie przybliżenie funkcji $F = f(K(i))$ parabolą $F_i - F_o = A(K(i))^m$, żeby błąd średni aproksymacji osiągnął minimum.

Postępując w znany sposób logarytmujemy obie strony równania /3/ i budujemy funkcję Y postaci,

$$Y = \sum_{i=1}^{n} \left[\lg(F_i - F_o) - \lg A - m \cdot \lg(K(i))\right]^2$$

podstawiając*) $B = \lg A$

$$j = \lg(K(i))$$

i obliczając pochodne otrzymujemy układ dwóch równań z dwiema niewiadomymi.

$$\frac{\partial Y}{\partial B} = -2\left\{\sum_{1}^{n} \lg(F - F_o) - nB - m\sum_{1}^{n} j\right\} = 0;$$

$$\frac{\partial Y}{\partial m} = -2\sum_{1}^{n} j^2\left\{\sum_{1}^{n} \lg(F - F_o) - B\sum_{1}^{n} j - m\sum_{1}^{n} j^2\right\} = 0$$

Rozwiązując ten układ otrzymujemy A oraz m, dla których obliczamy wartości F^*. /F^* obliczamy podstawiając do wzoru /3/ otrzymane wartości A i m z rozwiązania układu równań, dla rozwiązania których trzeba podstawić przewidywane wartości F_o/. Tak więc dla każdej wartości k mamy teraz parę wartości F oraz F^*. Obliczając:

$$\Delta F_o = \sum_{1}^{n} \left(F - F^*\right)^2$$ dla kilku różnych F_o możemy sprawdzić dla jakiej wartości F_o, ΔF_o jest najmniejsza. Ta wartość F_o stanowi poszukiwaną wielkość ekstremalną.

*) Wystarczy przyjąć pierwsze n wartości z otrzymanej statystyki. W rozwiązanych zadaniach praktycznych przyjmowano n = 5 ÷ 8.

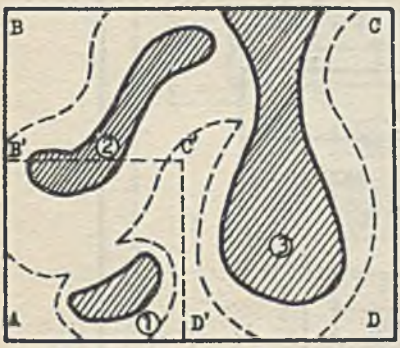
3.4. Przypadek obszaru wielospójnego

W praktyce inżynierskiej zdarzają się zagadnienia, których rozwiązanie może leżeć w jednym z nieznanych i niespójnych obszarów.

Jeżeli obszar jest wielowymiarowy, to na ogół nie potrafimy rozpoznać wzrokowo, bądź intuicyjnie, czy zachodzi wielospójność. Dlatego w razie najmniejszej wątpliwości należałoby zaczynać od tego przypadku. Postępowanie polega na wstępnym losowaniu przy przejściowo rozluźnionych warunkach.

Na istnienie zadań technicznych tego rodzaju i na ich ważność oraz trudności przy ich rozwiązywaniu zwrócił uwagę prof.Oderfeld, któremu zawdzięczam pomysł z "rozpęczaniem" obszaru.

Schemat przeszukiwania obszaru ilustruje sieć działań SD-2, a metodę poszukiwania zilustrujemy prostym przykładem.



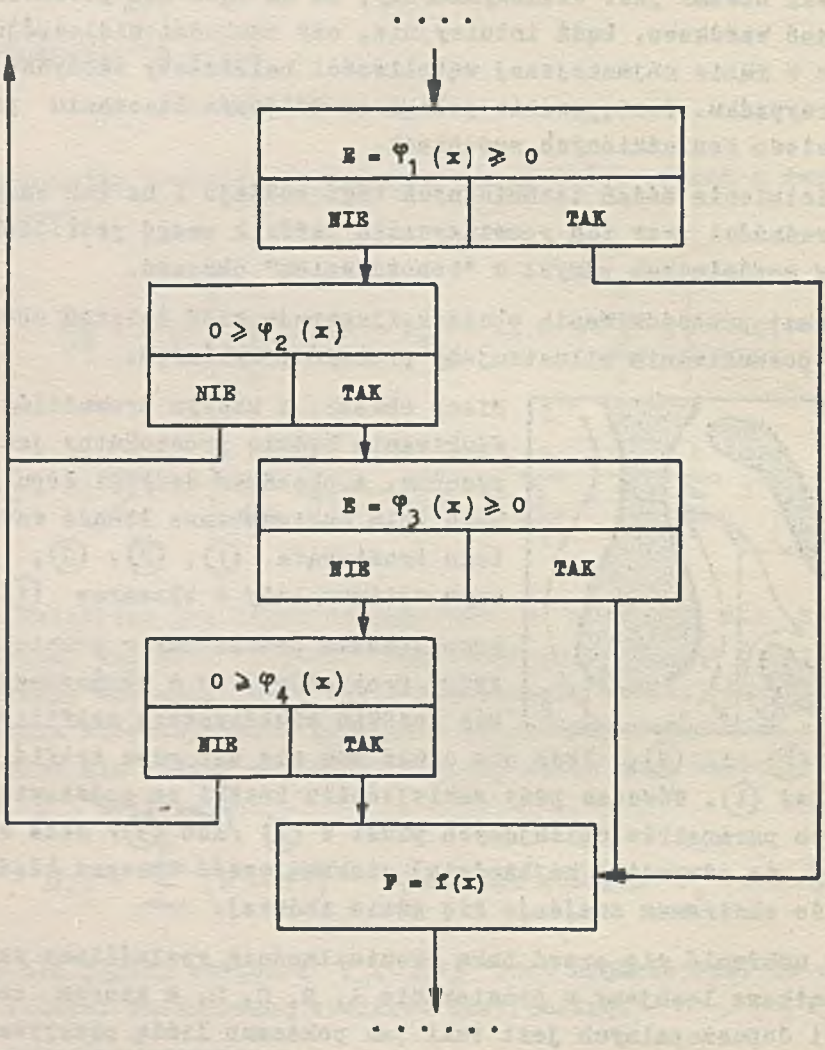
Niech obszar, w którym prowadzimy poszukiwania będzie prostokątny jak na rysunku, a obszarem decyzji dopuszczalnych pola zakreskowane leżące wewnątrz tego prostokąta ①, ②, ③, przy czym optimum leży w obszarze ①.

Poszukiwania prowadzimy w prostokącie ABCD. Przypuśćmy, że w skończonej liczbie losowań wielokrotnie trafiliśmy w obszar ② i ③, lecz nie udało nam się ani razu trafić w mały obszar ①. Wówczas przy zmniejszeniu kostki na podstawie wylosowanych parametrów opisujących punkt w ② /lub ③/ może się zdarzyć, że odrzucimy najbardziej ciekawą część obszaru błędnie sądząc, że ekstremum znajduje się gdzie indziej.

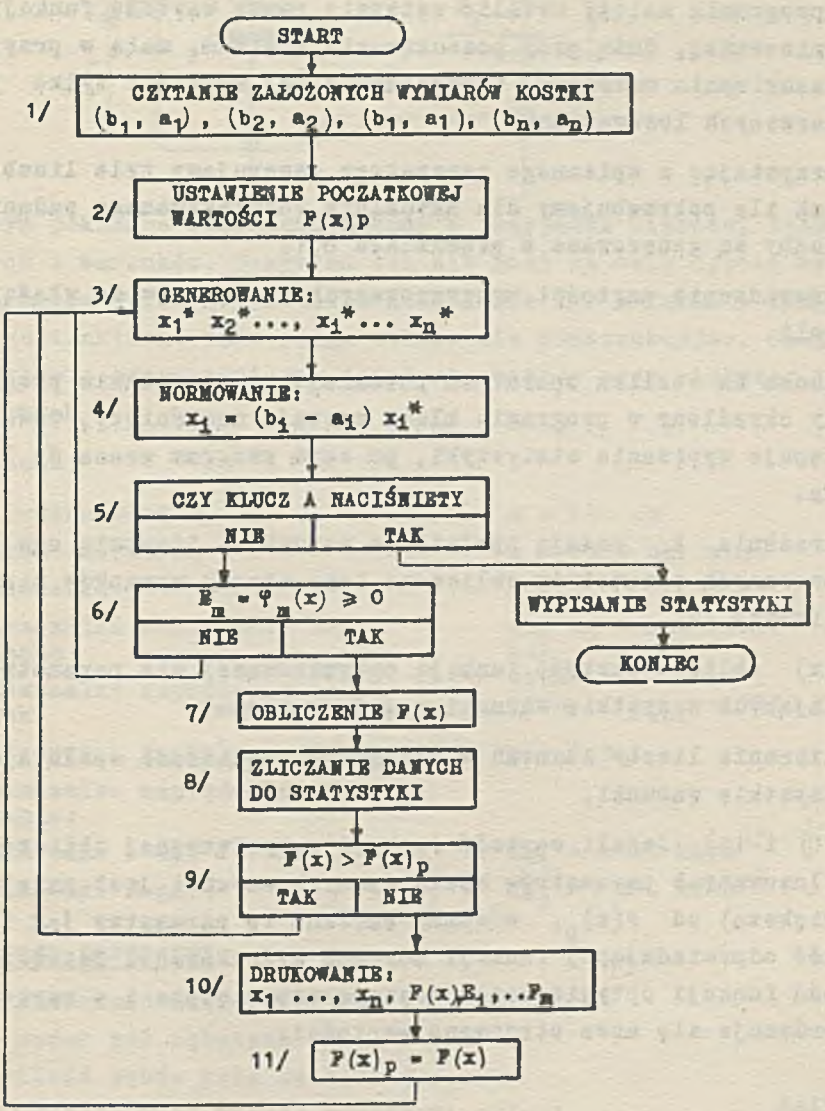
Aby uchronić się przed taką ewentualnością rozluźniamy warunki i początkowo losujemy w prostokącie A, B, C, D, w którym obszar decyzji dopuszczalnych jest taki jak pokazano linią przerywaną. "Rozluźnienie" oznacza zmianę prawej strony wszystkich lub niektórych nierówności. Wielkość rozluźnienia jest sprawą decyzji inżynierskiej.

Po zlokalizowaniu ekstremum można teraz zmniejszyć kostkę na AB´C´D´ i prowadząc w niej losowanie dla pierwotnych warunków najtrafniej określić ekstremum.

SIEĆ DZIAŁAŃ SD-2



SIEĆ DZIAŁAŃ SD-1



Objaśnienia do sieci działań SD-1

2) W programie należy ustalić wstępnie pewną wartość funkcji optymizowanej, dużą przy poszukiwaniu minimum, małą w przypadku poszukiwania maksimum. Korzystamy z tej wartości tylko przy pierwszych losowaniach.

3) Korzystając z opisanego generatora generujemy tyle liczb losowych ile potrzebujemy dla aktualnie rozwiązywanego zadania. Liczby są generowane z przedziału 0,1.

4) Sprowadzenie wartości wygenerowanych parametrów do właściwej skali.

5) Klucze na stoliku operatora pozwalają na sterowanie programem. Gdy określony w programie klucz zostaje naciśnięty, wówczas następuje wypisanie statystyki, po czym program wraca do losowania.

6) Wyrażenia E_m podają spełnienie warunków. Stanowią one dla wylosowanych parametrów obliczone lewe strony warunków nierównościowych /2/.

7) F(x) oblicza wartość funkcji optymizowanej dla parametrów spełniających wszystkie warunki nierównościowe.

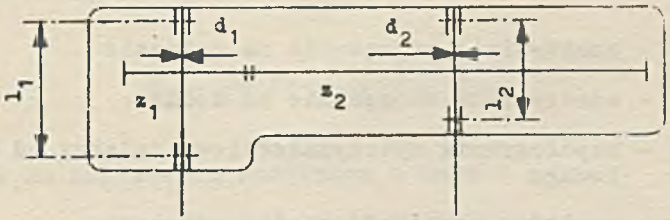
8) Zliczanie liczby losowań w różnych przedziałach spełniających wszystkie warunki.

9), 10) i 11) Jeżeli wartość funkcji optymizowanej obliczana dla wylosowanych parametrów spełniających warunki jest mniejsza (większa) od $F(x)_p$, wówczas zarówno te parametry jak i wartość odpowiadającej funkcji zostaną wydrukowane. Ostatnia wartość funkcji optymizowanej zostaje zapamiętana i z wartością tą porównuje się nowe otrzymane wartości.

Przykład

Reduktor 1-stopniowy /7 zmiennych/

W przykładzie tym rozpatrzono reduktor wg założonego schematu,



na którym pokazano działanie metody w przypadku większej ilości zmiennych i warunków. Przykład ten nie miał na celu ujęcia wszystkich warunków opisujących takie urządzenie i w związku z tym nie należy go traktować jako pełne obliczenie konstrukcyjne. Chodziło w nim tylko o przedstawienie działania metody przy większej ilości zmiennych.

Założono:

Moc przenoszoną	$N = 100$ kM
prędkość obrotową	$n = 1500$ obr min^{-1}
przełożenie	$i = 3$
dopuszczalne naprężenie na zginanie kół zębatych	dla kg = 800 $kGcm^{-2}$
dopuszczalne naprężenie na docisk	$p_{obl} = 5800$ $kGcm^{-2}$
współczynnik kształtu zęba	$q = 2,54$
dopuszczalne naprężenia na zginanie:	
dla wału 1-ego	$kg_1 = 1100$ $kGcm^{-2}$
dla wału 2-go	$kg_2 = 850$ $kGcm^{-2}$

Poszukiwane parametry

- b - szerokość wieńców zębatych
- m - moduł kół zębatych
- z - ilość zębów koła mniejszego
- l_1 - długość wału między podporami wału 1
- l_2 - długość wału między podporami wału 2
- d_1 - średnica wału 1
- d_2 - średnica wału 2

Oznaczenia

σ_g – rzeczywiste naprężenie na zginanie

p_s – rzeczywiste naprężenie na docisk

B – współczynnik wytrzymałościowy zależny od modułów Younga

λ – względna szerokość wieńca zębatego

f_1, f_2 – rzeczywiste strzałki ugięcia wałów podporowych 1 i 2

I_1, I_2 – biegunowe momenty bezwładności przekroju wału 1 i 2

fo_1, fo_2 – dopuszczalne strzałki ugięcia wałów podporowych 1 i 2

P – siła obwodowa w kG

M_{z1}, M_{z2} – zastępcze momenty działające na wał 1 i 2

W_{x1}, W_{x2} – osiowe wskaźniki wytrzymałościowe wału 1 i 2

Opis warunków

1. Warunek na zginanie

$$\sigma_g = \frac{2 \cdot M \cdot q}{b \cdot m^2 \cdot z} \leqslant k_g$$

2. Warunek na docisk

$$p_s^2 = \frac{2 \cdot B \cdot M}{m^2 \cdot z \cdot b} \leqslant p_{obl}^2$$

3. Warunek na graniczną ilość zębów $Z \geqslant Z_{gr} = 17$

4. Warunek na względną szerokość wieńca $\lambda \geqslant 5$

5. Warunek na względną szerokość wieńca $\lambda \leqslant 12$

6. Warunek gabarytowy $m(z_1 + z_2) \leqslant 160$

7. Warunek na strzałkę ugięcia wału 1

$$f_1 = \frac{1}{48} \cdot \frac{P \cdot l_1^3}{E\, I_1} \leqslant f_{01}$$

8. Warunek na strzałkę ugięcia wału 2

$$f_2 = \frac{1}{48} \cdot \frac{P \cdot l_2^3}{E\, I_2} \leqslant f_{o2}$$

9. Warunek na naprężenia zastępcze w wale 1

$$\sigma_g = \frac{M_{z_1}}{W_{x_1}} \leqslant k_{g_1}$$

10. Warunek na naprężenia zastępcze w wale 2

$$\sigma_g = \frac{M_{z_2}}{W_{x_2}} \leqslant k_{g_2}$$

$$M_z = \sqrt{M_g^2 + 0{,}75 \cdot M_s^2}$$

11. Warunek technologiczny $d_2 \geqslant 5$ cm

Funkcja kryterium.

Pełna objętość obu kół zębatych i wałów ma wynosić minimum

$$f(x) = 4\ mbz + d_1^2 \cdot l_1 + d_2^2 \cdot l_2 = \min$$

Tak więc zadanie to sprowadza się do znalezienia ekstremum funkcji f(x) w przestrzeni 7 wymiarowej ograniczonej 11-oma opisanymi warunkami plus 14 warunków narzuconych przez wymiary kostki. Mamy więc 7 zmiennych i 25 warunków.

R o z w i ą z a n i e:

Opisane wyżej warunki przedstawiono w postaci wymaganej przez algorytm i po uwzględnieniu stałych i wartości zadanych otrzymano je w następującej postaci:

1) $bm^2z - 12 \geqslant 0$

2) $bm^2z^2 - 530 \geqslant 0$

3) $z - 17 \geqslant 0$

4) $b - 5m \geqslant 0$

5) $12\,m - b \geqslant 0$

6) $40 - mz \geqslant 0$

7) $d_1^3 - \sqrt{0{,}0625 \cdot l_1/mz} + 1700 \geqslant 0$

8) $d_2^3 - \sqrt{0{,}0825 \cdot l_2/mz} + 1300 \geqslant 0$

9) $(l_1^3/mz \cdot d_1^4) - 192 \geqslant 0$

10) $(l_2^3/mz \cdot d_2^4) - 192 \geqslant 0$

11) $d_2 - 5 \geqslant 0$

Według przyjętej sieci działań SD-1 napisano program załącznik , składający się z 3 rozdziałów:

Rozdział 0 - czytający dane i ustalający stałe

Rozdział 1 - sprawdzający warunki i generujący ciąg liczb losowych

Rozdział 2 - drukujący wyniki.

Wstępne losowanie przeprowadzono w kostce opisanej następująco:

$$1 \leqslant b \leqslant 5$$

$$0{,}2 \leqslant m \leqslant 0{,}6$$

$$22 \leqslant z \leqslant 50$$

$$10 \leqslant l_1 \leqslant 17$$

$$9 \leqslant l_2 \leqslant 18$$

$$3 \leqslant d_1 \leqslant 7$$

$$3 \leqslant d_2 \leqslant 8$$

Wyniki tego losowania przedstawiono tablicą 1.

Tablica 1

Szerokość wieńca w cm	Moduł koła zębatego w cm	Ilość zębów napędzanego koła zębatego	Długość wałka, na którym osadzone jest mniejsze koło	Długość wałka, na którym osadzone jest większe koło	Średnica mniejszego wałka w cm	Średnica większego wałka w cm	Objętość reduktora wg definicji $_3$ w cm	Ilość losowań	Ilość losowań w obszarze
B	M	Z	L_1	L_2	D_1	D_2	OB	LI_1	LI_2
+3,0	+0,5	+35	+11,1	+10,5	+3,6	+6,9	+834,5	7	2
+2.7	+0,5	+49	+15,9	+12,9	+3,5	+5,3	+808,3	212	3
+3,1	+0,4	+34	+12,2	+10,5	+3,5	+5,5	+645,8	336	5
+2,8	+0,4	+34	+10,5	+ 9,2	+3,4	+5,7	+594,2	5110	123

Losowań przeprowadzono 5110; z tego wpadło w obszar 123. Najlepszy wynik osiągnięto w ostatnim losowaniu. Opierając się na najlepszym uzyskanym wyniku ustalono nowe wymiary kostki następująco:

$$2 \leqslant b \leqslant 3{,}6$$

$$0{,}3 \leqslant m \leqslant 0{,}5$$

$$28 \leqslant z \leqslant 40$$

$$9 \leqslant l_1 \leqslant 12$$

$$7{,}4 \leqslant l_2 \leqslant 11$$

$$2{,}8 \leqslant d_1 \leqslant 4$$

$$5 \leqslant d_2 \leqslant 6{,}4$$

Wyniki tego losowania przedstawiają się następująco:

Tablica 2

Szerokość wieńca w cm	Moduł koła zębatego w cm	Ilość zębów napędzanego koła zębatego	Długość wałka, na którym osadzone jest mniejsze koło	Długość wałka, na którym osadzone jest większe koło	Średnica mniejszego wałka w cm	Średnica większego wałka w cm	Objętość reduktora wg definicji w cm^3	Ilość losowań	Ilość losowań w obszarze
B	M	Z	L_1	L_2	D_1	D_2	OB	LI	LI
+2,5	+0,5	+35	+10,8	+9,7	+3,7	+6,9	+769,1	7	2
+2,6	+0,4	+34	+11,5	+9,6	+3,7	+5,5	+594,0	336	7

W kostce tej losowano 2137 razy. W obszar wpadło 68 losowań. Najlepszy wynik, który był 7 kolejnym w obszarze, osiągnięto w 336 losowaniu. Następnych 1801 losowań nie przyniosło poprawy. Wartość funkcji optimum /OB/ poprawiła się bardzo nieznacznie /o 0,2 cm^3/.

Na podstawie wyników losowania ponownie zmieniono wymiary kostki, którą tak ustawiono:

$$2 \leqslant b \leqslant 3{,}4$$
$$0{,}25 \leqslant m \leqslant 0{,}35$$
$$26 \leqslant z \leqslant 42$$
$$8{,}5 \leqslant l_1 \leqslant 13$$
$$7{,}4 \leqslant l_2 \leqslant 11{,}8$$
$$3 \leqslant d_1 \leqslant 6$$
$$5 \leqslant d_2 \leqslant 7$$

Wyniki tego losowania przedstawia tablica 3.

Tablica 3

Szerokość wieńca w cm	Moduł koła zębatego w cm	Ilość zębów napędzanego koła zębatego	Długość wałka, na którym osadzone jest mniejsze koło	Długość wałka, na którym osadzone jest większe koło	Średnica mniejszego wałka w cm	Średnica większego wałka w cm	Objętość reduktora wg definicji w cm^3	Ilość losowań	Ilość losowań w obszarze
B	M	Z	L_1	L_2	D_1	D_2	OB	LI	LI
+3,5	+0,4	+40	+ 9,9	+ 8,3	+5,7	+6,8	+917,3	16	2
+3,4	+0,4	+41	+12,4	+11,7	+4,1	+5,9	+832,0	19	3
+2,8	+0,3	+41	+ 9,6	+10,1	+4,2	+5,8	+664,5	164	5
+3,1	+0,3	+42	+11,9	+ 8,1	+4,2	+5,8	+658,7	2294	65
+2,8	+0,4	+37	+ 9,3	+ 8,7	+4,2	+5,2	+552,0	2442	68

W kostce tej przeprowadzono 8123 losowań, z czego 227 było w obszarze. Najlepszy wynik uzyskano w 2442 losowaniu /68 w obszarze/. Następne 5781 losowań nie poprawiło już wyniku. Tak więc parametry:

$$b = 2{,}8;\quad m = 0{,}4;\quad z = 37;\quad l_1 = 9{,}3;$$
$$l_2 = 8{,}7;\quad d_1 = 4{,}2;\quad d_2 = 5{,}2$$

przyjmujemy za optymalne. Osiągnięty wynik można by było starać się poprawić stosując metodę gradientową, co jest jednak kłopotliwe.

Można również ustalić teraz pewne parametry i losować w układzie o mniejszej ilości wymiarów. Nie było to jednak celem przykładu. Chodziło w nim bowiem jedynie o pokazanie jak działa metoda.

```
L1*4
ROZDZIAL:0
CALKOWITE:I,S
CALKOWITE:LIC1,LIC2
USTAW SKALE BINARNIE:35
CZYTAJ:W
SKALA DZIESIETNA PARAMETROW:6
USTAW SKALE DZIESIETNIE:6
CZYTAJ:GBE,DBE,GEM,DEM,GZET,DZET,GL1,DL1,GL2,DL2,GD1,DD1,GD2,DD2
PISZ NA BEBEN OD 0:GBE,DBE,GEM,DEM,GZET,DZET,GL1,DL1,GL2,DL2,GD1,DD1,-
GD2,DD2
SPACJI 3
TEKST:
B     M     Z     L1    L2    D1    D2    OB    LI    LI
LINIA 2
LIC1=1
LIC2=1
MBE=15000.
MEM=15000.
MZET=15000.
ML1=15000.
ML2=15000.
MD1=15000.
MD2=15000.
MO=15000.
BLOK(249):S
*)S(I)=0
POWTORZ:I=0(1)249
IDZ DO ROZDZIALU:1
KONIEC ROZDZIALU
ROZDZIAL:1
BLOK(0):W
CALKOWITE:LIC1,LIC2,ZETR,S,I
SKALA DZIESIETNA PARAMETROW:6
```

```
USTAW SKALE DZIESIETNIE:6
CZYTAJ Z BEBNA OD 0:GBE,DBE,GEM,DEM,GZET,DZET,GL1,DL1,GL2,DL2,-
GD1,DD1,GD2,DD2
BLOK(0):LIC1,LIC2,MBE,MEM,MZET,ML1,ML2,MD1,MD2,MO
BLOK(249):S
1A)BE=LOS(.)
EM=LOS(.)
ZET=LOS(.)
L1=LOS(.)
L2=LOS(.)
D1=LOS(.)
D2=LOS(.)
LIC1=LIC1+1
GDY KLUCZ 1:1B,INACZEJ NASTEPNY
BER=(GBE-DBE)xBE+DBE
EMR=(GEM-DEM)xEM+DEM
ZETR=(GZET-DZET)xZET+DZET
L1R=(GL1-DL1)xL1+DL1
L2R=(GL2-DL2)xL2+DL2
D1R=(GD1-DD1)xD1+DD1
D2R=(GD2-DD2)xD2+DD2
GDY (BERxEMR*2xZETR*2-530.)>0:NASTEPNY,INACZEJ 1A
GDY (BERxEMR*2xZETR-12.)>0:NASTEPNY,INACZEJ 1A
GDY (ZETR-17.)>0:NASTEPNY,INACZEJ 1A
GDY (BER-5xEMR)>0:NASTEPNY,INACZEJ 1A
GDY (12xEMR-BER)>0:NASTEPNY,INACZEJ 1A
GDY (40-EMRxZETR)>0:NASTEPNY,INACZEJ 1A
GDY (D1R*3-PWK(((0.0625xL1R)/EMRxZETR)+1700))>0:NASTEPNY,INACZEJ 1A
GDY D2R-5>0:NASTEPNY,INACZEJ 1A
GDY ((L1R*3/EMRxZETRxD1R*4)-192)>0:NASTEPNY,INACZEJ 1A
GDY ((L2R*3/EMRxZETRxD2R*4)-192)>0:NASTEPNY,INACZEJ 1A
OBJ=4xEMRxBERxZETR+D1R*2xL1R+D2R*2xL2R
GDY OBJ>950:1A,INACZEJ NASTEPNY
LIC2=LIC2+1
I=ENT(OBJ)
S(I)=S(I)+1
```

```
GDY OBJ>MO:1A,INACZEJ NASTEPNY
MBE=BER
MEM=EMR
MZET=ZETR
ML1=L1R
ML2=L2R
MD1=D1R
MD2=D2R
MO=OBJ
IDZ DO ROZDZIALU:2
1B) IDZ DO ROZDZIALU:3
PODPROGRAM:LOS(W)
JEZYK SAS
UM.958
MN.1B)
PM.958
UA.958
PW 20
SK*LOS(3)
1B)) -81713
     +37612
JEZYK SAKO
KONIEC ROZDZIALU

L1*4
ROZDZIAL:2
CALKOWITE:LIC1,LIC2,I,ZETR,S
SKALA DZIESIETNA PARAMETROW:6
USTAW SKALE DZIESIETNIE:6
BLOK(0):W
BLOK(0):GBE,DBE,GEM,DEM,GZET,DZET,GL1,DL1,GL2,DL2,GD1,DD1,GD2,DD2
BLOK(0):LIC1,LIC2,MBE,MEM,MZET,ML1,ML2,MD1,MD2,MO

BLOK(249):S
DRUKUJ(3.1):MBE,MEM
DRUKUJ(3.0):MZET
DRUKUJ(3.1):ML1,ML2,MD1,MD2,MO
DRUKUJ(5):LIC1,LIC2
LINIA
IDZ DO ROZDZIALU:1
KONIEC ROZDZIALU
```

ON APPROXIMATION OF MONTE-CARLO METHOD FOR SYNTHESIS OF MECHANISMS

Summary

The Monte-Carlo method has been applied for the evaluation of optimal dimensions of gearboxes. Principles of the method are explained, techniques of generating of pseudo-random numbers-presented, and the flow-diagram of ZAM 2 computer program - given.

The optimum design program, written in SAKO-code, is presented for one particular example, as well as the results of computations performed on a digital computer ZAM 2.

Instytut Maszyn Matematycznych
Algorytmy Nº 4

519.2:518.6

UWAGI O PEWNYM GENERATORZE KWADRATOWYM

Henryk GAJEWSKI
Ryszard ZIELIŃSKI

Pracę złożono 11.09.1964 r.

Dla otrzymania liczb losowych o rozkładzie równomiernym na odcinku $(0,1)$ stosuje się zwykle w maszynie ZAM-2 generator, który ma następującą postać w języku SAS [1]:

UM.R
MN.C
PM.R

gdzie: R - kolejne liczby pseudolosowe oraz C - pewna stała. Jeżeli:

$$R_0 \equiv 234.642.457.001$$
$$C \equiv 115.354.631.461$$

to generator ten nazywamy krótko generatorem RNA.

Generator RNA pozwala na uzyskanie poprawnych liczb losowych o rozkładzie równomiernym na odcinku $(0,1)$, ale losowość występowania różnych cyfr binarnych na poszczególnych pozycjach liczb uzyskiwanych z tego generatora nie była - jak nam wiadomo - analizowana. Co więcej, można zaobserwować pewną systematyczność w występowaniu określonego ciągu bitów na niektórych pozycjach liczb losowych.

Jeżeli p-tą liczbę uzyskaną z generatora RNA zapiszemy w postaci oktalowej:

$$R_p = q_{11}^p q_{10}^p q_9^p . q_8^p q_7^p q_6^p . q_5^p q_4^p q_3^p . q_2^p q_1^p q_0^p ,$$

to $q_0^p \equiv 1$ dla każdego p oraz

$$q_1^p \equiv \begin{cases} 0 & \text{dla } p = 0,4,8,\ldots \\ 6 & \text{dla } p = 1,5,7,\ldots \\ 4 & \text{dla } p = 2,6,10,\ldots \\ 2 & \text{dla } p = 3,7,11,\ldots \end{cases}$$

Oznacza to, że 6 ostatnich cyfr w zapisie binarnym powtarza się z okresem równym 4, natomiast ostatnie trzy cyfry binarne są zawsze jednakowe.

W niektórych zastosowaniach może to czynić omawiany generator nieprzydatnym.

Von Neuman zaproponował, opisany w pracach [2] i [3], kwadratowy*) generator liczb pseudolosowych o rozkładzie równomiernym. Na maszynie ZAM-2 generator ten – oznaczymy go krótko przez KGA – zrealizowano za pomocą ciągu rozkazów w języku SAS:

UM.R
MN.R
PW17
PM.R

przyjmując jako wartość początkową R = 234.642.457.001

Zwracamy uwagę na właściwości generatora KGA:

1. Dla każdego $p \geqslant 109938$ jest:

$$R_p \equiv 000.000.400.000$$

2. Nie ma dwóch takich różnych wskaźników $p,q < 109938$, aby

$$R_p \equiv R_q$$

Oznacza to, że wszystkie liczby losowe o numerze kolejnym mniejszym od 109938 są różne.

Z powyższego wynika, że generator ten może być wykorzystany tylko w takich przypadkach, gdy mniejsza od 109938 ilość liczb losowych jest wystarczająca.

*) Termin "generator kwadratowy" został przez nas przyjęty jako odpowiednik terminów angielskich "Middle Square" lub krótko "Mid-Square". Generator ten podany jest również w pracy [4] na str. 31.

Literatura

1. Maszyna ZAM-2, Opis maszyny, Kompedium programowania w języku SAS, opracowanie K. Piałkowski, J. Swianiewicz, Prace ZAM PAN, Warszawa 1962:C3.

2. METROPOLIS N: Phase Shifts-Middle Squares-Wave Equation, Symposium on Monte Carlo Methods, March 16 and 17, 1954.

3. TAUSSKY O., TODD J.: Generation and Testing of Pseudo-Random Numbers, Symposium on Monte Carlo Methods, March 16 and 17, 1954.

4. BUSLIENKO N.P. i in.: Metod statistiĉeskich ispytanij /metod Monte-Karlo/, Moskwa 1962.

NOTES ON A CERTAIN MIDDLE-SQUARE GENERATOR

Summary

In the paper a middle-square generator for ZAM-2 computer is discussed. Two properties of this generator with the inital value R_0 = 234.642.457.001 /in octal form/ are presented, namely, all random numbers R_p with $p \geqslant 109\ 938$ are equal to 000.000.400.000 and all random numbers R_p with $p < 109\ 938$ are different.

Instytut Maszyn Matematycznych
Algorytmy

681.14-523.8:519.2

TECHNIKA STOSOWANIA METOD MONTE-CARLO NA MASZYNACH CYFROWYCH

Elżbieta PLESZCZYŃSKA

Pracę złożono 16.10.1964 r.

Praca zawiera przegląd metod generowania na maszynie cyfrowej liczb losowych i realizacji zjawisk losowych, wykorzystywanych we wszelkich obliczeniach Monte-Carlo. W artykule, który ukaże się w następnym numerze Algorytmów, podana będzie dokumentacja niektórych metod generowania opracowanych dla ZAM-2.

1. WSTĘP

Wśród metod numerycznych szczególne miejsce zajmują metody Monte-Carlo. Są to metody przybliżonego rozwiązywania danego zagadnienia matematycznego lub fizycznego przy użyciu techniki statystycznego pobierania próby. Dokładniej mówiąc, dla rozwiązania danego zagadnienia buduje się i bada odpowiedni model statystyczny tak, aby uzyskać metodami statystycznymi estymację poszukiwanego rozwiązania. Jako przykład może służyć obliczanie całek oznaczonych, w szczególności całek wielokrotnych, które w metodach Monte-Carlo zostają potraktowane jako wartości oczekiwane odpowiednio definiowanych zmiennych losowych; dokonuje się losowania N niezależnych wartości tych zmiennych losowych, po czym ich średnią arytmetyczną przyjmuje się jako przybliżoną wartość całki.

Metoda taka jest powszechnie stosowana przy obliczaniu całek wielokrotnych.

Budowa i badanie modelu statystycznego w metodzie Monte-Carlo odbywa się za pomocą zwykłych środków obliczeniowych, a więc ołówka, papieru, tablic i różnych specjalnych urządzeń oblicze-

niowych, poczynając od suwaka logarytmicznego, kończąc na maszynie cyfrowej czy analogowej. Trzeba zatem wypracować metody statystycznego pobierania próby /a więc otrzymywania wartości zmiennych losowych i realizacji procesów losowych/ za pomocą owych zwykłych środków obliczeniowych.

Do takich ogólnie znanych metod należy posługiwanie się tablicami liczb losowych. Tablica liczb losowych o danym rozkładzie jest zbiorem wartości zmiennej losowej o tym rozkładzie i ma stanowić próbkę prostą. Hipotezę tę sprawdza się przy pomocy różnych testów statystycznych.

Zasadnicze znaczenie ma jednak wypracowanie metod statystycznego pobierania próby na szybkich maszynach liczących, w szczególności zaś na maszynach cyfrowych, które okazały się najbardziej przydatne do stosowania metod Monte-Carlo.

Najprostsza z tych metod polega na wprowadzeniu odpowiednich tablic liczb losowych do pamięci maszyny. Ze względu na ograniczoną ilość miejsc w pamięci operacyjnej z konieczności trzeba albo zdecydować się na wprowadzenie ich do innych dostępnych pamięci, którymi rozporządza dana maszyna /w szczególności bębnowej/, albo wczytywać liczby losowe z tablicy bardzo małymi partiami. Oba te sposoby są czasochłonne, a poza tym nie zawsze są opracowane odpowiednie tablice, zawierające dostateczny zasób liczb losowych.

Dlatego znacznie większą rolę odgrywają metody tak zwanego generowania liczb losowych i realizacji zjawisk losowych.

Niniejszy artykuł zawiera krótki przegląd tych metod, opracowany na podstawie cytowanych dalej źródeł. W języku polskim dotychczas nie ma takich opracowań, dlatego przegląd ten wydaje się potrzebny. Stanowi on wstęp do kolejnego artykułu dotyczącego tych zagadnień, zatytułowanego "Technika stosowania metod Monte-Carlo na ZAM-2", a przeznaczonego przede wszystkim dla tych użytkowników maszyn ZAM-2, którzy chcieliby przeprowadzić na niej obliczenia stosując metody Monte-Carlo, znają zaś jedynie programowanie w języku SAKO. Artykuł ten zawiera dokładne opisy z dokumentacją w SAS poszczególnych metod generowania opracowanych dotąd dla

ZAM-2, wraz z objaśnieniem sposobu korzystania z tych opracowań. Przeczytanie obu artykułów ma dać możność samodzielnego stosowania metod Monte-Carlo na maszynie ZAM-2.

2. GENEROWANIE LICZB LOSOWYCH O ROZKŁADZIE RÓWNOMIERNYM

Przy wszelkiej generacji na maszynie cyfrowej zasadniczą rolę odgrywa /jak pokażemy w 3 i 5/ źródło liczb losowych o rozkładzie równomiernym.

Zdarzają się maszyny wyposażone w takie źródło przez konstruktora. Istnieją więc rozmaitego rodzaju urządzenia, które przekazują do pamięci maszyny ciągi, złożone z niezależnych jednakowo-prawdopodobnych zer i jedynek; taki ciąg o ustalonej długości tworzy dwójkowe rozwinięcie liczby losowej o rozkładzie równomiernym na odcinku /0.1/. Urządzenia takie wykorzystują do tworzenia owego ciągu bitów np. szumy własne w lampach, procesy radioaktywne itp. Niektóre z tych urządzeń powodują, że w regularnych odstępach czasu pojawia się liczba losowa o rozkładzie równomiernym w specjalnym rejestrze lub ustalonej komórce pamięci maszyny. Bliżej opisano te zagadnienia w [2], str. 225 i 248.

Jeśli maszyna nie posiada takich urządzeń, wykorzystujących fizyczne zjawiska losowe, wyposaża się ją w tzw. generator liczb pseudolosowych, o następującej definicji. Rozważmy ciąg, którego wyrazy określone są rekurencyjnie:

$$r_{i+1} = f(r_0, \ldots, r_i),$$

zależą więc jedynie od postaci funkcji f i od wyrazów poprzednich, w szczególności od wyrazu r_0, i przyjmują wartości z przedziału $(0,1)$. Przy odpowiednim doborze funkcji f i wyrazu początkowego można otrzymać skończony ciąg $r_0, \ldots, r_n$, taki, że niektóre jego właściwości są analogiczne do $(n+1)$ - elementowej próbki prostej z populacji o rozkładzie równomiernym na odcinku $(0,1)$. Jeśli zastosować do takiego ciągu testy statystyczne badające ową hipotezę i jeśli te testy nie pozwolą nam jej odrzucić, wówczas ciąg ten nazywamy ciągiem liczb pseudolosowych o

rozkładzie równomiernym na odcinku $(0,1)$. Przykłady tego rodzaju ciągów można znaleźć w [1], skąd zaczerpnięto też ciąg używany do generowania na maszynie ZAM-2, opisany w [7]. Generatorem liczb pseudolosowych o rozkładzie równomiernym na odcinku $(0,1)$ - zwanym odtąd generatorem R - nazywamy program, po odwołaniu się do którego maszyna oblicza i rejestruje w pamięci jako wynik kolejny element wspomnianego wyżej rekurencyjnego ciągu. Po n-krotnym odwołaniu się do tego programu otrzymuje się n elementów ciągu.

Literatura dotycząca tworzenia i właściwości generatorów R stale rośnie /patrz np. [1] z 1954 r., [5] z 1963 r./.

3. GENEROWANIE LICZB LOSOWYCH O DANYM ROZKŁADZIE

Generatorem liczb losowych o danym rozkładzie nazywamy program, po odwołaniu się do którego maszyna oblicza i rejestruje w pamięci jako wynik liczbę należącą do zbioru wartości zmiennej losowej o tym rozkładzie. Po n-krotnym odwołaniu się do tego programu otrzymuje się ciąg liczb, który ma stanowić n-elementową próbkę prostą z populacji generalnej o tym rozkładzie.

Pokażemy, że mając dany generator R można skonstruować generator liczb losowych o dowolnym zadanym rozkładzie. Trzeba tu dodać, że konsekwencją "pseudolosowości" generatora R jest "pseudolosowość" generatorów skonstruowanych przy jego pomocy. Związane z tym problemy są omówione w następnym paragrafie. W tym paragrafie rozpatrzymy zagadnienie konstrukcji generatora liczb losowych o danym rozkładzie przy założeniu, że rozporządzamy idealnym generatorem R .

Niech więc R oznacza zmienną losową o rozkładzie równomiernym na odcinku $(0,1)$, $\{R_i\}$ - ciąg niezależnych zmiennych losowych o rozkładzie takim samym jak rozkład zmiennej losowej R , $\{r_i\}$ - ciąg wartości zmiennych losowych R_i .

Zauważmy na wstępie, że w maszynie cyfrowej operujemy tylko liczbami wymiernymi, że zatem liczby r_i , którymi rozporządzamy, są właściwie wartościami zmiennych losowych R_i^* , zwanych quasi-

równomiernymi, przybierających skończoną liczbę jednakowo-prawdopodobnych wartości $0, \frac{1}{2^k}, \ldots, \frac{2^k-1}{2^k}$, gdzie k jest ilością bitów w rozwinięciu binarnym liczby. Wartością średnią takich zmiennych losowych jest $\frac{1}{2}\left(1 - \frac{1}{2^k}\right)$, zaś wariancją $\frac{1}{12}\left(1 - \frac{1}{4^k}\right)$, podczas gdy dla R_i mamy odpowiednio $\frac{1}{2}$ i $\frac{1}{12}$. Spowodowane tym skutki mogą niekiedy mieć istotne znaczenie. Zagadnienie to omówiono w [2], str. 282. W niniejszej pracy poprzestajemy na zasygnalizowaniu tej kwestii i w dalszym ciągu nie uwzględniamy wynikłych stąd skutków.

Jedna z ogólnych metod konstrukcji generatora X liczb losowych x_i, będących wartościami ciągłej jednowymiarowej zmiennej losowej X o zadanej gęstości $f(x)$, oparta jest na twierdzeniu, że zmienna losowa $R = \int_{-\infty}^{X} f(t)dt$ ma rozkład równomierny na odcinku $(0,1)$. Z tego związku można wyznaczyć zmienną losową X jako funkcję zmiennej losowej R, i w ten sposób przyporządkować każdej wartości r_i wartość x_i spełniającą równanie

$$r_i = \int_{-\infty}^{x_i} f(t)dt \qquad / i = 1,2,\ldots / \qquad /1/$$

Ten ogólny sposób może niekiedy być jednak bardzo czasochłonny i dlatego nie można poprzestać na podaniu tego wzoru.

Weźmy jako przykład zmienną losową X o rozkładzie wykładniczym. Mamy wtedy

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{dla } x > 0 \\ 0 & \text{dla pozostałych,} \end{cases}$$

a zatem wzór /1/ przybiera postać:

$$r_i = \lambda \int_0^{x_i} e^{-\lambda t}\, dt ,$$

skąd

$$x_i = -\frac{1}{\lambda} \ln\left(1 - r_i\right)$$

Pozostawiamy dociekaniom czytelnika, dlaczego równie dobrze można stosować wzór

$$x_i = -\frac{1}{\lambda} \ln r_i$$

W tym więc przypadku zmienna losowa X dała się łatwo przedstawić w postaci funkcji zmiennej losowej R . Mimo to jednak w praktyce nie powinno się stosować takiej metody generacji liczb losowych o rozkładzie wykładniczym, gdyż można podać algorytm prostszy, opisany i uzasadniony w [4], a przytoczony w [7]. Można także skorzystać z faktu, że zmienna losowa o rozkładzie wykładniczym z $\lambda = 1$ daje się przedstawić jako połowa sumy kwadratów dwóch zmiennych losowych o rozkładzie normalnym ze średnią zero i wariancją 1. Jeśli zatem przyjąć, że dany jest odpowiedni ciąg liczb losowych o rozkładzie normalnym $\{n_j\}$, to obliczamy poszukiwaną liczbę x_i wg wzoru:

$$x_i = \frac{1}{2}\left(n_{2i}^2 + n_{2i+1}^2\right)$$

Metoda ta jest jednak bardzo czasochłonna.

Generacji liczb losowych o rozkładzie wykładniczym poświęcono tu więcej miejsca z kilku powodów: po pierwsze - rozkład wykładniczy występuje często w różnych zagadnieniach; po drugie - liczby losowe o tym rozkładzie stanowią punkt wyjścia np. przy generacji liczb losowych o rozkładzie Rayleigh'a /patrz [7], punkt 4.2.g/, czy realizacji procesu Poissona /[6]/; po trzecie - na tym przykładzie widać wyraźnie, jak wiele jest możliwości generowania i z jak wielu punktów widzenia trzeba te możliwości oceniać, przy czym ważna jest ocena czasu generowania.

Przykładem zmiennej losowej ciągłej, dla której korzystanie z równania /1/ przy generacji byłoby bardzo niewygodne ze względu na postać jej dystrybuanty, jest zmienna losowa o rozkładzie normalnym. Podana dalej metoda generacji liczb losowych n_i o rozkładzie normalnym $N(0,1)$ jest zarazem przykładem zastosowania

twierdzeń granicznych, w tym przypadku centralnego twierdzenia granicznego, na podstawie którego w szczególności suma n niezależnych zmiennych losowych o jednakowym rozkładzie ze skończoną wariancją ma rozkład asymptotycznie normalny. Jako owe niezależne zmienne losowe można by było przyjąć zmienne losowe R_i. Suma n tych zmiennych ma średnią $\frac{n}{2}$ i odchylenie średnie $\frac{\sqrt{n}}{2\sqrt{3}}$. Aby otrzymać zmienną losową o rozkładzie asymptotycznie normalnym $N(0,1)$, trzeba tę sumę standaryzować. Manipulacje przy standaryzowaniu stają się prostsze, jeśli zamiast R_i użyć zmiennych losowych R_i' o rozkładzie równomiernym na odcinku $(-1,1)$. Suma n zmiennych R_i' ma wartość średnią zero i odchylenie średnie $\sqrt{\frac{n}{3}}$. Przy generacji liczb n_i przyjmujemy $n = 12$, i wtedy odchylenie średnie jest równe 2. Ciąg liczb z generatora R, używany przy generacji, oznaczymy tutaj przez r_{ij} /$j = 1,2,...12$, $i = 1,2,.../$. Stąd $n_i = \frac{1}{2}\sum_{j=1}^{12}\left(2r_{ij} - 1\right)$. Z kolei liczby losowe o rozkładzie normalnym pozwalają nam generować np. liczby losowe o rozkładzie Rice'a /artykuł na ten temat ma się ukazać w Nr. 5 Algorytmów/.

Zwróćmy jeszcze uwagę na fakt, że otrzymane tą metodą liczby losowe n_i są ograniczone, mianowicie $|n_i| \leqslant 6$ /bezwzględna wartość sumy dwunastu wartości zmiennych R_i' nie może przekroczyć 12/, nie stanowi to jednak żadnego niedostatku tego generatora, gdyż dla X o rozkładzie $N(0,1)$ $P_r\left(|X| > 4\right) < 0{,}0003$.

Przejdę teraz do omówienia generatorów liczb losowych o rozkładzie skokowym.

Niech będzie dana zmienna losowa X o rozkładzie

$$P_r\left(X = j\right) = p_j \qquad \left(j = 0,1,2, ...\right)$$

Prawdopodobieństwo, że zmienna losowa X przyjmie wartość j

$\left(\text{równe} \quad \sum_{k=0}^{j} p_k - \sum_{k=0}^{j-1} p_k\right)$ jest równe prawdopodobieństwu, że zmienna losowa R o rozkładzie równomiernym na odcinku $(0,1)$ zawarta jest w przedziale $\left(\sum_{k=0}^{j-1} p_k \, , \sum_{k=0}^{j} p_k\right)$:

$$Pr\,(x=j) = Pr\left(\sum_{k=0}^{j-1} p_k \leqslant R \leqslant \sum_{k=0}^{j} p_k\right)$$

Zatem kolejnym wartościom r_i $(i = 1,2, \ldots\ldots)$ przyporządkowuje się taką wartość j , dla której:

$$\sum_{k=0}^{j-1} p_k \leqslant r_i \leqslant \sum_{k=0}^{j} p_k \, ,$$

przy czym przyjmuje się, że $\sum_{k=0}^{-1} p_k = 0$.

Warto zauważyć, że reguła ta dla rozkładu skokowego stanowi odpowiednik równania /1/ , które stosuje się dla rozkładu ciągłego.

W ten sposób generuje się np. liczby losowe o rozkładzie otrzymanym empirycznie i zadanym za pomocą histogramu. W ten też sposób można generować liczby o rozkładzie Poissona, a mianowicie liczbie r_i przyporządkowujemy taką wartość j , dla której

$$\sum_{k=0}^{j-1} \frac{\lambda^k}{k!} \leqslant e^{\lambda} r_i \leqslant \sum_{k=0}^{j} \frac{\lambda^k}{k!} \, .$$

Inny sposób generacji liczb losowych o rozkładzie Poissona wykorzystuje twierdzenie Poissona, które mówi, że jeśli p_n jest

prawdopodobieństwem zajścia zdarzenia A w pojedynczym doświadczeniu, to prawdopodobieństwo zajścia k zjawisk A w n niezależnych doświadczeniach dąży do $\frac{\lambda^k}{k!} e^{-\lambda}$ przy $n \to \infty$ i $n p_n \to \lambda$.

Obieramy zatem tak duże n , żeby $p_n = \frac{\lambda}{n}$ było dostatecznie małe /w praktyce: $0{,}1 \leqslant p_n \leqslant 0{,}2$ /. Z ciągu $\{r_i\}$ wybieramy n kolejnych liczb $r_{i_1}, r_{i_2}, \ldots, r_{i_n}$, każdą z tych liczb porównujemy z p_n i przyjmujemy ilość przypadków, w których $r_{ij} \leqslant p_n$ $(j = 1, \ldots, n)$, jako liczbę losową o rozkładzie Poissona z parametrem λ .

Na zakończenie należy wspomnieć o generacji wektorów losowych o n składowych, o funkcji gęstości $f(x_1, \ldots, x_n)$. Według [3] , str. 399, najczęściej zadaną gęstość rozkłada się na czynniki postaci:

$$g_1(x_1) \cdot g_2(x_1, x_2) \cdots g_n(x_1, x_2, \ldots, x_n),$$

gdzie $g_1(x_1)$ jest gęstością brzegową rozkładu zmiennej losowej X_1 , daną wzorem

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f(x_1, \ldots, x_n)\, dx_2 \ldots dx_n ,$$

$g_2(x_1, x_2)$ - warunkową gestością brzegową zmiennej losowej X_2 przy założeniu, że $X_1 = x_1$, itd. A więc wartość wektora losowego $x_1, \ldots, x_n$ uzyskuje się wyznaczając kolejno składowe, przy czym każdą z nich otrzymuje się jako wynik losowania jednowymiarowego.

I tak np. liczby losowe (x, y) o rozkładzie normalnym dwuwymiarowym $N(m_x, m_y, \sigma x, \sigma y, \varrho)$ otrzymuje się następująco: najpierw znajduje się liczbę losową x o rozkładzie normalnym $N(m_x, \sigma x)$, a następnie liczbę y o rozkładzie normalnym $N\left(m_y + \varrho \frac{\sigma y}{\sigma x}(x - m_x)\right)$, $\sigma y \sqrt{1 - \varrho^2}$, gdyż, jak wiadomo, taki jest warunkowy rozkład zmiennej losowej Y przy założeniu $X = x$.

4. SPRAWDZANIE GENERATORÓW LICZB PSEUDOLOSOWYCH

Przy sprawdzaniu generatorów liczb pseudolosowych należy odpowiedzieć na pytanie, czy ciąg liczb otrzymany z tego generatora można traktować jako próbkę prostą z populacji o danym rozkładzie /jakkolwiek wiemy, że kolejne jego wyrazy powstają według wzoru matematycznego; tym, których niepokoi takie postępowanie, przypominamy, że podobnie np. producenci sztucznego jedwabiu badają, czy ich tkanina ze względu na swoje właściwości może być używana jako jedwab, chociaż dobrze wiedzą, że powstała bez udziału motyla jedwabnika/. Jeśli metodami statystycznymi nie odrzucamy tej hipotezy, uważamy, że nie ma podstaw do dyskwalifikacji generatora.

Jednakże jednolite metody weryfikacji takiej hipotezy nie istnieją. Wybór testów sprawdzających następuje według uznania twórcy generatora. Stosuje się zwykle kilka różnych testów, zaś moc koniunkcji tych testów nie jest wyznaczana, stąd, znając poziomy istotności poszczególnych testów, niewiele można powiedzieć o poziomie istotności ich koniunkcji.

Wśród mieszaniny stosowanych metod korzystnie wyróżnia się metoda przyjęta w [4] przy sprawdzaniu generowania liczb pseudolosowych o rozkładzie wykładniczym. Autorzy [4] wychodzą z założenia, że należy badać

a/ niezależność występowania w ciągu /aby stwierdzić, czy dany ciąg można traktować jako próbkę prostą z jakiejś populacji/,
b/ zgodność empirycznego rozkładu z postulowanym,
c/ wartości pierwszych momentów.

W tym celu proponują obliczanie w każdej setce liczb wartości pięciu dystrybuant stosownie wybranych funkcji /statystyk/ owej setki liczb /opis tych statystyk przytoczono w [6]/; następnie przeprowadzają dla dystrybuant poszczególnych statystyk testy zgodności z rozkładem równomiernym na odcinku $(0,1)$ /taki bowiem rozkład mają dystrybuanty zmiennych losowych/.

Wydaje się istotne, aby wartości owych dystrybuant udostępnić do wglądu korzystającemu z liczb losowych, który w ten sposób może

wybrać do obliczeń poszczególne ciągi liczb /na podstawie których wyznaczano te wartości/ zgodnie z ideą losowania warstwowego.

Ten rodzaj losowania zilustrujemy przykładem. Przypuśćmy, że poszukiwana jest wartość parametru a, i znany jest sposób konstrukcji nieobciążonego estymatora $\hat{a}$ tego parametru, obliczanego na podstawie N-elementowej próbki prostej z populacji zmiennej losowej X. Niech $\sigma_{\hat{a}}$ oznacza odchylenie standartowe estymatora $\hat{a}$.

Jako źródło N niezależnych wartości zmiennej losowej X służą nam liczby z odpowiedniego generatora liczb losowych. Zwykłe losowanie polega na wzięciu N kolejnych liczb. Przypuśćmy jednak, że w poszczególnych setkach liczb generatora znamy wartości pewnej statystyki tych liczb. Można wtedy wykorzystać tę informację stosując losowanie, które stanowi szczególny przypadek losowania warstwowego. Opiszemy je dla $N = 200$. Wybieramy wtedy do próbki jedną setkę liczb, w której wartość owej statystyki jest mniejsza od mediany rozkładu tej statystyki, oraz jedną setkę, w której jest ona od mediany większa. Inaczej można to ująć tak: w jednej setce wartość dystrybuanty obliczanej statystyki ma być mniejsza od $\frac{1}{2}$, w drugiej setce - większa od $\frac{1}{2}$. Łatwo uogólnić podaną regułę losowania warstwowego na przypadek kilku setek w próbce, a także kilku statystyk. Wynika stąd, że losowanie warstwowe jest bardzo ułatwione, jeśli poszczególnym setkom liczb przyporządkowuje się wartości dystrybuant obliczanych statystyk, jak to ma miejsce w [4].

Można pokazać w wielu konkretnych przykładach, że estymator parametru a obliczany na podstawie tak wybranej próbki pozostaje nieobciążony i ma na ogół odchylenie średnie mniejsze od $\sigma_{\hat{a}}$. Przy odpowiednim przygotowaniu generatora losowanie warstwowe w nieznacznym tylko stopniu komplikuje organizację obliczeń.

Omówimy teraz trochę szerzej punkt a., gdyż wydaje się, że badanie niezależności jest powszechnie mniej znane niż badania postulowane w punktach b. i c.. Otóż najczęściej stosuje się któryś z testów tzw. teorii serii /np. bada się ilość serii wyrazów ciągu mniejszych od mediany czy wartości średniej w populacji, śre-

dnią długość serii wyrazów malejących w ciągu, itp./. Jednocześnie stosuje się jakiś test oparty na autokorelacji. Jeden z takich testów dla ciągu niezależnych zmiennych losowych X_i o jednakowych rozkładach normalnych podano w [8]. Wyznaczono tam rozkłady współczynników autokorelacji ϱ_k, zdefiniowanych następująco:

$$\varrho_k = \frac{X_1X_{k+1}+X_2X_{k+2}+\ldots+X_{n-k}X_n+X_{n-k+1}X_1+\ldots+X_nX_k - \frac{\left(\sum_{i=1}^{n} X_i\right)^2}{n}}{\sum_{i=1}^{n} X_i^2 - \frac{\left(\sum_{i=1}^{n} X_i\right)^2}{n}}$$

$$\text{dla } k = 1,2, \ldots,n-1$$

Asymptotyczny rozkład dla $k=1$ jest rozkładem normalnym ze średnią i wariancją równymi $\frac{1}{n-1}$.

Dla ciągu niezależnych zmiennych losowych o rozkładzie równomiernym na odcinku $(0,1)$ łatwo pokazać, że współczynnik a_k, dany wzorem

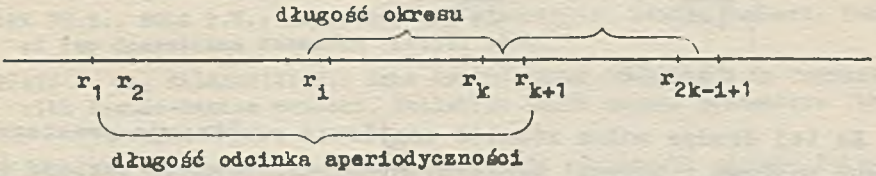
$$a_k = \sum_{i=1}^{\left[\frac{n}{k}\right]} R_{ki-k-1}R_{ki} \qquad \text{/n parzyste, } k=1,2,\ldots,n\text{/}$$

ma asymptotycznie dla dużych $\left[\frac{n}{k}\right]$ rozkład normalny o średniej $0,25\left[\frac{n}{k}\right]$ i wariancji $0,05\left[\frac{n}{k}\right]$. Można też łatwo wyznaczyć asymptotyczne rozkłady tego współczynnika dla ciągów zmiennych losowych o różnych innych rozkładach. Wartości współczynnika a_k oblicza się na maszynie cyfrowej szczególnie prosto.

Ponieważ generator liczb pseudolosowych o rozkładzie równomiernym /generator R/ stosuje się do budowy generatorów liczb pseudolosowych o wszelkich innych rozkładach, sprawdzenie go ma podstawowe znaczenie. Jeśli generator R jest dość wszechstronnie sprawdzony, można już nie badać budowanych generatorów. Ponieważ

jednak w praktyce nie rozporządzamy na ogół bardzo dobrymi generatorami R , badanie takie jest potrzebne.

W generatorach R trzeba dodatkowo badać tzw. długość okresu aperiodyczności. Wiadomo bowiem, że ciągi liczb $\{r_i\}$ otrzymane na podstawie różnych wzorów rekurencyjnych są z reguły ciągami okresowymi o następującej strukturze:



tzn. pierwsze k liczb są różne między sobą, zaś liczba r_{k+1} jest równa liczbie r_i $(1 \leqslant i \leqslant k)$, i ogólnie

$$r_{i+j+n(k-i+1)} = r_{i+j} \qquad \begin{matrix} j = 0,1,2,\ldots,k-i \\ n = 1,2,\ldots \end{matrix}$$

Oczywiście, do obliczeń można wykorzystywać tylko owe pierwsze k liczb. Wielkość k możemy wyznaczyć empirycznie, za pomocą specjalnego programu; są prace omawiające teoretyczne sposoby wyznaczania okresu.

5. GENEROWANIE REALIZACJI ZDARZEŃ LOSOWYCH I PROCESÓW STOCHASTYCZNYCH.

Jeśli chcemy w maszynie cyfrowej symulować wystąpienie jakiegoś zdarzenia A z prawdopodobieństwem p , generujemy liczbę r_i z generatora R i badamy, czy liczba r_i jest mniejsza od p , które to zdarzenie realizuje się również prawdopodobieństwem p:

$$P(R<p) = \int_0^p dt = p .$$

Jeśli $r_i < p$, twierdzimy, że zdarzenie A zrealizowało się.

Analogicznie twierdzimy, że zrealizowało się k-te z ciągu niezależnych zdarzeń losowych A_1, A_2, ..., o prawdopodobieństwach $p_j = P(A_j)$, jeśli wygenerowana liczba r_i spełnia nierówność:

$$\begin{cases} \sum_{j=1}^{k-1} p_j \leqslant r_i \leqslant \sum_{j=1}^{k} p_j & \text{dla } k = 2, 3, \ldots, \\ r_i < p_1 & \text{dla } k = 1. \end{cases} \qquad /2/$$

Na tej drodze można również znajdować realizacje prostego łańcucha Markowa o zadanej macierzy prawdopodobieństw przejść (p_{ij}): skoro wiemy, że układ dotąd znajdował się na przykład w stanie i_o ze zbioru możliwych stanów, przyjmujemy, że układ będzie w stanie k, jeśli wygenerowana liczba r_i spełnia odpowiednią z nierówności /2/, w których p_j zastąpiono przez $p_{i_o j}$. W tym przypadku realizacją procesu jest ciąg numerów przyporządkowanych poszczególnym stanom w łańcuchu Markowa. Ogólnie, przy generowaniu realizacji procesu stochastycznego aproksymujemy w maszynie cyfrowej funkcję należącą do zbioru realizacji tego procesu, dla skończonego przedziału parametru procesu. Jeśli realizacja procesu jest funkcją schodkową o skończonej ilości i wielkości skoków w każdym skończonym przedziale, jak to ma miejsce np. dla procesu Poissona, to może być reprezentowana w maszynie przez pary liczb (t_i, x_{t_i}), $i = 1, 2, \ldots$, gdzie t_i - wartości parametru, przy których następuje i-ty skok, zaś x_{t_i} - wartości procesu X_t dla $t = t_i$.

Dla jednorodnych procesów sygnałowych o przyrostach niezależnych przy danej intensywności sygnałów λ odległość dwu kolejnych sygnałów ma rozkład wykładniczy z parametrem λ, i wtedy generowanie realizacji polega na generowaniu odległości między skokami według tego rozkładu, oraz znajdowaniu wartości procesu w punkcie t_i (patrz [6]).

Literatura

1. MAYER H.A.: Symposium on Monte-Carlo Method, John Wiley and Sons, N.York - London, 1954.

2. BUSLENKO N. P., GOLENKO D.I., SOBOL I.M., SRAGOWICZ W.G., SZREJDER J.A.: Metod statističeskich ispytanij /metod Monte-Carlo/, Moskwa 1962.

3. BROWN W.: Metody Monte-Carlo, Nowoczesna matematyka dla inżynierów pod redakcją E.F.Beckenbacha, Warszawa 1962:I, 12.

4. CLARK CH.E., HOLZ B.W.: Exponentially Distributed Random Numbers, Published for Operations Research Office, 1960.

5. MATTEIS A. de, PALESCHINI B.: Some Arithmetical Properties in Connection with Pseudo-Random Numbers, Bolletino della unione matematica Italiana, Settembre 1963.

6. PLESZCZYŃSKA E.: Niektóre metody generowania realizacji procesu Poissona, Prace IMM, Algorytmy 1963:1,2

7. PLESZCZYŃSKA E.: Technika stosowania metod Monte-Carlo na ZAM-2 /w przygotowaniu do druku/.

8. ANDERSON R.L.: Distribution of the Serial Correlation Coefficient, The Annals of Mathematical Statistics, 1942:13, 1.

THE MONTE-CARLO TECHNIQUES FOR DIGITAL COMPUTERS

Summary

The paper contains a review of some more important methods of generation of random numbers on digital computers, generally used in Monte-Carlo techniques.

THEORY OF PROGRAMMING
AND
OTHER APPLICATION

Instytut Maszyn Matematycznych
Algorytmy N° 4

681.14-523.8

SORTING BY MEANS OF RANDOM ACCESS STORE

Jan WIERZBOWSKI

Received September 11th 1964.

Two sorting methods by means of random access store are considered. Some details of the effectiveness of this methods are given.

1. INTRODUCTION

Sorting of large volume information is one of the most essential problems which should be solved, when data processing system is considered. The well-known methods use for this purpose magnetic tapes, however sometimes the possibility of sorting without tapes is convenient. This paper describes a method using random access store instead of tapes. This method was programmed on the digital computer ZAM-2, and some details are given.

2. DESCRIPTION OF THE PROBLEM

Let us assume that data to be sorted are grouped into standard blocks $B_1, B_2, \ldots, B_n$. In each block B there is a sorting key K according to which blocks are to be ordered.

We shall write $B_1 < B_2$ when block B_1 precedes block B_2 in the required order, and $B_1 \equiv B_2$ when blocks B_1 and B_2 have the same sorting key. $B_1 \leq B_2$ means that $B_1 < B_2$ or $B_1 \equiv B_2$.

For simplification we shall assume that there are two fixed blocks B_0 and B_{-1}, which satisfy the condition

$$B_{-1} < B_i < B_0 \qquad \text{for} \qquad i = 1, 2, \ldots, n$$

These blocks are stored before sorting.

The problem may be now described as follows: blocks $B_1, B_2, \ldots, B_n$ should be stored in such a way which would enable to read these blocks in the required sequence. This problem will be solved by insertion of the block B_{i+1} into the string of blocks $B_{-1}, B_0, \ldots, B_i$ in a special manner. This insertion should be repeated for $i = 0, 1, \ldots, n-1$.

The block B with address A will be denoted by $B(A)$, and the address of block B will be denoted by $A(B)$. The block directly preceding block B will be called B^{-1}, and block B^1 will follow block B. It means that

$$B^{-1} \leqslant B$$

and if $B^{-1} < B'$ then $B \leqslant B'$

and

$$B \leqslant B^1$$

and if $B' < B^1$ then $B' \leqslant B$

3. STRING METHOD.

To each block B written in the random access store a special label will be assigned, containing the addresses of blocks B^{-1} and B^1. This label denoted by m consists of two parts:

$$m^{-1}(B) = A(B^{-1})$$

and

$$m^1(B) = A(B^1)$$

Blocks written in such a way form two strings as shown in fig. 1.

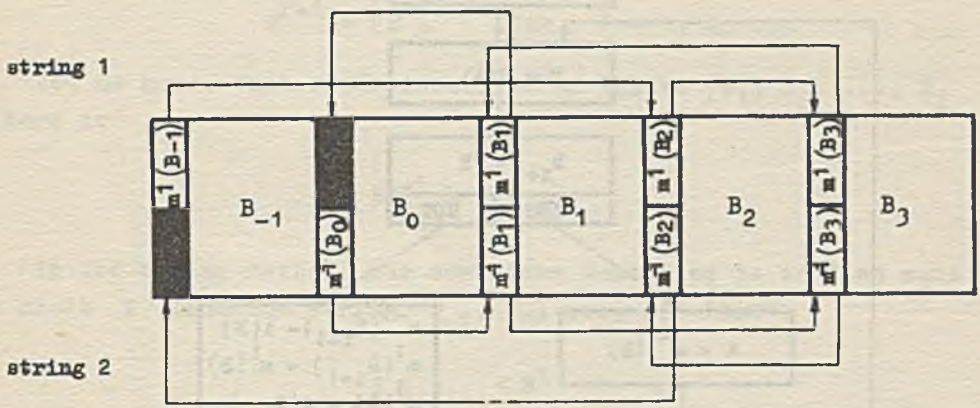


Fig. 1. An example of stored blocks $/B_2 < B_3 < B_1/$.

In the first string the first block is B_{-1}, the last one B_0, and in the second string conversely - block B_0 is the first one and B_{-1} - the last one.

The insertion of block B_{i+1} into string of blocks $B_{-1}, B_0, B_1, \ldots, B_i$ consists of:

a. finding such a block B from the string $B_{-1}, B_0, \ldots, B_i$ that

$$B \leqslant B_{i+1} < B^1$$

b. changing labels of blocks B and B^1

c. writing a labelled block B_{i+1}

Fig. 2 shows an algorithm for the described method.

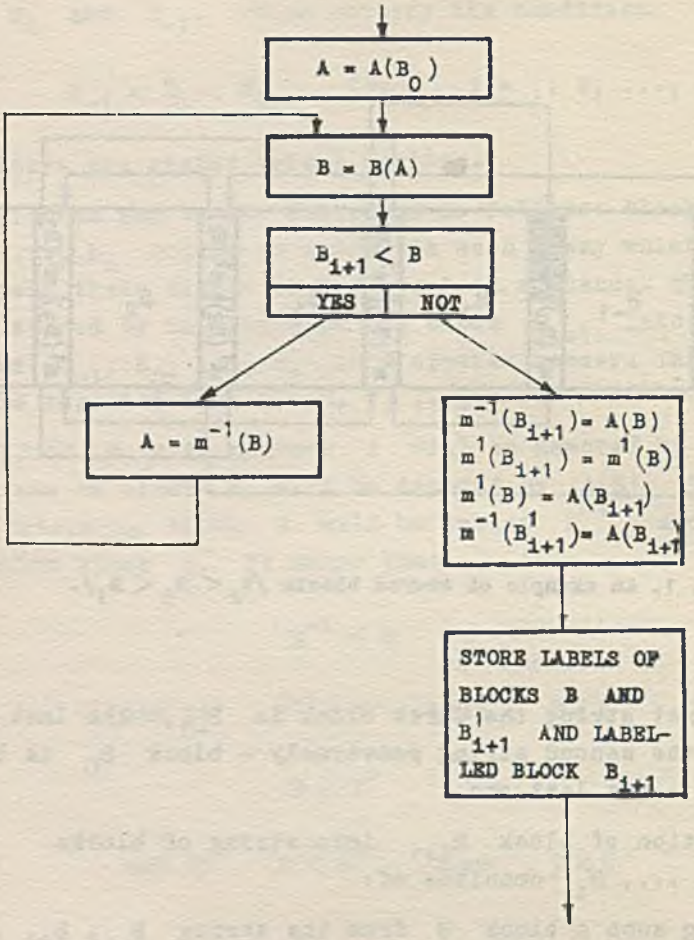


Fig. 2. Algorithm for the string method.

4. THE METHOD OF A STRING WITH "VOCABULARY".

Let us generalize the definition of B^1 as follows

$$B^0 = B$$

and

$$B^{r+1} = (B^r)^1 \quad \text{for} \quad r = 1, 2, \ldots$$

Let us note, that after insertion of blocks $B_1, B_2, \ldots, B_i$ there is

$$B_{-1}^{i+1} = B_0$$

In the string method, the most time consuming is to find such a block B from the string $B_{-1}, B_0, \ldots, B_i$ that

$$B \leqslant B_{i+1} < B^1$$

The time could be shortened if we could divide the whole string $B_{-1}, B_0, \ldots, B_i$ into some substrings. For this purpose let us assume that the computer storage contains a "vocabulary" S consisting of k items $S_1, S_2, \ldots, S_k$. Each item S_p /$1 \leqslant p \leqslant k$/ consists of a sorting key and the address of block B

$$B = B_{-1}^{r_p}$$

where

$$0 = r_0 < r_1 \leqslant r_2 \leqslant \ldots \leqslant r_k = i+1$$

Let us note that if before the insertion of the block B_{i+1} for a certain block B there is

$$B = B_{-1}^{r}$$

after the insertion

$$B = B_{-1}^{r} \quad \text{or} \quad B = B_{-1}^{r+1}$$

It results from this that values $r_1, r_2, \ldots, r_{k-1}$ must be changed from time to time.

We shall assume that the initial value /for $i = 0$/ will be

$$r_1 = r_2 = \ldots = r_k = 1$$

and that after the insertion of $i = \alpha k-1,\ 2\alpha k-1,\ \ldots$ blocks, the vocabulary will be updated according to the formula

$$r_{p+1} - r_p = r_1 \qquad \text{for} \quad p = 1, 2, \ldots, k-1$$

It means that after the insertion of $i = m\alpha k-1$ blocks

$$r_p = m\alpha p = p\,\frac{i+1}{k} \qquad \text{for} \quad p = 1, 2, \ldots, k$$

The vocabulary S being described, we may find the block B which satisfies the condition

$$B \leqslant B_{i+1} < B^1$$

in the following way:

a. we find the block $B_{-1}^{r_p}$ which satisfies the condition

$$B_{-1}^{r_p} \leqslant B_{i+1} < B_{-1}^{r_{p+1}} \qquad 0 \leqslant p < k$$

b. we find the block B from the substring

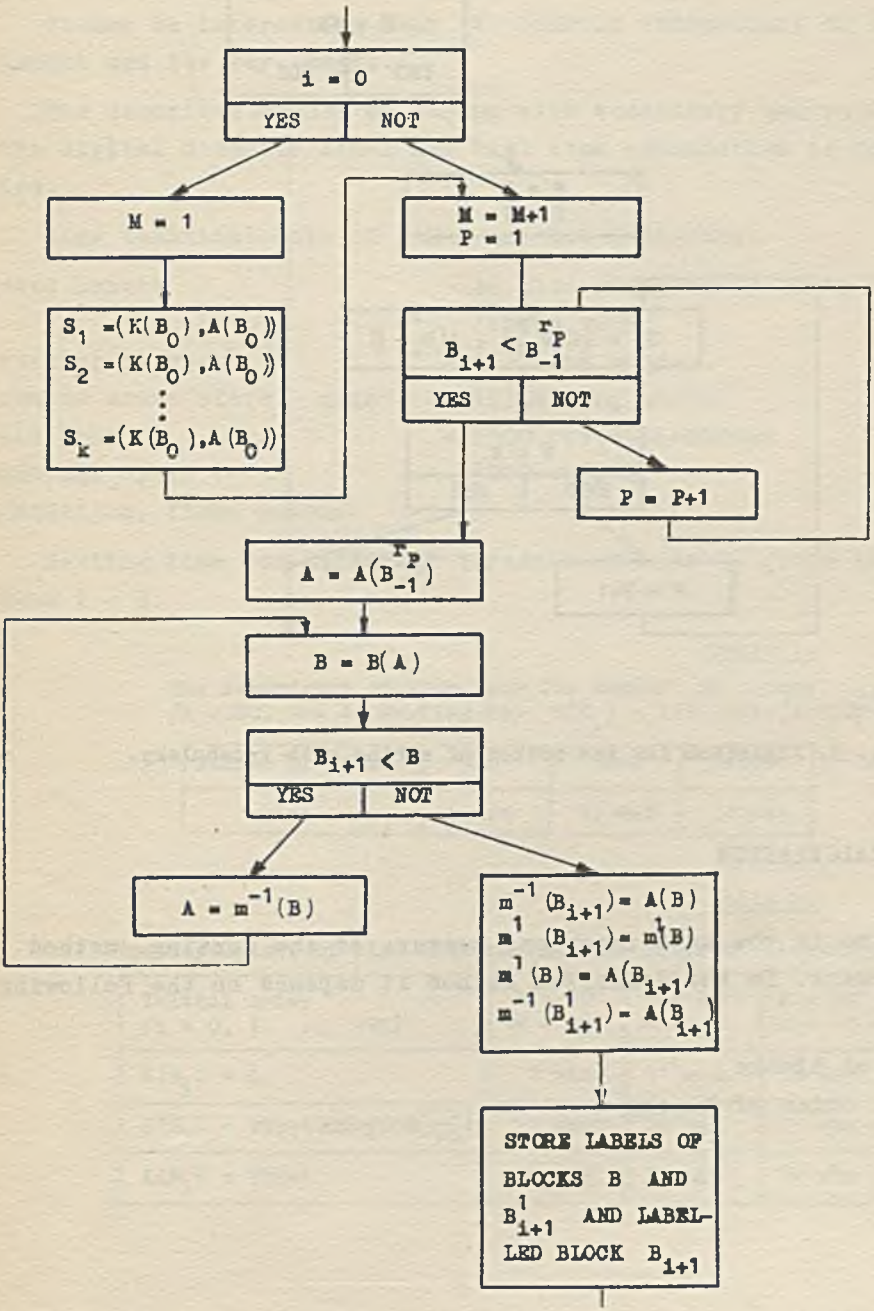
$$B_{-1}^{r_p},\ B_{-1}^{r_p+1},\ \ldots,\ B_{-1}^{r_{p+1}-1}$$

which satisfies the condition

$$B \leqslant B_{i+1} < B^1$$

This method is much faster than the string method.

Fig. 3 shows the algorithm for the method of string with vocabulary.



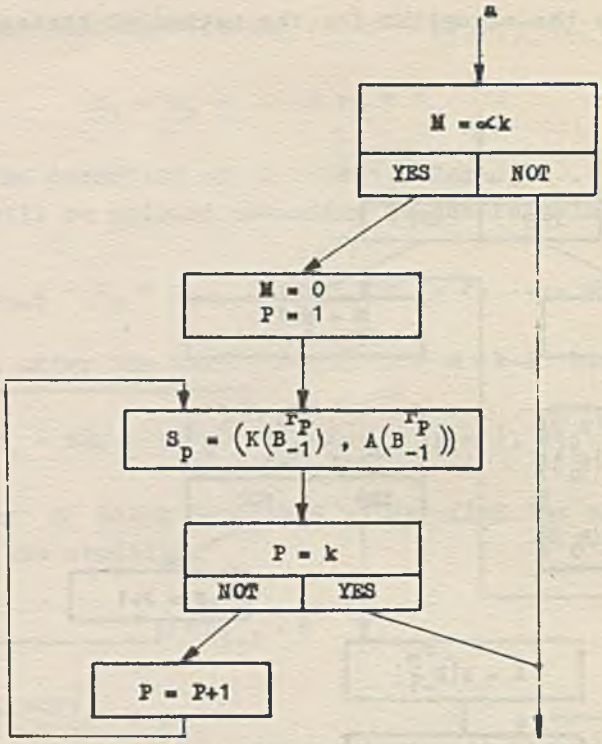


Fig. 3. Algorithm for the method of string with vocabulary.

5. TIME CALCULATION

The time is the most important measure of the sorting method effectiveness. In the described method it depends on the following elements:

- number of blocks
- initial order of blocks

- length of vocabulary /k/
- frequency of vocabulary updating /α/.

It may be interesting that the time is independent of the block length and the key length.

The described method of string with vocabulary was realized on the digital computer ZAM-2 and real time calculation is given below.

Some technical data of ZAM-2 are the following:

word length	- 36 bits /long word/ or 18 bits /short word/
working storage	- 512 long words
random acces store - magnetic drum	- 16384 long words
	- 1500 revolutions/min
average speed /addition, fixed point/	- 1000 op/sec

Sorting time for different variants of data is given in tables 1 - 3.

Table 1.

The dependence of time upon the number of items /k = 80, α = 1, sorting key $K(B_i) = I-1-100i+(I-1)E\left(\frac{100i}{I}\right)$

Number of items	500	1000	1500
time	6 min	13 min	22 min

Table 2.

The dependence of time upon the initial order of blocks.

Initial order /i = 0, 1, ..., 999/	k = 10 α = 16	k = 20 α = 1	k = 80 α = 1
$K(B_i) = i$	5 min	11 min	9 min
$K(B_i) = 999-100i+999E\left(\frac{1}{10}\right)$	17 min	21 min	13 min
$K(B_i) = 999-i$	36 min	22 min	17 min

Table 3a.

The dependence of time upon the length of vocabulary and frequency updating /$K(B_i)$ = 999-i, i = 0, 1, ..., 999/.

α \ k	10	20	40	80	160
1		22 min	16 min	17 min	26 min
2	29 min	20 min	19 min	27 min	
4	27 min	22 min	28 min		
8	28 min	31 min			
16	36 min				

Table 3b.

The dependence of time upon the length of vocabulary and frequency updating /$K(B_i)$ = 999-100i+999E($\frac{1}{10}$)/.

α \ k	10	20	40	80	160
1		21 min	14 min	13 min	17 min
2	23 min	15 min	13 min	17 min	
4	17 min	13 min	17 min		
8	14 min	17 min			
16	17 min				

References

1. McCRACKEN D.: Programming Business Computers, Wiley, New York 1959.

2. FLORES I.: Analysis of Internal Sorting, Journal of the Association for Computing Machinery, 1961:8, 1, 41-80.

3. BEAUCLAIR W. de: Das Sortiren von Magnetband-Daten in einfachen Buchungsanlagen, Elektronische Anlagen, 1961:2, 75-81.

Instytut Maszyn Matematycznych
Algorytmy Nr 4

513.83:518.6

ON A CERTAIN THEOREM OF GRAPH THEORY AND ITS APPLICATION TO AUTOMATIC PROGRAMMING

by Andrzej SALWICKI

Received March 23rd, 1964

Formulas describing the change of a matrix associated with multigraph after the shrinkage of a certain set of vertices are presented, as well as their application to translation from autocode into machine language.

The structure of a program for digital computer is convenient to be interpreted as a graph. The set of vertices of a graph is formed of instructions, the arcs are ordered pairs of instructions arranged according to the successiveness of their execution.

The purpose of this paper is to pose the problem of program segmentation into chapters, i.e. dividing the program into parts which can be entirely contained in the internal store of a computer, further being called Chapters. One Chapter having been performed, the next one is called into the internal store.

Such an approach to the question of segmentation may be slightly unconventional; thus, instead of dividing the program into chapters, we propose to join them if there is enough place in the internal store.

The following example explains it most clearly: Let a_{ij} denote the number of possible consecutive executions of j-th chapter after i-th, i.e. the number of so-called "passes" from the i-th chapter to the j-th.

Let T /sec./ be the total time wasted for calling in the chapters during the program execution.

Let's assume now that chapters "i" and "j" are joined together. Then, the time wasted for calling in chapters during the program execution is $T - t.\ (a_{ij} + a_{ji})$, where t is the time of calling in the chapter from the external storage /in sec/.

Joining 3 chapters, namely i, j, and k, into one permits us to save $t \cdot (a_{ij} + a_{ji} + a_{ik} + a_{ki} + a_{jk} + a_{kj})$ sec. and so on. Further, the operation of joining chapters will correspond to the operation of shrinkage of vertices.

If one begins to treat every instruction as a separate chapter, these chapters will require a possibly great capacity of the internal storage; the number of chapters being small.

The method we are suggesting, takes into account a number of connections of one chapter with another /the number of possible consecutive performances of j-th after the i-th instruction during the execution of the program/.

It is expressed by the matrix associated with the multigraph H corresponding to the program.

The theorem mentioned in the title describes the changes of matrix caused by the shrinkage of a certain number of chapters /instructions/.

The second part of the paper gives an example of solution to the problem of segmentation into chapters for autocodes of the type ALGOL or SAKO.

1. Given the set X of vertices and the set U of arcs. An ordered pair (x_i, x_j) of vertices corresponds to each arc $u \in U$. We say that the arc u joins the vertices x_i and x_j. The element x_i is called i n i t i a l v e r t e x of the arc u, x_j - its t e r m i n a l v e r t e x.

More than one arc $u \in U$, with the initial vertex x_i and the terminal vertex x_j, may correspond to an ordered pair (x_i, x_j).

An ordered pair $G = (X, U)$ is called o r i e n t e d m u l t i g r a p h. The multigraph is said to c o n t a i n a l o o p if such arc $u \in U$ and vertex $x \in X$ exist that the arc u joins the pair (x, x).

Further, we shall consider f i n i t e m u l t i g r a p h s, i.e. such that the set X contains a finite number of elements $\{x_1, x_2, \ldots, x_n\}$.

The number of arcs beginning with x_i and ending with x_j will be denoted by a_{ij}.

The square matrix $[a_{ij}]$ with n rows and n columns is called m a t r i x a s s o c i a t e d with multigraph G.

Let A be the subset of the set X of vertices in G.

The operation resulting in

- identifying all vertices of A with some fixed element $a \in A$,
- omitting all arcs joining the elements of A

is called shrinkage of subset $A \subset X$ in multigraph G.

The shrinkage of the subset $A \subset X$ leads from multigraph $G = (X, U)$ to multigraph $G' = (X - A \cup \{a\}, U')$, where U' is the set of arcs of multigraph G'. U' arises from U by means of:

- omitting all arcs beginning and ending in A,
- replacing the remaining arcs, the beginning or the end of which belongs to A, by arcs beginning /or ending/ in a.

Let us denote by $G_{(i,j)}$ the multigraph obtained from G by the shrinkage of vertices x_i and x_j.

We say that two multigraphs $G = (X, U)$ and $H = (Y, V)$ are i s o m o r p h i c, if there exists an one-to-one correspondence φ, carrying X into Y and U into V, such that if $x \in X$ is the beginning and $y \in X$ the end of an arc $u \in U$, then $\varphi(x) \in Y$ and $\varphi(y) \in Y$ are respectively the beginning and the end of the arc $\varphi(u) \in V$.

From the definition of shrinkage it is easily seen that multigraphs $G_{(i,j)}$ and $G_{(j,i)}$ are isomorphic. This property of shrinkage will be called c o m u t a t i v i t y.

T h e o r e m 1.

Given loopfree multigraph $G = (X, U)$. Let its set of vertices be numbered in an arbitrary but fixed way by natural numbers from 1 to n. Then the matrix associated with G contains n x n elements. Let the multigraph G' arise from G by the shrinkage of subset $A = \{x_m, x_{m+1}, \ldots, x_n\}$. The matrix associated with G' will contain m x m elements.

Let b_{ij} be the element of the matrix associated with the multigraph G', obtained from the loopfree multigraph G by means of shrinkage of all vertices with numbers $> m$ to the vertex x_m. Then

$$b_{ij} = \begin{cases} a_{ij} & \text{if } i < m, \quad j < m \\ \sum_{k=m}^{n} a_{ik} & \text{if } i < m, \quad j = m \\ \sum_{l=m}^{n} a_{l} & \text{if } i = m, \quad j < m \\ 0 & \text{if } i = j = m \end{cases} \qquad /1/$$

where $[a_{ij}]$ is the matrix associated of the multigraph G.
The proof of the theorem results from the definition of shrinkage.

It is evident that for i and j smaller than m

$$b_{ij} = a_{ij}$$

as neither x_i nor x_j belong to the shrunk subset A.

The number of arcs beginning in an arbitrary element $x_i (i < m)$, the end of which belongs to $A = \{x_m, \ldots, x_n\}$ equals

$$a_{i,m} + a_{i,m+1} + \ldots + a_{i,n}$$

this being the value of b_{im}.

Similarly for $j < m$

$$b_{m,j} = a_{m,j} + a_{m+1,j} + \ldots + a_{n,j}.$$

As all arcs beginning and ending in $A, (x_m \in A)$ are omitted while shrinking

$$b_{m,m} = 0$$

the proof is completed.

The following form of the theorem seems to be most convenient for practical applications:
Let $A \subset X$ be the sequence $\{x_{i_1}, \ldots, x_{i_k}\}$, and let the operation of shrinkage identify vertices $A \subset X$ with the vertex $x_{i_r} \in A$. Let's denote:

N_n – set of natural number smaller than $n + 1$
I – set $\{i_1, i_2, \ldots, i_r, \ldots, i_k\}$
J – set $I - \{i_r\}$

Let the set of vertices of the multigraph G be $X = \{x_1, x_2, \ldots, x_n\}$. If vertices belonging to A are shrunk to x_i, the number of vertices of multigraph G' equals $n-k+1$. We shall number them from 1 to $n - k + 1$. The function Ψ, defined below, determines the correspondence between the numbers of vertices not being shrunk in multigraph G, and the numbers of vertices in G'. Thus, let

$\Psi(1)$ – be the smallest element of the set $N_n - J$;
$\Psi(i)$ – the smallest element of the set $N_n - J - U_{j<i}\{\Psi(j)\}$ for every i $1 < i \leq n - k + 1$.

The theorem will now take the form:

The element b_{ij} of the matrix assooiated with multigraph G, obtained from the multigraph G by means of shrinkage of vertices of the subset $A \subset X$ to the vertex x_{i_r}, is of the form

$$b_{ij} = \begin{cases} a_{\psi(i),\psi(j)} & \text{if} \quad \psi(i) \neq i_r, \ \psi(j) \neq i_r \\ \sum_{l \in I} a_{l,\psi(j)} & \text{if} \quad \psi(i) = i_r, \ \psi(j) \neq i_r \\ \sum_{p \in I} a_{\psi(i),p} & \text{if} \quad \psi(i) \neq i_r, \ \psi(j) = i_r \\ 0 & \text{if} \quad \psi(i) = \psi(j) = i_r \end{cases} \qquad /2/$$

Let's note that the number of aros in multigraph G with assooiated matrix $[a_{ij}]$ equal

$$S = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}$$

We shall call the multigraph $G_A = (A, U_A)$ the submultigraph of (X, U) if:

1. $A \subset X$
2. U_A is oomposed of all aros with initial and terminal vertices belonging to A.

Let S(A) denote the number of aros in the multigraph G_A.

From the theorem we obtain the following:

Collorary 1.

The number S' of aros of the multigraph G = (X; U'), ob-

tained from the multigraph $G = (X,U)$ by means of shrinkage of the subset $A \subset X$, is equal to

$$S' = S - S(A)$$

The theorem may help us to solve the following problem.

Let the function f transform the family of subsets of the set X into the set of natural numbers

$$f : 2^X \to N$$

Let p be a fixed number. The subset $A = \{x_{i_1}, \ldots, x_{i_k}\}$ of X $A \in 2^X$ will be called a d m i s s i b l e if

$$f(A) \leqslant p$$

The decomposition $\{A_i\}$ of the set X will be called a d - m i s s i b l e if

$$f(A_i) \leqslant p \qquad \text{for every} \quad i.$$

For example, p will be interpreted as the number of places in the internal store of the computer, and the function f as a function that assigns the number of places in the internal store to the given set of instructions /chapter/.

Let the multigraph $G = (X, U)$ be given as well as some decomposition $\{A_i, i \in J\}$ of its set of vertices.

Let us determine the multigraph $G' = (X', U')$ as follows: the vertices of X' are elements of the decomposition $\{A_i, i \in I\}$; the set of multigraph arcs U' is formed by the arcs $u \in U$ of multigraph G, obtained by shrinkage of vertices in every element A_i of the decomposition $\{A_i\}$.

Let us denote by $a(A_i, A_j)$ the number of arcs connecting the element A_i with A_j in multigraph G'. Obviously, $a(A_i, A_i) = 0$. From the theorem we get:

Collorary 2

Let the element A_j of the decomposition $\{A_i\}$ be formed of vertices $\{x_{j1}, x_{j2}, \ldots, x_{ji_m}\}$.

Let us denote by W_j the set of indices $\{j_1, j_2, \ldots, j_{1_m}\}$; then:

$$a(A_i, A_j) = \sum_{l \in W_i} \sum_{k \in W_j} a_{lk}$$

Collorary 3

The number of arcs of the multigraph G' is

$$S' = S - \sum_{i \in I} S(A_i)$$

The above mentioned problem takes now the following form.

Given multigraph $G = (X, U)$, the function f and the number p; find such admissible decompositions $\{A_i, i \in I\}$ of the set X of vertices as to gain minimal number of arcs in the multigraph G'.

Let $\xi_i^j = \begin{cases} 1 & \text{if vertex } x_i \text{ is in j-th element of decomposition} \\ 0 & \text{in an opposite case} \end{cases}$

Let's assume that the decomposition contains n elements, some of which may be empty. The analytical form of the above problem is the following:

For the given square matrix $[a_{ij}]$, the number p and the function f

$$f : Q^n \to N$$

/where Q^n denotes a n-dimensional cube with the edge $[0,1]$/ find such integers $\xi_i^j (i,j = 1, 2, \ldots, n)$ that

$$1^\circ \quad \xi_i^j \geq 0 \qquad \text{for} \quad i,j = 1, 2, \ldots, n$$

$$2^\circ \quad \sum_{j=1}^{n} \xi_i^j = 1 \qquad \text{for} \quad i = 1, 2, \ldots, n$$

$$3^\circ \quad f(\xi_1^j, \ldots, \xi_i^j) < p \qquad \text{for} \quad j = 1, 2, \ldots, n$$

for which the sum

$$4^\circ \quad S = \sum_{i,k=1}^{n} a_{i_k} - \sum_{j=1}^{n} \sum_{l,m=1}^{n} a_{l_m} \cdot \xi_l^j \cdot \xi_m^j$$

reaches its minimum.

For the reason given in the second part of the paper, the methods of solving this problem will be not considered.

The problem may be posed in a slightly different way using the notion of an a d m i s s i b l e c o v e r i n g instead of admissible decomposition. Namely, for the given multigraph $G = (X, U)$ and the covering $\{B_i, i \in I\}$ of the set X the multigraph G' will be determined as follows:

- elements B_i of the covering $\{B_i, i \in I\}$ are vertices of the multigraph G'
- the arc $u \in U'$ joins the elements B_i and B_j - if
  1° its beginning belongs to B_i
  2° its end belongs to $B_j - B_i$.

It results from the above definition that the arc, joining two elements B_i and B_j, does not exist in G' if B_i is identic with B_j.

Collorary 4.

The number of arcs joining B_i with B_j is equal to

$$a(B_i, B_j) = \sum_{l \in W_i} \sum_{k \in W_j - W_i} a_{l_k}$$

where W_i and W_j denote the sets of such indices that if $l \in W_i$, then $x_l \in B_i$.

2. APPLICATION TO THE AUTOMATIC PROGRAMMING.

According to the above said, we shall treat the program as a multigraph. The number of arcs joining vertices x_i and x_j in this multigraph will be denoted by a_{ij} equalling the number of possible executions of the instruction "j", after the instruction "i".

The problem posed in the first part of this paper becomes interesting for the programmer after having admitted, that the function f makes the number of places in the computer memory correspondent to the set of instructions A. The number p gives the amount of places in the computer storage. The solution of this problem gives us the decomposition of the program into chapters - parts which can be entirely contained in the internal store. The waste of time for interchange of chapters for this decomposition being minimal.

Now let us remind the notions of the graph theory which may be useful further.

T h e p a t h in a graph is a sequence of arcs where every initial vertex is the terminal vertex of the previous arc, if such one exists. If x is the beginning of the first arc, and y the end of the last one, we say that the path connects vertices x and y. T h e p a t h is also the term used for the sequence

of vertioes $\{x_1, x_2, \ldots, x_k\}$ such that for $j = 1,2, \ldots, k-1$ the pair (x_j, x_{j+1}) is the arc of the graph.

T h e o i r o u i t of the graph is the path, the beginning of its first arc being the same as the end of the last one.

Further, we shall use the notions of program path and program loop.

T h e p r o g r a m p a t h will be the sequenoe of instructions being the path in a graph corresponding to the program. Henoe, the oirouit in the graph oorresponds to a l o o p in the program.

Let us denote by a_{ij} the element of the matrix associated with the multigraph /further denoted by H/, corresponding to the program. Notice that if a_{ij} differs from zero or one, it corresponds to arcs in various loops and is equal to the number of loop exeoutions.

In programs written in ALGOL or SAKO autocode the loops are build by means of instructions of the type.

1° { for i:= J step K until N do ...;
REPEAT from α : I = J(K)N

2° { if boolean expression then go to A else go to B;
IF conditional expression : 1A, ELSE 1B

In general, these instructions do not determine exactly the number of loop passes. However, let us temporarily assume that we are dealing with programs in which loops are build exclusively by means of such instructions of the type REPEAT /for/ that J,K,N are numbers written explicite in the statement REPEAT. Then, the matrix associated with multigraph H will be determined in two stages as follows:

At first, the square matrix $[a_{ij}]$ is to be determined

$$a_{ij} = \begin{cases} 0 & \text{if, directly after the performance of i-th instruction, the transfer to the execution of j-th instruction is impossible} \\ \frac{N-J}{K} & \text{if the i-th instruction is REPEAT FROM j-th instruction: I = J(K)N} \\ 1 & \text{if else} \end{cases}$$

$i,j = 1, 2, \ldots, n$; n = the number of instructions.

The matrix $[b_{ij}]$ associated with multigraph H is formed from the matrix $[a_{ij}]$ after the analysis of loops.

For every arc (x_i, x_j) we look for loops it belongs to. As to every loop corresponds one element a_{lk}, different from zero and one of the matrix $[a_{ij}]$, the list of loops, to which the arc (x_i, x_j) belongs, may be built of several indices l, k such that $0 \neq a_{lk} \neq 1$. This list will be denoted by

$$L_{ij} = (l_1, k_1; l_2, k_2; \ldots; l_m, k_m)$$

Let us divide the system L_{ij} of all loops into classes in such a way that if two loops belong to one class, one of them contains another.

Let us denote by $L_{ij}^{(p)}$ - p-th class of the decomposition of L_{ij} and let us suppose m to be the number of these classes.

Now we obtain the element b_{ij} of the matrix associated with a multigraph corresponding to the program,

$$b_{ij} = \sum_{p=1}^{m} \prod_{(l,k) \in L_{ij}^{(p)}} a_{lk}$$

if the arc (X_i, X_j) does not belong to any loop $b_{ij} = \operatorname{sgn}(a_{ij})$. Repeating this for $i, j = 1, 2, \ldots, n$ /n = numbers of instruc-

tions in the program/, we obtain the matrix associated with the multigraph H.

Now we have to solve the problem posed in the Part I so as to get the segmentation of the program as good as possible.

We have already mentioned that not all programs satisfy our assumptions on exact determination of the number of loop passes.

It seems to be worth to notice that in the majority of cases very rough approximations are sufficient. The increase of exactness may reveal no influence upon the result.

For such programs we are able to form only approximate associated matrices. One may approach the problem in many ways; let us mention two of them.

The first of the above mentioned ways is recommended while preparing big programs. It consists in computing elements of an associated matrix during the first run of the program.

The second one is rather suitable for translating the so-called scientific-technical programs written in such language as ALGOL or SAKO. It corresponds to the above described method, but the matrix $[a_{ij}]$ is defined in a somewhat different way:

$$a_{ij} = \begin{cases} 0 & \text{if directly after i-th instruction the j-th instruction can't be performed. If after i-th instruction the instruction j-th may be performed, one puts:} \\ \frac{N-J}{K} & \text{if } J,K,N, \text{ are numbers written in i-th instruction:} \\ & \qquad \text{REPEAT FROM j-th : I=J(K)N} \\ 1 & \text{if i-th instruction is not an instruction of the type REPEAT or IF} \\ \infty & \text{if else} \end{cases}$$

The character ∞ may be interpreted in a digital computer as an appropriately great number A. Next, we may act like in the

case of programs with an exactly determined number of loop passes. The matrix $[b_{ij}]$ can be formed using the matrix $[a_{ij}]$.

It seems that in many cases the following method of program segmentation into chapters is sufficient.

In the matrix associated with multigraph H the greatest element b_{ij} is chosen. The admissibility of joining i-th and j-th chapter shrinkage is checked. If so, we do it and we modify the associated matrix according to the theorem. If not, we look for the next greatest element of the matrix $[b_{ij}]$. We continue acting so till the moment when the shrinkage of chapters becomes possible.

For an exact solution of the problem, linear or other programming methods should be used. However, there appear two kinds of difficulties:

1° the function f may be not a convex one. It generally is not, as its values are the numbers of internal storage places, required for the execution of the given sequence of instructions

2° the solution must be an integer.

Moreover, the computing methods being so complex do not seem purposeful for translating programs, the operation time of which may appear shorter than the excessively developped translation time.

It seems that the results may be still obtained in this field by describing dependences between the exactness of an associated matrix and results of program segmentation into chapters. Better methods of solving the problem posed in part I, may be developped as well.

One more application of the theorem seems to be worth mentioning. In many cases, as for instance in ALGOL, the automation of decomposition is reached by means of mechanical segmentation of the program into parts of a similar fixed length. This however, should permit a simultaneous location of several segments in the computer internal storage.

During the operation of the program the transfer from one segment to another is completed by a special program. Then, a new problem arises, namely: while calling in a new segment from the external storage, one of the former segments must be overwritten. The choice of the proper segment is not a trifing matter. An inadequate exchange of segments may greatly increase the time of execution. The associated matrix of a multigraph, the vertices of which are the program segments, may help to solve the problem. The way of obtaining such a matrix is given by the theorem and collorary 2. For each segment the choice of the segments most tightly connected with it, according to the associated matrix, and forming a short list, would permit to remove the segments feably connected with the one called in. This may considerably reduce the time of program performance.

References

1. BERGE C.: Théorie des graphes et ses applications, 1958.

Instytut Maszyn Matematycznych
Algorytmy N° 5

513.83:518.6

ROZKŁAD KONTURÓW W GRAFIE SKOŃCZONYM ORAZ ZASTOSOWANIE TEORII GRAFÓW DO AUTOMATYCZNEGO PROGRAMOWANIA

Alfred SCHURMANN

Pracę złożono 12.02.1964 r.

W pracy opisano algorytm otrzymywania wszystkich konturów grafu skończonego i zorientowanego. Podano twierdzenie o ilości konturów między dwoma sąsiednimi wierzchołkami grafu. Końcowa część pracy przedstawia model dróg programu za pomocą grafu skończonego i zorientowanego. Daje to podstawę do stosowania wyżej wymienionego twierdzenia przy określaniu najlepszego miejsca do dzielenia programu na rozdziały /parag. 6, tw. 2/.

1. WSTĘP

Praca niniejsza jest próbą uporządkowania i uściślenia pewnych pojęć i problemów z zakresu teorii programowania. Pewne zagadnienia występujące w automatycznym programowaniu /np. podział programu na rozdziały, właściwy wybór segmentu z pamięci wewnętrznej/ nie dają się sformułować i rozwiązać bez stosowania innych działów matematyki.

To, że program można przedstawić za pomocą grafu, zauważył już Karp [2] w 1960 r. Schemat blokowy programu przedstawił jako graf skończony i zorientowany. Praca jego nie podaje jednak właściwie żadnych wniosków.

Autora zainteresowała interpretacja grafu jako modelu dróg programu, przy rozważaniach o podziale programu na rozdziały. Problem ten jest ściśle związany z drogami i pętlami programu. W pracy [4] i [3] autor opisał drogi programu za pomocą grafu skończonego

i zorientowanego. Dało to podstawę do analizy rozkładu pętli w programie [3], [4].

W niniejszej pracy powyższe zagadnienia podane są w odwrotnej kolejności. Najpierw formułuje się i rozwiązuje problem z teorii grafów, następnie podaje się jego interpretację w dziedzinie automatycznego programowania. I tak, w paragrafie 2 podane są potrzebne określenia i pojęcia z teorii grafów. W następnym paragrafie opisano algorytm otrzymywania z dowolnego grafu skończonego i zorientowanego G, grafu składającego się z samych konturów grafu G. Realizacja tego algorytmu w maszynie cyfrowej opisana jest w paragrafie 4.

Paragraf 5 przedstawia model dróg programu za pomocą grafu skończonego i zorientowanego G. Daje to podstawę do stosowania twierdzenia 1 /par. 3/, do określenia najlepszego miejsca do dzielenia programu na rozdziały /par. 6, tw. 2/.

2. OKREŚLENIA

2.1. Graf G*)

Dane są:

a. zbiór uporządkowany, niepusty

$$X = \left\{ X_1, X_2, \ldots, X_n \right\},$$

b. przekształcenie Γ zbioru X w X.

Grafem $G = (X, \Gamma)$ nazywamy parę uporządkowaną, która składa się ze zbioru X oraz przekształcenia Γ.

Elementy zbioru X nazywamy wierzchołkami grafu G. Para wierzchołków (X_i, X_j), gdzie $X_j \in \Gamma X_i$, oznacza łuk grafu; wierzchołek X_j jest następnikiem wierzchołka X_i. Drogą w grafie G nazywamy taki ciąg łuków, w którym koniec każdego poprzed-

*) Patrz [1].

niego łuku jest początkiem następnego łuku. Drogę skończoną $X_{a_1}, X_{a_2}, \ldots, X_{a_r}$, której pierwszy wierzchołek X_{a_1} równa się ostatniemu wierzchołkowi X_{a_r}, nazywamy konturem. Jeżeli żaden z łuków konturu nie występuje w nim dwa razy, to mówimy, że kontur jest elementarny.

2.2. Macierz wiązań *).

Graf $G = (X,\Gamma)$ opisany jest przez macierz wiązań $[a_{ij}]$, gdzie element a_{ij} jest określony:

$$a_{ij} = \begin{cases} 1, \text{ gdy w grafie } G \text{ istnieje łuk } (X_i, X_j) \\ 0, \text{ gdy nie istnieje łuk } (X_i, X_j). \end{cases}$$

2.3. Ściągnięcie zbioru wierzchołków *).

Zbiór $A \subset X$ został ściągnięty w grafie G do punktu $X_h \in A$, jeżeli wszystkie łuki między punktami zbioru A zostały pominięte, a wszystkie punkty zbioru A utożsamia się z punktem X_h.

Przez A_{t_h} będziemy oznaczali zbiór wierzchołków, które zostały ściągnięte do punktu X_{t_h}.

Symbole $A^+_{t_h}$ oraz $A^-_{t_h}$ oznaczają zbiory łuków przyporządkowane wierzchołkowi ściągnięcia X_{t_h} /ścisłe określenie podane jest w p. 2.5./. Po ściągnięciu zbioru A w grafie $G = (X,\Gamma)$ otrzymuje się p-graf, który będziemy oznaczali przez $(\bar{X}, \Gamma_p)$.

2.4. Kontur podstawowy.

Będziemy mówili, że między wierzchołkiem X_i oraz X_{i+1} przechodzi kontur podstawowy, gdy istnieje łuk (X_s, X_k), gdzie $s > i$ oraz $k \leqslant i$, który należy do konturu grafu G.

*) Patrz [1].

2.5. Zbiór A^+ i A^- *).

a. Niech zbiór $A \subset X$ nie zawiera punktów ściągnięcia. Przez A^+ będziemy rozumieli zbiór łuków wychodzących ze zbioru A, przez A^- – zbiór łuków wchodzących do zbioru A. W grafie przedstawionym na rys. 1a. zaznaczono zbiór $A_5 = \{X_2, X_4, X_5\}$. Dla tego przykładu:

$$A_5^+ = \{(X_2,X_3), (X_4,X_7), (X_5,X_8)\}$$

$$A_5^- = \{(X_6,X_5), (X_3,X_4), (X_1,X_2)\}$$

Gdy $A = A_{t_h}$, to $A^+ = A^+_{t_h}$ i $A^- = A^-_{t_h}$

b. Załóżmy, że jakieś podzbiory zbioru X zostały ściągnięte w grafie (X, Γ). Z grafu tego otrzymaliśmy pewien p-graf $(\bar{X}, \Gamma_p)$. Niech zbiór $A \subset \bar{X}$ posiada wierzchołki ściągnięcia $X_{j1}, \ldots, X_{ji}$, a zbiór $\bar{X} - A$ wierzchołki ściągnięcia $X_{k1}, \ldots, X_{km}$. Zbiory A^+ i A^- określa się w następujący sposób:

$$(X_e,X_s) \in A^+ \equiv ((X_e,X_s) \in (A^-_{k1} \cup \ldots \cup A^-_{km}) \vee$$

$$\vee X_s \in (\bar{X} - (A \cup \{X_{k1}, \ldots, X_{km}\}))) \wedge$$

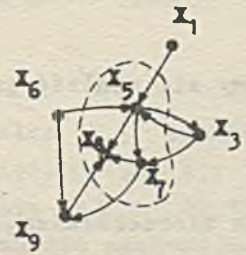
$$\wedge ((X_e, X_s) \in (A^+_{j1} \cup \ldots \cup A^+_{ji}) \vee$$

$$\vee X_e \in (A - \{X_{j1}, \ldots, X_{ji}\})) \wedge (X_e,X_s) \in G$$

Rys. 1a. Graf bez punktu ściągnięcia.

*) Definicja ta podana jest w pracy [3]

$$(X_e, X_s) \in A^- \equiv ((X_e, X_s) \in (A^+_{k1} \cup \ldots \cup A^+_{km}) \vee$$
$$\vee X_e \in (\bar{X} - (A \cup \{X_{k1}, \ldots, X_{km}\}))) \wedge$$
$$\wedge (X_e, X_s) \in (A^-_{j1} \cup \ldots \cup A^-_{ji}) \vee$$
$$\vee X_s \in (A - \{X_{j1}, \ldots, X_{ji}\})) \wedge (X_e, X_s) \in G,$$



Rys. 1b. Graf z punktem ściągnięcia X_5.

gdzie: $\cup$ jest znakiem sumy zbiorów,
$-$ – znakiem różnicy dwóch zbiorów,
$\wedge$ – znakiem iloczynu logicznego,
$\vee$ – znakiem sumy logicznej.

Na rysunku 1b. przedstawiono graf, który powstał z grafu na rys. 1a. poprzez ściągnięcie zbioru A_5 do punktu X_5. W grafie tym zaznaczono zbiór $A_8 = \{X_5, X_7, X_8\}$.

W przykładzie tym:

$$A^+_8 = \{(X_2, X_3), (X_7, X_9)\}$$
$$A^-_8 = \{(X_1, X_2), (X_3, X_4), (X_5, X_7), (X_6, X_5), (X_9, X_8)\}$$

2.6. Graf konturów [3].

(X_i, X_j) jest łukiem grafu (X, Γ_k) wtedy i tylko wtedy, gdy łuk (X_i, X_j) należy do dowolnego konturu grafu (X, Γ). (X, Γ_k) będziemy nazywali grafem konturów grafu (X, Γ).

Zatem, gdy w grafie (X,Γ) pominiemy łuki, które nie należą do żadnego konturu, to otrzymamy graf (X,Γ_k).

Określenia powyższe wprowadzają w problematykę paragrafu następnego, w którym opisana jest metoda otrzymywania grafu konturów (X,Γ_k) z grafu G [3].

3. GRAF KONTURÓW (X,Γ_k).

Mając graf G postaramy się znaleźć graf konturów (X,Γ_k).

Zadanie 1.

Niech X_j będzie takim wierzchołkiem, że:

a. w grafie (X,Γ) nie istnieje kontur składający się z wierzchołków ze zbioru $\{X_1,X_2,\ldots,X_{j-1}\}$,

b. istnieje dokładnie jeden łuk (X_j,X_i), gdzie $i \leqslant j$.

Znaleźć wszystkie kontury grafu (X,Γ) nie zawierające żadnego wierzchołka X_k, takiego że $j<k\leqslant n$.

Algorytm 1

Zadanie będzie rozwiązane jeżeli znajdziemy wszystkie drogi z X_i do X_j, które nie zawierają punktu X_k.

Znany z pracy [1] algorytm na znajdywanie drogi z punktu X_i do punktu X_j ma w naszym przypadku postać: od wierzchołka X_i iść zgodnie z kierunkiem wybranej drogi zorientowanej dopóki nie trafia się na wierzchołek X_k lub wierzchołek bez następnika, w tym przypadku należy wrócić tą samą drogą do najbliższego wierzchołka i wybrać drogę jeszcze nie zbadaną itd. W ten sposób dochodzi się do punktu X_j lub X_i. Jeżeli znajdziemy się w punkcie X_i, wtedy w grafie (X,Γ) nie istnieje kontur, który składa się tylko z wierzchołków ze zbioru $\{X_1,\ldots,X_j\}$.

Jeżeli znajdziemy się w punkcie X_j, wtedy istnieje kontur. Oznaczamy go przez $X_j, X_i, X_{i1}, X_{i2}, \ldots, X_{im}$. Pozostałe kontu-

ry elementarne znajdziemy szukając nowe drogi: z X_i do X_j; z X_{i1} do X_j; z X_{i2} do X_j itd.

Wszystkie kontury elementarne przechodzą przez wierzchołek X_j i tworzą w związku z tym jeden kontur.

Zadanie 2.

Znaleźć graf konturów grafu (X,Γ).

Algorytm 2.

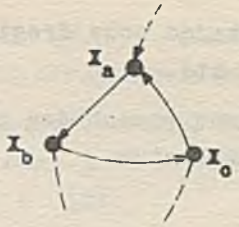
1. Począwszy od wierzchołka X_1, poprzez $X_2, X_3, \ldots$ szukamy pierwszego takiego wierzchołka X_j, którego następnikiem jest wierzchołek X_1, gdzie $1 \leq j$. Po znalezieniu X_j szukamy konturów elementarnych nie zawierających wierzchołka X_k, gdzie $k > j$. Problem ten jest rozwiązany w zadaniu 1. Jeżeli w grafie (X,Γ) nie istnieje kontur składający się z wierzchołków ze zbioru $\{X_1, \ldots, X_j\}$, wtedy dalej szukamy wierzchołka X_{j_1}, z następnikiem X_1, gdzie $1 \leq j_1$.

2. Przypuśćmy, że zostały znalezione kontury elementarne μ_e $(e = 1, \ldots, e_0)$. Niech A_j oznacza zbiór wierzchołków, które należą do tych konturów. Dana jest macierz $[C_{ij}]$ $(i, j = 1, 2, \ldots, n)$ o elementach równych zero. Kontury μ_e zapisujemy w macierzy $[C_{ij}]$ w następujący sposób: jeżeli (X_u, X_v) jest łukiem konturu μ_e, to $C_{uv} = 1$.

   W grafie (X,Γ) dokonujemy ściągnięcia zbioru A_j do punktu X_j. W otrzymanym p-grafie wierzchołkowi X_j należy przyporządkować zbiór A_j^+ i A_j^-.

3. Poczynając od X_j szukamy wierzchołka X_m o następniku X_s, gdzie $s \leq m$. Po znalezieniu wierzchołka X_m, tak samo jak w zadaniu 1, szukamy wszystkich konturów elementarnych nie przechodzących przez wierzchołek X_k, gdzie $k > m$.

4. Załóżmy, że zidentyfikowany kontur elementarny μ_e przechodzi przez punkt ściągnięcia X_a /p.rys. 2/. Konturowi μ_e odpowiada w grafie (X,Γ) kontur μ_e'. Kontur μ_e' różni się od



Rys. 2. Kontur μ_e.

konturu μ_e tym, iż zamiast przez punkt X_a przechodzi przez kontur o wierzchołkach ze zbioru A_a /rys. 3/.

Rys. 3. Kontur μ'_e.

Analogiczna odpowiedniość zachodzi wtedy, gdy kontur μ_e posiada więcej punktów ściągnięcia.

Na podstawie zidentyfikowanych konturów $\mu_e (e = 1,2, \ldots, e_1)$ zapisujemy w macierzy $[c_{ij}]$ kontury μ'_e. Z każdym łukiem (X_u, X_v) dowolnego konturu μ_e postępuje się zgodnie z opisem:

a. gdy X_u i X_v nie są punktami ściągnięcia, to $C_{uv} = 1$;

b. gdy X_u jest punktem ściągnięcia i X_v nie jest punktem ściągnięcia, to łuki (X_t, X_v) $(t = 1, 2, \ldots, m_1)$ należące do zbioru A_u^+ są łukami konturu μ_e'; elementom C_{tv} należy przyporządkować wartość 1;

c. gdy X_u nie jest punktem ściągnięcia i X_v jest punktem ściągnięcia, to łuki (X_u, X_p) $(p = 1, 2, \ldots, m_1)$ należące do zbioru A_v^- są łukami konturu μ_e'; elementom C_{up} należy przyporządkować wartość 1;

d. gdy X_u i X_v są punktami ściągnięcia, to łuki (X_t, X_p) należące do zbioru A_u^+ i A_v^- są łukami konturu μ_e'; elementom C_{tp} przyporządkowujemy wartość 1.

Oznaczmy przez A_m zbiór wierzchołków należących do konturów μ_e. W p-grafie należy dokonać ściągnięcia zbioru A_m do punktu X_m. Wierzchołkowi X_m przyporządkować zbiory A_m^+ i A_m^-. Otrzymaliśmy nowy p-graf, w którym nie istnieją kontury zawierające tylko wierzchołki X_{h-1}, gdzie $h \leq m$.

Postępując dalej według opisu w punktach 3 i 4 otrzymamy macierz $[C_{ij}]$, będącą macierzą wiązań grafu (X, Γ_k).

Z definicji konturu podstawowego oraz macierzy $[C_{ij}]$ wynika:

Twierdzenie 1.

Ilość konturów podstawowych między wierzchołkami X_i oraz X_{i+1} równa się

$$\sum_{k=1}^{i} \sum_{S=i+1}^{n} C_{sk}$$

4. REALIZACJA ALGORYTMU W MASZYNIE CYFROWEJ

Opis grafu G przy pomocy macierzy wiązań $[a_{ij}]$ nie zawsze jest dogodny w maszynie cyfrowej. Gdy w jednej komórce pamięci ma-

szyny umieścimy jeden element macierzy $[a_{ij}]$, to macierz ta zajmie stosunkowo dużo miejsca. Wydaje się, że w naszym przypadku najdogodniejszy będzie następujący zapis grafu:

$$\Gamma X_i = \{N(X_i)\},$$

gdzie $\{N(X_i)\}$ oznacza zbiór następników wierzchołka X_i.

Realizacja algorytmu

1. Poczynając od punktu X_1 szukać wierzchołka X_j o następniku X_k, gdzie $k \leq j$. Zgodnie z algorytmem 1 znaleźć kontury elementarne nie zawierające wierzchołka X_i, gdzie $i > j$. Załóżmy, że znaleziony został kontur $\mu = X_{f1}, X_{f2}, \ldots, X_{fr}$. Łuki $(X_{f_u}, X_{f_{u+1}}) \in \mu$, gdzie $f_u > f_{u+1}$, pamiętać w bloku LPW *). Zamiast ściągnięcia wierzchołków konturu μ dokonać następującej modyfikacji grafu:

$$\Gamma X_{f1} = \Gamma X_{f2} = \ldots = \Gamma X_{fr} = \left(\{N(X_{f1})\} \cup \{N(X_{f2})\} \cup \ldots \cup \{N(X_{fr})\}\right) - \{X_{f1}, X_{f2}, \ldots, X_{fr}\}$$

Zaznaczyć wierzchołki konturu μ i przyporządkować im zbiór:

$$\bar{A}^+_{f1} = \bar{A}^+_{f2} = \ldots = \bar{A}^+_{fr}, \quad \text{gdzie} \quad (X_t, X_s) \in \bar{A}^+_{f1} \equiv$$

$$\equiv (t > s) \wedge X_s \in \Gamma X_{f1} \wedge X_t \in \mu \wedge (X_t, X_s) \in G$$

*)W bloku LPW pamiętamy łuki w ten sposób, żeby po zakończeniu działania algorytmu zachodziło:

$$LPW(i) = \sum_{k=1}^{i} \sum_{s=i+1}^{n} C_{sk}$$

2. Poczynając od wierzchołka X_j szukać punktu X_m o następniku X_s, gdzie $s \leq m$. Zgodnie z algorytmem 1 szukać konturu, nie zawierającego wierzchołka X_i, gdzie $i > m$.

Przypuśćmy, że zidentyfikowany został kontur $\mu_1 = X_{p1}, X_{p2}, \ldots, X_{p_e}$.

Weźmy dowolny łuk $(X_{p_v}, X_{p_{v+1}}) \in \mu_1$. Gdy X_{p_v} nie należy do uprzednio zidentyfikowanego konturu i $p_v > p_{v+1}$, to zapamiętać w bloku LPW łuk $(X_{p_v}, X_{p_{v+1}})$.

Gdy X_{p_v} należy do uprzednio zidentyfikowanego konturu, to został mu przyporządkowany zbiór łuków $\bar{A}^+_{p_v}$. Zapamiętać w bloku LPW wszystkie łuki $(X_t, X_{p_{v+1}}) \in \bar{A}^+_{p_v}$ $(t = 1,2, \ldots, m)$.

W aktualnym grafie należy dokonać następującej modyfikacji:

$$\Gamma X_{p1} = \Gamma X_{p2} = \ldots = \Gamma X_{p_e} = (\{N(X_{p1})\} \cup \ldots \cup \{N(X_{p_e})\}) -$$

$$- \{X_{p1}, X_{p2}, \ldots, X_{p_e}\}$$

Niech $X_{j_1}, X_{j_2}, \ldots, X_{j_w}$ będą wierzchołkami konturu μ_1, należącymi do ^{2}konturów ^{w}uprzednio zidentyfikowanych. Wierzchołkom tym zostały przyporządkowane odpowiednie zbiory $\bar{A}^+_{j_1}, \bar{A}^+_{j_2}, \ldots, \bar{A}^+_{j_w}$. Punktom $X_{p_1}, X_{p2}, \ldots, X_{p_e}$ oraz wierzchołkom tych konturów, który zawierają punkty $X_{j_1}, X_{j_2}, \ldots, X_{j_w}$, należy przyporządkować zbiór łuków $\bar{A}^+_p$ określony w ten sposób:

$$(X_r, X_c) \in \bar{A}^+_p \equiv (r > c) \wedge X_c \in \Gamma X_{p_1} \wedge (X_r, X_c) \in G \wedge$$

$$\wedge ((X_r, X_c) \in (\bar{A}^+_{j_1} \cup \bar{A}^+_{j_2} \cup \ldots \cup \bar{A}^+_{j_w}) \vee$$

$$\vee X_r \in (\{X_{p_1}, \ldots, X_{p_e}\} - \{X_{j_1}, X_{j_2}, \ldots, X_{j_w}\}))$$

W grafie zaznaczyć wierzchołki konturu μ_1. Dalej postępować według opisu w punkcie 2.

j-ty element bloku LPW wskaże nam ile konturów podstawowych znajduje się między wierzchołkiem X_j i X_{j+1}.

5. MODEL DRÓG PROGRAMU

W pracy tej rozważane są programy, które nie modyfikują swojej własnej struktury.

Zdanie programu można podzielić na zdania operacyjne i zdania sterujące [2]. Dowolne zdanie programu może być oznaczone etykietą. Zdania operacyjne wykonują się w kolejności ich występowania w programie. Zdanie sterujące można określić jako funkcję, zależną od danych wejściowych oraz zdań programu, której wartościami są etykiety. W dalszym ciągu przez zdanie sterujące będziemy rozumieli funkcję:

$$Zs(Y) = \begin{cases} \alpha_1 \\ \alpha_2 \\ \cdot \\ \cdot \\ \cdot \\ \alpha_N \end{cases}$$

gdzie Y jest pewną niewiadomą zależną od zdań programu i danych wejściowych, a $\alpha_1, \alpha_2, \ldots, \alpha_N$ są ustalonymi etykietami programu. Zdanie sterujące jest punktem, z którego rozchodzą się drogi do $\alpha_1, \alpha_2, \ldots, \alpha_N$. O tym, którą z tych dróg przebiegać będzie wykonywanie programu decyduje zmienna Y, jednakże nie zmienia ona dróg programu. Punktami, w których drogi programu rozchodzą się lub zbiegają są, oprócz zdań sterujących, etykiety. Są to punkty węzłowe dróg programu.

Etykiety oraz zdania sterujące oznaczmy w takiej kolejności, w jakiej występują w programie, przez $X_1, X_2, \ldots, X_n$. Drogi pro-

gramu opisuje graf zorientowany $G = (X, \Gamma)$, gdzie X jest zbiorem uporządkowanym $\{X_1, X_2, \ldots, X_n\}$, a Γ przekształceniem zdefiniowanym:

$$\Gamma X_{i-1} = \begin{cases} \{X_i\}, & \text{gdy } X_{i-1} \text{ jest etykietą} \\ \{\alpha_1, \alpha_2, \ldots, \alpha_N\}, & \text{gdy } X_{i-1} \text{ jest zdaniem sterującym } Zs(Y) = \begin{cases} \alpha_1 \\ \alpha_2 \\ \ldots \\ \alpha_N \end{cases} \end{cases}$$

gdzie α_k jest elementem zbioru X.

W tej interpretacji grafu, drogi grafu są drogami programu, a kontury - pętlami programu. Podany model dróg programu pomija zdania operacyjne. Na podstawie tego modelu nie wiemy ile zdań operacyjnych znajduje się pomiędzy dowolnym punktem X_e a X_{e+1}. W dalszych rozważaniach informacja ta nie będzie nam potrzebna.

Przykład:

Dany jest program w Algolu:

```
begin real   J,B,M,R; integer  I;
A : I := 0 ;
    read I, B;
B:M:= M + sin (B + I);
  I:= I + 1;
  print M ;
  if I<100 then go to B ;
  if J = 1.4 then go to B1 ;
  R := (M+R) x J ;
  if R>50.6 then go to A ;
```

```
  print R ;
B1: end
```

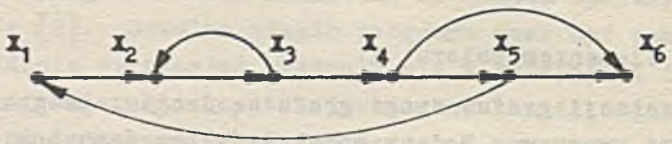
Wierzchołkami programu są:

X_1 = A ; X_2 = B ; X_3 = if I < 100 then go to B ;

X_4 = if J = 1.4 then go to B1;

X_5 = if R > 50.6 then go to A ; X_6 = B1.

Graf odpowiadający temu programowi ma postać:



Rys. 4.

6. PODZIAŁ PROGRAMU NA ROZDZIAŁY

Rozdziałem programu nazywamy część programu, która, przetłumaczona na język maszyny, mieści się w pamięci wewnętrznej.

Podczas wykonywania programu prawie zawsze rozdział z pamięci wewnętrznej odwołuje się do rozdziału z pamięci zewnętrznej /np. bębnowej/. Załóżmy, że odwołanie to następuje poprzez drogę ..., X_k, X_j, ... Gdy łuk (X_k, X_j) nie należy do pętli, to mamy do czynienia tylko z jednokrotnym sprowadzeniem rozdziału z pamięci zewnętrznej do pamięci wewnętrznej. Ilość odwołań tego typu jest mała i dlatego możemy je pominąć.

Gdy łuk (X_k, X_j) należy do pętli /jest to tzw. pętla wielorozdziałowa [4] /, to za każdym przebiegiem pętli następuje przynajmniej dwukrotne sprowadzenie rozdziału z pamięci zewnętrznej. Czas sprowadzenia rozdziału do pamięci wewnętrznej jest stosunkowo duży. Dlatego należy program tak podzielić na rozdziały, żeby przeciąć jak najmniejszą ilość pętli wielorozdziałowych. Przy in-

terpretacji dróg programu jako dróg grafu, pętla wielorozdziałowa jest konturem podstawowym*). Zatem na podstawie twierdzenia 1 otrzymujemy:

Twierdzenie 2.

Na odcinku programu od punktu X_a do X_b najlepsze miejsce do dzielenia programu na rozdziały znajduje się między punktem X_i oraz X_{i+1} $(a \leq i < b)$, gdy

$$\sum_{k=1}^{i}\sum_{s=i+1}^{n} C_{sk} = \min_{a\leq j<b}\left(\sum_{k=1}^{j}\sum_{s=j+1}^{n} C_{sk}\right)$$

7. WNIOSKI KOŃCOWE

Z praktycznego punktu widzenia twierdzenie 2 całkowicie nie rozwiązuje problemu. Nie uwzględnia liczby przebiegów w danej pętli. Jest to wielkość niewiadoma, zależna od danego programu oraz jego danych wejściowych.

Wydaje się, że opisany algorytm otrzymywania grafu konturów można będzie również stosować przy dzieleniu programu na segmenty. Algorytm ten pozwala określić do jakich pętli dany segment należy. Przy wyborze segmentu z pamięci wewnętrznej, podczas wykonywania programu, należałoby wyrzucić ten segment, który nie należy do pętli aktualnie wykonywanej.

Na zakończenie chciałbym podziękować doc. dr Z. Pawlakowi za szereg cennych uwag.

*)Patrz [4] str. 77.

Literatura

1. BERGE C.: Théorie des graphes et ces applications /Teorija grafov i jejo primienienija/, Izdat. Inostr.Literat., Moskva 1962.

2. KARP R.M.: A Note on the Application of Graph Theory to Digital Computer Programming, Information and Control 1960:2, 3, 179-190.

3. SCHURMANN A.: The Application of Graphs to the Analysis of Distribution of Loops in a Program, Information and Control 1964:7, 3, 275-282.

4. SCHURMANN A.: On the Application of Graph Theory to Determine the Number of Multisection Loops in a Program, Prace IMM, Algorytmy 1964:2, 3, 73-81.

THE DISTRIBUTION OF CYCLES IN A FINITE GRAPH AND THE APPLICATION OF GRAPHS TO COMPUTER PROGRAMMING

Summary

A finite and oriented graph G is considered. The first part of the paper contains some definitions used in the essential parts 3 - 7. Part 3 describes the algorithm for obtaining all cycles of graph G. This permits to determine the number of cycles between two neighbouring vertexes of the graph G /theorem 1/. Part 4 presents the way of realizing the algorithm in a digital computer.

It is shown in paper [3] and [4] that the program pathes are described by the oriented graph G. The loops of any program are cycles of the graph. It permits to find the best division of the program into sections /theorem 2/.

Druk „Znak" Zl. 1001/1259 Nakł. 560