AUTOMATIC SYNTHESIS OF THE QUASI-MINIMAL MULTIPLE-OUTPUT SWITCHING CIRCUITS

by

Ryszard S. Michalski

VI International Symposium on Information Processing (FCIP 70),
pp. 1-7, Yugoslavia, Bled, September 23-26, 1970.

VI International Symposium on Information Processing (FCIP 70), Yugoslavia, Bled, September 23-26, 1970

r.s. michalski
institute of automatic control, polish academy of science
warsaw, poland

AUTOMATIC SYNTHESIS OF THE QUASI-MINIMAL MULTIPLE-OUTPUT SWITCHING CIRCUITS

The paper describes a program for automatic synthesis of multiple output switching circuits and results of investigations of it. The program is based on the algorithm A^q (presented at the FCIP 69 Symposium) for general solution of covering problem.

The program, written in LYA²AS language, gives the minimal or approximately minimal form of a switching function of n variables and m outputs, where $n + m \leq 31$. In the case of approximately minimal form the program gives an estimate of maximal possible distance between the obtained form and the minimal one.

1. Introduction

At the Vth FCIP 69 Symposium we presented an algorithm A^q for the so-called quasi-minimal solution of the generally stated covering problem [1] . This paper presents an application of algorithm A^q to the automatic synthesis of minimal forms of incompletely specified multiple-output switching functions and some of the results achieved by a computer program based on the described algorithm.

The program gives a so-called quasi-minimal form (which is minimal or approximately minimal) of a switching function with n variables and m outputs so that $n+m \leq 31$. In the case of approximately minimal form the program gives an estimate of maximal possible distance between the obtained form and the minimal one, expressed in terms of the number of components and literals.

2. Application of Algorithm A^q to the Synthesis of Minimal Forms of Switching Functions

2.1. General notes

We shall assume that the Reader is acquainted with paper [1], where we described the algorithm A^q and determined such concepts as: the image T(f) of a mapping f (which is a diagram of 2^n cells along with the values from the set $\{0,1,*\}$), a cover of image T(f), a complex of cells, a star G(e), a quasi-extremal L^q of star G(e) and others. Consequently, we shall make use here of the above-mentioned concepts as well as the system of notation used in [1] , without any additional explanation.

In the general statement of the covering problem given in [1], by a complex was meant a subset of cells (or subcells), which satisfied certain assumed conditions. We shall now define the concept of a complex in the concrete problem of synthesis of minimal forms of switching functions.

Let $X_j = (\dot{x}_1, x_2, \ldots, \dot{x}_n)$, $j \in \{0,1,..,2^n-1\}$, $\dot{x}_i \in \{0,1\}$, $i=1,2,..,n$, denote a sequence of values of the input variables $x_1, x_2, \ldots, x_n$ such that $j = \sum_{i=1}^{n} \dot{x}_i 2^{n-i}$.

Let $\mathbf{X}$ denote the set of all X_j sequences. A switching function $f(x_1,..,x_n)$ is then the mapping of the set $\mathbf{X}$ into set $\{0,1,*\}^m$:

$$f: \quad \mathbf{X} \longrightarrow \{0,1,*\}^m \qquad (1)$$

where * denotes an unspecified value 0 or 1, and m - the number of function outputs.

A one-output switching function can be uniquely determined by means of any two of the three sets F^1, F^0, F^* designating the indices j of the sequences X_j at which the function f takes the values 1, 0, *, respectively. An m-output switching function is equivalent to a set of m one-output switching functions f^k, $k=1,2,..,m$ of the same input variables. Consequently, it can be determined by m above-mentioned pairs of sets, e.g. $(F^{1,k}, F^{0,k})$, $k=1,2,\ldots,m$.

Let us assume that $m = 1$. If the value $f(X_j)$, $j=0,1,\ldots,2^n-1$ is assigned to the cell e^j of the diagram [1] , where e^j is the cell with number j, the image T(f) uniquely represents the function f. In the problem of the synthesis of a disjunctive normal form (d.n.f.) of f a complex of cells corresponds to a single product of literals $x_i^{\delta_i}$, $x_i^{\delta_i} = \begin{cases} x_i, & \text{if } \delta_i = 1 \\ \bar{x}_i, & \text{if } \delta_i = 0 \end{cases}$.

In the case of an m-output switching function every cell e^j of the diagram is divided into m subcells e^{jk}, k=1,2,..,m and to each subcell we assign the corresponding value of f^k, k=1,2,..,m. A complex of subcells now corresponds to a single product K with the set $I \subseteq \{1,2,..,m\}$ of subnumbers k indicating the functions f^k which this product implies.

As has been stated in [1], in order to apply algorithm A^1 to a concrete problem like this one, it is necessary to have a rule for determining the complexes and a method based on this rule for generating a star G(e), i.e. a set of maximal complexes covering the cell e (or subcell e in the case of $m > 1$). If we interpret the diagram as the logical diagram described in [2,3], a complex of cells has a very simple geometrical interpretation and in the case of hand calculations a star G(e) can be easily determined visually. However, to carry out the algorithm in machine calculations we need to work out an analytical method for generating stars which, at the same time, is suitable for a digital computer.

2.2. An Algorithm for Generating a Star G(j)

A star G(e) of a cell $e \in F^1$ is a set of all maximal complexes covering cell e or, in the case of multiple-output switching functions, subcell e. In the latter case we shall call the star G(e) a multiple-output star. For the purpose of machine calculations rather that determining a star for a cell e, we shall determine it for a product K(e) corresponding to this cell. In this case a star, denoted as G(j), where j - number of cell e, is a set of all prime implicants of a given function f which are included in product K(e).

We will describe here an algorithm G^2 for generating star G(j), which is based only on the knowledge of the sets F^1 and F^0. So we do not need to know and record in the computer the set F^*, which can be very large in practice, especially in the case of numerous input variables.

First we assume that m = 1; the case of $m > 1$ will be considered later. Let $j \in F^1$. A product $K(j) = x_1^{\sigma_1} x_2^{\sigma_2} \cdot\cdot x_n^{\sigma_n}$, such that $\sum_{i=1}^{n} \sigma_i 2^{n-i} = j$ is a fundamental product of the function f. Let $F^0 = \{j_l\}_{l=1}^{z}$. Let $K_1, K_2, \ldots, K_z$ denote products corresponding to numbers j_1, that is $K_1 = x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_n^{\sigma_n}$, where $\sum_{i=1}^{n} \sigma_i 2^{n-i} = j_1$.

An implicant of f included in K(j), that is an element of G(j), is a product consisting of literals of K(j) and not included in any product K_1. The implicant is a prime implicant of f if it is minimal under inclusion, which means that it itself does not include any other implicant of f. Let $\tilde{K}$ and $\tilde{K}_l$, l=1,2,..,z denote sets of literals of products K and K_1, respectively.

Algorithm G^2:

1. Determine sets $\tilde{D}_l = \tilde{K} \setminus \tilde{K}_l$, l=1,2,..,z
2. Set up a function:

$$f_g = \bigwedge_{l=1}^{z} D_l \qquad (2)$$

where $D_l = \bigvee_{x_i^{\sigma_i} \in \tilde{D}_l} x_i^{\sigma_i}$

3. Find the irredundant disjuntive normal form $D(f_g)$ of function f_g. It can be done by logicelly multiplying each sum D_l in (2) by all the others and applying the absorption laws:

$$x_a^{\sigma_a} (x_a^{\sigma_a} \vee x_b^{\sigma_b} \vee x_c^{\sigma_c} \vee \ldots) = x_a^{\sigma_a} \qquad (3)$$

$$x_a^{\sigma_a} \vee x_a^{\sigma_a} x_b^{\sigma_b} x_c^{\sigma_c} \ldots = x_a^{\sigma_a} \qquad (4)$$

The star G(j) is the set of all components of $D(f_g)$. The proof of the above statement is in [3].

Let us determine now a star $G(j^k)$ in case of $m > 1$, where j^k corresponds to subcell e^{jk}, $j \in F^{1,k}$, $k \in \{1,2,..,m\}$. The star $G(j^k)$ is a set of all multiple-output prime implicants of f which are included in product K(j) and implying function f^k. An m-output switching function f can be determined by means of pairs of sets: $(F^{1,k}, F^{0,k})$, k=1, 2,..,m. First we determine the set $I(j^k) = \{k_l\}_{l=0}^{w}$ of indices k such that sets $F^{\cdot,k}$ do not include number j. Of course $k \in I(j^k)$ and $w-1 \leq m$. Let $k_0 = k$.

Let $\{I_s\}_{s=0}^{2^w-1}$ be the family of all subsets of $I(j^k)$ which include index k and $F_s^0 = \{j_l^s\}_{l=1}^{z_s}$ - the intersection $\bigcap_{k \in I_s} F^{0,k}$.

Now we realise the algorithm G^2 substituting the products K_1, l=1,2,..,z first with K_1^0, l=1,2,..,z_0, then with K_1^1, l=1,2,..,z_1 ... and so on up to $K_1^{2^w-1}$, l=1,2,..,z_{μ}, $\mu = 2^w - 1$, where K_1^s is a product corresponding to j_1^s. Instead of one function f_g we have now a sequence $f_g^0, f_g^1, \ldots, f_g^{2^w-1}$.

The set of all components of each $D(f_g^s)$ we will call a substar $G^s(j^k)$. Elements of $G^s(j^k)$ will be denoted as a concatenation $K(j)I_s$. The set I_s we will call a set of indices of product $K(j)$. The star $G(j^k)$ is determined as a set-theoretic sum:

$$G(j^k) = \bigcup_{s=1}^{2^r-1} G^s(j^k) \qquad (5)$$

2.3. Realization of Algorithm A^q in case of machine minimization of switching functions

First we consider the case of $m = 1$. Suppose we are given an one-output switching function f, which is determined by means of sets F^1 and F^0. Knowing the rule for generating a star $G(j)$ we directly realize algorithm according to its flow-diagram given in [1]. However, since we start from the knowledge of sets F^1 and F^0, and not the image $T(f)$, we must make some corrections. The corrected flow-diagram of algorithm A^q presents fig.1.

F^1, F^0, F^p, M^q - in the flow-diagram denote variables, whose values are sets. The beginning values of F^1 and F^0 are the previously defined sets F^1 and F^0. F^p and M^q are auxiliary variables.

$O^p(F; j_1)$ denotes the operation of choosing the smallest number from the current value of variable F and assigning the notation j_1 to it.

K^q is a product corresponding to a quasi-extremal in [1], i.e. a product of $G(j)$ which is included in the maximal number of products corresponding to elements of the current value of F^1.

Henceforth, the elements of F^1 such that products corresponding to them include a product K will be simply reffered to as elements covered by K.

K^m is a product corresponding to complex L^m in [1], i.e. minimal cost product of $G(j)$.

K^q is the set of all elements of F^1 covered by K^q.

$G^u(j)$ is the set of all elements of F^1 which are covered by any of the products of $G(j)$.

$z(K)$ is a cost of product K determined as the number of literals in it.

The final value of variable M^q is a set of products - components of a quasi-minimal form $M^q(f)$ of the function f. The final values of Δ and δ are the maximal possible difference between the form obtained and the minimal one in number of components and literals, respectively.

Now we shall consider the case of $m > 1$. Suppose we are given an m-output switching function f, which is determined by means of pairs of sets $(F^{1,k}, F^{0,k})$, $k=1,2,\ldots,m$. Each single pair $(F^{1,k}, F^{0,k})$ determines an one-output function f^k. Let us assign to each number $j \in F^{1,k}$ the index $k : F^{1,k} = \{j_1^k, j_2^k \ldots j_i^k \ldots\}$, $k=1,2,\ldots,m$. Hence, the following sum is produced:

$$F^1 = \bigcup_{k=1}^{m} F^{1,k} \qquad (6)$$

and ordered in the following way: the numbers of different value are ordered by relation $<$, the numbers of the same value are ordered by relation $<$ referring to their indices k.

Example : Let $F^{1,1} = \{1,3,5,8\}$
$F^{1,2} = \{1,2,6,8,9\}$
$F^{1,3} = \{2,3,4,8\}$

Then: $F^1 = \{1^1, 1^2, 2^2, 2^3, 3^1, 3^3, 4^3, 5^1, 6^2, 8^1, 8^2, 8^3, 9^3\}$.

The algorithm A^q (fig.1) is now realized in reference to the above-defined set F^1. The stars $G(j^k)$, $j^k \in F^1$ are constracted according to the algorithm described in 2.2. The cost z(K) is determined as a sum of number of literals in it plus number of functions f^k which this product implies (number of elements in set I conected with the product).

The result of algorithm is a cover $M^q(f)$ - the final value of M^q - consisting of terms $K_i I_i$, $i=1,2,\ldots$, where K_i is a product and I_i - the set of indices indicating functions f^k implied by this product. From the cover $M^q(f)$ a quasi-minimal form $M^q(f^k)$ of each f^k is determined as:

$$M^q(f^k) = \bigvee_{\{i: k \in I_i\}} K_i \,, \quad k=1,2,\ldots,m \qquad (7)$$

In this case ($m > 1$), however, the cover $M^q(f)$ can have some redundant numbers in sets I_i, and, if this occours, some redundant literals in K_i. Then, before we determine the final forms $M^q(f^k)$, we must check whether or not this redundancy occurs, and if it does, then it must be reduced.

2.4. Reduction of Redundancy in $M^q(f)$

First the redundancy in sets I_i is tested. This is carried out according to the algorithm presented in fig.2.

The variables N and F^p are auxiliary variables. $c(G)$ is the number of elements in G.

$\tilde{G}$ is the set of numbers of $F^{1,k}$ which are covered by the product of set G.

This algorithm may seem to be somewhat complicated, but in fact it is very easy to perform, because the sets - values of G-may consist of one or very small number of elements. A result of the algorithm are sets M_k, $k = 1,2,..,m$. If a set $M_{\hat{k}}$, $\hat{k} \in \{1,2,..,m\}$, does not include a product K_i, whose set of indices I_i contains $\hat{k}$, the index $\hat{k}$ is removed from I_i.

Each product K_i, whose set I_i has been reduced, is then checked for redundant literals. The first literal is removed and the remainder is tested to see whether it covers any number of the set $\bigcap_{k \in I_i} F^{0,k}$. If it does not cover any of such a number, the remainder is still an implicant of f. Then the next literal is removed (from the remainder) and the previous procedure repeated.

If the remainder does cover a number of the above set, the literal cannot be removed from the product, so the next literal is removed from the product and the above procedure is again repeated. This is continued until the last literal of the product is considered.

The algorithm described above does not guarantee that the shortest implicant is obtained, however, it always gives an irredundant one. To find, in general, the shortest implicant, the inspection of the other subproducts can be required. Having reduced the redundancy in $M^q(f)$ the parameters Δ (when whole products happened to be redundant) and δ are appropriately corrected and according to (7) the forms $M^q(f^k)$ are determind.

2.5. Improving Algorithm Results by an Adaptive Iterative Process

Let us assume that the algorithm has been realized and a cover $M^q(f)$ determined, but values of Δ and δ are considered to be too large. If the algorithm is repeated whilst selecting the other quasi-extremals from stars and/or generating stars of numbers j other than the smallest in the sets specified by values of F^p (as is done in the first execution of the algorithm), then a simpler cover $M^q(f)$ and smaller parameters Δ and δ may be obtained.

A natural question arises: How should the algorithm be repeated in order to obtain a better solution, approximating the minimal one and, at the same time, having Δ and δ as small as possible ? While the algorithm is being executed some knowledge of the function itself is obtained, e.g. the knowledge of numbers of elements in stars G(j), of the number of disjoint stars and other information. If needed, some additional information of the function can also be measured. Repeating the algorithm all this information could be used to obtain a better solution. So a certain adaptive process should be realized. But how it could be done ? Here there are many different heuristic approaches possible, some of which have been tested in our programs. We will describe here one approach, which is very simple and at the same time usually produces good results in practice.

During the first execution of the algorithm numbers of elements in generated stars G(j) and the number $\mathcal{G}_1$ of elements in the family of disjoint stars are recorded. Then, after the execution of the algorithm, the elements of F^1, for which the stars were generated, are ordered by relation $<$ referred to numbers of elements in their stars. Then they are put at the begining of the set F^1. Thus, the next execution of the algorithm starts with the elements of F^1, whose stars have an initially small but subsequentely ever-increasing number of elements. Although the advantages of this procedure can be easily explained, the scope of this paper does not permit us to delve into this subject.

In the second execution parameter Δ is calculated as:

$$\Delta = c(M^q) - \max(\mathcal{G}_1, \mathcal{G}_2) \quad (8)$$

where $\mathcal{G}_2$ is the number of elements in the family of disjoint stars obtained in the second execution.

In the case of $\mathcal{G}_2 > \mathcal{G}_1$ parameter δ can be approximately evaluated by substraction from the value of δ obtained as in fig. 1 the number of literals in the last (or more precisely the shortest) $\mathcal{G}_2 - \mathcal{G}_1$ products of the now obtained cover. Analogously to the above the third and subsequent executions of the algorithm can be performed, till the parameters Δ and δ become small enough for practice or the cost of consecutive covers stops decreasing.

3. Results of the Computer Investigations of the Algorithm

The algorithm for switching functions minimization described above has been programed in the LYAPAS language [4,5] for the ODRA 1204 computer. Two programs have been developed: 1) A^q-NORMIN 17a (this is the notation of the last version of different modifications of the program,
2) A^q-NORMIN w1 .

The first program permits the minimization of only one-output switching functions (up to 31 variables) and was designed mainly for the theoretical investigations of the algorithm A^q (e.g. making it possible to measure many different parameters characterizing the minimization process).

The second program realizes the minimization of multiple-output switching functions. The input information of a switching function to be minimized in the program consists of the pairs of sets $(F^{1,k}, F^{0,k})$, $k = 1,2,\ldots,m$. Elements of these sets are recorded in the following way: excluding one bit of a LYAPAS word (32 bits) for the auxiliary purposes, the reminder is divided into 2 parts: in the first part a number j is contained; in the second is contained a binary number whose elements of value 1 indicates sets $F^{1,k}$ to which the number j belongs. Thus, the maximal number of input variables n and outputs m of a switching function that can be minimized in the program must fill out the inequality:

$$n + m \leq 31 \qquad (9)$$

An element K_iI_i of a star $G(j^k)$ is recorded in a single word, which is divided into 2 parts (excluding one auxiliary bit): in the first part is contained a binary number whose elements of value 1 indicate literals from the product K(j) included in K_i and in the second - a binary number, whose elements of value 1 indicate functions f^k implied by the product.

In order to record the entire star $G(j^k)$ (what is needed[1]) to determine products K^q and K^m of the star) it suffices to record all the elements of said star in the way described above plus one additional word recording the product K(j). Consequently we thus arrive at an extremely economical way of recording prime implicants of the function.

Table I presents the results of minimizing a sampling of 16 switching functions achieved by program A^q-NORMIN 17a. Functions to be minimized were determined by means of sets F^1 and F^0. Numbers of elements in these sets are given in columns $c(F^1)$ and $c(F^0)$. The column Iter. contains numbers of algorithm iterations performed for each function. The column $c(M^q)$ contains numbers of elements in obtained covers, but in the case when more then 1 iteration for a given function was performed it containes 2 numbers: first indicates the number of elements in the cover obtained in the first iteration and the second one - in the simplest cover obtained after all iterations. Corresponding to the above covers values of parameters Δ and δ are given in columns denoted by Δ and δ . Column c(G) contains the average numbers of elements in the stars generated in the last iteration of the algorithm.

Table II presents results of the minimization of 8 multiple-output switching functions. Column $c(M^{q,k})$ contains numbers of components in forms $M^q(f^k)$, $k=1,2,..,m$, determined from cover $M^q(f)$. Some of the functions considered in the investigations have been taken from practice, while others have been randomly produced by a special function-generator.

At the presently operational translator of the LYAPAS language for the ODRA 1204 computer the programs in this language are performed by the computer at the speed of approximately 2000 oper./sec. This speed is about 20 times slower then the execution speed of the programs in the machine language. The decrease of speed was caused by the fact that to represent one LYAPAS word (32 bits) in the computer two machine words (24 bits) were taken. At the above speed the execution time of the programs were: e.g. for functions 3,7,12,13 and 16 from table I - 0.5, 2, 8, 20 and 80 min., respectively, and for functions 1,3,5,7 and 8 from table II - 0.25, 3.5, 14, 1.3 and 7 min., respectively.

---

1) Worth noting is the fact that in the case of $m > 1$ to determine products K^q and K^m it is sufficient to record at once only substars $G^s(j^k)$.

4. Conclusion

The algorithm and computer programs described in the paper belong to a number of algorithms and programs developed in the Institute of Automatic Control of the Polish Academy of Sciences for the automatic synthesis of combinational switching circuits. Among them a program for the synthesis of three-level networks on NOR and NAND elements, so-called TANT networks, based on the algorithm A^q, has also been developed. The latter one will be described in a separate paper.-

References

1. Michalski R.S., On the quasi-minimal solution of the general covering problem, Proceedings of the Vth Yugoslav International Symposium on Information Processing (FCIP 69), Bled, October 8-11, 1969, vol. A3, pp.125-128.

2. Michalski R.S., Recognition of total or partial symmetry in a completely or incompletely specified switching function, Vth Congress of IFAC, Warsaw, June 16-21, 1969, Finite Automata and Switching Systems, vol.27, pp.109-129.

3. Michalski R.S., Synteza wyrażeń minimalnych i rozpoznawanie symetrii funkcji logicznych, Prace Instytutu Automatyki PAN, z.91, 1970.

4. Logicheskii yazyk dlya predstavleniya algoritmov sinteza releinykh ustroistv, Izd. "Nauka", Moskva 1966.

5. Michalski A., Wiewiórowski T., Odra Ljapas, Computation Centre of the Polish Academy of Sciences Reports, vol.4, Warszawa, 1970.-

n	m	No	c(F¹)	c(F⁰)	Iter.	c(K^q)	Δ	δ	c(G)
6	1	1	14	40	1	7	0	0	1,2
		2	30	30	7	15/11	5/0	23/0	2,3
		3	30	30	2	12/12	2/0	7/0	1,5
		4	50	13	2	15/13	4/2	14/8	3,5
7	1	5	50	50	3	23/20	5/2	18/13	2,2
		6	50	75	2	24/23	3/1	18/6	1,9
		7	65	63	5	26/24	5/2	29/13	2,4
		8	100	28	5	24/23	8/2	37/13	3,8
8	1	9	100	35	2	26/24	6/3	28/16	3,6
		10	100	70	6	31/30	8/2	45/12	3,8
		11	100	100	6	43/40	11/3	63/20	2,5
		12	150	60	6	37/34	7/5	80/27	4
10	1	13	100	75	2	26/24	14/7	73/39	18,0
15	1	14	40	25	3	8/7	5/3	20/12	165,0
20	1	15	30	15	2	5/5	4/3	10/9	360,0
25	1	16	25	15	1	4	1	4	397,0

TABLE I

n	m	No	$o(P^{1,k})$ k=1,2,...,m	$o(P^{0,k})$	Iter.	$o(M^q)$	$o(M^{q,k})$	Δ
4	4	1	7 8 11 7	8 8 5 8	1	14	3 6 6 3	1
6	2	2	25 27	27 27	1	25	15 17	5
	3	3	25 27 30	27 27 25	1	29	15 17 17	5
	4	4	25 27 30 33	27 27 25 20	1	36	15 18 18 14	8
	6	5	25 27 30 33 31 22	27 27 25 20 21 32	1	49	13 14 13 12 15 12	12
	8	6	25 27 30 33 31 22 24 28	27 27 25 20 21 32 29 27	1	60	14 14 15 12 18 13 14 13	19
7	5	7	28 12 12 6 6	60 76 75 82 82	1	11	7 2 2 2 2	0
	11	8	68 32 24 12 18 12 14 6 14 6 38	28 64 72 84 78 84 82 84 82 90 58 90	3	24/23	13 10 4 4 6 4 4 2 4 2 8	6/1

TABLE II.

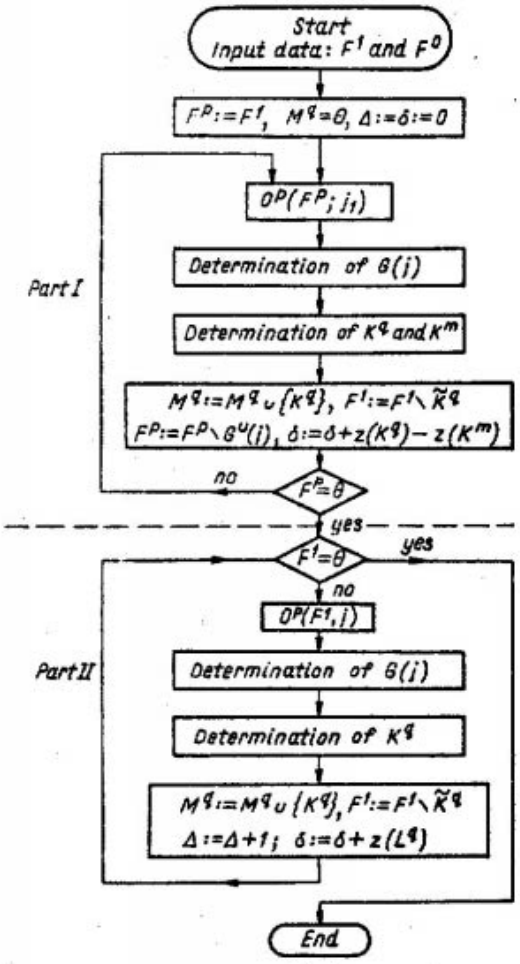
Start
Input data: F^1 and F^0
$F^P := F^1$, $M^q = \theta$, $\Delta := \delta := 0$
$O^P(F^P; j_1)$
Determination of $G(j)$
Determination of K^q and K^m
$M^q := M^q \cup \{K^q\}$, $F^1 := F^1 \setminus \tilde{K}^q$
$F^P := F^P \setminus G^U(j)$, $\delta := \delta + z(K^q) - z(K^m)$
$F^P = \theta$
no
yes
Part I
$F^1 = \theta$
yes
no
$O^P(F^1, j)$
Determination of $G(j)$
Determination of K^q
$M^q := M^q \cup \{K^q\}$, $F^1 := F^1 \setminus \tilde{K}^q$
$\Delta := \Delta + 1$; $\delta := \delta + z(L^q)$
Part II
End

Fig. 1

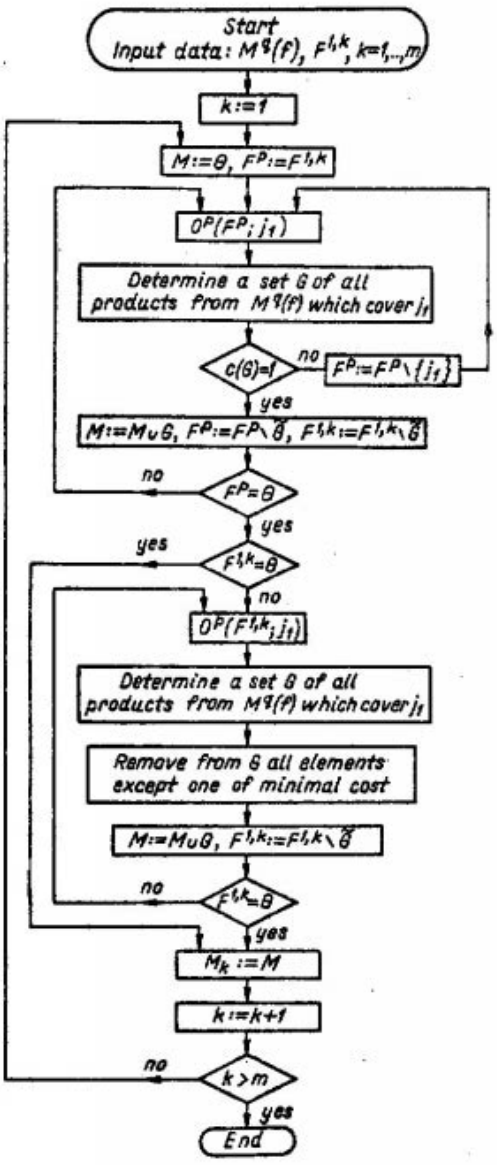
Start
Input data: $M^q(f)$, $F^{1,k}$, $k=1,..,m$
$k := 1$
$M := \theta$, $F^P := F^{1,k}$
$O^P(F^P; j_1)$
Determine a set G of all products from $M^q(f)$ which cover j_1
$c(G) = 1$
no
$F^P := F^P \setminus \{j_1\}$
yes
$M := M \cup G$, $F^P := F^P \setminus \tilde{G}$, $F^{1,k} := F^{1,k} \setminus \tilde{G}$
no
$F^P = \theta$
yes
yes
$F^{1,k} = \theta$
no
$O^P(F^{1,k}; j_1)$
Determine a set G of all products from $M^q(f)$ which cover j_1
Remove from G all elements except one of minimal cost
$M := M \cup G$, $F^{1,k} := F^{1,k} \setminus \tilde{G}$
no
$F^{1,k} = \theta$
yes
$M_k := M$
$k := k+1$
no
$k > m$
yes
End

Fig. 2