AN ABSTRACT OF THE THESIS OF

Mária Edith Csermelyi Balogh for the Ph.D. in Applied Mathematics
(Name) (Degree) (Major)

Date thesis is presented August 14, 1964

Title ON SCHEDULING ALGORITHMS

Redacted for privacy

Abstract approved
(Major professor)

General techniques for constructing Scheduling Algorithms are described as well as their applications to two diverse problems, namely the scheduling at a university and the imitation of the activity of a taxonomist in microbiology.

ON SCHEDULING ALGORITHMS

by

MÁRIA EDITH CSERMELYI BALOGH

A THESIS

submitted to

OREGON STATE UNIVERSITY

in partial fulfillment of
the requirements for the
degree of

DOCTOR OF PHILOSOPHY

June 1965

APPROVED:

Redacted for privacy

Professor of Mathematics

In Charge of Major

Redacted for privacy

Chairman of Department of Mathematics

Redacted for privacy

Dean of Graduate School

Date thesis is presented August 14, 1964

Typed by Illa W. Atwood

ACKNOWLEDGEMENTS

The author wishes to thank Professor H. E. Goheen for his generous assistance in the completion of this thesis, Dr. J. Beard, Dr. B. Y. Kersh and Dr. Priscilla Kilbourn for their consultation and advice.

The major part of this work was done in connection with a production programming effort for the Oregon State Board of Higher Education through the State Grant Account No. 2-1756, "Scheduling Students by Computers."

TABLE OF CONTENTS

	Page
Introduction	1
Part I. Theory	4
1.1. Definition, Phrasing of the Problem.	5
1.2. Fundamental Algorithm GA'.	9
1.3. An Algorithm for the Elimination of Resolvable Conflicts: CA'.	12
1.4. An Estimation for the Number of Necessary Tests to be Performed Using the Algorithms GA'. and CA'.	15
1.5. Scheduling-Compiler-Algorithm: SA.	20
Part II. Practical Applications	23
2.1. Timetable Construction and Student Sectioning.	23
2.2. n-ary Tree Search.	30
2.3. System Analysis for Timetable Construction and Student Sectioning.	34
2.4. System Analysis for the n-ary Tree Search in Taxonomy.	44
2.5. Master Algorithm GA for Timetable Construction, Sectioning and n-ary Tree Search in Terms of Elementary Operators.	47
2.6. CA Algorithm in Terms of Elementary Operators.	60
2.7. SA Algorithm in Terms of Elementary Operators.	63
Part III. Solution of the Taxonomic Problem Using Turing Programming	67
3.1. Description of the Turing Programming.	67
3.2. Turing Program for the n-ary Tree Search in Taxonomy	80
Bibliography	80

ON SCHEDULING ALGORITHMS

INTRODUCTION

For some years now computers have been proposed for administrative purposes at colleges and universities, such as student sectioning and master schedule construction. Some new trends in secondary education makes the process of sectioning very laborious and costly and therefore the automation of these processes will be inevitable (6).

Practical methods using computers have been published to assign college students to sections according to a previously prepared master schedule (3, 11), and these methods proved to be efficient at Purdue University (2).

However, according to our knowledge, no theoretical investigation has been published so-far which would make the generalization and the practical adaptation possible to the different circumstances of different colleges. The construction of a master schedule manually is a tedious process, and the automation of it is generally considered to be a difficult problem. Computer methods of constructing master schedules for secondary schools have been proposed (8) based on iteration of boolean matrices, and limited tests (9 teachers, 9 classes, 9 hours, each teacher meets each class once) have been

carried out successfully (5). By another method described in (1) using a heuristic approach, master schedules have been produced partly by computer and completed by hand. The difficulty of the combinatorial approaches is that the number of required trials grows in an alarming way with the increase of the number of elements. Thus, using these methods, the proposed solutions are far from practical applicability for colleges.

We do not follow the methods mentioned above. We have considered the construction of the master schedule and student sectioning as a union of assignment problems, which are based on the same principal problem, which we will call the Scheduling Problem.

In Part 1 we introduce an abstract terminology to formulate the Scheduling Problem in its general and rigorous form. For this Problem we construct abstract algorithms to find the solutions in the sense that the algorithms give a solution of the abstract Scheduling Problem if there exists any or it indicates that there is no solution.

Based on this theory we are able to give an algorithmic procedure for the solution of practical problems, in particular we design a programming system for the automation of the registration process: teacher assignment, classroom assignment, and student sectioning. The given algorithms can be easily translated into machine language programs for any digital computer with tape units. An essential feature of these algorithms is that we are able to keep track of the

operations during processing and to change initial data to reduce rejects.

Moreover we have used the fact that the n-ary Tree Search is a special case of our abstract Scheduling Problem which can be solved also by the given master algorithm. The output of this algorithm depends on the ordering of the input data. We give in Part 3 another solution for the n-ary Tree Search using Turing Programming, which is independent of the ordering of the input data. The given Turing Program simulates the methods used by a taxonomist.

PART 1. THEORY

1.1. Definitions, Phrasing of the Problem.

Consider a finite set S of n elements, each element of which is completely described in the following form:

$$e \equiv k\, d_1\, d_2, \ldots, d_\ell\, t.$$

Here k is called the kernel of the element, d_i $(i = 1, \ldots, \ell)$ the details of the element and t the resource-index of the element (t is a non-negative integer). Let A be a subset of S. We say, that A is compatible, if certain restrictions on the details of its elements are satisfied, and we say that A is realistic, if the t-index in each of its element is positive.

We want to investigate the following problem:
Given a finite set H of kernels:

$$H = \{k_1, k_2, \ldots, k_g\}.$$

Construct a compatible, realistic subset A of S by completing each kernel in H to an element of S. We will call problems of this kind Scheduling Problems.

Obviously a necessary condition to find such a subset A is, that $H \subset K$, where K is the set of all kernels of the elements of S.

To formulate the Scheduling Problem more precisely we

introduce some notations and definitions.

Let S be represented as an arrangement of the form:

$$
\begin{array}{l}
k_1\ {}^1d_{11}\ {}^1d_{12}\ \ldots,\ {}^1d_{1\ell}\ {}^1t_1 \\
k_1\ {}^1d_{21}\ {}^1d_{22}\ \ldots,\ {}^1d_{2\ell}\ {}^1t_2 \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
k_1\ {}^1d_{r_1 1}\ {}^1d_{r_1 2}\ \ldots,\ {}^1d_{r_1 \ell}\ {}^1t_{r_1}
\end{array}
$$

$$
\begin{array}{l}
k_2\ {}^2d_{11}\ {}^2d_{12}\ \ldots,\ {}^2d_{1\ell}\ {}^2t_1 \\
k_2\ {}^2d_{21}\ {}^2d_{22}\ \ldots,\ {}^2d_{2\ell}\ {}^2t_2 \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
k_2\ {}^2d_{r_2 1}\ {}^2d_{r_2 2}\ \ldots,\ {}^2d_{r_2 \ell}\ {}^2t_{r_2} \\
\vdots \\
k_s\ {}^sd_{11}\ {}^sd_{12}\ \ldots,\ {}^sd_{1\ell}\ {}^st_1 \\
k_s\ {}^sd_{21}\ {}^sd_{22}\ \ldots,\ {}^sd_{2\ell}\ {}^st_2 \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
k_s\ {}^sd_{r_s 1}\ {}^sd_{r_s 2}\ \ldots,\ {}^sd_{r_s \ell}\ {}^st_{r_s},
\end{array}
$$

where $k_i \neq k_j$ if $i \neq j$, for all $i, j \in \{1, \ldots, s\}$.

Define S_{k_i} for fixed k_i by $S_{k_i} = \{e \mid e \in S\}$. Note, that in the given arrangement: r_i is the number of elements in S_{k_i}; ℓ is the number of details of any element of S_{k_i}. To each kernel $k_j \in K$ there exists a unique $r_j x \ell+1$ matrix Ψ_j, such that each row-vector of Ψ_j can complete k_j to an element of S.

$$k_j \longleftrightarrow \Psi_j = \begin{pmatrix} {}^{j}d_{11} & \cdots & {}^{j}d_{1\ell} & {}^{j}t_1 \\ \vdots & & & \vdots \\ {}^{j}d_{r_j 1} & \cdots & {}^{j}d_{r_j \ell} & {}^{j}t_{r_j} \end{pmatrix}.$$

Denote by ${}^{j}Z_i$ the i-th column-vector of Ψ_j, $i = 1,..,\ell$ and $j = 1,..,s$ and by ${}^{j}V_i$ the i-th row-vector of Ψ_j, $i = 1,..,r_j$ and $j = 1,..,s$.

To describe the above mentioned restrictions more precisely, let us introduce relations defined on the set of elements of row-vectors and column-vectors.

Definition: R is a row-relation if and only if for any given element $r \in R$ there exist elements ${}^{i}d_{jk}$ and ${}^{i}d_{jk+1}$ such that $r = [{}^{i}d_{jk}, {}^{i}d_{jk+1}]$ for ${}^{i}d_{jk}, {}^{i}d_{jk+1} \in {}^{i}V_j$, $(j = 1,..,r_i)$, $(i = 1,..,s)$, where $[a,b]$ denotes an ordered pair.

The construction of n-place (or n-ary) row-relations R_n under this definition may be exemplified by considering ternary, that is three-place relations.

A set R_3 is a ternary row-relation if and only if R_3 is a relation and for any given $r \epsilon R_3$ there exist ${}^{i}d_{jk}$, ${}^{i}d_{jk+1}$ and ${}^{i}d_{jk+2}$ such that

$$r = [[{}^{i}d_{jk}, {}^{i}d_{jk+1}], {}^{i}d_{jk+2}],$$

for ${}^{i}d_{jk}, {}^{i}d_{jk+1}, {}^{i}d_{jk+2} \epsilon {}^{i}V_j$, $j = 1,..,r_i$, $i = 1,..,s$.

Definition: C is a column-relation if and only if for any given $c \epsilon C$ there exist ${}^{i}d_{jk}$ and ${}^{i}d_{j+1k}$, such that

$$c = [{}^{i}d_{jk}, {}^{i}d_{j+1k}]$$

for ${}^{i}d_{jk}, {}^{i}d_{j+1k} \epsilon {}^{i}Z_k$, $i = 1,..,s$, $k = 1,..,\ell$. We define the n-ary column-relations C_n similarly to the above given definition of R_n.

Definition: M is a mixed row-column relation if and only if for any given $m \epsilon M$ there exist ${}^{i}d_{jk}$, ${}^{p}d_{j+qk}$, ${}^{i}d_{jk+1}$ and ${}^{p}d_{j+qk+1}$ such that

$$m = [({}^{i}d_{jk} - {}^{p}d_{j+qk}), ({}^{i}d_{jk+1} - {}^{p}d_{j+qk+1})]$$

for ${}^{i}d_{jk}, {}^{i}d_{jk+1} \epsilon {}^{i}V_j$ and ${}^{p}d_{j+qk}, {}^{p}d_{j+qk+1} \epsilon {}^{p}V_{j+q}$

for some i, and $p \epsilon \{1,..,s\}$,

$j \epsilon \{1,..,r_i\}$,

$j+q \epsilon \{1,..,r_p\}$,

$k \epsilon \{1,..,\ell\}$.

In order to simplify notations, we introduce some new notations:

$[{}^{i}d_{jk}, {}^{i}d_{jk+1}] \in R_2$ if and only if $R_2(j,k)$, i.e. R_2 is a binary relation on the j-th row-vector between the k-th and (k + 1)-st elements.

$[[{}^{i}d_{jk}, {}^{i}d_{jk+1}], {}^{i}d_{jk+2}] \in R_3$ if and only if $R_3(j,k)$, i.e. R_3 is a ternary relation among the neighbouring elements of the j-th row-vector starting with the k-th element. $[{}^{i}d_{jk}, {}^{i}d_{j+1k}] \in C_2$ if and only if $C_2(k,j)$, i.e. C_2 is a binary relation between the neighbouring elements of the k-th column-vector starting with the j-th element.

$[({}^{u}d_{jk} - {}^{v}d_{ik}), ({}^{u}d_{jk+1} - {}^{v}d_{ik+1})] \in M$ if and only if $M(i,j;k)$, i.e. M is a mixed relation among the two neighbouring elements of the i-th and the j-th row-vector starting with the k-th element

$u, v \in \{1, .., s\}$.

In connection with the above defined relations we will use logical conditions, we denote them by τ's, which are satisfied if and only if the relation to which they refer does exist.

E.g. $\tau\{C_2(k,j)\}$ is satisfied if and only if $C_2(k,j)$.

$\tau\{C_2(k,j)\}$ is not satisfied if and only if $\sim C_2(k,j)$,

where $\sim$ is the symbol used for negation.

Definition: τ is a composite condition if there exists a condition δ such, that τ is to be tested if and only if δ is satisfied; in notation: $\tau \mid \delta$.

Let ω be some logical combination (union, intersection) of

simple and composite conditions. Clearly ω is also a logical condition.

Consider $\mathbf{R}$ the set of relations defined for a Scheduling Problem. Let us denote the set of logical combinations of the corresponding logical conditions by Ω. Clearly, the elements of Ω are independent from each other in the sense, that testing one does not require the testing of any other.

Now we can formulate the Scheduling Problem in the above developed terminology:

For a given set of kernels:

$$H = \{k_1, k_2, .., k_g\}$$

find a corresponding set of row-vectors $\{{}^1V_{v_1}, {}^2V_{v_2}, .., {}^gV_{v_g}\}$ with $v_i \in \{1, 2, .., r_i\}$, $i = 1, .., g$, such that all elements of Ω' are satisfied for $\Omega' \subset \Omega$ and corresponding to $\mathbf{R}' \subset \mathbf{R}$.

1.2. Fundamental Algorithm GA'.

The following algorithm GA' will give a required set of row-vectors, if such exists, otherwise it indicates that there is a resolvable or unresolvable conflict among the elements of the set. We call a conflict resolvable if there exists an algorithm which can resolve it. We will use Lyapunov's operator programming to describe the algorithm GA'. Before giving the algorithm we present a short description of the Lyapunov operator programming (10, Sec. 32). This

programming method is based upon the fact that an algorithm can be broken into a certain succession of more or less independent steps, so that each such step may be programmed, and the solution of the problem is obtained as the union of its partial programs. The stages into which the process of solution may be divided are called operators. Corresponding to the character of the stages the operators are conditional (logical or predicate) or unconditional (e.g. arithmetic) operators. The operating scheme describes the succession and interconnection of the operators. We shall denote operators by capitals. For the convenience in describing logical schemes the operators depicting a scheme are written in one line with the following rules:

1) The ordinal number of an operator in a given scheme is expressed by an index of the operator. The numbering of the operators is the obvious one.

2) If the signs for the operators stand in order in the scheme, then the operator written on the left transfers control to the operator written on the right.

3) If a semicolon stands between two signs for operators in order, then from the operator written on the left there is no transfer of control to the operator written on the right.

4) Transfer of control to an operator not directly to the right is designated by a symbol "$\overset{k}{\lceil}$" and "$\underset{k}{\rfloor}$" respectively placed after the operator transferring control (k is the ordinal number of

the operator to which control is transferred), and by the symbol "$\overset{\ell}{\urcorner}$" and "$\underset{\ell}{\llcorner}$" respectively, placed before the operator receiving control (ℓ is the ordinal number of the operator transferring control).

5) The transfer of control not to the next operator in order may be accomplished by one of the operations of a conditional operator $_{\lambda}X_{\ell}$ on the basis of the value of a signal ω, which has the value 1 if the condition is fulfilled and 0 otherwise. If $\omega = 1$, then the logical operator transfers control to the successive operator or to an operator X_k indicated by the scheme:

$$_{\lambda}X_{\ell} \ \overset{k}{\ulcorner} \ X_{\ell+1} \ldots X_{k-1} \ \overset{\ell}{\urcorner} \ X_k.$$

If $\omega = 0$, then the logical operator transfers control to some X_p according to the scheme:

$$_{\lambda}X_{\ell} \ \underset{p}{\llcorner} \ X_{\ell+1} \ldots X_{p-1} \ \underset{\ell}{\llcorner} \ X_p.$$

We use the abbreviations for the combinations of the symbols: $\overset{k}{\ulcorner}\underset{i}{\llcorner}$ and $\underset{\ell}{\llcorner}\overset{q}{\urcorner}$ respectively, the symbols $\overset{k}{\underset{i}{\llcorner}}$ and $\underset{\ell}{\overset{q}{\urcorner}}$ respectively. The partial program belonging to an operator X_{ℓ}^{i} in the scheme can be described again by another operating scheme with operators X_{ℓ}^{i} or $_{\lambda}X_{\ell}^{i}$ (λ indicates logical operator) called elementary operators, where the commands of these are simple enough to be translated to any machine language.

Let the operator F_0 input the program and data, F_1 set $j = 1$

and F_2 set $i = 1$. Let the operator F_3 choose ${}^{j}V_i$ (by a table look-up procedure). The operator P_4 checks the logical condition: ${}^{j}t_i > 0$; if this condition is not fulfilled, then F_9 substitutes i for $i + 1$. P_{10} checks the condition: $i \leq r_j$; if this condition is not satisfied, then F_{13} will indicate that the request is not realistic. The operator P_5 checks the logical conditions in Ω' for the already chosen ${}^{j}V_i$'s. If the conditions are not satisfied, the logical operator P_{11} checks the condition: $i = r_j$; if the condition is not fulfilled, then branch back to F_9. F_{12} indicates that the conflict is not resolvable by the algorithm GA'. The operator F_6 indicates, that so far there is no conflict and F_7 substitutes j for $j+1$. The operator P_8 checks the logical condition: $j \leq g$; if this condition is not fulfilled, then F_{14} puts the result out. The operator F_{15} stops the program. The algorithm is the following:

$$F_0 F_1 \overset{8}{\urcorner} F_2 \overset{10}{\urcorner} F_3 P_4 \underset{9}{\llcorner} P_5 \underset{11}{\llcorner} F_6 F_7 P_8 \underset{14}{\sqsubset} ;$$

$$\underset{4,11}{\llcorner} F_9 P_{10} \underset{13}{\sqsubset} ; \underset{5}{\llcorner} P_{11} \underset{9}{\llcorner} F_{12} \underset{10}{\llcorner} F_{13} \underset{8}{\llcorner} F_{14} F_{15}.$$

1.3. An Algorithm for the Elimination of Resolvable Conflicts: CA'.

In the algorithm GA', the operator F_{12} indicates a conflict, which is not resolvable by GA'.

Assume 1) That F_{12} indicates the conflict at the processing

of the n-th request: k_n;

2) That for the preceding $n-1$ requests, the corresponding set of row-vectors, which satisfy the requirements in the phrasing of the problem, is the following: $\{{}^1V_{q_1}, {}^2V_{q_2}, .., {}^{n-1}V_{q_{n-1}}\}$, where (as it is given in the general formulation) ${}^kV_{q_k}$ is the q_k-th row-vector of Ψ_k, $q_k \in \{1, 2, .., r_k\}$, $k = 1, .., (n-1)$.

In order to simplify notations we give the following definition:

Let P_n be the set of any n compatible row-vectors, i.e. $({}^1V_{q_1}, {}^2V_{q_2}, .., {}^nV_{q_n}) \in P_n$ if and only if $\{{}^1V_{q_1}, {}^2V_{q_2}, .., {}^nV_{q_n}\}$ are compatible. Using this formalism, we can phrase Assumption 1, as:

$$({}^1V_{q_1}, {}^2V_{q_2}, .., {}^{(n-1)}V_{q_{(n-1)}}) \in P_{(n-1)} \text{ but}$$

$$({}^1V_{q_1}, {}^2V_{q_2}, .., {}^{(n-1)}V_{q_{(n-1)}} {}^nV_{q_n}) \notin P_n,$$

for all $q_n \in \{1, 2, .., r_n\}$.

The recursive algorithm CA', given below will construct a compatible set of row-vectors: ${}^1V_{p_1}, {}^2V_{p_2}, .., {}^nV_{p_n}$ ($p_i \in \{1, 2, .., r_i\}$, $i = 1, .., n$), if there exists any, or it will show, that no such compatible set exist, ($p_i \in \{1, .., r_i\}$, $i = 1, .., n$). Again we will use the Lyapunov notation to describe the algorithm CA'.

Let the operator G_0 open the algorithm. G_1 sets $t = 1$. G_2 substitutes $q_{(n-t)}$ for $q_{(n-t)}+1$. Q_3 checks the logical

condition: $q_{(n-t)} \leq r_{(n-t)}$; if the condition is not satisfied, G_8 will increase t by 1. The logical operator Q_9 checks the condition: $t \geq n$; if the condition does not hold, then branch back to G_2. The operator Q_4 checks the logical condition: $({}^1V_{q_1}, {}^2V_{q_2}, \ldots, {}^{(n-t)}V_{q_{(n-t)}}) \in P_{(n-t)}$; if the condition is not satisfied, then go to G_2. The operator G_5 substitutes t for t-1. Q_6 checks the condition: $t \geq 0$; if the condition is not satisfied, then G_{11} indicates, that a solution is obtained. G_7 substitutes q_{n-t} for 1; G_{10} indicates, that there exist no solution at all. G_{12} closes the algorithm and operator G_{13} stops it.

The Algorithm CA':

$$G_0 G_1 \underset{4,9}{\rfloor} G_2 Q_3 \underset{8}{\lfloor} \overset{7}{\lceil} Q_4 \underset{2}{\lfloor} G_5 Q_6 \underset{11}{\lfloor} G_7 \overset{4}{\lceil};$$

$$\underset{3}{\rfloor} G_8 Q_9 \underset{2}{\lfloor} G_{10} \overset{12}{\lceil}; \quad \underset{6}{\rfloor} G_{11} \overset{10}{\lceil} G_{12} G_{13}.$$

By the construction of CA' it can be seen easily that CA' tests systematically the compatibility of the possible arrangements ${}^1V_{q_1}, {}^2V_{q_2}, \ldots, {}^nV_{q_n}$, (what the algorithm GA' did not test previously until it obtains a solution, and in the case of unresolvable incompatibility it tests <u>every</u> possible arrangement before arriving at a conclusion.

1.4. An Estimation of the Number of Necessary Tests to be Performed Using the Algorithms GA' and CA'.

As we have seen, the algorithms GA' and CA' lead to a result which is a compatible solution or the conclusion that such solution does not exist. We wish to answer the following question: What is the maximum number of logical tests M_n which are necessary to get a result? (The subscript n of M_n indicates that there is a conflict at the processing of the n-th request.)

We prove the following

Theorem:

$$M_n = \sum_{j=2}^{n} (r_{j-1} - q_{j-1}) r_j \delta_{n-j} + \sum_{j=3}^{n} r_j + q_1 r_2, \quad n \geq 3,$$

where the δ's are given by the following recursion:

$$\delta_0 = 1$$

$$\delta_{i+1} = 1 + r_{n-i} \delta_i \qquad (i = 0, \ldots, (n-3)).$$

We use mathematical induction in the proof. In order to illustrate the method of our calculation (and also the working of CA') we apply the algorithm CA' for $n = 3$.

In the following we shall use some abbreviations:

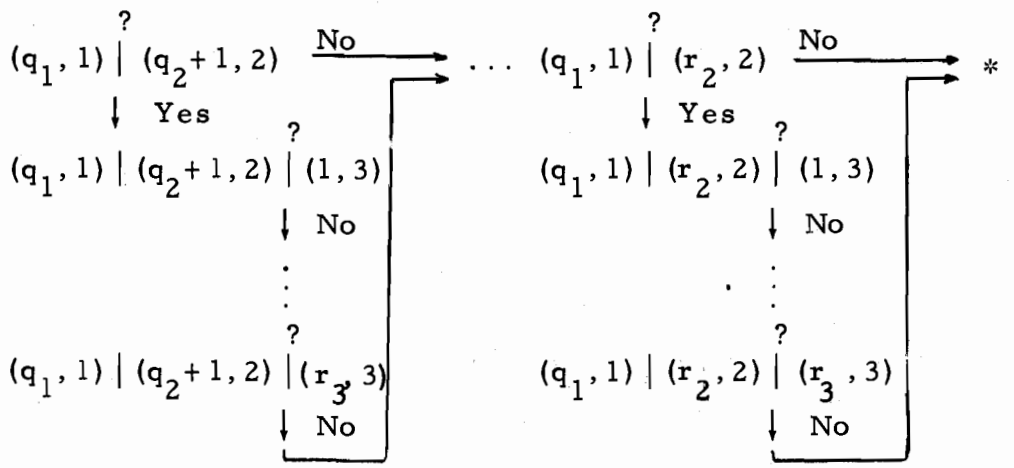
$({}^1V_{q_1}, {}^2V_{q_2}) \in P_2$ if and only if $(q_1, 1) \mid (q_2, 2)$

$({}^1V_{q_1}, {}^2V_{q_2}) \notin P_2$ if and only if $(q_1, 1) \nmid (q_2, 2)$.

The question mark above the bar: $\overset{?}{|}$ indicates a logical question.

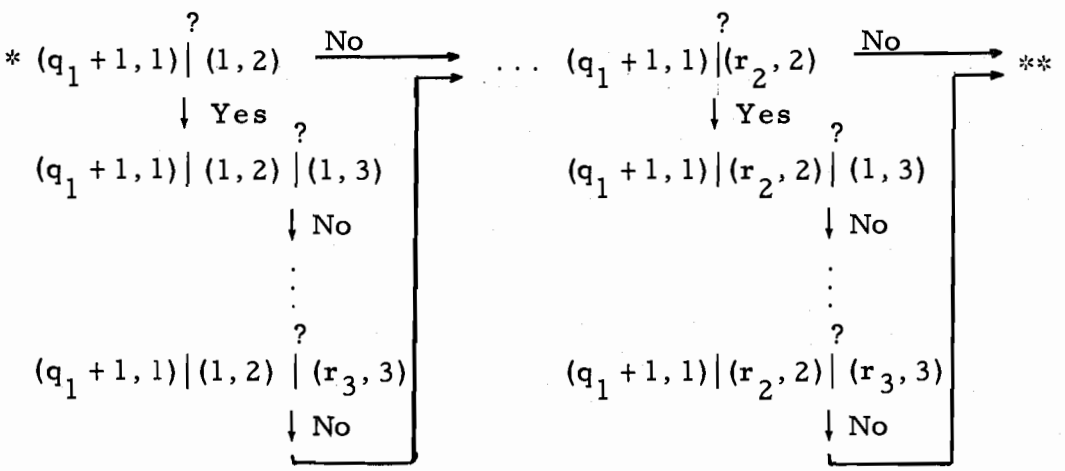
$$(q_1, 1) \overset{?}{|} (q_2, 2) \not| (i, 3) \quad \text{for all } i = 1, 2, \ldots, r_3.$$

The number of necessary tests to obtain this result: $(q_1 - 1)r_2 + q_2 + r_3$.



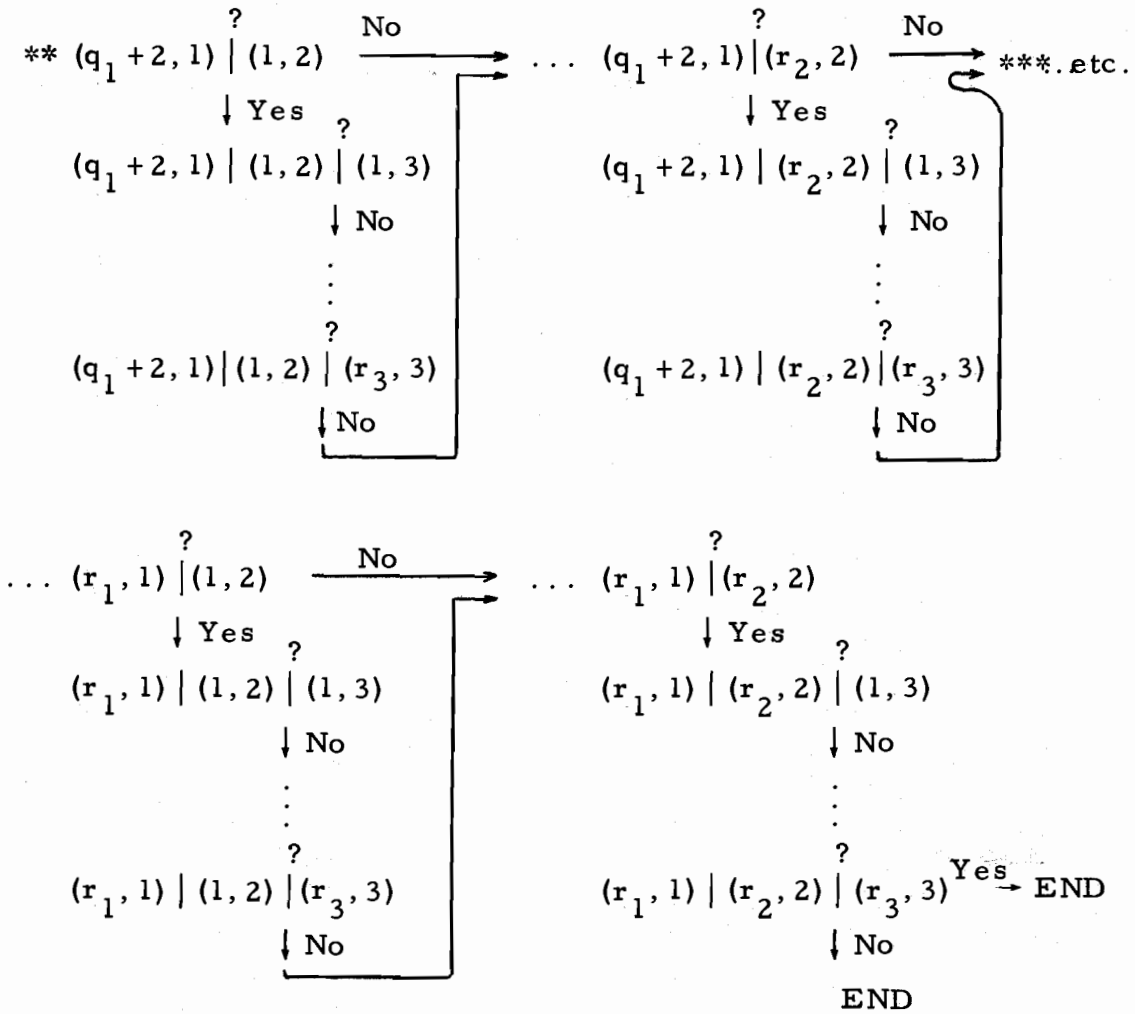
The maximal number of necessary tests until * :

$$[(q_1 - 1)r_2 + q_2 + r_3] + [(r_2 - q_2) + (r_2 - q_2)r_3].$$



The maximal number of necessary tests between * and ** is

$[r_2 + r_2 r_3]$.



The maximal number of necessary steps between * and END:

$[(r_1 - q_1)(r_2 + r_2 r_3)]$.

Thus the maximal number of necessary tests in the process:

$$M_3 = [(q_1 - 1) + q_2 + r_3] + [(r_2 - q_2) + (r_2 - q_2) r_3] +$$

$$+ [(r_1 - q_1)(r_2 + r_2 r_3)] =$$

$$= (r_1 - q_1)(r_2(1 + r_3)) + (r_2 - q_2) r_3 + r_3 + q_1 r_2.$$

We show that whenever the formula for M_i is true for $i = n$ then it is correct for $i = n+1$. At first expand the formula given for M_n.

$$M_n = (r_1 - q_1) r_2 [1 + r_3(1 + r_4(1 + r_5(\ldots(1 + r_n)\ldots)] +$$

$$+ (r_2 - q_2) r_3 [1 + r_4(1 + r_5(\ldots(1 + r_n)\ldots)] + \ldots$$

$$\ldots + (r_i - q_i) r_{i+1} [1 + r_{i+2}(1 + \ldots(1 + r_n)\ldots)] + \ldots$$

$$\ldots + (r_{n-2} - q_{n-2}) r_{n-1} [1 + r_n] +$$

$$+ (r_{n-1} - q_{n-1}) r_n + \sum_{j=3}^{n} r_j + q_1 r_2.$$

Now apply algorithm CA' to the set of $n+1$ row-vectors: $\{{}^1V_{q_1}, {}^2V_{q_2}, \ldots, {}^{n+1}V_{q_{n+1}}\}$. Then we will obtain for M_{n+1} M_n plus the following additional terms:

$$[r_{n+1} + q_n - r_n] + [r_n - q_n + (r_n - q_n) r_{n+1}] +$$

$$+ [(r_{n-1} - q_{n-1}) r_n r_{n+1}] + \ldots + [(r_i - q_i) r_{i+1} \cdots r_{n+1}]$$

$$+ \ldots + [(r_1 - q_1) r_2 r_3 \cdots r_{n-1} r_n r_{n+1}].$$

After some elementary algebraic operations we get:

$$M_{n+1} = (r_1 - q_1)\, r_2\, [1 + r_3(1 + r_4(\ldots(1 + r_{n+1})\ldots)] +$$

$$+ (r_2 - q_2)\, r_3\, [1 + r_4(1 + \ldots(1 + r_{n+1})\ldots)] + \ldots$$

$$\ldots + (r_i - q_i)\, r_{i+1}\, [1 + r_{i+2}(1 + \ldots(1 + r_{n+1})\ldots)] + \ldots$$

$$\ldots + (r_{n-2} - q_{n-2})\, r_{n-1}\, [1 + r_n(1 + r_{n+1})] +$$

$$+ (r_{n-1} - q_{n-1})\, r_n\, [1 + r_{n+1}] +$$

$$+ (r_n - q_n)\, r_{n+1} + \sum_{j=3}^{n+1} r_j + q_1 r_2 .$$

Thus

$$M_{n+1} = \sum_{j=2}^{n+1} (r_{j-1} - q_{j-1})\, r_j\, \delta_{n+1-j} + \sum_{j=3}^{n+1} r_j + q_1 r_2$$

with

$$\delta_0 = 1$$

$$\delta_{i+1} = 1 + r_{n+1-i}\, \delta_i, \qquad (i = 0, \ldots, (n-2)).$$

And so our formula for M_n is proved.

We can show by elementary manipulations that:

$$M_n \geq \sum_{j=3}^{n} r_j + r_1 r_2 \quad \text{with equality for } q_j = r_j \text{ for all } j$$

and

$$M_n \leq \sum_{t=2}^{n} \prod_{i=1}^{t} r_i \quad \text{with equality for } q_j = 1 \text{ for all } j.$$

1.5. Scheduling-Computer-Algorithm: SA.

In the algorithm GA' operator P_5 and in algorithm CA' operator Q_4 checks whether Ω', the set of logical conditions is satisfied or not in the already chosen j row-vectors ($\Omega' \subset \Omega$).

The algorithms GA^i and GA^i with CA^i are mappings of the set of matrices $\{\Psi_i, i = 1, .. s\}$ into the set of row-vectors $\{{}^iV_j, j = 1, ... r_i, i = 1, .. s\}$ in such a way that $\Psi_i \rightarrow {}^iV_j$ for some $j \in \{1, .. r_i\}$, and all conditions in Ω^i are satisfied ($\Omega^i \subset \Omega$) for the image set. Of course, it is possible, that the image set is empty, i.e. using an earlier terminology, the problem has no solution; if the image set is not empty, it is a solution of the problem.

Considering all possible subsets of Ω, we can construct the set of all possible mappings of this kind, i.e. all possible problems with different requirements and the algorithms to solve them. The corresponding set of GA^i's and CA^i's generates the class of image sets.

Now we can construct master algorithms GA and CA for the set of all conditions Ω, such that any algorithm GA^i and CA^i will be a particular algorithm written for a particular $\Omega^i \subset \Omega$.

We present an algorithm SA, which will construct the particular GA^i and CA^i for a given Ω^i from the master algorithms GA and CA, for a given set of constants given in a particular problem. The construction of the master algorithm is quite simple: in the

above given algorithms GA' and CA' let operators P_5 and Q_4, respectively, check whether the conditions in set Ω are satisfied or not.

Assume, that Ω has N elements: $\omega_1, \ldots, \omega_N$. Then operators P_5 and Q_4, respectively, consist of N tests or subroutines. Each element of Ω is the logical combination of some k simple and/or composite conditions, and each of these conditions is connected with a fixed number of details in a fixed ordering; we will refer to the values of these details and to the position of these details in the fixed ordering, as the parameters of the Scheduling Problem. These parameters were kept fixed so far; varying them we can generate a class of Scheduling Problems for each Ω^i. These classes of Scheduling Problems are those we have to deal with in any practical application. Thus, in order to generate practically usable algorithm's GA^i and CA^i, we have to be able to construct these for some given values of the parameters of the particular Scheduling Problem.

Given a subset of Ω with the corresponding fixed constants:

$$\omega_{z_1}(\bar{c}_1^{z_1}, \bar{c}_2^{z_1}, \ldots, \bar{c}_{\ell_{z_1}}^{z_1}),\ \omega_{z_2}(\bar{c}_1^{z_2}, \bar{c}_2^{z_2}, \ldots, c_{\ell_{z_2}}^{z_2}),$$

$$\ldots\ \omega_{z_n}(\bar{c}_1^{z_n}, \bar{c}_2^{z_n}, \ldots, \bar{c}_{\ell_{z_n}}^{z_n}),$$

where $\bar{c}_i^{z_i}$ is a fixed value of the parameter $c_j^{z_i}$ for $z_i \in \{1, \ldots N\}$, $i = 1, \ldots, n$, $j = 1, \ldots, \ell_{z_i}$, we describe the algorithm SA.

Operator H_0 opens the algorithm. H_1 sets $i = 1$ and H_2 sets $j = 1$. The logical operator X_3 checks the logical condition: $z_i = j$; if the condition is not satisfied H_7 increases the value of j by 1. X_8 checks the logical condition: $j \leq N$; if the condition is not satisfied operator H_9 indicates an error stop. H_4 substitutes $\bar{c}_k^{z_i}$ for c_k^j, for all $k = 1, \ldots \ell_{z_i}$. H_5 substitutes i for $i+1$. Logical operator X_6 checks the condition: $i \leq n$; if the condition is not satisfied operator H_{10} indicates that the required algorithm has been generated, operator H_{11} closes the algorithm and H_{12} stops it.

The Algorithm SA:

$$H_0 H_1 \overset{6}{\urcorner} H_2 \overset{8}{\urcorner} X_3 \underset{7}{\llcorner} H_4 H_5 X_6 \underset{10}{\boxed{}}; \underset{3}{\llcorner} H_7 X_8 \underset{9}{\boxed{}};$$

$$\underset{8}{\llcorner} H_9; \underset{6}{\llcorner} H_{10} H_{11} H_{12}.$$

It is easy to see that the output of the algorithm SA is independent of the ordering of the input sequence. The output is, of course, the required scheduling algorithm composed of a GA and a CA.

PART 2. PRACTICAL APPLICATIONS

2.1. Timetable Construction and Student Sectioning.

There is a wide range of applications of the above given theory. In this section we are going to elaborate one: the automation of the registration process at colleges and universities.

The registration process consists of four major steps:

α) Compiling a list of sections to be offered with time-schedules and the maximal number of students allowed in each section.

β) Assigning classrooms to each section.

γ) Assigning instructors to each section.

δ) Assigning students to each section.

We assume, that the α) step above has been done manually. The automation of β), γ) and δ) will illustrate an application of the Scheduling Algorithms. From the point of view of the theory these three problems are similar, differing only in the definition of the Ω_i's, i.e. the set of restrictions to be satisfied.

We will set up master algorithms GA and CA for the set Ω defined as:

$$\Omega = \{\text{TCFT, LOC, CONS, LOC}\,|\,\text{CONS, BALANCE, MAXLOAD, SEAT}\}.$$

The elements of Ω are defined as:

1) Time conflict:

$$TCFT({}^{1}V_{v_1}, {}^{2}V_{v_2}, \ldots, {}^{g}V_{v_g}) = \bigcap_{i=1}^{(\overline{n}\,\overline{m})} \tau\{C_g(i,1)\}.$$

The parameters: m consecutive time intervals in n unit time interval. We represent it as a time-vector with (nm) components. (In practical examples usually this means n days with m hours in each day.) A bar above the parameters indicates a fixed value of the parameters, what we call the constants of the problem.

2) Restriction on location:

$$LOC({}^{1}V_{v_1}, {}^{2}V_{v_2}, \ldots {}^{g}V_{v_g}) = \bigcap_{i=1}^{g} \tau\{R_2(i, \overline{\ell})\}.$$

The constants: the column index ℓ, where the ℓ-th and $(\ell+1)$-st details describe the location code.

3) Consecutive in time:

$$CONS({}^{v}V_j, {}^{w}V_k) = TCFT({}^{v}V_j, B\,{}^{w}V_k) \cup TCFT({}^{v}V_j, J\,{}^{w}V_k),$$

where B is an operator such that:

$$B\,{}^{w}V_k = B({}^{w}d_{k1}, {}^{w}d_{k2}, \ldots, {}^{w}d_{k\ell}) = ({}^{w}d_{k2}, \ldots, {}^{w}d_{k,\ell}, 0)$$

and J is an operator such that:

$$J\,{}^{w}V_k = B({}^{w}d_{k1}, {}^{w}d_{k2}, \ldots, {}^{w}d_{k\ell}) = (0, {}^{w}d_{k1}, \ldots, {}^{w}d_{k\ell-1}).$$

4) Restriction on location if consecutive in time:

$$LOC|CONS({}^{v}V_j, {}^{w}V_k) = \tau\{M(j,k;\ell)\,|\,CONS({}^{v}V_j, {}^{w}V_k)\}$$

5) Restriction on the number of busy time-intervals in each unit interval:

$$BALANCE({}^{1}V_{v_1}, \ldots, {}^{g}V_{v_g}) = \bigcup_{j=0}^{\overline{n}-1} \tau\{R_{\overline{m}}(g+1, j\overline{m}+1)\}$$

for $V_{g+1} \equiv S({}^{1}V_{v_1}, \ldots, {}^{g}V_{v_g})$, where S is the vector-sum operator.

6) Restriction on the total number of busy time-intervals:

$$MAXLOAD({}^{1}V_{v_1}, \ldots, {}^{g}V_{v_g}) = \tau\{R_{(\overline{m}\ \overline{n})}(g+1, 1)\}$$

for $V_{g+1} \equiv S({}^{1}V_{v_1}, \ldots, {}^{g}V_{v_g})$, where S is the vector-sum operator.

7) Restriction on seat number available:

$$SEAT({}^{1}V_{v_1}, \ldots, {}^{g}V_{v_g}) = \tau\{C_g(\overline{q}, 1)\},$$

where s, the resource-index is stored in the q-th column. Now we are prepared to give the particular subsets of Ω for the problems β), γ) and δ) denoting them by Ω_β, Ω_γ, and Ω_δ correspondingly.

$$\Omega_\beta = \{TCFT(\overline{n}, \overline{m}),\ SEAT(\overline{q})\}$$

$$\Omega_\gamma = \{TCFT(\overline{n}, \overline{m}), MAXLOAD(\overline{m}, \overline{n}),\ LOC(\overline{\ell}),\ LOC\,|\,CONS(\overline{n}, \overline{m}, \overline{\ell})\}$$

$$\Omega_\delta = \{TCFT(\overline{n}, \overline{m}), BALANCE(\overline{n}, \overline{m}),\ MAXLOAD(\overline{n}, \overline{m}),\ LOC\,|\,CONS(\overline{n}, \overline{m}, \overline{\ell})\}.$$

After each element of Ω we gave the constants in parentheses. The algorithm SA generates from the master algorithms GA and CA

the particular algorithms GA_i and CA_i for all $i \epsilon \{\alpha, \beta, \gamma\}$.

The input for GA_β and CA_β consists of:

1) The timetable generated in α) ${}_\alpha S_\beta$ with elements of the form:

$$e_{{}_\alpha S_\beta} = hk\, d_1 d_2 \ldots d_{nm}\, s,$$

where h is the code for the description of the characteristics of the classroom required for the section (e.g. room is adequate for demonstration of physical experiments, or for teaching mathematics, etc.); k is the section number, d_1, $d_2 \ldots d_{nm}$ is the time-vector, s is the resource index (e.g. number of students in the section); h is the kernel of e_β

2) The set S_β, the set of classrooms, with elements of the form:

$$e_{S_\beta} = h\, d_1 d_2 \ldots d_{nm}\, d_q,$$

where d_q is the number of seats in the classroom; h is the kernel of e_{S_β}.

The output of GA_β and CA_β consists of:

1) The modified timetable ${}_\beta S_\gamma$ with the listing of assigned classrooms,

2) The modified listing of the set of classrooms where the time-vector in each element indicates when the classroom is occupied.

The input for GA_γ and CA_γ consists of:

1) The set of courses that an instructor is willing to teach, with elements of the form:

$$e = k f_1 f_2 \ldots f_y$$

where k is the course-number, $f_1, \ldots, f_y$ are the personal data of the instructor including such as: code of office, teaching load, etc., k is the kernel of the element;

2) The modified timetable ${}_\beta S_\gamma$ in the elements of the form:

$${}_\beta^e S_\gamma = h\, k\, d_1 d_2 \ldots d_{nm}\, s\, d_\ell\, d_{\ell+1}$$

where the ordered pair $(d_\ell, d_{\ell+1})$ is the classroom code, k is the kernel of the element.

The output of GA_γ and CA_γ consists of:

1) The modified timetable ${}_\gamma S_\delta$ with the names of the assigned instructors;

2) The teaching schedule for the instructors, $\{ts\}$.

The input of GA_δ and CA_δ consists of:

1) The set of courses that a student requests with elements of the form:

$$e_\delta = k f_1 f_2 \ldots f_t$$

where k is the requested course number, $f_1, f_2 \ldots f_t$ are the identifying data of the student, k is the kernel of e_δ;

2) The modified timetable ${}_{\gamma}S_{\delta}$ with elements of the form:

$$ {}^{e}_{\gamma}S_{\delta} = h\,k\,d_1\,d_2 \ldots d_{nm}\,s\,d_{\ell}\,d_{\ell+1}\,d_i $$

where d_i is the code of the instructor's name; k is the kernel of ${}^{e}S_{\delta}$.

The output of GA_{δ} and CA_{δ} consists of:

1) The modified timetable ${}_{\delta}S$, where the resource index s is decreased by one in each section to which the student has been assigned;

2) The set of individual schedules for the students, {ss}.

We illustrate the above described system of algorithms in the following flow-diagrams:

Compiling by SA:

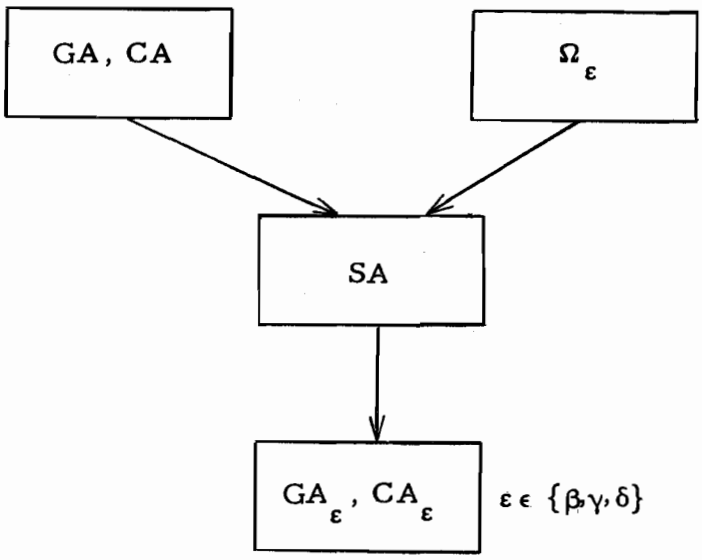


Figure 1

Compiled programming system for Scheduling:

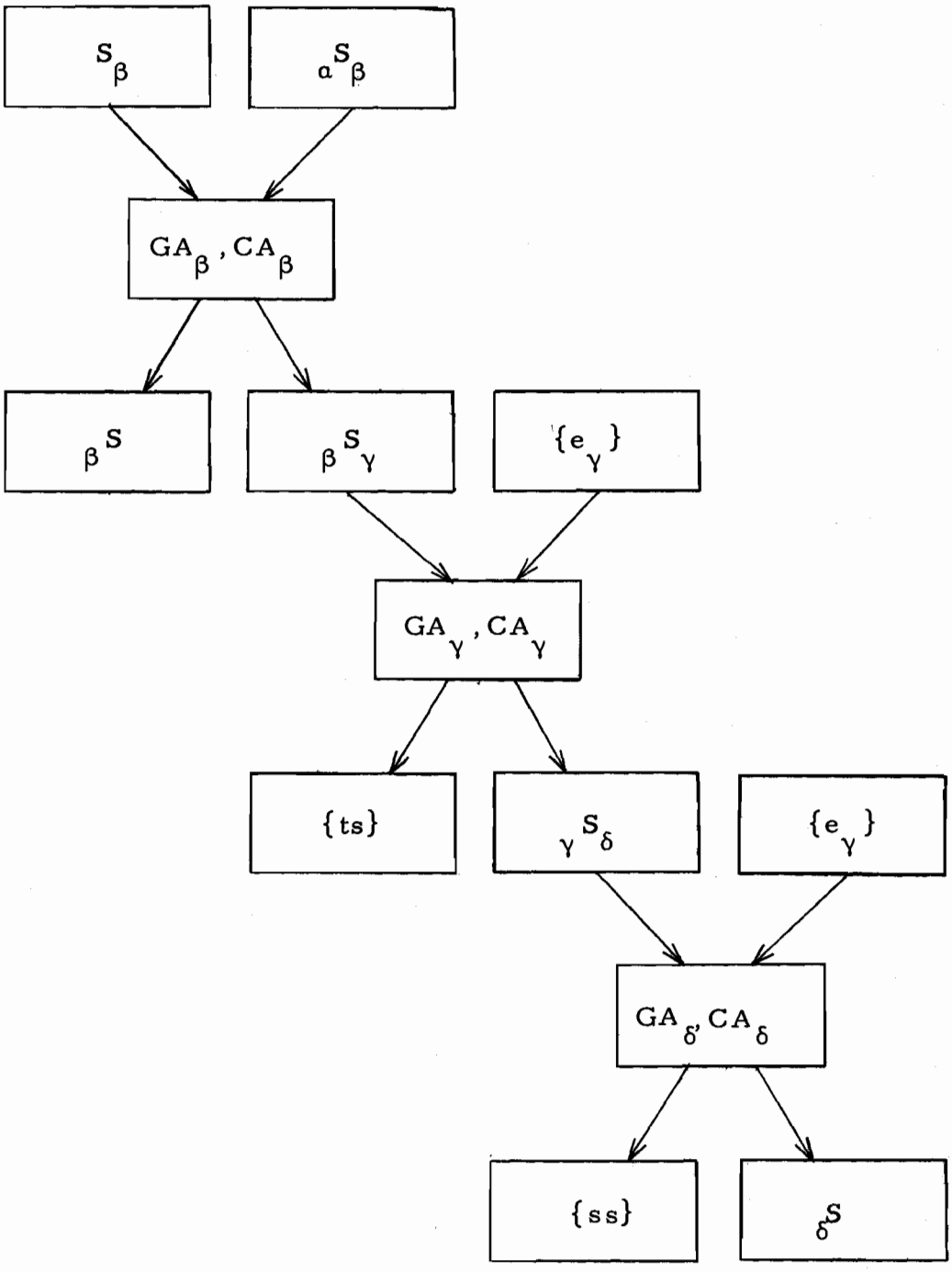


Figure 2

2.2. n-ary Tree Search.

In this section we discuss another practical application of the Scheduling Algorithm: the automation of the Search problem.

First we introduce a terminology. Most of our terms are standard in graph theory [14, p. 1-21, 58-77], but we use some of the definitions given by Sussenguth [15], namely the definition of tree allocation and value of a node. The name of a node is introduced here for the first time to our knowledge.

A graph comprises a set of nodes and a set of unilateral associations specified between pairs of nodes. If node i is associated with node j, the association is called a branch from initial node i to terminal node j. A path is a sequence of branches such that the terminal node of each branch coincides with the initial node of the succeeding branch. A circuit is a path in which the initial node coincides with the terminal node. A tree is a graph which contains no circuit and has at most one branch entering each node. A root of a tree is a node which has no branches entering it, and a leaf is a node which has no branches leaving it. A root is said to lie on the first level of the tree and a node which lies at the end of a path of length j - 1 from a root is on the j-th level. An n-ary tree is a a tree which has at least one leaf on the n-th level and has at least one node on the (n - 1)-st level with at least two branches leaving it. The set of nodes which lie at the end of a path of length one from

node x comprises the filial set of node x and x is the parent node of that set. An item is the basic unit which is processed in some data processing system. The item has two parts: the key is that part which distinguishes the item from all other items; the base is that part which is not the key. A set of items is a file. In the tree allocation the items of the file and the leaves of the tree are placed in a one-one correspondence. After breaking the keys of the file items into several disjoint subsets each of which is made to correspond to a tree node: the 1st subset corresponds to a node on the 1st level, etc. The subset corresponding to the node is called the value of the node. The set of experiments whose results are the value of the node is called the name of the node. If a key is composed of h subsets, the path to its corresponding leaf has length h - 1. Thus, if all of the keys of the file have the same number of disjoint subsets, h, the tree will have leaves only at the h-th level, otherwise there will be leaves on several different levels. It is meaningful to associate a partial key with each node. It is composed of the values of the nodes in the path leading from a root to a given node taken in the order of the levels.

The classification and search problem consists of two major steps:

1) Subdivision of a file of items into different classes using some given principles; this is the classification part;

2) Identification of a given item, as a specified element of one of the above defined classes; this is the search part.

We assume that the 1st step above has been done, in the form of a tree allocation. In particular we will consider trees constructed for a taxonomical problem. In this problem we know only the base of the query item and in order to determine its key elements we have to perform some sets of experiments on it. The sets of results of these experiments are the key elements of the query item. The name of any parental node describes the set of experiments to be performed and a subset of the results is the set of values of the filial nodes. This symbolism represents the dependence of a set of experiments on the results of a preceding one. Iterating (n - 1) times the algorithm GA_T we can identify the query item as a leaf of a given n-ary tree, if it is one of the leaves, otherwise iterating at most (n - 1) times the algorithm GA_T will show that the query item is none of the leaves.

We can generate GA_T from the master algorithm GA by algorithm SA for $\Omega = \Omega_\varepsilon$ (we do not need algorithm CA), where Ω_ε is the set of restrictions for the taxonomic n-ary Tree Search:

$$\Omega_\varepsilon = \{\text{IDENT}(\bar{p}, \bar{q})\} \text{ where } \text{IDENT} = \bigcap_{i=1}^{g} \tau\{R_{\bar{p}}(i, \bar{q})\} .$$

The input for GA_T consists of:

1) The set S with elements of the form:

$$e_S = E(i_1,i_2,\ldots,i_j)\, W(i_1,i_2,\ldots,i_j,i_{j+1})\, E(i_1,i_2,\ldots,i_j,i_{j+1})$$

for all $i_{j+1} = (1,2,3,\ldots,n(i_1,i_2\ldots,i_j))$ and for all $j = 1,2,\ldots,n-1$, where $E(i_1,i_2,\ldots,i_j)$ is the name of a parent node, $W(i_1,i_2,\ldots,i_j,i_{j+1})$ is the value of one of its filial nodes, $E(i_1,i_2,\ldots,i_j,i_{j+1})$ is the name of the filial node, $n(i_1,i_2,\ldots,i_j)$ is the number of the filial nodes of node $E(i_1,i_2,\ldots,i_j)$; $E(i_1,i_2,\ldots,i_j)\, W(i_1,i_2,\ldots,i_j,i_{j+1})$ is the kernel of e.

2) A set of experiments with the result-set which is a partial key of the query item: $e = E(i_1,i_2,\ldots,i_j)\, W(i_1,i_2,\ldots,i_j,i_{j+1})$ where every i_k is a fixed element of the set $\{1,2,\ldots,n(i_1,i_2,\ldots,i_j)\}$ for a fixed j and for all $k = 1,2,\ldots,j+1$; $E(i_1,i_2,\ldots,i_j)\, W(i_1,i_2,\ldots,i_j,i_{j+1})$ is the kernel of the element.

The output of GA_T is a set with elements of the form:

$$\sigma = E(i_1,i_2,\ldots,i_j)\, W(i_1,i_2,\ldots,i_j,i_{j+1})\, E(i_1,i_2,\ldots,i_j,i_{j+1})$$

where $E(i_1,i_2,\ldots,i_j,i_{j+1})$ is the name of the (i_{j+1}) the filial node of node $E(i_1,i_2,\ldots,i_j)$ i.e. the next set of experiments to be performed on the query item. In particular: if $j+1 = n$, then $E(i_1,i_2,\ldots,i_n)$ is the name of a leaf which identifies the query item.

2.3. System Analysis for Timetable Construction and Student Sectioning.

In this section we will give the detailed system analyses for the problems described in the preceding chapter.

Classroom Assigning Program: CAP.

The purpose of the program CAP is to assign classrooms to each section in a previously established timetable. CAP accepts a list of sections and a list of classrooms on tapes, takes into account time conflicts and seat limitations, and produces a list of sections with the assigned classrooms ($_{\beta}S_{\gamma}$) on tape and a printed list of classrooms ($_{\beta}S$).

Input and Output of Information.

The course and section timetable file will be prepared on punched cards, one card for each section, the set of section cards headed by a course card to which the sections belong. Then they are to be numbered (a code number will be assigned to each section for machine use only) and transferred to magnetic tape in K blocks. (Each block must be able to be stored in the main core memory of the computer.)

The card output (course cards with the assigned code number) will serve as master cards for the preparation of teacher request cards used in program TSP (Teacher Sectioning Program) and of

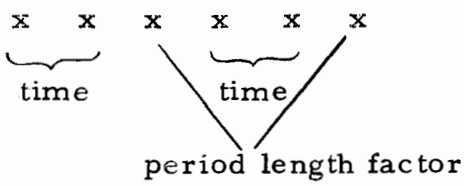
student request cards used in program SSP (Student Sectioning Program).

The list of classrooms, a set of prepunched cards, will be transferred to tape during Phase 1 of the operation.

There are two principal files of information: the section file and the classroom file. An item in the section file is described by the Section Identification Number, Maximal Number of Students, Meeting Times, and by the Balance Indicator. The Section Identification Number is assigned to each section by a subroutine (SEINB). This is a six-digit number composed of a 4-digit course Identification Number and a 2-digit Section Number (00-99). The modification of the section file, using this numbering, is fairly easy. SEINB assigns the Section Identification Numbers in an ascending order.

The Maximal Number of Students (MNS) is a 3-digit number assigned to each section previously.

The Meeting Times (denoted by TIME) are expressed by five six-column fields, each of the five fields corresponds to one day. The format of each field is given in the following scheme:



The P.M. hours are expressed as 2-digit numbers. The period

length factor is a digit, greater than or equal to 2, expresses the length of the period as a multiple of 30 minutes. The last three columns defines the meeting time, if the class meets twice on the same day, otherwise it remains blank.

The Balance Indicator (BI) is a 1-digit code, 0 or 1 indicating whether this course will be included in checking BALANCE (defined in 2.1).

An item in the classroom file is defined by the Maximal Number of Seats, by the Time Vector, by the Building and Room Code and by the Classroom Characteristics. The Classroom Characteristics are described by a 6-digit number-code describing the classroom; this plays the role of the table argument in the table look-up procedure.

The Time Vector is expressed by five ten-column fields, where the 1st column of the 1st field denotes the 1st time period (8-9) on Monday, a 1 (0) in this column indicates that the room is (not) occupied at this time, etc..

The Building and Room Code is a 5-digit number composed of a 2-digit Building Code and a 3-digit Room number. The Building Code (B) defines the position of the building relative to a coordinate system.

The following Figure (Figure 3) is a diagram of general procedure from Classroom File and Section File of punched cards to a

modified Section File on tape and a printed modified Classroom File.

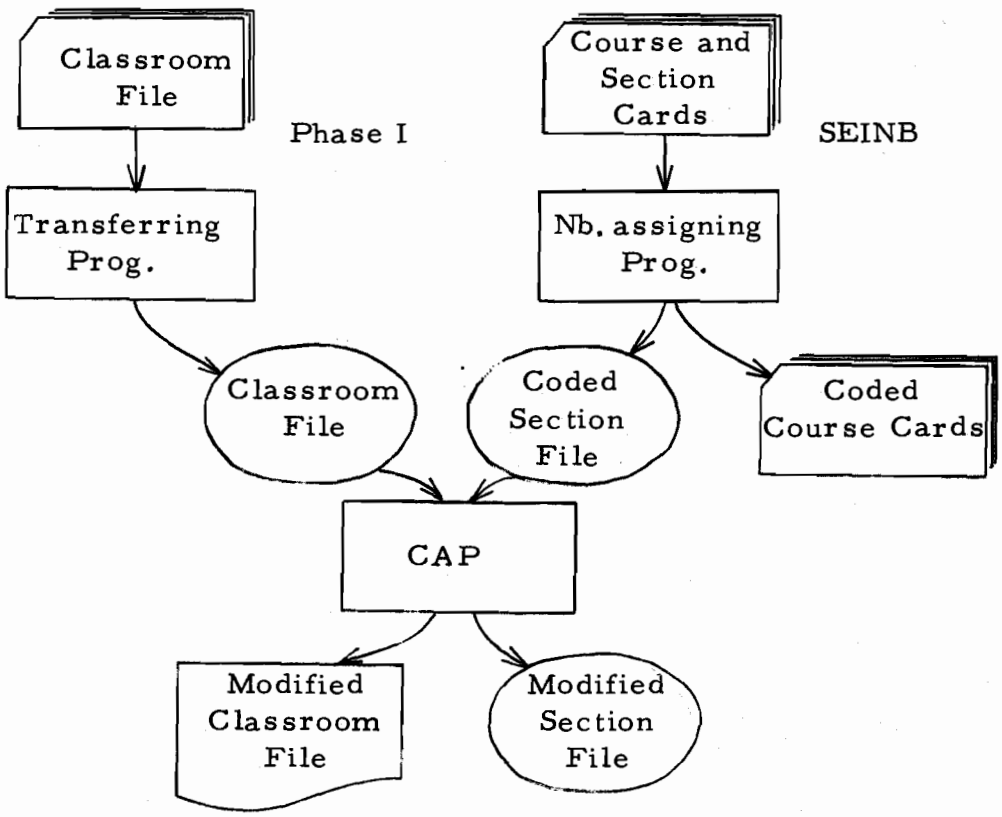


Figure 3.

Teacher Scheduling Program: TSP.

This program will assign an instructor to each section in the modified Section File which is an output of program CAP. TSP accepts instructor request as a set of prepunched cards, one card for each request headed by a personal identification card. TSP schedules the instructor in at most K runs (KC being the size of the modified Section File, and C the size of the part of the core

memory to be used to store information concerning the section file.)

TSP takes into account time conflicts, request for free time for the instructor, keeps the total number of teaching periods under a given limit and sets a given bound for the walking time between consecutive class periods and a given bound for the distance between the instructor's office and the classroom. TSP will produce finished instructor schedules and a completed Section File with the names of the assigned instructors.

Input and Output of Information.

The modified Section File is the tape output of program CAP. The teacher request, a set of prepunched course cards will be transferred to tape during Phase I.

An item in the modified Section File is described by the Section Identification Number, Maximal Number of Students, Meeting Times, Balance Indicator and by the Building and Room Code.

An item in the Instructor Request File is described by the Instructor's Name, Term, Teaching Load, Office Building and Room Code on the Identification Card and by a list of requests, each of them consisting of the following data: Subject Name, Catalog Number and Course Identification Number on a set of cards.

The Instructor's Name is described by a code up to 24 characters, the Term by a 1-digit code, the Teaching Load (TL) by a 2-digit

number and the Office Building and Room Code by a 5-digit number; the Office Building Code is denoted by OB. The Subject Name is given by a code up to 23 characters, the Catalog Number by a 3-digit number, and the Course Identification Number by a 6-digit number.

The following Figure 4 is a flow diagram for the program TSP:

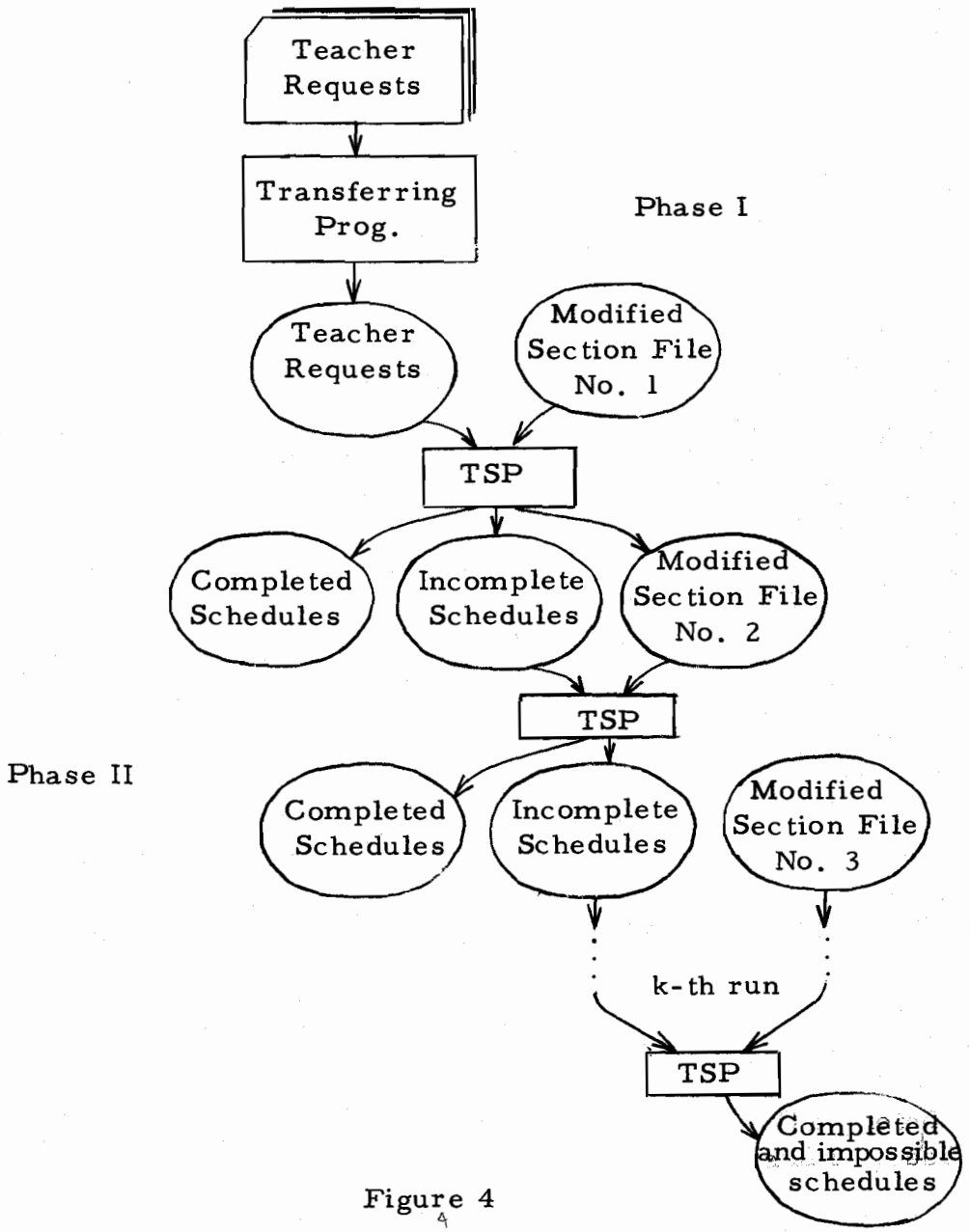


Figure 4

Student Scheduling Program: SSP.

The purpose of the program SSP is to schedule students to the completed Section File. There are no limitations on the number of students or on the size of the Section File. SSP accepts student requests as a set of prepunched cards, one card for each request, headed with a personal identification card, and schedules the student in at most K runs. SSP takes into account time conflicts, request for free time for the student, tries to balance student load and sets a bound for the walking time between consecutive class periods.

SSP will produce finished student schedules, class cards and class lists. Periodic console operations allow us to change the Section File (e.g. we may include sections with no seats in the Section File, which could be opened up, if necessary). If for some reason a student cannot be scheduled, then

a) His requests will be rejected, if it is due to some "request error" discovered during Phase 1;

b) An incomplete schedule will be worked out for him otherwise.

By ordering the request cards the student may indicate the priority of his requests. (The 1st cards have the best chance to avoid rejection because of time conflict.)

Input and Output of Information.

The completed Section File is the tape output of program TSP. The student request, a set of prepunched course cards, will be checked and transferred to tape during Phase I of the operation.

After the scheduling is completed, the file on the tape will be sorted, so that the student schedules will be grouped by schools and the printer will print these files as the student schedules. Another sorting of the "schedule" tape will provide the class lists printed out and the punched class cards.

There are two principal files of information: the completed Section File and the Student Request File. The Completed Section File is the same as in program TSP. An item in the Student Request File can be described by the data on the Identification Card, namely by the S-Code, Student Name, Term, Identification Number, Classification, Status, School and Major, and by the C-Code, and by the data on the set of request cards, namely by the Subject Name, Catalog Number, Course Identification Number and by the Credit Code.

The S-code is described by a 1-digit code, the Student Name by a code up to 24 characters, the Term by a 1-digit code, the Identification Number by a 6-digit number, the Classification by a code U or G, the Status by a code, GA or RA or Staff or SP, the School and Major by a 3-digit code and the C-code by a 1-digit code.

The Credit Code is described by a 2-digit number.

Figure 5 on page 43 is a diagram of general procedure from Student Request File of punched cards and completed Section File on tapes to printed schedules and punched class cards.

Data Preparation.

Phase I transfers the input data from cards to tape and assigns working areas to each data unit and to each block for machine use and for checking purposes. Phase I prepares the following two types of files:

1) The RFile, consists of N blocks of length B. Each block consists of r(variable) R-place requests followed by a one-place code T, then a 50-place area TRY, a 100-place area DISTANCE and a 10-place area TOTAL. In each request there is a 6-digit number (called NUMBER) with zeros in its last two digits and a one-place Request Indicator, RI, followed by the rest of the descriptors of the particular Request File. We denote data in the RFile by prefixing an R to the name of the datum followed by a number in parentheses giving the ordinal number of the request to which the datum belongs. E.g. RNUMBER (X20) means NUMBER in the X20-th request of the RFile.

2) The SFile, consists of the data of the table input; we denote datum in the SFile by prefixing an S letter to the name of the datum.

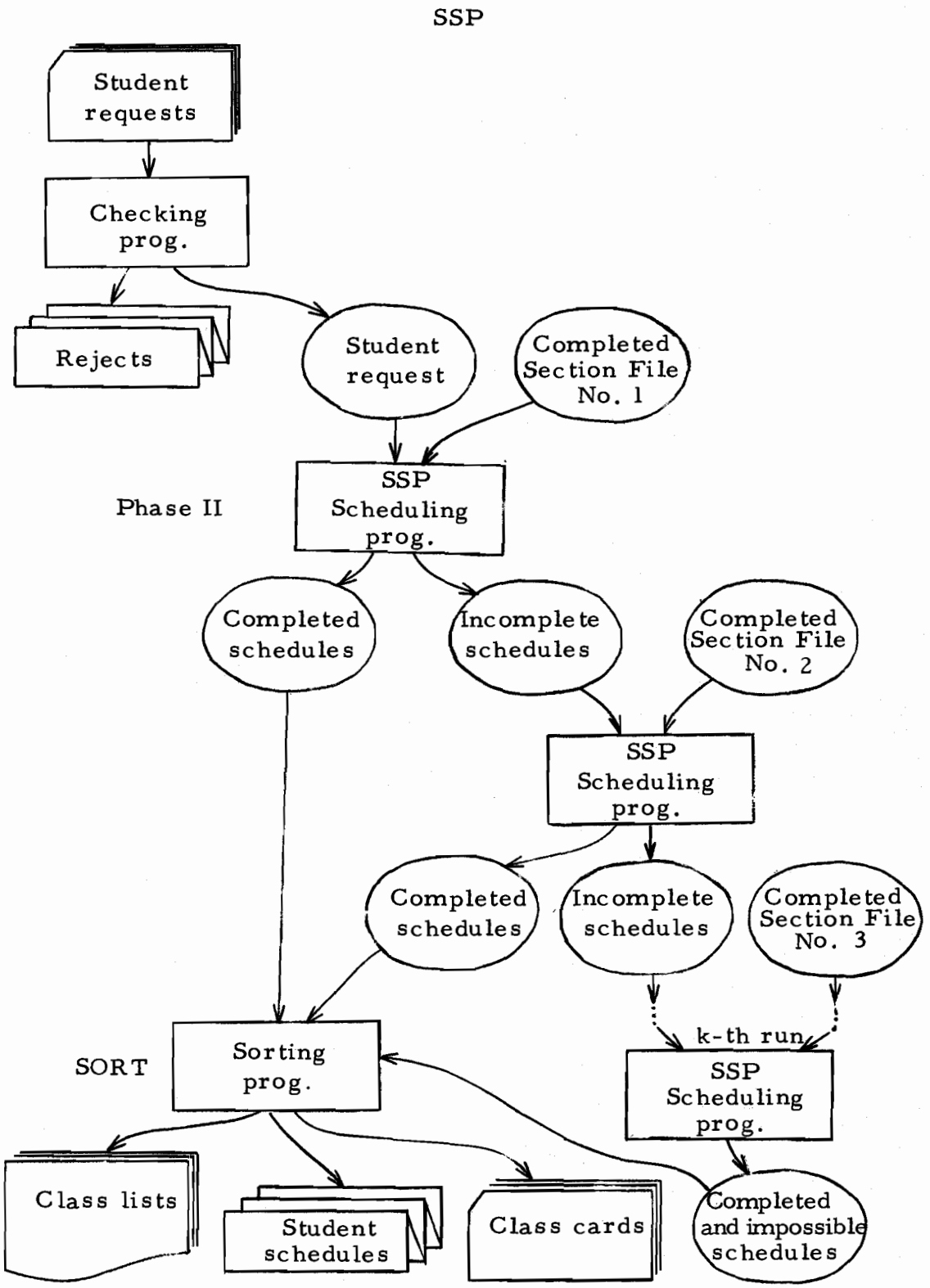

SSP
Student requests
Checking prog.
Rejects
Student request
Completed Section File No. 1
Phase II
SSP Scheduling prog.
Completed schedules
Incomplete schedules
Completed Section File No. 2
SSP Scheduling prog.
Completed schedules
Incomplete schedules
Completed Section File No. 3
k-th run
SORT
Sorting prog.
SSP Scheduling prog.
Class lists
Student schedules
Class cards
Completed and impossible schedules

Figure 5

2.4. System Analysis for the n-ary Tree Search in Taxonomy.

Tree Search Program: TSP.

Let the file S be an n-ary tree. The iteration of TSP will identify a given item e as a specified element of S if it is in S, otherwise it states that e is not in S.

The program TSP accepts a list of the Tree File elements on tape and a list of the Query File elements (i.e. partial keys with the name of the node or set of experiments to which they belong) on card; identifies the name of the node in the tree allocation whose value is the given list of partial keys (if any) and prints out the name of the identified node or indicates that the item is not on the list. If the tree S consists of n levels, then we have to repeat TSP (n - 1) times, in order to obtain a definite answer in the sense described above.

Input and Output of Information.

The Tree File will be prepared on punched cards, k cards for the name of the parent node, m cards for the value of a filial node and n cards for the name of the filial node. Then a 6-digit code will be assigned to each element for machine use only, and the data will be transferred to tape.

The Query File will be prepared on punched cards, k cards for the name of the parent node (code of the performed experiments) and

m cards for the value of one of its filial nodes to be identified (the results of the performed experiments).

The Output File is a set of u cards describing the name of the identified filial node (the set of experiments to be performed next) if any, otherwise a printed statement, that the query item is not on the list.

An item in the Tree File is described by the Item Identification Number, Name of a Parent Node, Value of a Filial Node and by the Name of the Filial Node.

Item Identification Number is a 6-digit number assigned to each item in the list in an increasing order, such that the first four digits are the same for items belonging to the same parental node and the last two digits are code-numbers from 01 to 99.

The name of a Parent Node is expressed by p q-place codes with $p \cdot q = 80k$, where each code describes an experiment performed.

The value of a Filial Node (VF) is expressed by p t-place codes with $p \cdot t = 80m$, where each code describes the result of each experiment.

The name of a Filial Node is expressed by n q-place codes with $u \cdot q = 80n$, where each code describes an experiment to be performed.

We can describe the elements of the Query File by a descriptor

consisting of: Item Identification Number, Name of Parent Node, Value of Filial Node. The elements of the Output File are described by a descriptor consisting of: Item Identification Number and Name of Filial Node, as has been done above.

The following flow diagram (Figure 6) illustrates the iterated Tree Search Procedure. In the first run TSP puts out the name of

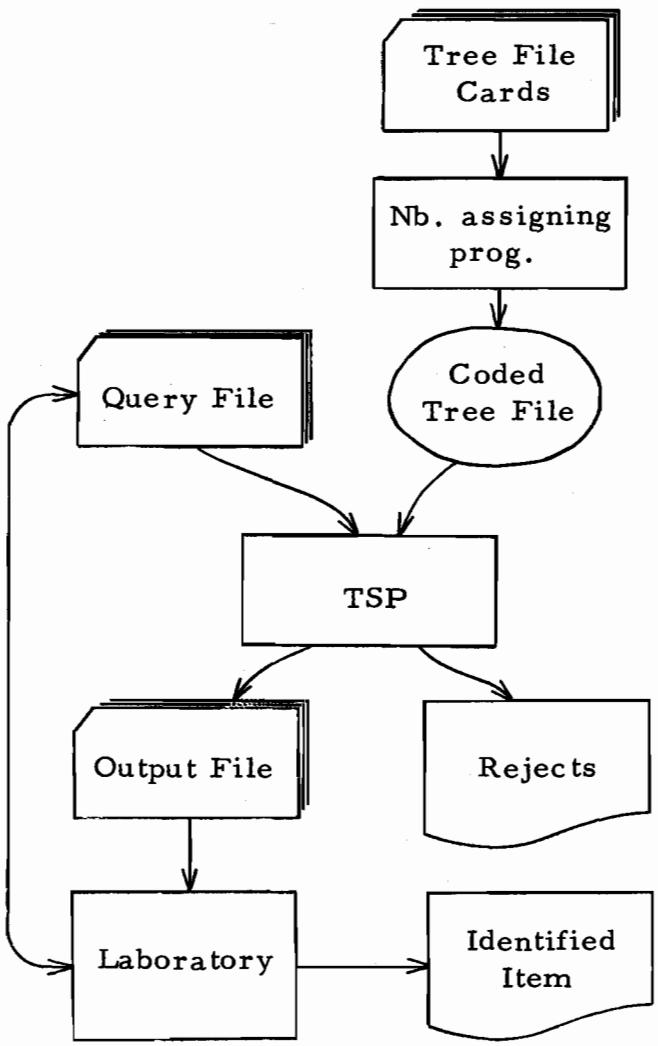


Figure 6

the node on the 2nd level whose value agrees with the value on the query item, and sends the output (which is a coded description of a set of experiments to be performed) to the laboratory or puts out the query item on the Reject File. The results of the required experiments will be the input for TSP in the 2nd run, etc. This process is terminated after at most (n-1) runs.

2.5 Master Algorithm GA for Timetable Construction, Sectioning and n-ary Tree Search in Terms of Elementary Operators.

We will work out the master algorithm GA giving every practical detail in Lyapunov's notation. This will describe the logic of the programming system and can be programmed for any digital computer with tape units when translated into the machine language.

Now we give a specified description of the set of restrictions Ω (described in 2.1) for this particular case.

For TCFT we define the g-place column-relation as follows:

$$C_g(i,1) \iff \sum_{j=1}^{g} i_j \leq 1,$$

where i_j is the j-th element of the i-th column-vector of the matrix defined by the requested row vectors.

For LOC the binary row-relation is defined by:

$$R_2(i,\ell) \iff (\ell_i - c_1)^2 + ((\ell+1)_i - c_2)^2 < a,$$

where ℓ_i is the ℓ-th element of the i-th row-vector, a is a constant, c_1, c_2 are given as ROB in the request.

In LOC|CONS the mixed relation is defined as follows:

$$M(j,k;\ell) \iff (\ell_j - \ell_k)^2 + ((\ell+1)_j - (\ell+1)_k)^2 < w,$$

where ℓ_j is the ℓ-th element of the j-th row-vector and w is a given constant.

The m-place row-relation in BALANCE is defined by:

$$R_m(g+1,\ jm+1) \iff \sum_{i=1}^{m} \ell_{jm+i} < v,$$

where ℓ_j is the j-th element of the (g+1)-st row-vector and v is a given constant.

In MAXLOAD the m·n-place row-relation is defined as:

$$R_{m\,n}(g+1,1) \iff \sum_{j=1}^{m\cdot n} \ell_j < q,$$

where ℓ_j is the j-th element of the (g+1)-st row-vector and q is given as RTL in the request.

For SEAT the g-place column-relation is defined by:

$$C_g(q,1) \iff \ell_i > p_i, \text{ for all } i = 1,..,g,$$

where ℓ_i is the i-th element in the q-th column-vector and p_i is given as RMNOS in the request.

The p-place row-relation is defined for IDENT as follows:

$$R_p(i,q) \iff (\ell_q, \ell_{q+1}, \ldots, \ell_p) = u,$$

where ℓ_q is the q-th element of the i-th row-vector and u is given as RVF in the request.

In the following we assume that K = 1; otherwise we will repeat the procedure K-times.

Now we describe the operators of GA as sequences of elementary operators, i.e. machine instructions. We have denoted the operators of GA by F_i and P_j. We will denote their elementary operators by F_i^k and P_j^ℓ, the predicate elementary operators by ${}_\lambda F_i^k$ and ${}_\lambda P_j^\ell$.

$\underline{F_0}$ in Terms of Elementary Operators.

Let the operator F_0^1 define a 4-valued SWITCH for values 1, 2, 3 and 4. The operator F_0^2 defines a C character area STABLE and F_0^3 defines a B character area READIN, F_0^4 defines an m·n place area ADDEND, F_0^5 defines a 2·m·n place area DISTTEST, F_0^6 defines an R place area PROBE, F_0^7 defines a 2n place area PROBTOTAL and F_0^8 defines 2-place areas called X1, X2, X6, X7, X8, X9, X10, X11, X12, X20 and sets them equal to zero. The logical operator ${}_\lambda F_0^9$ checks the logical condition:

SWITCH = 1;

if the condition is not satisfied, then ${}_\lambda F_0^{14}$ checks the condition:

SWITCH = 2;

if the condition is not fulfilled λF_0^{19} tests the condition:

SWITCH = 3;

if the condition is not satisfied, F_0^{24} tests the condition:

SWITCH = 4;

if the condition is not satisfied, then F_0^{25} indicates an error halt. The operator F_0^{25} inputs Tree File, F_0^{26} inputs Query File, F_0^{27} sets STABLE = Tree File and F_0^{28} sets READIN = Query File. The operator F_0^{10} inputs the Classroom File and F_0^{11} inputs the Section File. Operator F_0^{12} sets STABLE = Classroom File and F_0^{13} sets READIN = Section File. The operator F_0^{15} inputs Section File and F_0^{16} inputs Teacher Request File. F_0^{17} sets STABLE = Section File and F_0^{18} sets READIN = Teacher Request File. The operator F_0^{20} inputs Section File and F_0^{21} inputs Student Request File. F_0^{22} sets STABLE = Section File and F_0^{23} sets READIN = Teacher Request File.

$$F_0 = F_0^1 F_0^2 F_0^3 F_0^4 F_0^5 F_0^6 F_0^7 F_0^8 \lambda F_0^9 \underset{14}{\lfloor\!\!\lfloor\!\!\lfloor\!\!\lfloor} F_0^{10} F_0^{11} F_0^{12}$$

$$F_0^{13} \overset{F_1^1}{\lceil\!\!\lceil\!\!\lceil\!\!\lceil} ; \underset{9}{\rfloor\!\!\rfloor\!\!\rfloor\!\!\rfloor} \lambda F_0^{14} \underset{19}{\lfloor\!\!\lfloor\!\!\lfloor\!\!\lfloor} F_0^{15} F_0^{16} F_0^{17} F_0^{18} \overset{F_1^1}{\lceil\!\!\lceil\!\!\lceil\!\!\lceil} ;$$

$$\underset{14}{\rfloor\!\!\rfloor\!\!\rfloor\!\!\rfloor} \lambda F_0^{19} \underset{24}{\lfloor\!\!\lfloor\!\!\lfloor\!\!\lfloor} I_0^{20} F_0^{21} F_0^{22} F_0^{23} \overset{F_1^1}{\lceil\!\!\lceil\!\!\lceil\!\!\lceil} ;$$

$$\underset{19}{\rfloor\!\!\rfloor\!\!\rfloor\!\!\rfloor} \lambda F_0^{24} \underset{29}{\lfloor\!\!\lfloor\!\!\lfloor\!\!\lfloor} F_0^{25} F_0^{26} F_0^{27} F_0^{28} \overset{F_1^1}{\lceil\!\!\lceil\!\!\lceil\!\!\lceil} ; \underset{24}{\rfloor\!\!\rfloor\!\!\rfloor\!\!\rfloor} F_0^{29}:$$

Remarks. The above used files are defined in sections (2.3; 2.4). We assumed that the periods start at i o'clock for i = 8, 9, ... 17, ... m. With a small amount of work GA can be adjusted to any other arrangement.

F_1 and F_2 in the Terms of Elementary Operators.

Operator F_1^1 reads in the file READIN, F_2^1 puts in STABLE. (STABLE is ordered according to the increasing SNUMBER's.)

$$F_1 F_2 = F_1^1 F_2^2.$$

Operator F_3 in Terms of Elementary Operators.

The logical operator ${}_\lambda F_3^1$ checks the condition:

$$\text{RNUMBER (X20)} \neq \text{T};$$

if the condition is not satisfied then go to F_{14}^1. The operator F_3^2 scans STABLE to find agreement with RNUMBER (X20) i.e. it looks up equal, high or endmark in STABLE. The operator F_3^3 transfers the row-vector of STABLE starting with the SNUMBER found by F_3^2, into PROBE. ${}_\lambda F_3^4$ checks the logical condition:

$$\text{PNUMBER} \neq \text{endmark};$$

if the condition is not satisfied then go to ${}_\lambda F_3^6$. ${}_\lambda F_3^5$ tests the logical condition:

$$a_k = b_k , \qquad k = 3, 4, 5, 6,$$

for PNUMBER $= \sum_{k=1}^{6} a_k 10^k$ and RNUMBER (X20) $= \sum_{k=1}^{6} b_k 10^k$;

if the condition is not satisfied then go to ${}_\lambda F_3^6$. The logical operator ${}_\lambda F_3^6$ checks the condition:

$$RRI\,(X20) = f;$$

if the condition is not satisfied then ${}_\lambda F_3^7$ checks the condition:

$$RRI\,(X20) = 0$$

if the condition is not fulfilled then go to F_{12}. F_3^8 indicates that RNUMBER (X20) is not in STABLE and stops the operation (error halt).

$$F_3 = {}_\lambda F_3^1 \;\underset{F_{14}^1}{\rfloor}\; F_3^2\, F_3^3\, {}_\lambda F_3^4 \;\underset{6}{\lfloor}\; {}_\lambda F_3^5 \;\overset{P_4^1}{\underset{6}{\ulcorner}}\;;\; \underset{1,4,5}{\rfloor}\; {}_\lambda F_3^6 \;\overset{F_{13}^1}{\underset{7}{\ulcorner}}\;;$$

$${}_\lambda F_3^7 \;\underset{F_{12}^1}{\lfloor}\; F_3^8 .$$

<u>P_4 in the Terms of Elementary Operators.</u>

${}_\lambda P_4^1$ tests the condition:

$$SWITCH = 1;$$

if the condition is not fulfilled, then ${}_\lambda P_4^2$ tests the condition:

$$SWITCH = 2;$$

if the condition is not satisfied, then ${}_\lambda P_4^3$ checks the condition:

$$SWITCH = 3;$$

if the condition is not satisfied go to P_5^{45}. $_{\lambda}P_4^3$ tests the condition:

$$PMNOS > 0;$$

if the condition is not fulfilled, then P_4^5 puts an f into RRI(X20).

$$P_4 = {}_{\lambda}P_4^1 \underset{2}{\overset{P_5^1}{\square}}; \underset{2}{\square} {}_{\lambda}P_4^2 \underset{3}{\overset{P_5^1}{\square}}; \underset{3}{\square} {}_{\lambda}P_4^3 \underset{4}{\overset{P_5^{45}}{\square}}; {}_{\lambda}P_4^4 \underset{5}{\overset{P_5^1}{\square}};$$

$$\underset{4}{\square} P_4^5 \overset{F_9^1}{\square} .$$

P_5 in Terms of Elementary Operators.

$_{\lambda}P_5^1$ tests the logical condition:

$$SWITCH = 1;$$

if the condition is not fulfilled, then $_{\lambda}P_5^3$ checks the condition:

$$SWITCH = 2;$$

if this condition is not satisfied, then $_{\lambda}P_5^5$ tests the condition:

$$PBI = 1;$$

if the condition is not fulfilled, then branch to P_5^{14}. $_{\lambda}P_5^2$ checks the logical condition:

$$RMNOS(X20) \leq PMNS;$$

if the condition is not satisfied, then branch to F_9^2. $_{\lambda}P_5^4$ checks the condition:

$$(x_0 - x_1)^2 + (y_0 - y_1)^2 < a$$

for (x_1, y_1) = PB and (x_0, y_0) = ROB and a is a given constant;

if the condition is not satisfied, then go to F_9^3. The operator P_5^6 adds 3 to X1, and P_5^7 adds 1 to X2. P_5^8 transfers the sum: PTIME (X1) + PTIME (X1 + 3) to PROBTOTAL(X2). $_\lambda P_5^9$ checks the logical condition:

$$\text{PROBTOTAL}(X2) > g,$$

where g is a constant; if the condition is not satisfied, then P_5^{10} adds 1 to X2, and $_\lambda P_5^{11}$ tests the condition:

$$X2 > n;$$

if the condition is not satisfied then P_5^{12} adds 6 to X1 and branches to P_5^8. $_\lambda P_5^{13}$ checks the condition:

$$\sum_{i=1}^{2n} (\text{PTIME}(3i)) + \sum_{i=1}^{n} \text{TOTAL}(i) \leq v,$$

where v is a constant; if the condition is not satisfied then go to F_9^5. P_5^{14} sets X6 equal to 2, P_5^{15} sets X7 equal to 1. P_5^{16} sets X8 equal to 8. P_5^{17} adds X11 to X7. $_\lambda P_5^{18}$ checks the logical condition:

$$\text{PTIME}(X6) = X8;$$

if the condition is not satisfied, then P_5^{22} adds 1 to X7 and P_5^{23} adds 1 to X8. P_5^{24} tests the condition:

$$X8 = 8 + m;$$

if the condition is not fulfilled, then go to $_\lambda P_5^{18}$. P_5^{19} adds 1 to X9. $_\lambda P_5^{20}$ checks the condition:

$$X9 = 9;$$

if the condition is not satisfied, then go to P_5^{22}, if the condition is satisfied, then P_5^{21} indicates an error. $_\lambda P_5^{25}$ checks the condition:

X9 = even;

if the condition is not fulfilled, then go to $_\lambda P_5^{27}$. P_5^{26} adds 1 to X10. $_\lambda P_5^{27}$ checks the condition:

TRY (X7 + X10) = 1;

if the condition is not fulfilled, then P_5^{28} writes a 1 in ADDEND(X7 + X10), if the condition is fulfilled then go to $F_9^6 \cdot {}_\lambda P_5^{29}$ checks the condition:

PTIME (X6 + 1) = X9 + 1;

if the condition is not satisfied, then go to P_5^{19}. P_5^{30} adds 1 to X12. $_\lambda P_5^{31}$ checks the condition:

X12 = 2n;

if the condition is satisfied, then go to P_5^{35}, if the condition is not satisfied, then $_\lambda P_5^{32}$ tests the condition:

X12 = even;

if the condition is not satisfied, then go to P_5^{15}. P_5^{33} adds 10 to X11, and P_5^{34} adds $3 \cdot$X12 to X6 and branches to P_5^{14}. P_5^{35} sets X12 equal to 1. $_\lambda P_5^{36}$ tests the condition:

ADDEND (X12) = 1;

if the condition is not satisfied, then P_5^{38} adds 1 to X12. $_\lambda P_5^{39}$ checks the condition:

$$X12 = m \cdot n;$$

if the condition is not satisfied, then branch to λP_5^{36}; if the condition is satisfied, then go to F_6. P_5^{37} writes PB into DISTEST (X12). λP_5^{40} checks the condition:

$$X12 = 1;$$

if the condition is not satisfied, then λP_5^{42} tests the condition:

$$DISTANCE\ (X12 - 1) = \text{blank};$$

if the condition is satisfied then go to P_5^{38}, if not, then λP_5^{43} checks the condition:

$$([DISTANCE\ (X12 - 1)]_1 - [DISTEST\ (X12)]_1)^2 +$$

$$([DISTANCE\ (X12 - 1)]_2 - [DISTEST\ (X12)]_2)^2 < w$$

where w is a given constant. If the condition is not satisfied then go to F_9^7. λP_5^{41} checks the condition:

$$DISTANCE\ (X12 + 1) = \text{blank};$$

if the condition is not fulfilled, then λP_5^{44} tests the condition:

$$([DISTANCE\ (X12 + 1)]_1 - [DISTEST\ (X12)]_1)^2 +$$

$$([DISTANCE\ (X12 + 1)]_2 - [DISTEST\ (X12)]_2)^2 < w,$$

if the condition is satisfied then go to P_5^{38}; if not, then branch to F_9^7. λP_5^{45} checks the logical condition:

$$PVF = SVF;$$

if the condition is not satisfied, go to F_9^8. Otherwise go to F_{14}^1.

$$P_5 = {}_{\lambda}P_5^1 \underset{3}{\llcorner} {}_{\lambda}P_5^2 \overset{14}{\underset{F_9^2}{\sqcap}}; \underset{1}{\lrcorner} {}_{\lambda}P_5^3 \underset{5}{\llcorner} {}_{\lambda}P_5^4 \overset{14}{\underset{F_9^3}{\sqcap}}; \underset{3}{\lrcorner} {}_{\lambda}P_5^5 \underset{14}{\llcorner} P_5^6$$

$$P_5^7 \overset{12}{\urcorner} P_5^8 {}_{\lambda}P_5^9 \overset{F_9^4}{\underset{10}{\sqcap}}; \underset{9}{\lrcorner} P_5^{10} {}_{\lambda}P_5^{11} \overset{13}{\underset{12}{\sqcap}}; \underset{11}{\lrcorner} P_5^{12} \overset{8}{\sqcap};$$

$$\overset{11}{\urcorner} {}_{\lambda}P_5^{13} \underset{F_9^5}{\llcorner} \overset{2,4,34}{\underset{5}{\sqcap}} P_5^{14} \underset{32}{\lrcorner} P_5^{15} P_5^{16} P_5^{17} \underset{24}{\lrcorner} {}_{\lambda}P_5^{18}$$

$$\underset{22}{\llcorner} \underset{29}{\lrcorner} P_5^{19} {}_{\lambda}P_5^{20} \underset{22}{\llcorner} P_5^{21}; \underset{18,20}{\lrcorner} P_5^{22} P_5^{23} {}_{\lambda}P_5^{24} \underset{18}{\llcorner} P_5^{25}$$

$$\underset{27}{\llcorner} P_5^{26} \underset{25}{\lrcorner} {}_{\lambda}P_5^{27} \overset{F_9^6}{\underset{28}{\sqcap}}; \underset{27}{\lrcorner} P_5^{28} {}_{\lambda}P_5^{29} \underset{19}{\llcorner} P_5^{30} {}_{\lambda}P_5^{31} \overset{35}{\underset{32}{\sqcap}};$$

$$\underset{31}{\lrcorner} {}_{\lambda}P_5^{32} \underset{15}{\llcorner} P_5^{33} P_5^{34} \overset{14}{\sqcap} \overset{31}{\urcorner} P_5^{35} \underset{39}{\lrcorner} {}_{\lambda}P_5^{36} \underset{38}{\llcorner} P_5^{37}$$

$$\underset{36,42,44}{\lrcorner} P_5^{38} {}_{\lambda}P_5^{39} \overset{F_6^1}{\underset{36}{\sqcap}}; {}_{\lambda}P_5^{40} \underset{42}{\llcorner} {}_{\lambda}P_5^{41} \underset{44}{\llcorner} \underset{40}{\lrcorner} {}_{\lambda}P_5^{42} \overset{38}{\underset{43}{\sqcap}};$$

$$\underset{42}{\lrcorner} {}_{\lambda}P_5^{43} \underset{F_9^7}{\llcorner} \underset{41}{\lrcorner} {}_{\lambda}P_5^{44} \overset{38}{\underset{F_9^7}{\sqcap}}; \underset{{}_{\lambda}P_4^3}{\lrcorner} P_5^{45} \overset{F_{14}^1}{\underset{F_9^8}{\sqcap}};$$

F_6, F_7 and P_8 in terms of elementary operators:

F_6^1 indicates, that so far there is no conflict. F_6^2 adds ADDEND to TRT, F_6^3 adds DISTEST to DISTANCE, F_6^4 adds PROBTOTAL to TOTAL, F_6^5 adds PROBE to READIN(X20), F_6^6 sets X1, X2, X6, X7, X8, X9, X10, X11, X12 equal to zero. F_6^7 clears ADDEND, DISTEST, PROBTOTAL and PROBE. F_7^1 adds 1 to X20. $_\lambda P_8^1$ tests the logical condition:

READIN is not the last block in the R File;

if the condition is not fulfilled, then go to I_{14}^3.

$$F_6F_7 = F_6^1\,F_6^2\,F_6^3\,F_6^4\,F_6^5\,F_6^6\,F_6^7\,F_7^1\left\lceil {}_\lambda F_3^1\right.;\quad P_8 = \underset{I_{14}^1,\,I_{14}^2}{\lfloor\!\rfloor}\ {}_\lambda P_8^1\left\lceil \frac{F_1^1}{F_{14}^3}\right.;$$

F_9 in Terms of Elementary Operators.

F_9^1 writes an f into RRI(X20), F_9^2 writes an s into RRI(X20),

F_9^3 writes an ℓ into RRI(X20), F_9^4 writes a b into RRI(X20),

F_9^5 writes an m into RRI(X20), F_9^6 writes a t into RRI(X20),

F_9^7 writes an n into RRI(X20), F_9^8 adds 1 to PNUMBER.

F_9^9 scans the STABLE to find agreement with PNUMBER, i.e. it looks up equal, high or endmark in STABLE. F_9^{10} transfers the row-vector of STABLE starting with SNUMBER found by F_9^9 into PROBE and branches to $_\lambda F_3^3$.

$$F_9 = \overset{P_4^4}{\urcorner} F_9^1 \lceil^8 ; \underset{\lambda P_5^2}{\lfloor} F_9^2 \lceil^8 ; \underset{\lambda P_5^4}{\lfloor} F_9^3 \lceil^8 ; \overset{\lambda P_5^9}{\urcorner} F_9^4 \lceil^8 ;$$

$$\underset{\lambda P_5^{13}}{\lfloor} F_9^5 \lceil^8 ; \overset{\lambda P_5^{27}}{\urcorner} F_9^6 \lceil^8 ; \underset{\lambda P_5^{43}\ \lambda P_5^{44}}{\lfloor} F_9^7 F_9^8$$

$$F_9^9 F_9^{10} \overset{\lambda F_3^3}{\lceil} .$$

P_{10}, P_{11}, F_{12} and F_{13} in Terms of Elementary Operators.

The operators P_{10} and P_{11} are included in F_3. F_{12}^1 indicates that the conflict is not resolvable by GA and F_{12}^2 transfers READIN to REJECT. F_{13}^1 indicates that the request is not realistic.

$$F_{12}; F_{13} = \underset{\lambda F_3^7}{\lfloor} F_{12}^1 \overset{F_{13}^1}{\urcorner} F_{12}^2 \overset{F_{14}^2}{\lceil} ; \underset{\lambda F_3^6}{\lfloor} F_{13}^1 \overset{F_{12}^2}{\lceil} .$$

F_{14} in Terms of Elementary Operators.

F_{14}^1 puts out READIN onto the Completed List. F_{14}^2 puts out READIN onto the Reject List. F_{14}^3 indicates that the operation is completed.

$$F_{14} = \underset{\lambda F_3^1}{\overset{\lambda P_4^3}{\lfloor}} F_{14}^1 \overset{\lambda P_8^1}{\lceil} ; \underset{F_{12}^2}{\lfloor} F_{14}^2 \overset{\lambda P_8^1}{\lceil} ; \underset{\lambda P_8^1}{\lfloor} F_{14}^3 .$$

2.6 CA Algorithm in Terms of Elementary Operators.

The objective of the CA algorithm is to eliminate resolvable conflicts in GA indicated by the operator F_{12}^{1}. At this stage of the program there are two alternatives: to insert CA or to reject the request by operator F_{12}^{2}.

G_1^1 subtracts 1 from X20. G_1^2 puts the X20-th block of READIN into PROBE. $_{\lambda}G_1^3$ tests the condition:

$$PBI = 1;$$

if the condition is not fulfilled, then go to G_1^{11}. G_1^4 sets X1 equal to 3. G_1^5 sets X2 equal to 1. G_1^6 transfers the sum

$$PTIME(X1) + PTIME(X1 + 3) \text{ to } PROBTOTAL.$$

G_1^7 adds 1 to X2. $_{\lambda}G_1^8$ tests the condition:

$$X2 > \bar{n};$$

if the condition is not fulfilled, then G_1^{10} adds 6 to X1 and go to G_1^6. G_1^9 subtracts PROBTOTAL from TOTAL and go to G_1^{11}. G_1^{11} sets X6 equal to 2, G_1^{12} sets X7 equal to 1, G_1^{13} sets X8 equal to 8, and G_1^{14} adds X11 to X7. G_1^{15} checks the condition:

$$PTIME(X6) = X8;$$

if the condition is not satisfied, then G_1^{19} adds 1 to X7 and G_1^{20} adds 1 to X8. $_{\lambda}G_1^{21}$ tests the condition:

$$X8 = 8 + m;$$

if the condition is not satisfied, then go to G_1^{15}. G_1^{16} adds 1 to X9,

and $_{\lambda}G_1^{17}$ checks the condition:

$$X9 = 9;$$

if the condition is not fulfilled, then go to G_1^{19}. G_1^{18} indicates an error halt. $_{\lambda}G_1^{22}$ tests the condition:

X9 is even;

if the condition is not satisfied, then G_1^{24} writes a 1 in ADDEND(X7 X10). G_1^{23} adds 1 to X10, and $_{\lambda}G_1^{25}$ checks the condition:

$$PTIME\,(X6 + 1) = X9 + 1;$$

if the condition is not fulfilled, go to G_1^{16}. G_1^{26} adds 1 to X12, and $_{\lambda}G_1^{27}$ tests the condition:

$$X12 = 2n;$$

if the condition is satisfied, go to G_1^{31}, if not then, $_{\lambda}G_1^{28}$ tests the condition:

X12 is even;

if the condition is not satisfied, go to G_1^{12}. G_1^{29} adds 10 to X11. G_1^{30} adds $3 \cdot X12$ to X6 and goes to G_1^{11}. G_1^{31} sets X12 equal to 1. $_{\lambda}G_1^{32}$ checks the condition:

$$ADDEND\,(X12) = 1;$$

if the condition is not fulfilled, then G_1^{34} adds 1 to X12. $_{\lambda}G_1^{35}$ tests the condition:

$$X12 = m{\cdot}n;$$

if the condition is not fulfilled go to $_{\lambda}G_1^{32}$. G_1^{36} subtracts ADDEND from TRY. G_1^{37} subtracts DISTEST from DISTANCE, and branches

to G_2^1. G_1^{33} writes PB into DISTEST and goes to G_2^{34}. G_2^1 adds 1 to PNUMBER. G_2^2 scans STABLE to find agreement with PNUMBER. G_2^3 transfers the row-vector of STABLE starting with the SNUMBER found by G_2^2 into PROBE and goes to $_\lambda Q_3^1$. $_\lambda Q_3^1$ tests the condition:

$$\text{PNUMBER} \neq \text{endmark};$$

if the condition does not hold go to G_8^1. G_8^1 subtracts 1 from X20. $_\lambda Q_9^1$ tests the condition:

$$X20 \leq 0;$$

if the condition is not satisfied, go to G_1^2. $_\lambda Q_3^2$ tests the condition:

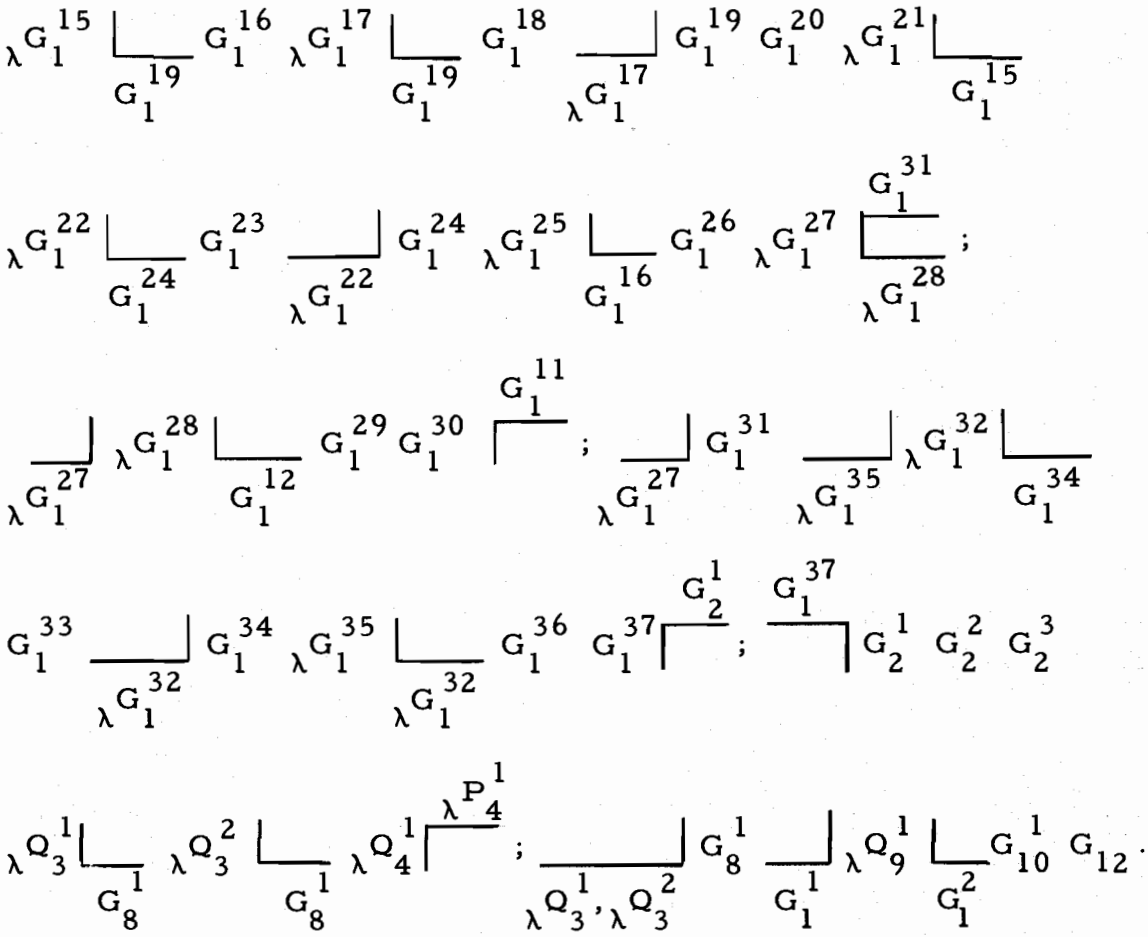
$$a_k = b_k, \qquad k = 3, 4, 5, 6$$

for PNUMBER $= \sum_{k=1}^{6} a_k 10^k$ and RNUMBER (X20) $= \sum_{k=1}^{6} b_k 10^k$;

if the condition is not satisfied then go to G_8^1. Q_4 branches to $_\lambda P_4^1$. The operators G_5, Q_6, G_7 and G_{11} are included in GA. G_{10}^1 indicates that there exists no solution at all.

$$\underset{F_{12}^1}{\lfloor\!\rfloor} G_1^1 \; \underset{\lambda Q_9^1}{\lfloor\!\rfloor} G_1^2 \; {}_\lambda G_1^3 \underset{G_1^{11}}{\lfloor\!\_} G_1^4 \; G_1^5 \; \overset{G_1^{10}}{\lceil\!\rceil} G_1^6 \; G_1^7 \; {}_\lambda G_1^8 \underset{G_1^{10}}{\lfloor\!\_}$$

$$G_1^9 \overset{G_1^{11}}{\lceil} \; ; \; \underset{\lambda G_1^8}{\lfloor\!\rfloor} G_1^{10} \overset{G_1^6}{\lceil} \; ; \; \overset{G_1^9}{\lceil\!\rceil} G_1^{11} \; \underset{\lambda G_1^{28}}{\lfloor\!\rfloor} G_1^{12} \; G_1^{13} \; G_1^{14} \; \underset{\lambda G_1^{21}}{\lfloor\!\rfloor}$$



2.7. SA Algorithm in Terms of Elementary Operators.

In the master algorithm GA we built in a branch point, called SWITCH, with four positions, each of which selects one subset of Ω. By some additional positions and testing we could easily include any other subset of Ω from the remaining $2^7 - 4$. The first four positions of the switch give the four algorithms we are interested in. We want to emphasize, that SA is invariant with respect to the permutations of the input elements.

H_0^1 defines an R-place area RESTRICT, H_0^2 defines an M-place area MASTER, H_0^3 defines an S-place area CONDITION, H_0^4 defines two 1-place area Y1 and Y2, and sets them equal to zero. H_0^5 defines the 1-place areas FLAG i, i = 1,...7, and sets them equal to zero. H_0^6 inputs Master Algorithm GA, H_0^7 inputs Restrict File, and H_0^8 inputs Condition File. H_1^1 reads in CONDITION(Y1), and H_2^1 reads in RESTRICT(Y2). ${}_\lambda X_3^1$ tests the condition:

CONDITION(Y1) = RESTRICT(Y2);

if the condition is not fulfilled, then H_7^1 adds 1 to Y2. ${}_\lambda X_8^1$ tests the condition:

$$Y2 \leq 7;$$

if the condition is not satisfied, H_9 indicates an error halt. H_4^1 sets FLAG(Y2) = 1, and H_5^1 adds 1 to Y1. ${}_\lambda X_6^1$ tests the condition:

$$Y1 \leq 4;$$

if the condition is not fulfilled, ${}_\lambda H_{10}^1$ tests the condition:

FLAG 1 = 1;

if the condition is not satisfied, ${}_\lambda H_{10}^3$ tests the condition:

FLAG 2 = 1;

if the condition is not fulfilled, go to H_9. H_{10}^2 sets SWITCH = 4 and goes to H_4^2. ${}_\lambda H_{10}^4$ tests the condition:

FLAG 4 = 1;

if the condition is not fulfilled, ${}_{\lambda}H_{10}^{8}$ tests the condition:

FLAG 3 = 1;

if the condition is not satisfied go to H_9. ${}_{\lambda}H_{10}^{5}$ tests the condition:

FLAG 6 = 1;

if the condition is not fulfilled, go to H_9. ${}_{\lambda}H_{10}^{6}$ tests the condition:

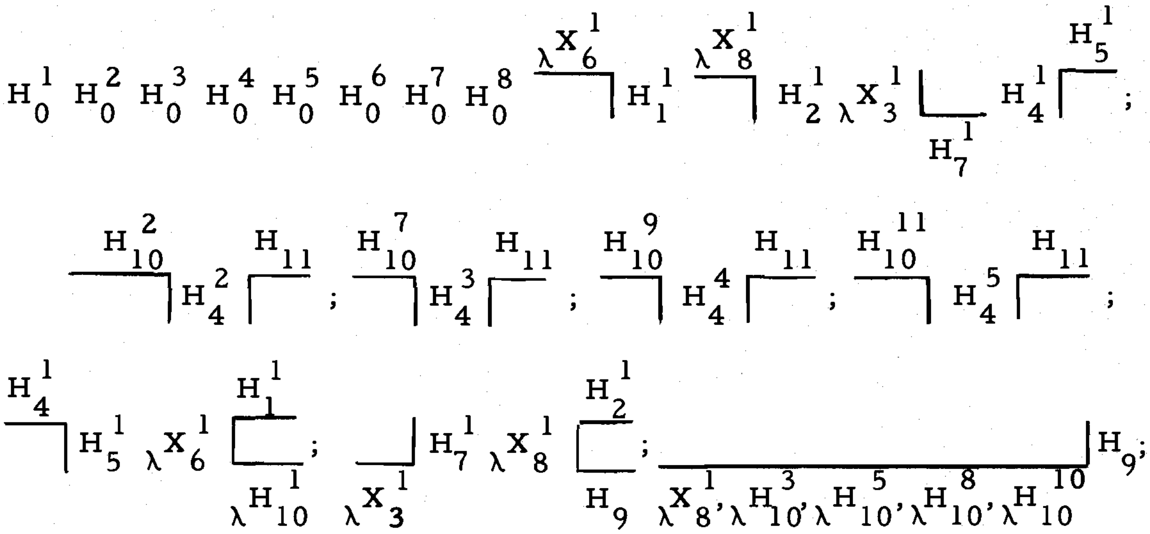
FLAG 7 = 1;

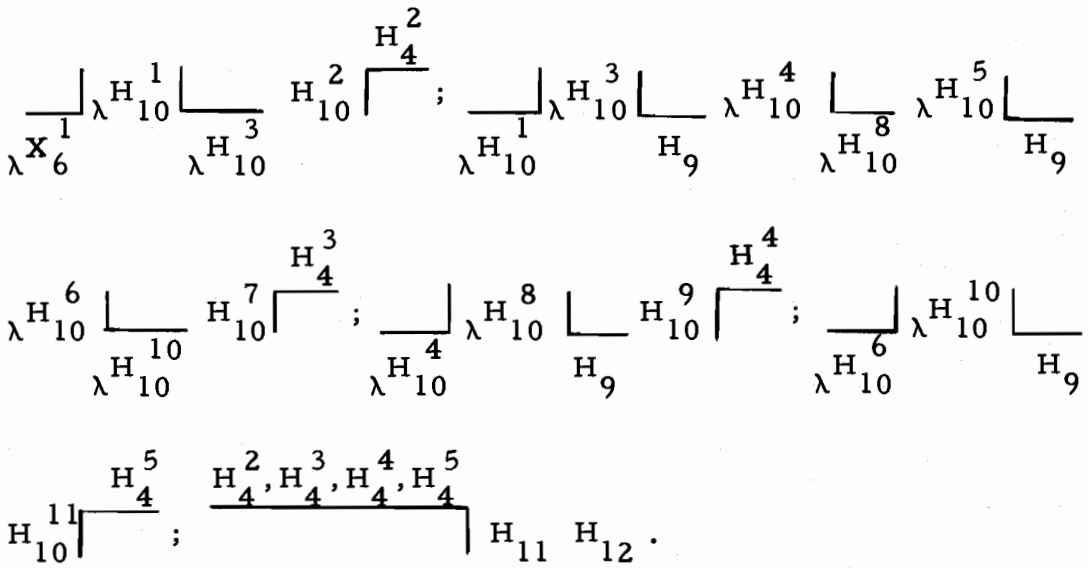
if the condition is not satisfied, ${}_{\lambda}H_{10}^{10}$ checks the condition:

FLAG 5 = 1;

if the condition is not satisfied, go to H_9. H_{10}^{7} sets SWITCH = 3 and goes to H_4^3, H_{10}^{9} sets SWITCH = 1 and goes to H_4^4, and H_{10}^{11} sets SWITCH = 2 and goes to H_4^5. H_4^2 sets $p = \overline{p}$ and $q = \overline{q}$, H_4^3 sets $m = \overline{m}$, $n = \overline{n}$ and $\ell = \overline{\ell}$, H_4^4 sets $m = \overline{m}$, $n = \overline{n}$ and $q = \overline{q}$, and H_4^5 sets $m = \overline{m}$, $n = \overline{n}$ and $\ell = \overline{\ell}$.

H_{11}^{1} outputs the modified GA algorithm.





PART III. SOLUTION OF THE TAXONOMIC PROBLEM USING TURING PROGRAMMING

3.1. Description of the Turing Programming.

We solved the n-ary Tree Search Problem in Taxonomy by the GA algorithm. We have to emphasize that the algorithm is not invariant with respect to the permutation of the elements of the sets of experiments and of their corresponding results, and therefore the GA algorithm does not simulate the methods used by a taxonomist.

We present here another algorithm, a Turing Program for the n-ary Tree Search in Taxonomy, which imitates the taxonomist, as a human automaton.

The Turing Machine was invented by A. M. Turing in 1936 (18) as a theoretical tool for investigating effectively computable functions. In another respect v. Neumann pointed out (13), that the Turing Machine is a machine which can be caused to imitate the behavior of any other machine. Thus, it is possible to construct Turing Programs, i.e. code instruction systems for the Turing Machine, which make it simulate specified automata.

The Turing Machine has proved to be a useful tool for direct study of certain kind of algorithms, frequently with direct symbolic imitation of certain human conceptualization (4, 12, 15).

The Turing Machine consists of a tape, which can be extended unboundedly and divided into successive squares, each square bearing

a symbol, and a machine capable of reading and writing on the tape, one symbol at a time. The machine has a finite number of states and at any time it must be in one of the states, scanning one square on the tape. The machine is able to erase a symbol on the scanned square, to replace it by another (or the same), to move one square to the right R, or one to the left L, or to remain on the same place P. Its action is determined by its present state and by the symbol it is reading. The machine works on a prepared tape (input) and after it stops (if at all) the tape gives the output of the computation.

It is convenient to mark that part of the tape the machine is working on by end markers: h.

The Turing Program includes a list of symbols (alphabet) which is recognizable by the machine, a list of internal states, which the machine can assume and a table of quintuplets of the form: symbol scanned, present state, new state, motion (R, L or P), and new symbol to replace scanned symbol before motion is carried out. The machine stops if the motion is P, the present state is identical with the new state and the symbol scanned is identical with the new symbol.

With these preliminary remarks, we are prepared to describe a solution to the taxonomic problem. It is to be emphasized that while this automaton bears a resemblance to the GA algorithm, it is indeed more flexible since input conventions are not so restrictive for it.

3.2. Turing Program for the n-ary Tree Search in Taxonomy.

In the following we consider a four-level Tree, such that the maximal number m of the Filial Nodes belonging to the same Parental Node is at most 6. The generalization to the n-ary Tree with $m > 6$ is trivial by extending the alphabet.*

Let the alphabet for the Turing Machine be: 0, 1, 2, 3, 4, 5, 6, σ, τ, Q, M, ϕ, *, with $b_i \in P$, $c_j \in P$, $i = 1,..,k$, $j = 1,..,\ell$

where b_i's are describing the value of the node E_N we compare with, c_i's describes the values of the rest of the nodes belonging to the same parental node N. The input for the Turing Machine is of the form:

$$\phi\ M\ .\ .\ .\ .\ Q\ *\ .\ .\ .\ .\ .\ .\ .\ .\ .\ *\ \phi\ \phi\ .\ .\ .\ .\ \phi$$

N-placed code describing the name of node E_N we compare with	value of the node to be identified	N + 3 blank places.

If there is at least one disagreement with the value of node E_N, then the output of the Turing Machine is the name of the next filial

---

* With the choice $n = 4$, $m = 6$ the Turing Program given in this section can be applied immediately to the classification of the bacterial flora found in the gastrointestinal tracts of various animals. The biological part of this problem has been worked out by P. Kilbourn at Oregon State University, Department of Microbiology.

node belonging to the same parental node N, if any, otherwise error halt.

If there is complete agreement, i.e. the results of the experiment is the value of node E_N, then the output of the Turing Machine is the name of the 1st filial node of E_N printed out. The name of the filial node of E_N is a code for a set of experiments to be performed in the laboratory.

We give two examples to illustrate the input and output format.

Example 1.

Let the input of the Turing Machine be the string:

$$\phi\ M\ A\ 1\ 0\ 0\ Q * a\ b\ c\ d\ e * \phi\ \phi\ \phi\ \phi\ \phi\ \phi\ \phi.$$

If in the string a b c d e there is at least one disagreement with the value of E_n (coded as A 1 0 0), then the output is the name of the next filial node (coded as A 2 0 0) belonging to the parental node of E_n, followed by the value of node to be identified. The output is the string

$$\sigma\ M\ A\ 2\ 0\ 0\ Q * a\ b\ c\ d\ e * \tau\ \phi\ \phi\ \phi\ \phi\ \phi\ \phi.$$

Example 2.

Let the input of the Turing Machine be the string:

$$\phi\ M\ A\ 5\ 0\ 0\ Q * a\ b\ c\ d\ e * \phi\ \phi\ \phi\ \phi\ \phi\ \phi\ \phi.$$

If the string a b c d e agrees with the value of E_n (coded as A 5 0 0), then the output is the name of the filial node of E_n (coded

as A 5 1 0). The output is the string

$$M\,\phi\,\phi\,\phi\,\phi\,\phi\,\phi\,\phi\,\phi\,\phi\,\phi\,\phi\,\sigma\,M\,A\,5\,1\,0\,\tau\,\phi.$$

When we get to the last level and we get complete agreement, then we indicate by a mark ! after the name of the item, that no more test is necessary. We may say, that it indicates a final stop.

We give the description of the Turing Machine in the form of a list of quintuplets in Table 1. The symbol % in the listing denotes the name of the Filial Node of E_N with the smallest ordinal number.

TABLE 1

1 * : 2 R *	2 c_1 : 3 L c_1	4 ϕ : 5 R σ
1 M : 1 R M	⋮	4 M : 4 L M
1 Q : 1 R Q	2 c_ℓ : 3 L c_ℓ	4 1 : 4 L 1
1 0 : 1 R 0		4 2 : 4 L 2
1 1 : 1 R 1	3 * : 3 L *	4 3 : 4 L 3
1 2 : 1 R 2	3 b_1 : 3 L b_1	4 4 : 4 L 4
1 3 : 1 R 3	⋮	4 5 : 4 L 5
1 4 : 1 R 4	3 b_k : 3 L b_k	4 6 : 4 L 6
1 5 : 1 R 5	3 Q : 3 L Q	4 A : 4 L A
1 6 : 1 R 6	3 0 : 3 L 0	4 σ : 5 R σ
1 A : 1 R A	3 1 : 4 L 2	
	3 2 : 4 L 3	5 * : 6 R *
2 * : 8 R *	3 3 : 4 L 4	5 M : 5 R M
2 b_1 : 2 R b_1	3 4 : 4 L 5	5 Q : 5 R Q
⋮	3 5 : 4 L 6	5 0 : 5 R 0
2 b_k : 2 R b_k	3 6 : 4 L 7	5 1 : 5 R 1

TABLE 1 (Continued)

5 2 : 5 R 2
5 3 : 5 R 3
5 4 : 5 R 4
5 5 : 5 R 5
5 6 : 5 R 6
5 A : 5 R A

6 * : 6 R *
6 b_1 : 6 R b_1
⋮
6 b_k : 6 R b_k
6 c_1 : 6 R c_1
⋮
6 c_ℓ : 6 R c_ℓ
6 ϕ : 7 L T
6 T : 7 L T

7 * : 7 L *
7 b_1 : 7 L b_1
⋮
7 b_k : 7 L b_k
7 c_1 : 7 L c_1
⋮
7 c_ℓ : 7 L c_ℓ
7 M : 7 P M
7 Q : 7 L Q
7 0 : 7 L 0
7 1 : 7 L 1

7 2 : 7 L 2
7 3 : 7 L 3
7 4 : 7 L 4
7 5 : 7 L 5
7 6 : 7 L 6
7 A : 7 L A

8 ϕ : 9 R σ

9 ϕ : 10 R M

10 ϕ : 11 R A

11 ϕ : 12 R %

12 ϕ : 13 R %

13 ϕ : 14 R %

14 ϕ : 14 L T
14 M : 14 L M
14 0 : 14 L 0
14 1 : 14 L 1
14 2 : 14 L 2
14 3 : 14 L 3
14 4 : 14 L 4
14 5 : 14 L 5
14 6 : 14 L 6

14 A : 14 L A
14 σ : 15 L σ
14 T : 14 L T

15 * : 15 L ϕ
15 b_1 : 15 L ϕ
15 b_2 : 15 L ϕ
⋮
15 b_k : 15 L ϕ
15 c_1 : 15 L ϕ
⋮
15 c_ℓ : 15 L ϕ
15 M : 15 P M
15 Q : 15 L ϕ
15 0 : 15 L ϕ
15 1 : 15 L ϕ
15 2 : 15 L ϕ
15 3 : 15 L ϕ
15 4 : 15 L ϕ
15 5 : 15 L ϕ
15 6 : 15 L ϕ
15 A : 15 L ϕ

For the n-ary Tree Search in Taxonomy we have to construct several Turing Machines differing only in their alphabets. This task is a boring, time-consuming, repetitive one. We should therefore hope to construct a Turing Machine which will write these Turing Machines automatically. We describe the generator Turing Machine in general terms.

Let T_a be a Turing Machine with an alphabet $\{a_1, a_2, .., a_k, x_1, x_2, .., x_{t-k}\}$, where $a_1, a_2, .., a_k$ are variables. Let $B_i = \{{}^1b_i, {}^2b_i, .., {}^{s_i}b_i\}$ be the set of values, that variable a_i may assume $(i = 1, .., k)$.

Consider the ordered k-tuple $({}^1\ell, {}^2\ell, .. {}^k\ell)$ of constants we can obtain by choosing an arbitrary element successively from B_1, then from B_2, etc.

We wish to construct a Turing Machine T_b, which will be able to rewrite T_a by substituting ${}^i\ell$ for a_i for $i = 1, .., k$.

Denote the generated Turing Machine by $T_a({}^1\ell, {}^2\ell, .., {}^k\ell)$.

The alphabet of T_b is

$$[(\text{the alphabet of } T_a) \cup \bigcup_{i=1}^{k} B_i \cup \{\phi, *, h\}].$$

Let the input for T_b be the string:

$$h * {}^1\ell, {}^2\ell, .., {}^k\ell * \text{ the quintuplets of } T_a\ h,$$

and let the output be the string:

$$h * \phi\phi \ldots . \phi * \text{ the quintuplets of } \tau_a({}^1\ell, {}^2\ell, \ldots {}^k\ell)\, h.$$

The number of states in τ_b is

$$s_1 + s_2 + \ldots + s_k + 2s_k.$$

Assuming that the number of symbols in the alphabet of τ_a is t, τ_b will have $t(s_1 + s_2 + \ldots + 3s_k) + (s_1 + s_2 + \ldots + 3s_k)^2$ quintuplets. Usually this number is too large for a computer to handle, so we need to break τ_b down into a sequence of Turing Machines of smaller size:

$$\tau_b = \tau_1\, \tau_2 \cdots \tau_j \cdots \tau_k.$$

Let the alphabet for τ_j be

$$[\text{The alphabet of } \tau_a \cup B_j \cup \{\phi, *, h\}].$$

The number of quintuplets is:

$$t(s_j + 2) + (s_j + 2)^2.$$

If the size of τ_j is still too large, we can break down τ_j also into a chain of Turing Machines:

$$\tau_j = \tau_{j1}\, \tau_{j2} \cdots \tau_{ji} \cdots \tau_{jm},$$

where the alphabet for τ_{ji} is

$$[\text{The alphabet of } \tau_a \cup B_{ji} \cup \{\phi, *, h\}],$$

where $\bigcup_{i=1}^{m} B_{ji} = B_j$. Let the number of elements of B_{ji} be n_{ji}.

The number of quintuplets for τ_{ji} is:

$$t(n_{ji} + 2) + (n_{ji} + 2)^2.$$

The list of quintuplets for the generating Turing Machine is given in Table 2. The mark ? in the list indicates an error-halt.

TABLE 2

0	h	:	1	R	h
1	h	:	(s_1+1)	L	h
1	a_1	:	1	R	$^1b_1$
1	a_2	:	1	R	a_2
⋮					
1	a_k	:	1	R	a_k
1	x_1	:	1	R	x_1
1	x_2	:	1	R	x_2
⋮					
1	x_{t-k}	:	1	R	x_{t-k}
1	*	:	1	R	*
1	$^1b_1$	:	1	R	ϕ
1	$^2b_1$	:	2	R	ϕ
1	$^3b_1$	:	3	R	ϕ
⋮					
1	$^{s_1}b_1$	:	s_1	R	ϕ

1	$^1b_2$	:	1	R	$^1b_2$
1	$^2b_2$	:	1	R	$^2b_2$
⋮					
1	$^{s_2}b_2$	:	1	R	$^{s_2}b_2$
⋮					
1	^1b_k	:	1	R	^1b_k
1	^2b_k	:	1	R	^2b_k
⋮					
1	$^{s_k}b_k$	:	1	R	$^{s_k}b_k$

2	h	:	(s_1+1)	L	h
2	a_1	:	2	R	$^2b_1$
2	a_2	:	2	R	a_2
⋮					
2	a_k	:	2	R	a_k
2	x_1	:	2	R	x_1

TABLE 2 (Continued)

$2\ x_2 : 2\ R\ x_2$

$\vdots$

$2\ x_{t-k} : 2\ R\ x_{t-k}$

$2\ * : 2\ R\ *$

$2\ {}^{1}b_2 : 2\ R\ {}^{1}b_2$

$2\ {}^{2}b_2 : 2\ R\ {}^{2}b_2$

$\vdots$

$2\ {}^{s_2}b_2 : 2\ R\ {}^{s_2}b_2$

$\vdots$

$2\ {}^{1}b_k : 2\ R\ {}^{1}b_k$

$2\ {}^{2}b_k : 2\ R\ {}^{2}b_k$

$\vdots$

$2\ {}^{s_k}b_k : 2\ R\ {}^{s_k}b_k$

$\vdots$

$s_1\ h : (s_1+1)\ L\ h$

$s_1\ a_1 : s_1\ R\ {}^{s_1}b_1$

$s_1\ a_2 : s_1\ R\ a_2$

$\vdots$

$s_1\ a_k : s_1\ R\ a_k$

$s_1\ x_1 : s_1\ R\ x_1$

$s_1\ x_2 : s_1\ R\ x_2$

$\vdots$

$s_1\ x_{t-k} : s_1\ R\ x_{t-k}$

$s_1\ * : s_1\ R\ *$

$s_1\ {}^{1}b_2 : s_1\ R\ {}^{1}b_2$

$s_1\ {}^{2}b_2 : s_1\ R\ {}^{2}b_2$

$\vdots$

$s_1\ {}^{s_2}b_2 : s_1\ R\ {}^{s_2}b_2$

$\vdots$

$s_1\ {}^{1}b_k : s_1\ R\ {}^{1}b_k$

$s_1\ {}^{2}b_k : s_1\ R\ {}^{2}b_k$

$\vdots$

$s_1\ {}^{s_k}b_k : s_1\ R\ {}^{s_k}b_k$

$(s_1+1)\ h : (s_1+1)\ P\ ?$

$(s_1+1)\ a_1 : (s_1+1)\ L\ a_1$

$(s_1+1)\ a_2 : (s_1+1)\ L\ a_2$

$\vdots$

$(s_1+1)\ a_k : (s_1+1)\ L\ a_k$

$(s_1+1)\ x_1 : (s_1+1)\ L\ x_1$

$(s_1+1)\ x_2 : (s_1+1)\ L\ x_2$

$\vdots$

$(s_1+1)\ x_{t-k} : (s_1+1)\ L\ x_{t-k}$

$(s_1+1)\ * : (s_1+1)\ L\ *$

$(s_1+1)\ {}^{1}b_2 : (s_1+1)\ L\ {}^{1}b_2$

$(s_1+1)\ {}^{2}b_2 : (s_1+1)\ L\ {}^{2}b_2$

$\vdots$

$(s_1+1)\ {}^{s_2}b_2 : (s_1+1)\ L\ {}^{s_2}b_b$

$\vdots$

$(s_1+1)\ {}^{1}b_k : (s_1+1)\ L\ {}^{1}b_k$

$(s_1+1)\ {}^{2}b_k : (s_1+1)\ L\ {}^{2}b_k$

$\vdots$

TABLE 2 (Continued)

$(s_1+1)\ {}^{s_k}b_k : (s_1+1)\ L\ {}^{s_k}b_k$

$(s_1+1)\ \phi : (s_1+2)\ R\ \phi$

$(s_1+2)\ * : z\ R\ \sigma$

$(s_1+2)\ {}^{1}b_2 : (s_1+3)\ R\ \phi$

$(s_1+2)\ {}^{2}b_2 : (s_1+4)\ R\ \phi$

$(s_1+2)\ {}^{3}b_2 : (s_1+5)\ R\ \phi$

$\vdots$

$(s_1+2)\ {}^{s_2}b_2 : (s_1+s_2)\ R\ \phi$

$\vdots$

$(s_1+2)\ {}^{1}b_k : (s_1+2)\ R\ {}^{1}b_k$

$(s_1+2)\ {}^{2}b_k : (s_1+2)\ R\ {}^{2}b_k$

$(s_1+2)\ {}^{3}b_k : (s_1+2)\ R\ {}^{3}b_k$

$\vdots$

$(s_1+2)\ {}^{s_k}b_k : (s_1+2)\ R\ {}^{s_k}b_k$

$(s_1+3)\ h : (s_1+s_2+3)\ L\ h$

$(s_1+3)\ a_2 : (s_1+3)\ R\ {}^{1}b_2$

$\vdots$

$(s_1+3)\ a_k : (s_1+3)\ R\ a_k$

$(s_1+3)\ x_1 : (s_1+3)\ R\ x_1$

$(s_1+3)\ x_2 : (s_1+3)\ R\ x_2$

$\vdots$

$(s_1+3)\ x_{t-k} : (s_1+3)\ R\ x_{t-k}$

$(s_1+3)\ * : (s_1+3)\ R\ *$

$\vdots$

$(s_1+3)\ {}^{1}b_k : (s_1+3)\ R\ {}^{1}b_k$

$(s_1+3)\ {}^{2}b_k : (s_1+3)\ R\ {}^{2}b_k$

$(s_1+3)\ {}^{3}b_k : (s_1+3)\ R\ {}^{3}b_k$

$\vdots$

$(s_1+3)\ {}^{s_k}b_k : (s_1+3)\ R\ {}^{s_k}b_k$

$(s_1+4)\ h : (s_1+s_2+3)\ L\ h$

$(s_1+4)\ a_2 : (s_1+4)\ R\ {}^{2}b_2$

$\vdots$

$(s_1+4)\ a_k : (s_1+4)\ R\ a_k$

$(s_1+4)\ x_1 : (s_1+4)\ R\ x_1$

$(s_1+4)\ x_2 : (s_1+4)\ R\ x_2$

$\vdots$

$(s_1+4)\ x_{t-k} : (s_1+4)\ R\ x_{t-k}$

$(s_1+4)\ * : (s_1+4)\ R\ *$

$\vdots$

$(s_1+4)\ {}^{1}b_k : (s_1+4)\ R\ {}^{1}b_k$

$(s_1+4)\ {}^{2}b_k : (s_1+4)\ R\ {}^{2}b_k$

$\vdots$

$(s_1+4)\ {}^{s_k}b_k : (s_1+4)\ R\ {}^{s_k}b_k$

$\vdots$

$(s_1+s_2+2)\ h : (s_1+s_2+3)\ L\ h$

$(s_1+s_2+2)\ a_2 : (s_1+s_2+2)\ R\ {}^{s_2}b_2$

$\vdots$

$(s_1+s_2+2)\ a_k : (s_1+s_2+2)\ R\ a_k$

TABLE 2 (Continued)

$(s_1+s_2+2)\ x_1 : (s_1+s_2+2)\ R\ x_1$
$\vdots$
$(s_1+s_2+2)\ x_{t-k} : (s_1+s^2+2)\ R\ x_{t-k}$
$(s_1+s_2+2)\ * : (s_1+s_2+2)\ R\ *$
$(s_1+s_2+2)\ {}^1b_k : (s_1+s_2+2)\ R\ {}^1b_k$
$\vdots$
$(s_1+s_2+2)\ {}^{s_k}b_k : (s_1+s_2+2)\ R\ {}^{s_k}b_k$

$(s_1+s_2+3)\ h : (s_1+s_2+3)\ P\ ?$
$(s_1+s_2+3)\ a_1 : (s_1+s_2+3)\ L\ a_1$
$(s_1+s_2+3)\ a_2 : (s_1+s_2+3)\ L\ a_2$
$\vdots$
$(s_1+s_2+3)\ a_k : (s_1+s_2+3)\ L\ a_k$
$(s_1+s_2+3)\ x_1 : (s_1+s_2+3)\ L\ x_1$
$(s_1+s_2+3)\ x_2 : (s_1+s_2+3)\ L\ x_2$
$\vdots$
$(s_1+s_2+3)\ x_{t-k} : (s_1+s_2+3)\ L\ x_{t-k}$
$(s_1+s_2+3)\ * : (s_1+s_2+3)\ L\ *$
$\vdots$

$(s_1+s_2+3)\ {}^1b_k : (s_1+s_2+3)\ L\ {}^1b_k$
$(s_1+s_2+3)\ {}^2b_k : (s_1+s_2+3)\ L\ {}^2b_k$
$\vdots$
$(s_1+s_2+3)\ {}^{s_k}b_k : (s_1+s_2+3)\ L\ {}^{s_k}b_k$
$(s_1+s_2+3)\ \phi : (s_1+s_2+4)\ R\ \phi$

$(s_1+s_2+4)\ * : Z\ R\ \sigma$
$\vdots$

$Z\ h : Z\ P\ h$
$Z\ a_1 : Z\ R\ a_1$
$Z\ a_2 : Z\ R\ a_2$
$\vdots$
$Z\ a_k : Z\ R\ a_k$
$Z\ x_1 : Z\ R\ x_1$
$Z\ x_2 : Z\ R\ x_2$
$\vdots$
$Z\ x_{t-k} : Z\ R\ x_{t-k}$
$Z\ * : Z\ R\ \tau$
$Z\ {}^1b_1 : Z\ R\ {}^1b_1$
$\vdots$
$Z\ {}^{s_1}b_1 : Z\ R\ {}^{s_1}b_1$
$Z\ {}^1b_2 : Z\ R\ {}^1b_2$
$Z\ {}^2b_2 : Z\ R\ {}^2b_2$
$\vdots$
$Z\ {}^1b_k : Z\ R\ {}^1b_k$
$Z\ {}^2b_k : Z\ R\ {}^2b_k$
$\vdots$
$Z\ {}^{s_k}b_k : Z\ R\ {}^{s_k}b_k$

There is growing need indicated in publications of the life sciences for the automation of the mechanic part of the work of a taxonomist (7). We quote a typical phrasing of the problem by T. L. Jahn (9):

> Could we code a complete description of an organism on a card, drop it into a machine, and have the machine tell us its proper name and code number. . . . The only requirement would be, that we teach the machine the method of classification to be used. This teaching could consist of feeding it a piece of magnetic tape on which we had coded either the whole taxonomic system or some suitable section of it.

The above given method gives a possible solution to this question.

BIBLIOGRAPHY

1. Appleby, J. S., D. V. Blake and E. A. Newman. Techniques for producing school timetables on a computer and their application to other scheduling problems. Computer Journal 3:237-245. 1960.

2. Blakesley, J. F. Automation in college management. College and University Business 27:39-44. 1959.

3. Bossert, W. H., and J. B. Harmon Student sectioning on the IBM 7090. Cambridge, Mass., IBM Corp., March 1963. 73 numb. leaves.

4. Coffin, R. W., H. E. Goheen and W. R. Stahl. Simulation of a Turing Machine on a digital computer. In: AFIPS Conference Proceedings, Western Joint Computer Conference. Las Vegas, Nev., Baltimore, Md. Spartan Books, 1964. p. 35-43.

5. Csima, J. and C. C. Gotlieb. Tests on a computer method for constructing school time-tables. Communications of the ACM 7(3):160-163. March 1964.

6. Egbert, R. L. The computer in education: Malefactor or benefactor. In: AFIPS Conference Proceedings Fall Joint Computer Conference. Baltimore, Md. Spartan Books, 1963. p. 619-630.

7. Ehrlich, P. R. Systematics in 1970. Journal of Parasitology 48:157-158. 1962.

8. Gotlieb, C C. The construction of class-teacher time-tables. In: Proceedings of IFIP Congress 62, Munich. Amsterdam, North Holland Pub. Co., 1963. p. 73-77.

9. Jahn, T. L The use of computer in systematics. Journal of Parasitology 48:656-663. 1962.

10. Kitov, A. I. and Krinitskiy, N. A. Elektronnye Tsifroviye Mashini i Programmirovaniye. (Electronic digital machines and programming). State Publishing House for Physico-Mathematical Literature, Moscow, USSR (1959), 572 p. (In Russian)

11. Lukey, I. K. and Hubbard, D. A. Student scheduling technique using defined increments of time. IBM 1620 General Program Library. Cambridge, Mass., IBM Corporation, n.d. 90 numb. leaves.

12. McNaughton, R. The theory of automata, a survey. Advances in Computers 2:379-421. 1961.

13. Neumann von, J. The computer and the brain. New Haven, Yale University Press, 1958. 82 p.

14. Ore, O. Theory of graphs. Providence, Rhode Island, American Mathematical Society, 1962. 270 p.

15. Stahl, W. R., R. W. Coffin and H. E Goheen. Simulation of biological cells by systems composed of string-processing finite automata. In press.

16. Sussenguth, E H. Use of tree structures for processing files. Communications of ACM 5:273-279. 1963.

17. Trakhenbrot, B. A. Algorithms and the machine solution of problems. (In Russian) Available under the title: Algorithms and Automatic Computing Machines. Boston, Heath, 1963. 112 p. (Topics in Mathematics.)

18. Turing, A. M. On computable numbers, with an application to the Entscheidungs problem. Proceedings of the London Mathematical Society, Ser 2, 42:230-265. 1936. 43:544-546. 1937.