DOCUMENT RESUME

ED 099 973 EA 006 590

AUTHOR Nunamaker, J. F., Jr.; And Others
TITLE Processing Systems Optimization Through Automatic Design and Reorganization of Program Modules.
INSTITUTION Purdue Univ., Lafayette, Ind. Herman C. Krannert Graduate School of Industrial Administration.
REPORT NO Pap-391
PUB DATE Jan 73
NOTE 47p.

EDRS PRICE MF-$0.75 HC-$1.85 PLUS POSTAGE
DESCRIPTORS Bibliographies; Computer Oriented Programs; Computers; Computer Storage Devices; Cybernetics; *Electronic Data Processing; Electronic Equipment; Information Processing; *Information Storage; *Information Systems; Performance Criteria; Systems Analysis; *Systems Approach; Systems Concepts; *Systems Development

ABSTRACT

A methodology is described for an automatic design system initially defined in terms of logical processes or program modules. Processes and files are grouped and reorganized in such a way as to produce an optimal design with respect to a specific target machine. Performance criteria for the optimal design are defined in terms of transport volume savings and core memory requirements. Starting with a graph theoretic representation of the interaction between processes (or modules) and files, the methodology consists of two components: (1) a generator of feasible alternatives and (2) a procedure for reorganization and core generation for specific groupings. Not only can an optimal design for the processing system be generated; but due to reorganization techniques, the resultant modules (defined from specific process groupings) may approach the computational efficiency expected of an integrated program. (Author)

PROCESSING SYSTEMS OPTIMIZATION THROUGH AUTOMATIC DESIGN AND REORGANIZATION OF PROGRAM MODULES

by

J. F. Nunamaker, Jr.
W. C. Nylin, Jr.
and
Benn Konsynski

Paper No. 391 - January 1973

Institute for Research in the
BEHAVIORAL, ECONOMIC, and
MANAGEMENT SCIENCES

KRANNERT GRADUATE SCHOOL OF
INDUSTRIAL ADMINISTRATION

Purdue University
West Lafayette, Indiana

PROCESSING SYSTEMS OPTIMIZATION THROUGH AUTOMATIC DESIGN AND REORGANIZATION OF PROGRAM MODULES

by

J. F. Nunamaker, Jr.

W. C. Nylin, Jr.

and

Benn Konsynski

Purdue University

December, 1972

Presented at the 4th International Conference on Computer and Information Science. Proceedings to be published by ACADEMIC PRESS in 1973.

ABSTRACT

PROCESSING SYSTEMS OPTIMIZATION THROUGH AUTOMATIC DESIGN AND REORGANIZATION OF PROGRAM MODULES

J.F. Nunamaker, Jr., W.C. Nylin, Jr., and Benn Konsynski

A methodology is described for the automatic design of a processing system initially defined in terms of logical processes or program modules. Processes and files are grouped and reorganized in such a way as to produce an optimal design with respect to a specific target machine. Performance criteria for the optimal design is defined in terms of transport volume savings and core memory requirements.

Starting with a graph theoretic representation of the interaction between processes (or modules) and files, the methodology consists of two components: (1) a generator of feasible alternatives and (2) a procedure for reorganization and code generation for specific groupings. The generator for the feasible alternatives uses an implicit enumeration algorithm to optimize process groupings in an efficient manner. The objective is to group processes into modules which minimize the interaction between modules while still satisfying the logical requirements of the program and the physical constraints of the hardware. Finally, after the program modules have been specified, program and file reorganization will be performed to further optimize the design. Reorganization includes the combination of similar data passes on the same file to minimize transport volume and the merging of loops to enable elimination of code and of intermediate data files.

The code generator will then accept the optimal program design and produce an optimized source language program for the target machine. Consequently, not only can an optimal design for the processing system be generated; but due to reorganization techniques, the resultant modules (defined from specific process groupings) may approach the computational efficiency expected of an integrated program.

PROCESSING SYSTEMS OPTIMIZATION THROUGH AUTOMATIC DESIGN AND REORGANIZATION OF PROGRAM MODULES

J.F. Nunamaker, Jr.
Purdue University
Lafayette, Indiana

William C. Nylin, Jr.
Southern Methodist University
Dallas, Texas

and

Benn Konsynski, Jr.
Purdue University
Lafayette, Indiana

I. INTRODUCTION

It is recognized that perhaps the single most important problem which faces a computer user is that of conversion of programs to another machine. This is true even for programs written in "machine independent", high-level source languages. Changes in the system configuration; e.g., hardware, operating system, or file structure, may have altered the operating environment significantly so that the programs no longer take advantage of the strength of the configuration.

For whatever reasons that make the conversion necessary, such as the replacement of an obsolete machine or the requirement to run the program on additional machines, the situation is applicable to many users. It is also recognized that many users have considered automatic conversion of computer programs with techniques such as emulation and simulation. However, very few users have seriously attacked the problem of the optimal reorganization and design of the program when moving it from one machine to another.

Many users are of the opinion that anything less than 100% automatic conversion is not worth considering; however, it can be stated emphatically that less-than-complete conversion tools are useful and the redesign and reorganization are necessary for efficient operation of the resulting program modules.

The transferability problem touches on all aspects of software design; specific methodology from decompiling, graph models of programs, operations research search techniques, and problem statement languages are used to form an approach to the problem.

II. METHODOLOGY

What is needed is a methodology for converting, redesigning, and reorganizing programs from one machine to another as a result of stated performance criteria.

This paper discusses a software system for the design and reorganization of computer programs, and a methodology is described for the automatic design of a processing system initially defined in terms of logical processes or program modules. Processes and files are grouped and reorganized in such a way as to produce an optimal design with respect to a specific target machine. Performance criteria for the optimal design are defined in terms of transport volume savings, core memory requirements, and input/output requirements.

Transport volume of a system is a measure of performance that is related to total processing time. Processing time is a non-decreasing function of transport volume; therefore, it is desirable to decrease the transport volume of a set of program modules. It was shown by Nunamaker [1,2] that there exists a class of process groupings which result in a reduction of transport volume when two or more processes are grouped. Using a simple case as an example, the transport volume is reduced when two processes are grouped if the output of one is the input to the other process. As a result of the grouping of the processes into a composite program module, the core requirement will be increased and the input/output requirements of the system may be affected.

In this paper the assumption is made that we are starting with a well-defined problem, and that the set of processes can be described in terms of a directed graph. In addition, we know other information such as frequency, volumes, etc.

Building on the graph theory representation of the interaction between processes (or modules) and files, the methodology consists of two components: (1) a generator of feasible alternatives and (2) a procedure for reorganization and code generation for specific groupings. The generator for the feasible alternatives uses an implicit enumeration algorithm to generate alternative groupings in an efficient manner. The objective is to group processes into modules which minimize the interaction between modules while still satisfying the logical

requirements of the program and the physical constraints of the hardware. Finally, after the program modules have been specified, program and file reorganization will be performed to further optimize the design. Reorganization includes the combination of similar data passes on the same file to minimize transport volume and the merging of loops to enable elimination of code and intermediate data files.

The Program Module Generator presents a very large set of feasible program modules for the target machines; it is the task of the selection algorithm and the reorganizer to construct a reasonably good set of program modules. The code generator then accepts the optimal program design and creates the optimized physical code for the target machine. Thus, the Program Module Generator chooses from among all conceivable combinations of processes for program modules and selects the "best" design after considerable interaction with the program reorganizer.

Consequently, not only can an optimal design for the processing system be generated, but due to reorganization techniques, the resultant modules (defined from specific process groupings) may approach the computational efficiency expected of an integrated program.

An overview of the methodology for the automatic design and reorganization of program modules is shown in Figure 1. The specific subject of this paper begins with the assumption that a Problem Definition exists and is shown below the dotted line in Figure 1. The problem definition, generation, and translation into a problem statement is the subject of another paper [2].

III. DEFINITIONS

A programming system, PS = (PR,F,T,E), is defined as a set of processes (PR), files (F), the control flow of the files(T), and the relationships of the set of processes and files (E). These definitions are extensions of the work of Langefors [3] and Briggs [4].

pr - Process--A well-defined task representing a pass over one or more data files; where $PR=(pr_1, pr_2, \ldots, pr_n)$.

f - File--The data input or output of a process; where $F=(f_1, f_2, \ldots, f_k)$.

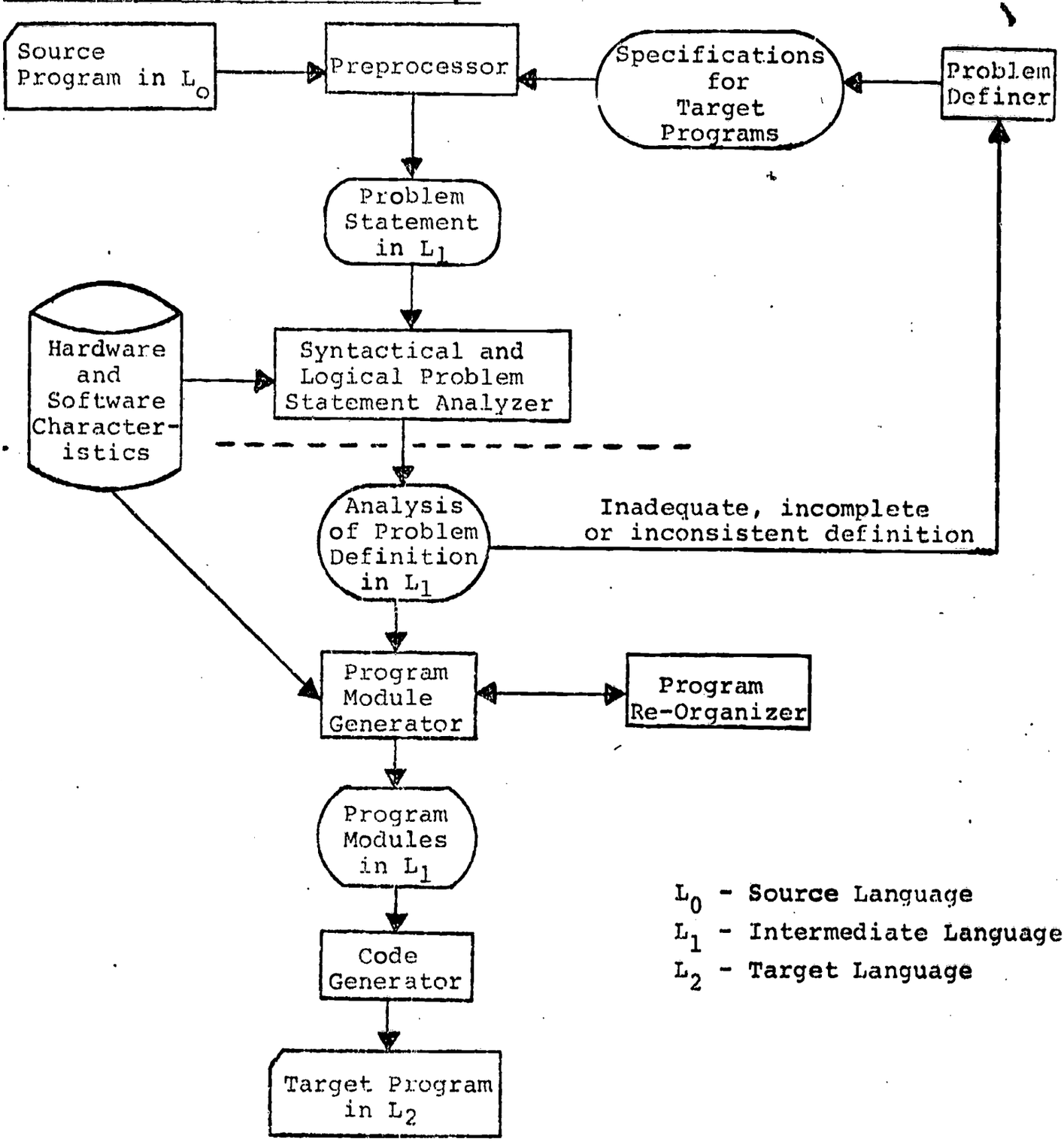


Figure 1. Overview of a System for the Automatic Design and Reorganization of Program Modules

T - Control Flow Matrix--The control flow precedence relationship of the programming system.

t_{ij} = 1 if control flow can pass directly from f_i to f_j.

t_{ij} = 0 otherwise.

E - Incidence Matrix--Processes and files.

e_{ij} = 1 if f_j is an input to pr_i.

e_{ij} = -1 if f_j is an output of pr_i.

e_{ij} = 0 if there is no incidence between f_j and pr_i.

The control flow of a network is described by T, and the data relationship of the processes and files in a network is described by E.

From the Incidence Matrix we can define the concept of transport volume. Transport volume is one component of the performance criteria which are used to evaluate alternative program module design. Performance criteria for program module design is a function of the following compcnents: (1) processing time, (2) transport volume, (3) core size, and (4) the number and type of input/output units required.

Let v_j be the volume of file f_j; ℓ_i, the number of logical inputs and outputs of pr_i; and mp_j, the multiplicity of file transport for f_j. Thus mp_j represents the number of times f_j is an input or output of a set of processes; cm_i represents the core memory required by pr_i.

$$\ell_i = \sum_{j=1}^{k} |e_{ij}| \; ; \; i = 1,2,\ldots,n.$$

$$mp_j = \sum_{i=1}^{n} |e_{ij}| \; ; \; j = 1,2,\ldots,k.$$

The transport volume for f_j is:

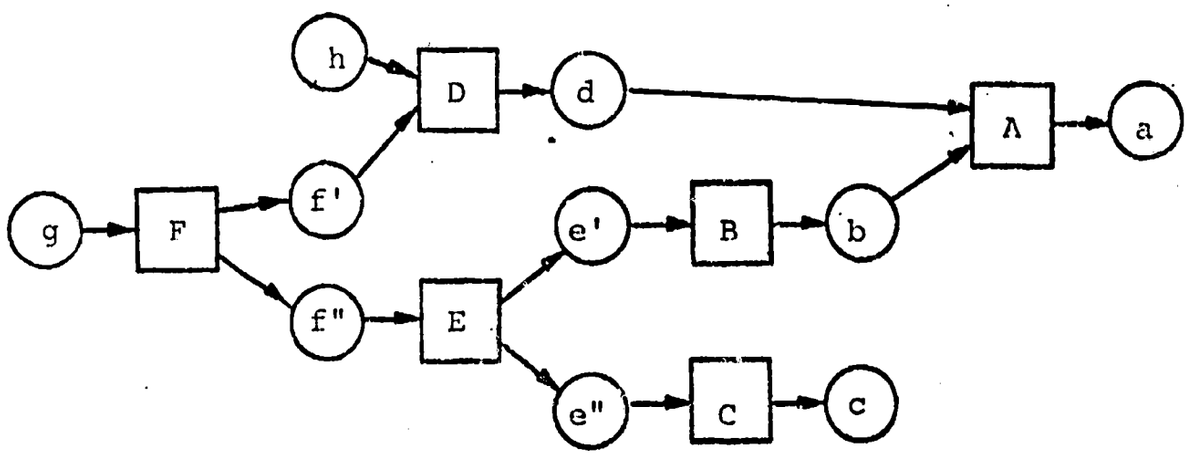
$$tv_j = mp_j \cdot v_j.$$

The transport volume for the set of data files is:

$$TV = \sum_{j=1}^{k} tv_j.$$

Let pr_i be represented by a $\square$ and f_j by a $\bigcirc$.

The Incidence Matrix (E) and the associated incidence graph for a programming system of six processes and ten files is shown in Figure 2.



Files

Processes	a	b	c	d	e'	e"	f'	f"	g	h	ℓ_i	cm_i
A	-1	1	0	1	0	0	0	0	0	0	3	30
B	0	-1	0	0	1	0	0	0	0	0	2	10
C	0	0	-1	0	0	1	0	0	0	0	2	15
D	0	0	0	-1	0	0	1	0	0	1	3	20
E	0	0	0	0	-1	-1	0	1	0	0	3	10
F	0	0	0	0	0	0	-1	-1	1	0	3	10
mp_j	1	2	1	2	2	2	2	2	1	1		
v_j	30	10	20	50	10	10	10	20	30	50		
tv_j	30	20	20	100	20	20	20	40	30	50		

Figure 2. Incidence Graph and Matrix E for a programming system of 6 processes and 10 files.

This example is also used later in Section VII.

We now must define additional matrices needed for the grouping procedure.

The E matrix of processes and files is used to generate the data flow Precedence Matrix of processes P. Note that a distinction is made between the control flow T of the programming system and the precedence relationship of the processes with respect to data flow.

P - Precedence Matrix: Processes

$p_{ij} = 1$ if pr_i is a direct precedent of pr_j.

$p_{ij} = 0$ if otherwise.

P can be reconstructed from E as follows:

$p_{ij} = 1$ if and only if $\exists\ \ell$ ∋ $e_{i\ell} = -1$ and $e_{j\ell} = 1$.

The Precedence Matrix (P) of processes for the example of Figure 2 is shown in Figure 3.

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	1	0	0	0	0	0
C	0	0	0	0	0	0
D	1	0	0	0	0	0
E	0	1	1	0	0	0
F	0	0	0	1	1	0

Figure 3. Precedence Matrix of Processes P.

The R, R*, and G matrices are generated for the entire set of Processes.

R - Reachability Matrix: Processes

The R matrix is used to check precedence violations in the grouping procedure

$$R = P \vee P^2 \vee \ldots . \vee P^{q-1}$$

where q is the index of the nilpotent matrix P; i.e., when $P^q = 0$.

$r_{ij} = 1$ if pr_i has any precedence relationship with pr_j.

$r_{ij} = 0$ otherwise.

The Reachability Matrix (R) of Processes for the example of Figure 2 is shown in Figure 4.

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	1	0	0	0	0	0
C	0	0	0	0	0	0
D	1	0	0	0	0	0
E	1	1	1	0	0	0
F	1	1	1	1	1	0

Figure 4. Reachability Matrix of Processes R.

R* - Partial Reachability Matrix: Processes

The R* matrix is used to calculate the G Matrix.

$R^* = P^2 \vee P^3 \vee \ldots \vee P^{q-1}$

r^*_{ij} = 1 if pr_i has a higher (2 or more) order precedence with pr_j.

r^*_{ij} = 0 otherwise.

The Partial Reachability Matrix (R*) of Processes for the example of Figure 2 is shown in Figure 5.

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	1	0	0	0	0	0
F	1	1	1	0	0	0

Figure 5. Partial Reachability Matrix of Processes R*.

G - Feasible Process Pairs Grouping Matrix: Processes

If $g_{ij} = -1$, there exists higher (2 or more) order relationships between pr_i and pr_j; and pr_i cannot be combined with pr_j. If $g_{ij} = 0$, there is no precedence ordering; and pr_i can be combined with pr_j. This indicates a feasible but not necessarily profitable grouping. If $g_{ij} = 1$, there is either a direct precedence relationship, and pr_i can and should be combined with pr_j since this indicates a feasible and profitable grouping; or there is an immediate reduction in logical input/output requirements when pr_i and pr_j are grouped.

$g_{ij} = -1$ if r^*_{ij} or $r^*_{ji} = 1$ or $i=j$.

$g_{ij} = 0$ if $r^*_{ij} = 0$ and $r^*_{ji} = 0$ and $p_{ij} = 0$ and $p_{ji} = 0$; except when $(p_{i\ell}=1$ and $p_{j\ell}=1)$ or $(p_{\ell i}=1$ and $p_{\ell j}=1)$.

$g_{ij} = 1$ if $r^*_{ij} = 0$ and $r^*_{ji} = 0$ and $\left[(p_{ij}=1) \text{ or } (p_{ji}=1) \text{ or } (p_{i\ell}=1 \text{ and } p_{j\ell}=1) \text{ or } (p_{\ell j}=1 \text{ and } p_{\ell i}=1)\right]$.

pr_ℓ has a first order precedence or succedence relationship with pr_i and pr_j.

The Feasible Process Pairs Grouping Matrix (G) for the example of Figure 2 is shown in Figure 6.

	A	B	C	D	E	F
A	-1	1	0	1	-1	-1
B	1	-1	1	1	1	-1
C	0	1	-1	0	1	-1
D	1	1	0	-1	1	1
E	-1	1	1	1	-1	1
F	-1	-1	-1	1	1	-1

Figure 6. Feasible Process Pairs Grouping Matrix G.

A list of all feasible pairs for grouping of processes is constructed from the G Matrix and passed to the generator of alternative groupings.

It is known that by grouping processes into a composite process called a program module, the multiple input and output of files can be reduced. A program module PM_i is created by combining and reorganizing processes $pr_{i_1}, pr_{i_2}, \ldots, pr_{i_k}$. Let L_0 be the source language; L_1, the intermediate language; and L_2, the target language. Note that a situation may exist in which a single language could serve as L_0, L_1, and L_2.

Define procedure RG which maps processes in language L_0 into reorganized modules in language L_1. RG performs the following tasks:

1. Conversion of the individual processes from L_0 to L_1 if $L_0 \neq L_1$.
2. Transformation of the processes written in L_1 into a syntactically and logically correct module in L_2.
3. Reorganization of the module.

Thus, one can define a program module PM_i using RG.

$PM_i = RG(pr_{i_1}, \ldots, pr_{i_k})$ is a <u>feasible program module</u> if $pr_{i_j} \in PR, 1 \leq j \leq k$, and PM_i satisfies all the constraints for a valid subprogram in language L for the target machine. Thus, a feasible program module must satisfy the core memory and logical constraints of the program.

$M_i = \{pr_{i_1}, pr_{i_2}, \ldots, pr_{i_k}\}$ is a <u>feasible grouping</u> if PM_i is a feasible program module.

$\delta = \{M_1, M_2, \ldots, M_q\}$ is a <u>cover</u> for the programming system PS if:

i) M_i, $1 \leq i \leq q$ is a feasible grouping.

ii) $M_1 \cup M_2 \cup \ldots \cup M_q = PR$.

iii) $M_i \cap M_j = \emptyset$ for $i \neq j \; \forall \; j<q, j<i, i \leq q$.

Note that $PM_i = RG(pr_i)$ must be a feasible module for $1 \leq i \leq n$ for a cover for PS to exist. We can now define the set of all possible covers (Δ) for PS. (i.e., $\Delta = \{\delta \mid \delta$ is a cover of PS$\}$.)

IV. PROCESS GROUPING CONCEPT

In generating an efficient design, it is necessary to decrease the transport volume (total number of characters read in and written out of main memory) in order to reduce the processing time. If file volumes remain constant, in order to decrease the transport volume, the multiplicity (the number of times a file is input and output) of file transport must be decreased. After the Program Modules are specified, the files are consolidated for the purpo of reducing the number of input/output files required and for better utilization of storage in auxiliary memory. Process grouping is shown to correspond to a grouping of rows of the Incidence Matrix, and file consolidation is shown to correspond to a grouping of columns.

Program module design is concerned with the reduction of processing time and can be summarized by the two methods by which the processing time can be reduced. The generator of alternatives determines which operations (Processes) will be grouped into Program Modules:

1. Group Processes which eliminate the writing out and the reading in of a file. Consider the example in which the output of one process is the input to another process, as shown in Figure 7.

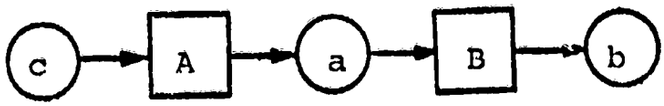


Figure 7. The output f_a if pr_A is the input to pr_B.

The transport volume of f_a is eliminated when pr_A and pr_B are grouped as shown in Figure 8.

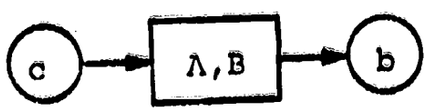


Figure 8. Grouping of pr_A and pr_B of Figure 7.

2. Group Processes which require the same file as shown in Figure 9.

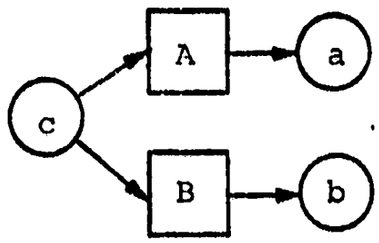


Figure 9. pr_A and pr_B read a common input file f_c.

The transport volume is reduced when pr_A and pr_B are grouped since F_c is read only once as shown in Figure 10.

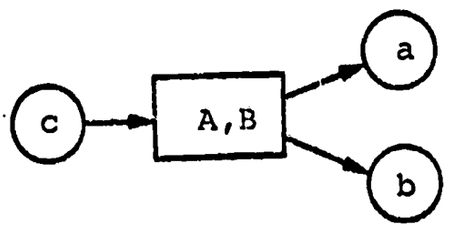


Figure 10. Grouping of pr_A and pr_B of Figure 9.

It may be profitable also to group f_a and f_b as shown in Figure 11.

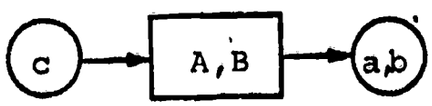
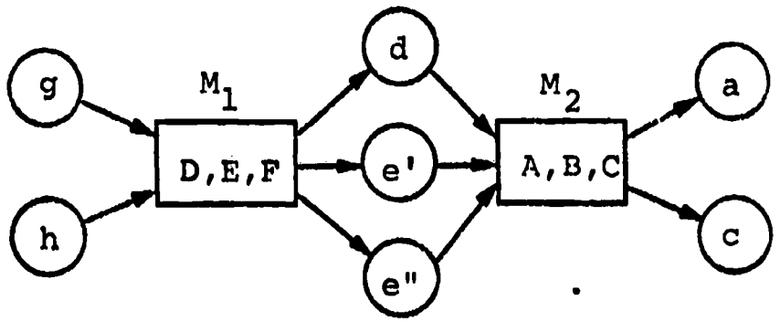


Figure 11. Grouping of f_a and f_b of Figure 10.

The objective is to reduce total transport volume and thus total processing time.

The concept of process grouping is illustrated with the example from section VII of the paper and the Incidence Matrix and graph for the example is shown in Figure 2. The transport volume TV for the example is 350 units, the core memory required is 30 units, and the maximum input/output requirement for any process is 3. The crucial items in determining which processes to group into modules are the transport volume and main memory size. It is desirable to produce modules (subject to the memory constraint) minimizing the transport volume for the programming system.

Consider as an alternative design combining pr_A, pr_B, and pr_C into one program module and pr_D, pr_E, and pr_F into a second program module. The resulting transport volume TV is 270, the core memory will be no larger than 55 units, and the maximum number of input/output processes for each program module has increased to 5. This alternative design is shown in Figure 12.



	a	c	d	e'	e"	g	h	ℓ_i	cm_i
M_1	0	0	-1	-1	-1	1	1	5	40
M_2	-1	-1	1	1	1	0	0	5	55
mp_j	1	1	2	2	2	1	1		
v_j	30	20	50	10	10	30	50		
tv_j	30	20	100	20	20	30	50		

Figure 12. Grouping Processes of Figure 2 into two Program Modules.

The transport volume for the example of Figure 2 has been reduced from 350 units to 270 units, the core requirement has increased from 30 units to 55 units for program module size and the maximum input/output requirement has increased from 3 to 5.

Several assumptions are made in the graphical representation of the processes and files. Control flow over a file is assumed to exist for any process. Multiple data passes over an input and output file as illustrated in Figure 13 are shown as follows in Figure 14 for the computation of transport volume.

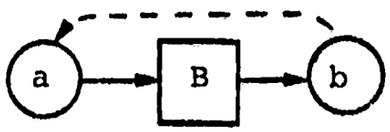


Figure 13. Multiple data passes over an input file and output file.

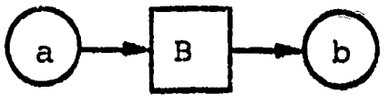


Figure 14. Graphical representation of the examples of Figure 13 for the purpose of computing transport volume.

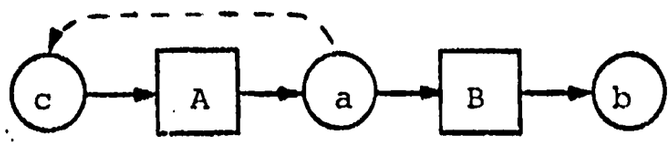
In other words, an element of file "a" is read and an element of file "b" is written. Control then reads the next element of "a" and writes the next element of "b". Therefore, cycles within a process are not shown in the Incidence Graph, but are assumed to exist in the Incidence Graph and are shown in the control flow graph.

In addition, the situation may exist in which control flow actually passes completely from file "a" to file "b". This is the case when the entire file "a" is read before file "b" is written. For example, the process may involve a sort on file "a", or a complete read of file "a" may be required to compute various sums that are dependent on the content of file "a".

Both cases are represented as having the same incidence matrix for purposes of computing the transport volume. The T or control flow matrix reflects occurrences of multiple data passes over a file.

The P matrix and E matrix are the same for both cases and the T matrix is different. This is illustrated as follows in Figure 15.

It can be noted that although the transport volume for specific files has been eliminated by the grouping, storage (in main memory) for the information contained in those files is still necessary. It is the purpose of reorganization to automatically restructure the combined processes towards the reduction of the number of loops over that information and thus possibly eliminate the need to maintain it



T

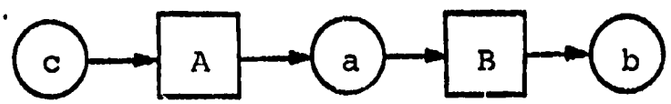
	a	b	c
a	0	1	1
b	0	0	0
c	1	0	0

P

	A	B
A	0	1
B	0	0

E

	a	b	c
A	-1	0	1
B	1	-1	0



T

	a	b	c
a	0	1	0
b	0	0	0
c	1	0	0

P

	A	B
A	0	1
B	0	0

E

	a	b	c
A	-1	0	1
B	1	-1	0

Figure 15. Illustration of the usefulness of the T Matrix.

between two data passes. This can be accomplished by restructuring the loops (consecutive data passes) in an attempt to provide only one loop over the original data file. Consequently, only a specific record of that information file may need to exist for each pass through the merged loop. That is, the information in that record could be computed and used by the same pass, and no longer be necessary upon completion of that pass. Such is the case in the example presented in Figure 2 and described in a later section in which the combination and reorganization of processes E and F allow for the elimination of the Customer Transaction File f".

A record of the Warehouse Transaction File g was used to compute each record in the Customer Transaction File f". Similarly, a record of f" was used to compute each record in the Customer Order File e' and the Customer Payment File e". By grouping and reorganizing two

processes, E and F, the individual records for e' and e" can be computed from a record of f" (which was just computed from a record of g). Consequently, only the storage for a record of f" is necessary in comparison to storage for the entire file. In addition, it may be possible to eliminate the record of f" and compute those for e' and e" directly from g. Such a procedure requires the following subtasks:

1. Combining processes into logically and syntactically correct modules.
2. Comprehensive control and data flow analysis.
3. Restructuring the module to merge data passes (or loops).
4. Elimination of unnecessary files (or data).

V. PROCESS GROUPING DETERMINATION

The problem of assigning the various processes to modules is a complex combinatorial problem complicated by the fact that savings from reorganization cannot be predetermined. However, prediction of savings in transport volume which result from process groupings can be accomplished.

The G Matrix relates the profitable binary groupings; i.e., those in which a savings in transport volume is incurred.

An interaction matrix is created to reflect the transport volume savings encountered in a binary grouping of processes. This savings matrix or S matrix is computed as follows:

$$S_{ij} = S_{ji} = \sum_{k=1}^{m} v_k \cdot \begin{bmatrix} 1 \text{ if } e_{ik} = 1 \text{ and } e_{jk} = 1. \\ 2 \text{ if } (e_{ik} = -1 \text{ and } e_{jk} = 1) \text{or} (e_{ik} = 1 \text{ and } e_{jk} = -1) \\ 0 \text{ otherwise.} \end{bmatrix}$$

Where k is the file subscript of the Incidence Matrix.

If the binary grouping is immediately infeasible, we maintain the potential savings we would encounter if the grouping is made at a later time.

The TV savings which result from a grouping of n processes is then reflected in the sum of the savings of binary groupings of the n processes. (NOTE: There are $\frac{n(n-1)}{2}$ binary combinations of processes.)

Thus, our S matrix is presented in Figure 16 for the example problem given in Figure 2.

S	A	B	C	D	E	F
A	-	20	0	100	0	0
B	20	-	0	0	20	0
C	0	0	-	0	20	0
D	100	0	0	-	0	20
E	0	20	20	0	-	40
F	0	0	0	20	40	-

Figure 16. Matrix of transport savings for pairwise groupings for the example of Figure 2.

TVS is the Transport Volume Savings function for any grouping M.

$$TVS(M) = \sum_{\substack{pr_i \in M,\ pr_j \in M \\ i<j}} S_{ij}$$

Thus, from above:

$$TVS\ (DEF) = S_{DE} + S_{EF} + S_{DF} = 0 + 40 + 20 = 60$$

$$TVS\ (ABD) = S_{AB} + S_{BD} + S_{AD} = 20 + 0 + 100 = 120$$

Note that this short-cut method for computing Transport Volume Savings (TVS) is based on the assumption that a file is input to no more than two processes. If a file is input to three or more processes, the Transport Volume Savings must be computed directly from the Incidence Graph of the proposed design.

Transport Volume Savings is supplemented by gains occurring as a result of reorganization.

The objective is to minimize transport volume subject to core constraints and feasible grouping consideration. This may be stated as follows:

If $C(M_i)$ is the core requirement for module $RG(M_i)$ and CT is the core constraint,

$$C(M_i) \leq CT\ ,\ 1 \leq i \leq q$$

It is important to note the interaction effects encountered are of a non-convex nature; i.e., there exist local optima which are not necessarily global optima; thus, to guarantee a global optimum, the entire solution space must be considered.

The grouping procedure was first formulated as a Quadratic Assignment Algorithm [5] in order to implicitly truncate unprofitable solutions from the solution space and facilitate speed of convergence to a good solution. However, optimality cannot be guaranteed for this problem as we formulated it using the Quadratic Assignment Algorithm. The process grouping procedure described in the next section is formulated as a straightforward enumeration scheme. The number of feasible designs is not too large for problems of 50 processes or less when memory and precedence relationships are used as constraints on Module size.

A. GENERATION OF FEASIBLE PROCESS GROUPINGS TO FORM MODULES

The G matrix is used to create a listing of binary groupings. The binary pairs (i,j) of the upper triangular portion of the matrix are selected if $g_{ij} = 1$ and it is not true that $\exists \ell \ni p_{i\ell}$ and $r_{\ell j} = 1$. The consequence of allowing the (i,j) pair into the pair list given the above condition is the generation of infeasible groupings. Consider the following precedence graph of processes as shown in Figure 17.

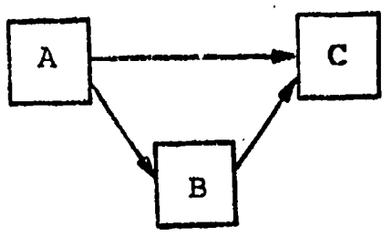


Figure 17. Precedence graph of processes.

We observe that $p_{AB} = 1$ and $r_{BC} = 1$ hold; thus, AC is eliminated from the pair list; otherwise {AC,B} appears to be a feasible design, which it is not. By eliminating the pair {AC} from the feasible list of pairs, we create only feasible designs. (NOTE: {ABC} is still a feasible grouping.)

Once a list of feasible pairs is created, feasible modules of size 3, ..., n, are generated by the following algorithm.

A list Y' of feasible groups of size k+1 is generated using the list Y of groups of size k by selecting an element of Y and comparing it with every element of the list of pairs and adding a process to the

grouping if that process is in a pair with an element of the group and no output of any process of the element of Y is input, or reachable as input, to the process to be added to the grouping. This means that if $y \in Y$ and $pr_i \notin y$ then $y \cup pr_i$ is a feasible grouping if $\exists\ pr_j \in y \ni g_{ij} = 1$ and for all $pr_k \in y$ if $\exists\ pr_\ell \notin y$ and $\ell \neq i$ then $p_{k\ell} = 1$ implies $r_{\ell i} \neq 1$ and $p_{\ell k} = 1$ implies $r_{i\ell} \neq 1$.

This is to say that a process is added to the grouping if there is no output from a process of the group which is indirect input to the candidate process; i.e., there is no process not in the grouping which accepts output from the group and generates input to the candidate process. An example is shown in Figure 18 of the problems that

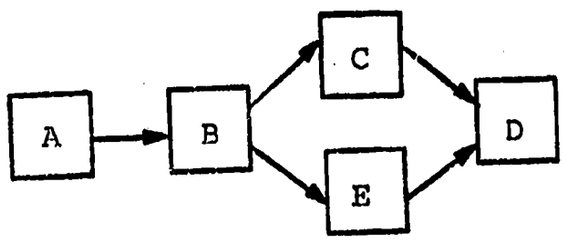


Figure 18. Illustration of Potential Grouping Problem.

can arise in the grouping procedure. If {A B C} is a grouping, then we see that the grouping {A B C D} violates our rules for acceptance of the candidate D for $p_{BE}=1$ and $r_{ED}=1$ and yet $E \notin \{A\ B\ C\}$ and $E \neq D$. Thus {A B C D} is not a feasible grouping. We can see, however, that {A B C E} and {A B C D E} are feasible groupings.

From the list of feasible pairs, all feasible triples are generated. Two pairs are grouped if they have one process in common. From valid pairs and triples, designs of size 4, ..., n, are generated.

This procedure generates modules from size 2 up to n where n is the number of elements of PR. The result is a list of all possible feasible process groupings M, with the exception of the individual processes. Thus, in the example we call this listing β, and

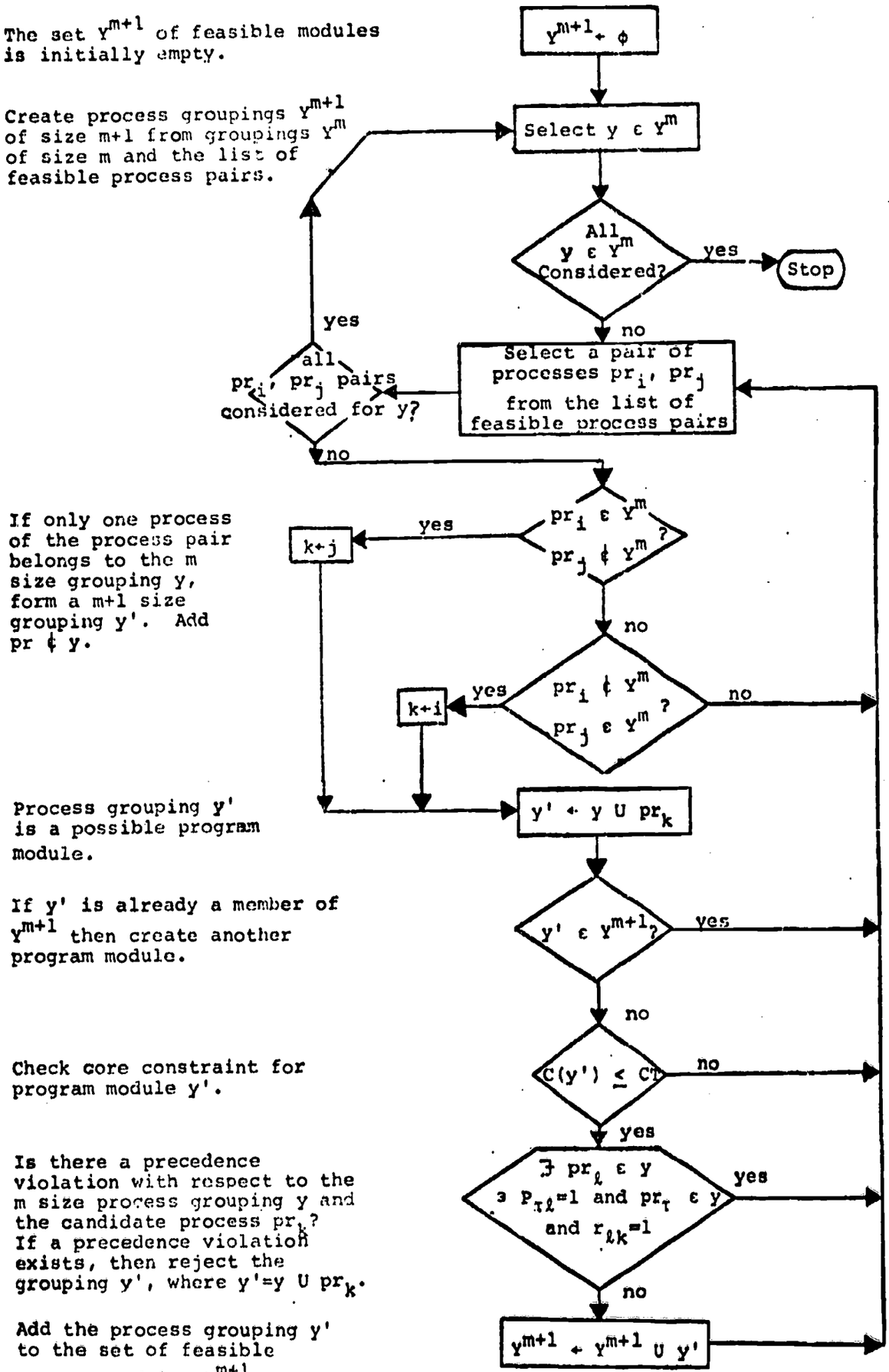


Figure 19. Generation of Feasible Process Groupings To Form Modules.

The set of feasible systems designs α^{m+1} is initially empty.

Create groupings θ^{m+1} of size m+1 from groupings θ^{m} of size m.

If candidate module b has a process in common with θ^{m} reject the candidate grouping $\theta^{m} \cup b$.

Module grouping θ^{m+1} is a possible programming systems design.

If θ^{m+1} is already a member of α^{m+1} then create another grouping of modules to form a new systems design.

Is there a precedence violation with respect to the m size module grouping θ^{m} and the candidate module b? If a precedence violation exists, then reject the grouping θ^{m+1}, where $\theta^{m+1} = \theta^{m} \cup b$.

Add the module grouping θ^{m+1} to the set of feasible systems designs α^{m+1}.

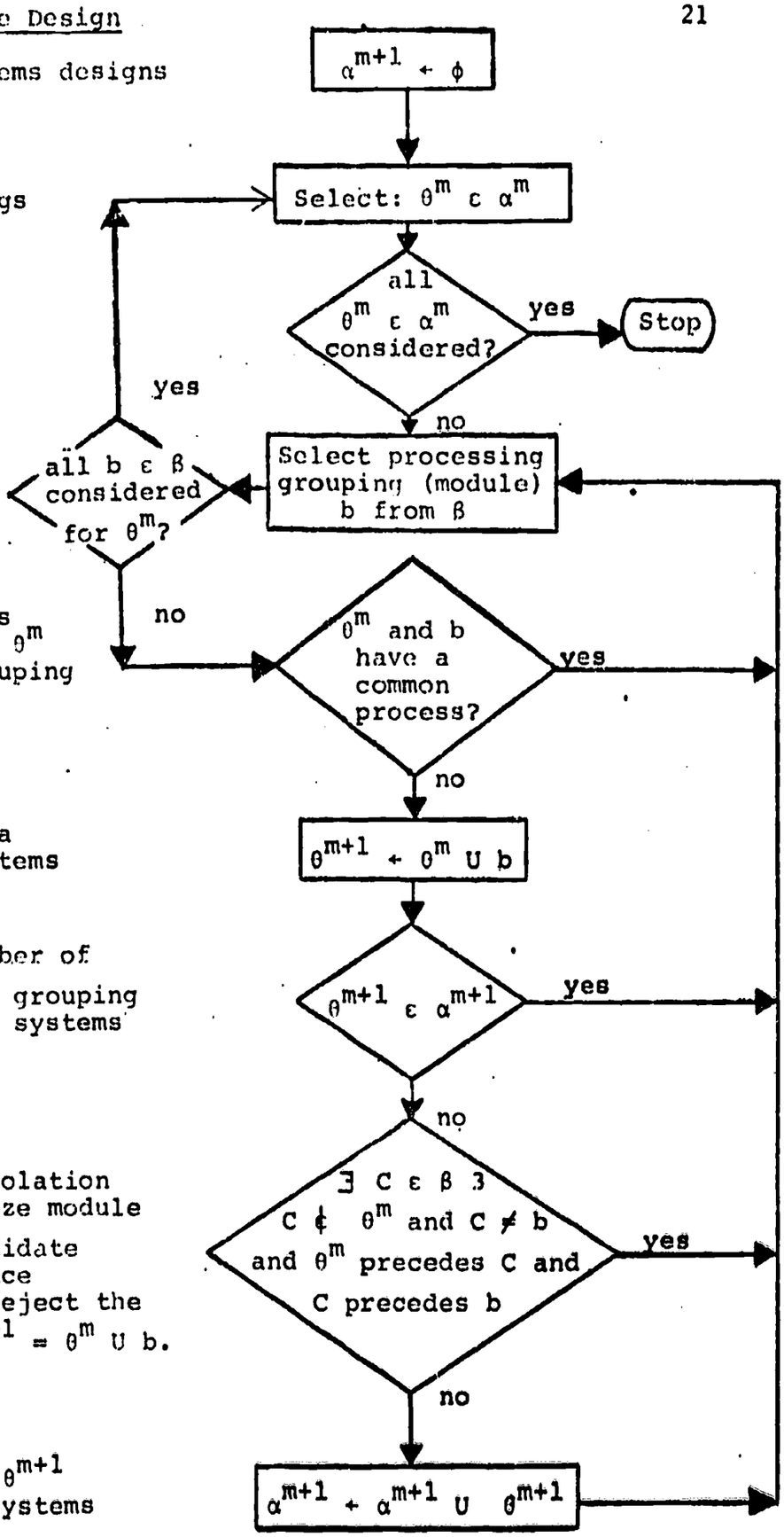


Figure 20. Generation of Alternative System Designs

$$\beta = \begin{bmatrix} \{AB\} \\ \{AD\} \\ \vdots \\ \{EF\} \\ \{ABC\} \\ \vdots \\ \{DEF\} \\ \{ABCD\} \\ \vdots \\ \{ABCDEF\} \end{bmatrix}$$

the list of all feasible groupings M is the union of the set β and the set PR or M = β U PR. See the flow chart of Figure 19.

B. GENERATION OF ALTERNATIVE SYSTEM DESIGNS

A similar procedure is used to select all feasible combinations of modules (process groupings) that form a cover for the programming system PS. Note that all elements of β satisfy the system's core constraint so modules may be combined freely so long as no modules in the cover have any common processes. The core constraint of a system is satisfied as long as the core requirement of the largest program module in the design does not exceed the constraint.

The procedure seeks to generate sets $\{\alpha^m\}$, m = 1, ..., s, where α^m is a set that contains all possible combinations of m modules and $s \leq n$ is the maximum number of modules that can be combined. Clearly, $\alpha^1 = \beta$. Proceeding by inductive definition, α^{m+1} is generated from α^m. To generate α^{m+1}, for each element $\theta^m \in \alpha^m$, θ^m is combined with any element $b \in \beta$ for which b has no process in common with the modules of θ^m. Then, $\theta^{m+1} = \theta^m \cup b$ becomes an element of α^{m+1}. This procedure is shown in Figure 20.

Further, as above, there must be only direct precedence between the set θ^m and the candidate module b; i.e., the case as illustrated in Figure 21 does not occur.

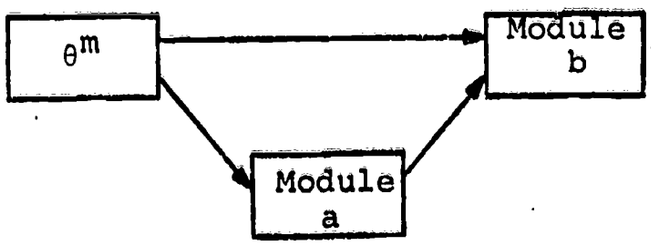


Figure 21. Illegal Grouping Situation.

Thus, for example, if

$\{BC,AD\} \in \alpha^2$ and $\{EF\} \in \beta$ and $\{BC,AD\} \cap \{EF\} = \phi$

Then $\{BC,AD,EF\} \in \alpha^{2+1} = \alpha^3$.

Further, if

$\{ABC\} \in \alpha^1$ and $\{EF\} \in \beta$ and $\{ABC\} \cap \{EF\} = \phi$

Then $\{ABC,EF\} \in \alpha^{1+1} = \alpha^2$

Note in the first case a cover of PS is formed.

After all combinations have been enumerated, a cover δ is formed from each combination $\theta^m \in \alpha^m$ as follows: We let $\theta^k \in \alpha^k = \{M_1, \ldots, M_k\}$, where $M_i \in \beta$ and feasible module $M_i = \{pr_{i_1}, \ldots, pr_{i_{n_i}}\}$. M_i contains n_i processes.

Then the residual processes can be defined as:

$$b' = \left\{ \{pr\} \mid pr \in \left[PR - \bigcup_{j=1}^{n_i} (M_i) \right] \right\}; \text{ thus } \delta = \theta^k \cup b'.$$

In other words, a feasible design is any element θ^m of any α^m combined with all ungrouped processes as individual modules. For example, $\delta = \{ABCD, E, F\}$ is a feasible design for the programming system diagrammed in Figure 2.

C. TRANSPORT VOLUME SAVINGS CALCULATION

The transport volume savings for any design can be calculated by examining each module of the design as stated earlier and the savings for the design is the sum of the savings for any modules. Thus, if $\delta = \{M_1, M_2, \ldots, M_q\}$, then the transport volume savings for δ is

$$TVS(\delta) = \sum_{i=1}^{q} \sum_{\substack{pr_j \in M_i,\ pr_k \in M_i \\ j<k}} s_{jk}$$

The optimal design with respect to transport volume savings can be designated as follows:

$$\delta_{opt} = \max_{\delta} \left(TVS(\delta) \right)$$

The core requirements for a design can be designated as the maximum requirements for any module of the design; thus, it is

$$\text{Maximum} \quad [\, C(M_1),\ C(M_2),\ \ldots,\ C(M_q)\,] \ .$$

The list of designs are then sorted in ascending order by core and descending order by transport volume savings and the x best designs (highest transport volume savings) are saved for each range of core; i.e., top ten for range 20K-30K bytes of memory, top ten for range 31K-40K bytes of memory, etc. The core constraint may be as a result of a partition size or an arbitrary constraint on module size. The number of designs (x) to save for each range of core memory is arbitrary.

VI. COMBINING PROCESSES

The combining of processes in L_0 into program modules in L_1 requires the translation of those processes into L_1. This translation can be made either before or after the processes are combined. However, if the translation is made first, then the procedures for combining program modules are over the same language as the reorganization procedures. This would enable new processes to be added to already reorganized modules. Procedures necessary for combining modules include those to resolve identifier conflicts, interface the modules with respect to external files, and perform structural modifications necessary to produce the desired syntactically and logically correct module.

Nylin [6] has discussed techniques which have been used for the implementation of these procedures. In addition, other techniques can be used to take advantage of any commonality which may exist between the modules being combined. Yershov [7,8] described techniques to efficiently utilize storage by allowing certain variables to use the same memory location. Similarly, algorithms exist for detecting common data storage areas and to eliminate redundant procedure definitions [6].

Much of the control flow information necessary for reorganization can be accomplished with existing techniques such as analysis using Boolean connectivity matrices first described by Prosser [9], or the interval method described by Cocke and Schwartz [10] and Allen [11].

Techniques for additional control flow information which can be used in reorganization are described by Nylin [6]. Existing techniques [12, 13, 14] can be used to compute variable "usage" information utilizing the data gathered in the control flow analysis.

The control flow and data flow information procedures are necessary to locate loops (data passes) which may be candidates to be merged. Of particular interest are loop pairs in which one loop is always executed the same number of times as a specific branch of the other. Thus, one loop computes the control parameters for the other. If this can be determined and if additional data flow information allows the loops to be merged, one of the loops can be eliminated and its body moved into the other loop. This procedure may include the replacement of an induction variable as well as the redefinition of certain program variables necessary as a result of merging the loops. It should be noted that the complexity of the control flow within either loop is not a factor in the ability to merge them.

Once the loops are merged, it may be possible to apply subsumption techniques [6,13] to eliminate unnecessary stores into variables. That is, the definition of a particular variable (which could be the internal representation of a data file) may be able to replace subsequent occurrences of that variable. Thus, due to the merging of two loops (data passes), it is no longer necessary to maintain information which is only utilized by a specific pass through the merged loop. Hence, an intermediate file (used for communication between the loops) can be eliminated.

Once a decision is made as to which processes are to form a module, they must be integrated by some automatic procedure. This combining of specific processes to form a program module that can be executed on the target machine involves several steps. First, the processes must be able to be represented in an intermediate language that has the following attributes:

1. An ability to measure the memory required to implement the processes on the target machine.
2. The ability to automatically analyze the grouped processes and perform reorganization procedures on their loop and file structure.
3. The ability to map the grouped processes into the desired programs on the target machine preserving their reorganized structure.

It should be noted that the intermediate language (Problem Statement Language) could be either the original language or the desired language for the target machine. It may even be desirable to compile the final grouping in the intermediate language to produce object code directly. This could be especially advantageous if the processes are described by a high-level problem statement language. That is, there exists no need for any other level of documentation since new modifications to the system would be made at the problem statement level. Thus, when changes are made to the existing processes or new processes are added, the final modular programming system can be automatically regenerated.

Clearly, this is necessary to guarantee that the system remains optimal and that no errors are introduced by adding code to a module consisting of multiple processes.

Once the processes are represented in the intermediate language L_1, they must be combined into a logically and syntactically correct program module for L_1. The program modules (processes represented in L_1) can be automatically combined to form larger modules in L_1. These multipass modules could be automatically analyzed; and, if possible, reorganized to combine multiple passes over the same file. In addition, in some cases the file could also be eliminated. The elimination of such files not only increases the efficiency of the resultant module but it decreases the memory it requires [6].

In addition to the directed graph representation of the set of programs, the following information is assumed to be available for module reorganization.

1. Process documentation
2. Source deck or list of processes
3. Operating instructions
4. File usage
5. Input and output test data
6. Frequency of process cycles

VII. EXAMPLE

The example below is a system of processes which creates a warehouse shipping schedule.

The input is considered to be a transaction file containing Receiving Reports, Customer Orders, and Customer Payments. The transactions are divided into a receiving file and a customer transaction file.

The receiving reports are used to update the inventory on hand file, while the customer orders and payments are separated and a payment summary produced.

The Incidence Matrix for this example is shown in Figure 2.

PROCESSES

A - Shipping Schedule Generator
B - Order file sorting for Scheduler
C - Customer Payment Summary Generator
D - Inventory Update
E - Separate Customer Payments from Customer Orders
F - Separate Receiving Report from Customer Transactions

FILES

a - Shipping Schedule Report
b - Customer orders sorted by item
c - Customer Payment Summary
d - Updated Inventory
e' - Customer Orders
e" - Customer Payments
f' - Receiving Report
f" - Customer Transaction
g - Warehouse Transaction
h - Old Inventory Master

The files e', e", and f' are described below:

RECEIVING REPORT (f')

Columns	Data
1 - 2	'RV'
3 - 7	Vendor Number
8 - 27	Vendor Name
28 - 47	Vendor Address
48 - 55	Value of Goods
56 - 60	Component Number
61 - 65	Quantity Received
66 - 71	Date Received
72 - 77	Blank
78 - 79	Warehouse

CUSTOMER ORDER (e')

Columns	Data
1 - 2	'CØ'
3 - 7	Customer Number
8 - 27	Customer Name
28 - 47	Customer Address
48 - 55	Value of Goods
56 - 60	Component Number
61 - 65	Quantity Received
66 - 71	Delivery Date
72 - 77	Order Number

CUSTOMER PAYMENT (e")

Columns	Data
1 - 2	'CP'
3 - 7	Customer Number
8 - 27	Customer Name
28 - 47	Customer Address
48 - 55	Amount Paid
56 - 71	Blank
72 - 77	Order Number

The P, R*, R, and G Matrices for the above example are given in Figures 3,4,5, and 6 respectively.

The transport volume savings Matrix S is given in Figure 16; thus, the total transport volume for this example is 350 units. The procedure detailed in Figure 18 was executed for CT=50 with the resultant module groupings:

$$M_1 = \{A,D\}$$

$$M_2 = \{B,C,E,F\}$$

$$C(M_1) = 50 \leq 50$$

$$C(M_2) = 45 \leq 50$$

With the organizing completed, the final transport volume was 170. This resulted in a savings of 180 units and only 40 units more than the absolute minimum of 130 units. If the core constraint is relaxed, the minimum transport volume is obtained when all processes are grouped into a single module.

To illustrate combining processes into modules utilizing reorganization techniques, consider processes D, E, and F. Modules representing each of these processes can be represented by the following COBOL procedure divisions.

PROCESS D

```
     OPEN INPUT OLD-INVENTORY-FILE, RECEIVING-REPORT-FILE,
          OUTPUT UPDATE-INVENTORY-FILE.
     REWIND RECEIVING-REPORT-FILE.
LABEL. READ RECEIVING-REPORT-FILE  AT END GO TO CLOSER.
     PERFORM UPDATE INVENTORY-FILE.
     GO TO LABEL.
CLOSER.  CLOSE ALL FILES.
```

PROCESS E

```
     OPEN INPUT CUSTOMER-TRANSACTION-FILE, OUTPUT CUSTOMER-PAYMENT-
          FILE, CUSTOMER-ORDER-FILE.
     REWIND CUSTOMER-TRANSACTION-FILE  AT END GO TO CLOSER.
          IF CODE OF CUSTOMER-TRANSACTION EQUAL 'P' THEN WRITE CUSTOMER-
          PAYMENT-REC FROM CUSTOMER-TRANSACTION ELSE WRITE CUSTOMER-
          ORDER-REC FROM CUSTOMER-TRANSACTION.
          GO TO LABEL.
     CLOSER.  CLOSE ALL FILES.
```

PROCESS F

```
     OPEN INPUT WAREHOUSE-TRANSACTION-FILE, OUTPUT RECEIVING-REPORT-
          FILE, CUSTOMER-TRANSACTION-FILE
     REWIND WAREHOUSE-TRANSACTION-FILE
LABEL.  READ WAREHOUSE-TRANSACTION-FILE  AT END GO TO CLOSER.
     IF CODE OF WAREHOUSE-TRANSACTION EQUAL 'R' THEN WRITE RECEIVING-
     REPORT-REC FROM WAREHOUSE-TRANSACTION ELSE WRITE CUSTOMER-
     TRANSACTION-REC FROM WAREHOUSE-TRANSACTION.  GO TO LABEL.
CLOSER.  CLOSE ALL FILES.
```

By combining and reorganizing Processes E and F into one module, the following integrated module is generated.

MODULE E-F

```
     OPEN INPUT WAREHOUSE-TRANSACTION-FILE, OUTPUT RECEIVING-REPORT-
          FILE, CUSTOMER-ORDER-FILE, CUSTOMER-PAYMENT-FILE.
     REWIND WAREHOUSE-TRANSACTION-FILE.
LABEL.  READ WAREHOUSE-TRANSACTION-FILE  AT END GO TO CLOSER.
     IF CODE OF WAREHOUSE-TRANSACTION EQUAL 'R' THEN WRITE RECEIVING-
     REPORT-REC FROM WAREHOUSE-TRANSACTION ELSE IF CODE OF WAREHOUSE-
     TRANSACTION EQUAL 'P' THEN WRITE CUSTOMER-PAYMENT-REC FROM
     WAREHOUSE-TRANSACTION.
     WRITE CUSTOMER-ORDER-REC FROM WAREHOUSE-TRANSACTION.
     GO TO LABEL.
CLOSER.  CLOSE ALL FILES.
```

Thus, the processes are able to be combined with the elimination of the Customer Transaction File (file f").

Similarly, Processes D and F can be combined and reorganized to eliminate the Receiving Report File (file f'). The resultant module is as follows:

MODULE D_F

```
     OPEN INPUT WAREHOUSE-TRANSACTION-FILE, OLD-INVENTORY-FILE,
          OUTPUT CUSTOMER-TRANSACTION-FILE, OLD-INVENTORY-FILE.
     REWIND WAREHOUSE-TRANSACTION-FILE.
LABEL.  READ WAREHOUSE-TRANSACTION-FILE  AT END GO TO CLOSER.
     IF CODE OF WAREHOUSE-TRANSACTION EQUAL 'R' THEN UPDATE
     INVENTORY-FILE ELSE WRITE CUSTOMER-TRANSACTION-REC FROM
     WAREHOUSE-TRANSACTION.  GO TO LABEL.
CLOSER.  CLOSE ALL FILES.
```

The ability to combine Processes D and F (eliminating file f') and E and F (eliminating file f") does not guarantee that both files can be eliminated by grouping Processes D, E, and F. That is, certain

program variable dependencies existing between Processes D and E may prohibit the reorganization of the total grouping. However, if such dependencies do not exist, then Processes D, E, and F may be combined and reorganized to produce the resultant module.

MODULE D_E_F

```
     OPEN INPUT WAREHOUSE-TRANSACTION-FILE, OLD-INVENTORY-FILE,
          OUTPUT OLD-INVENTORY-FILE, CUSTOMER-ORDER-FILE, CUSTOMER-
          PAYMENT-FILE.
     REWIND WAREHOUSE-TRANSACTION-FILE.
LABEL. READ WAREHOUSE-TRANSACTION-FILE  AT END GO TO CLOSER.
     IF CODE OF WAREHOUSE-TRANSACTION EQUAL 'R' THEN UPDATE-INVENTORY-
          FILE ELSE
     IF CODE OF WAREHOUSE-TRANSACTION EQUAL 'P' THEN WRITE CUSTOMER-
          PAYMENT-REC FROM WAREHOUSE-TRANSACTION ELSE
     WRITE CUSTOMER-ORDER-REC FROM WAREHOUSE-TRANSACTION.  GO TO LABEL.
CLOSER.  CLOSE ALL FILES.
```

VIII. CONCLUSIONS

A methodology is described for the automatic design of a processing system initially defined in terms of logical processes or program modules. Processes and files are grouped and reorganized in such a way as to produce an optimal design with respect to a specific target machine. Performance criteria for the optimal design is defined in terms of transport volume savings and core memory requirements.

Starting with a graph theoretic representation of the interaction between processes (or modules) and files, the methodology consists of two components: (1) a generator of feasible alternatives and (2) a procedure for reorganization and code generation for specific groupings. The generator for the feasible alternatives uses an implicit enumeration algorithm to optimize process groupings in an efficient manner. The objective is to group processes into modules which minimize the interaction between modules while still satisfying the logical requirements of the program and the physical constraints of the hardware. Finally, after the program modules have been specified, program and file reorganization will be performed to further optimize the design. Reorganization includes the combination of similar data passes on the same file to minimize transport volume and the merging of loops to enable elimination of code and of intermediate data files.

The code generator will then accept the optimal program design and produce an optimized source language program for the target machine. Consequently, not only can an optimal design for the processing system be generated; but due to reorganization techniques, the resultant modules (defined from specific process groupings) may approach the computational efficiency expected of an integrated program.

Although an automatic reorganizer has not been developed for COBOL, one has been implemented for Pilot (a subset of Neliac) on a C.D.C.6500 at Purdue University. This language (Pilot) could represent the intermediate language L_1 into which processes written in COBOL could be translated before they are combined and reorganized.

Another way in which this methodology could be used is to select designs that are optimal with respect to a particular pricing scheme. For example, the program design which may be executed the most efficiently

(with respect to transport volume) on a specific configuration could require main memory that would be disadvantageous to the user according to a particular pricing scheme that penalizes the user for larger memory requirements. By generating designs for various memory constraints, such alternative designs are available.

The methodology described in this paper could be used to break up programs into modules or overlays and adds a new dimension to program scheduling since we can now address the following question: "What is the optimal size of a program module?"

ACKNOWLEDGMENT

This work was supported, in part, by Grant Number GJ31572 from the Office of Computing Activities of the National Science Foundation and, in part, by Professor Daniel Teichroew, Director of the ISDOS Project, University of Michigan.

REFERENCES

1. Nunamaker, J.F., Jr. On the Design and Optimization of Information Processing Systems, Ph.D. Dissertation, Case Institute of Technology, March 1969.

2. Nunamaker, J.F., Jr. A Methodology for the Design and Optimization of Information Processing Systems, AFIPS Proceedings, Spring Joint Computer Conference, Volume 38, May 1971.

3. Langefors, Borge. Information System Design Computations Using Generalized Matrix Algebra, BIT, 5,2, 1965.

4. Briggs, R.B. A Mathematical Model for the Design of Information Management Systems, M.S. thesis, University of Pittsburgh, 1966.

5. Graves, Glenn and A. Whinston, "An Algorithm for the Quadratic Assignment Problem, Management Science, Vol. 17, No. 7, March 1970.

6. Nylin, W.C., Jr. Structural Reorganization of Multipass Computer Programs, Ph.D. Dissertation, Purdue University, June 1972.

7. Yershov, A.P. "ALPHA--An Automatic Programming System of High Efficiency," JACM, 13, January 1966, p. 17.

8. Yershov, A.P. The ALPHA Automatic Programming System, New York: Academic Press, 1971.

9. Prosser, R.T. "Application of Boolean Matrices to the Analysis of Flow Diagrams," Proceedings of the Eastern Joint Computer Conference, 1959, p. 133.

10. Cocke, John and J.T. Schwartz. Programming Languages and Their Compilers, 2nd Revised Version, Courant Institute of Mathematical Sciences, New York University, 1969.

11. Allen, Frances E. "Control Flow Analysis," ACM SIGPLAN Notices, 5, July, 1970, p. 1.

12. Allen, Frances E. "Program Optimization," Annual Review in Automatic Programming, Vol. V, 1965, p. 239.

13. Lowery, Edward S. and C.W. Medlock. "Object Code Optimization," CACM, 12, January 1969, p. 13.

14. Mendicino, Samuel F., et.al. "The LRLTRAN Compiler," CACM, 11, November 1968, p. 747.

The following is a listing of Institute Papers which are still in supply. Copies may be obtained from the Secretary of the Institute Paper and Reprint Series, Krannert Graduate School of Industrial Administration, Purdue University, West Lafayette, Indiana 47907.

(When requesting copies, please specify paper number.)

Paper No.	Title and Author(s)
83	A CLASS OF UTILITY FUNCTIONS ADMITTING TYRNI'S HOMOGENEOUS SAVING FUNCTION, Peter Jason Kalman.
101	CLASSIFICATION OF INVESTMENT SECURITIES USING MULTIPLE DISCRIMANANT ANALYSIS, Keith V. Smith.
123	A NOTE ON KONDRATIEFF CYCLES IN PREWAR JAPAN, Charles R. Keen.
138	BOREDOM VS. COGNITIVE REAPPRAISAL IN THE DEVELOPMENT OF COOPERATIVE STRATEGY, Marc Pilisuk, Paul Skolnick, Kenneth Thomas and Reuben Chapman.
144	ON IMPLICATIONS OF PRODUCTIVITY COEFFICIENTS AND EMPIRICAL RATIOS, Harry Schimmler.
147	DEPTH, CENTRALITY AND TOLERANCE IN COGNITIVE CONSISTENCY, Marc Pilisuk.
148	THE GENERAL INCONGRUITY ADAPTATION LEVEL (GIAL) HYPOTHESIS--II. INCONGRUITY MOTIVATION TO AFFECT, COGNITION, AND ACTIVATION-AROUSAL THEORY, Michael J. Driver and Siegfried Streufert.
150	PORTFOLIO REVISION, Keith V. Smith.
154	HEROES AND HOPELESSNESS IN A TOTAL INSTITUTION: ANOMIE THEORY APPLIED TO A COLLECTIVE DISTURBANCE, Robert Perrucci.
155	REGIONAL ALLOCATION OF INVESTMENT: A FURTHER ANALYSIS, Akira Takayama.
158	TWO CLASSICAL MONETARY MODELS, Cliff Lloyd.
161	THE PURCHASING POWER PARITY THEORY: IN DEFENSE OF GUSTAV CASSEL AS A MODERN THEORIST, James M. Holmes.
162	HOW CHARLIE ESTIMATES RUN-TIME, John M. Dutton and William H. Starbuck.
163	PER CAPITAL CONSUMPTION AND GROWTH: A FURTHER ANALYSIS, Akira Takayama.
164	THE PROBABILITY OF A CYCLICAL MAJORITY, Frank De Meyer and Charles R. Plott.

Paper No.	Title and Author(s)
166	THE CLASSROOM ECONOMY: RULES, RESULTS, REFLECTIONS, John A. Carlson.
167	AN ACTIVITY MODEL OF THE FIRM UNDER RISK, Carl R. Adams.
169	TAXES AND SHARE VALUATION IN COMPETITIVE MARKETS, Vernon L. Smith.
171	PROGRAMMING, PARETO OPTIMUM AND THE EXISTENCE OF COMPETITIVE EQUILIBRIA, Akira Takayama and Mohamed El-Hodiri.
178	ON THE STRUCTURE OF OPTIMAL GROWTH PROBLEM, Akira Takayama.
180	A NEW APPROACH TO DISCRETE MATHEMATICAL PROGRAMMING, G. W. Graves and Andrew B. Whinston.
181	EXPERIMENTING WITH THE ARMS RACE, Marc Pilisuk and Paul Skolnick.
186	REGIONAL ALLOCATION OF INVESTMENT: CORREGENDUM, Akira Takayama.
187	A SUGGESTED NEW MONETARY SYSTEM: THE GOLD VALUE STANDARD, Robert V. Horton.
193	MULTI-COMMODITY NETWORK FLOWS WITH MULTIPLE SOURCES AND SINKS, B. Rothchild and Andrew Whinston.
198	OPTIMAL DISPOSAL POLICIES, Carl Adams.
202	SOME FORMULAS ENCOUNTERED IN THE DEDUCTIVE ANALYSIS OF THIRD-ORDER AUTOGRESSION PROCESS, R. L. Basmann and R. J. Rohr.
215	A CONVERGENT PARETO-SATISFACTORY NON-TATONNEMENT ADJUSTMENT PROCESS FOR A CLASS OF UNSELFISH EXCHANGE ENVIRONMENTS, John O. Ledyard.
217	ON A "CONCAVE" CONTRACT CURVE, Akira Takayama.
218	THE EFFECTS OF FISCAL AND MONETARY POLICIES UNDER FLEXIBLE AND FIXED EXCHANGE RATES, Akira Takayama.
219	A MATCHING THEOREM FOR GRAPHS, D. Kleitman, A. Martin-Lof, B. Rothchild and A. Whinston.
224	GENERALIZED OPINION LEADERSHIP IN CONSUMER PRODUCTS: SOME PRELIMINARY FINDINGS, Charles W. King and John O. Summers.
226	THE FIRM AS AN AUTOMATION - I., Edward Ames.
227	SECOND-BEST SOLUTIONS, PEAK-LOADS AND MARGINAL COST PRICE POLICIES FOR PUBLIC UTILITIES, Robert A. Meyer, Jr.
228	EQUIPMENT REPLACEMENT UNDER UNCERTAINTY, Robert A. Meyer, Jr.

Paper No.	Title and Author(s)
233	ECONOMIC EFFECTS OF UNIFORM CONSUMER CREDIT CODE: A COMMENT, David C. Ewert.
234	OPTIMAL ADVERTISING EXPENDITURE IMPLICATIONS OF A SIMULTANEOUS-EQUATION REGRESSION ANALYSIS, Leonard J. Parsons and Frank M. Bass.
237	OPPOSITION OF PREFERENCES AND THE THEORY OF PUBLIC GOODS, Robert A. Meyer, Jr.
238	THE TAXATION OF RESTRICTED STOCK COMPENSATION PLANS, G. W. Hettenhouse and Wilbur G. Lewellen.
239	DECOMPOSABIE REGRESSION MODELS IN THE ANALYSIS OF MARKET POTENTIALS, Frank M. Bass.
241	OPPORTUNITY COSTS AND MODELS OF SCHOOLING IN THE NINETEENTH CENTURY, Lewis Solmon.
242	ESTIMATING FREQUENCY FUNCTIONS FROM LIMITED DATA, Keith C. Brown.
246	ON OPTIMAL CAPITAL ACCUMULATION IN THE PASINETTI MODEL OF GROWTH, S. C. Hu.
250	MONEY, INTEREST AND POLICY, P. H. Hendershott and George Horwich.
251	ON THE PEAK-LOAD PROBLEM, Akira Takayama.
253	A NOTE ON TECHNICAL PROGRESS, INVESTMENT, AND OPTIMAL GROWTH, Sheng Cheng Hu.
254	MANUFACTURERS' SALES AND INVENTORY ANTICIPATIONS: THE OBE COMPUTATIONAL PROCEDURES, John A. Carlson.
256	TWO ALGORITHMS FOR INTEGER OPTIMIZATION, Edna Loehman, Tuan Ph. Nghiem and Andrew Whinston.
260	AGE-DEPENDENT UTILITY IN THE LIFETIME ALLOCATION PROBLEM, Kenneth Avio.
261	AFFECTIVE AND VALUATIONAL CONSEQUENCES OF SELF-PERCEIVED UNIQUENESS DEPRIVATION: I. HYPOTHESES AND METHODOLOGICAL PRESCRIPTIONS, Howard Fromkin.
262	AFFECTIVE AND VALUATIONAL CONSEQUENCES OF SELF-PERCEIVED UNIQUENESS DEPRIVATION: II. EXPERIMENTALLY AROUSED FEELINGS OF SELF PERCEIVED SIMILARITY AS AN UNDESIRABLE AFFECTIVE STATE, Howard Fromkin.

Paper No.	Title and Author(s)
263	AFFECTIVE AND VALUATIONAL CONSEQUENCES OF SELF-PERCEIVED UNIQUENESS DEPRIVATION: III. THE EFFECTS OF EXPERIMENTALLY AROUSED FEELINGS OF SELF PERCEIVED SIMILARITY UPON VALUATION OF UNAVAILABLE AND NOVEL EXPERIENCES, Howard Fromkin.
264	AIR POLLUTION AND HOUSING: SOME FINDINGS, Robert J. Anderson, Jr., and Thomas D. Crocker.
265	APPLICATION OF REGRESSION MODELS IN MARKETING: TESTING VERSUS FORECASTING, Frank M. Bass.
267	A LINEAR PROGRAMMING APPROACH TO AIRPORT CONGESTION, Donald W. Kiefer.
268	ON PARETO OPTIMA AND COMPETITIVE EQUILIBRIA, PART I. RELATIONSHIP AMONG EQUILIBRIA AND OPTIMA, James C. Moore.
269	ON PARETO OPTIMA AND COMPETITIVE EQUILIBRIA, PART II. THE EXISTENCE OF EQUILIBRIA AND OPTIMA, James C. Moore.
271	A COMPARISON OF THREE MULTI-PRODUCT, MULTI-FACILITY BATCH SCHEDULING HEURISTICS, David R. Denzler.
272	A REPRESENTATION OF INTEGER POINTS IN POLYHEDRAL CONE, Ph. Tuan Nghiem.
273	LINE OF BUSINESS REPORTING - A METHODOLOGY FOR ESTIMATING BENEFITS, Russell M. Barefield.
274	MARKETING APPLICATIONS OF SELF-DESIGNATED OCCUPATION SKILL VARIABLES, E. A. Pessemier and D. J. Tigert.
275	THE FULL-EMPLOYMENT INTEREST RATE AND THE NEUTRALIZED MONEY STOCK, Patric H. Hendershott.
276	SOME APPLICATIONS OF THE CHANGE OF BASE TECHNIQUE IN INTEGER PROGRAMMING, Ph. Tuan Nghiem.
277	A WELFARE FUNCTION USING "RELATIVE INTENSITY" OF PREFERENCE, Frank DeMeyer and Charles R. Plott.
279	RACE AND COMPETENCE AS DETERMINANTS OF ACCEPTANCE OF NEWCOMERS IN SUCCESS AND FAILURE WORK GROUPS, Howard L. Fromkin, Richard J. Klimoski, and Michael F. Flanagan.
280	LEADERSHIP, POWER AND INFLUENCE, Donald C. King and Bernard B. Bass.
281	RECENT RESULTS IN THE THEORY OF VOTING, Charles R. Plott.
282	DISAGGREGATION OF ANALYSIS OF VARIANCE FOR PAIRED COMPARISONS: AN APPLICATION TO A MARKETING EXPERIMENT, E. A. Pessemier and R. D. Teach.

Paper No.	Title and Author(s)
283	MARKET RESPONSE TO INNOVATION, FURTHER APPLICATIONS OF THE BASS NEW PRODUCT GROWTH MODEL, John V. Nevers.
284	PROFESSIONALISM, UNIONISM, AND COLLECTIVE NEGOTIATION: TEACHER NEGOTIATIONS EXPERIENCE IN CALIFORNIA, James A. Craft.
285	A FREQUENCY DOMAIN TEST OF THE DISTURBANCE TERM IN LINEAR REGRESSION MODELS, Thomas F. Cargill and Robert A. Meyer.
286	EVALUATING ALTERNATIVE PROPOSALS AND SOURCES OF NEW INFORMATION, Edgar A. Pessemier.
287	A MULTIVARIATE REGRESSION ANALYSIS OF THE RESPONSES OF COMPETING BRANDS TO ADVERTISING, Frank M. Bass and Neil E. Beckwith.
288	ASSESSING REGULATORY ALTERNATIVES FOR THE NATURAL GAS PRODUCING INDUSTRY, Keith C. Brown.
289	TESTING AN ADAPTIVE INVENTORY CONTROL MODEL, D. Clay Whybark.
291	THE LABOR ASSIGNMENT DECISION: AN APPLICATION OF WORK FLOW STRUCTURE INFORMATION, William K. Holstein and William L. Berry.
294	AN EFFICIENT BRANCH AND FOUND ALGORITHM FOR THE WAREHOUSE LOCATION PROBLEM, Basheer M. Khumawala.
295	THE INTERACTION OF GROUP SIZE AND TASK STRUCTURE IN AN INDUSTRIAL ORGANIZATION, Robert C. Cummins and Donald C. King.
296	PROJECT AND PROGRAM DECISIONS IN RESEARCH AND DEVELOPMENT, Edgar A. Pessemier and Norman R. Baker.
298	SEGMENTING CONSUMER MARKETS WITH ACTIVITY AND ATTITUDE MEASURES, Thomas Hustad and Edgar Pessemier.
299	R & D MANAGERS' CHOICES OF DEVELOPMENT POLICIES IN SIMULATED R & D ENVIRONMENTS, Herbert Moskowitz.
300	DILUTION AND COUNTER-DILUTION IN REPORTING FOR DEFERRED EQUITY, Charles A. Tritschler.
301	A METHODOLOGY FOR THE DESIGN AND OPTIMIZATION OF INFORMATION PROCESSING SYSTEMS, J. F. Nunamaker, Jr.
303	ON PRODUCTION FUNCTIONS AND ELASTICITY OF SUBSTITUTION, K. R. Kadiyala.
304	AN EXPERIMENTAL INVESTIGATION OF DECISION MAKING IN A SIMULATED RESEARCH AND DEVELOPMENT ENVIRONMENT, Herbert Moskowitz.

Paper No.	Title and Author(s)
305	A NOTE ON MONEY AND GROWTH, Akira Takayama.
307	AN EXPERIMENTAL STUDY OF RELATIONSHIPS BETWEEN ATTITUDES, BRAND PREFERENCE AND CHOICE, Frank M. Bass, Edgar A. Pessemier, and Donald R. Lehmann.
309	WAGES AND HOURS AS SIGNIFICANT ISSUES IN COLLECTIVE BARGAINING, Paul V. Johnson.
311	AN EFFICIENT HEURISTIC ALGORITHM FOR THE WAREHOUSE LOCATION PROBLEM, Basheer M. Khumawala.
312	REACTIONS TO LEADERSHIP STYLE AS A FUNCTION OF PERSONALITY VARIABLES, M. H. Rucker and D. C. King.
313	FIRE FIGHTER STRATEGY IN WAGE NEGOTIATIONS, James A. Craft.
314	TESTING DISTRIBUTED LAG MODELS OF ADVERTISING EFFECT - AN ANALYSIS OF DIETARY WEIGHT CONTROL PRODUCT DATA, Frank M. Bass and Darrall G. Clarke.
316	AN EMPIRICAL INVESTIGATION OF THE RELIABILITY AND STABILITY OF SELECTED ACTIVITY AND ATTITUDE MEASURES, Edgar Pessemier and Albert Bruno.
317	BEHAVIOR OF THE FIRM UNDER REGULATORY CONSTRAINT: CLARIFICATIONS, Mohamed El-Hodiri and Akira Takayama.
320	DEPRECIATION POLICY AND THE BEHAVIOR OF CORPORATE PROFITS, Russell M. Barefield and Eugene E. Comiskey.
321	LABORATORY RESEARCH AND THE ORGANIZATION: GENERALIZING FROM LAB TO LIFE, Howard L. Fromkin and Thomas M. Ostrom.
322	LOT SIZING PROCEDURES FOR REQUIREMENTS PLANNING SYSTEMS: A FRAMEWORK FOR ANALYSIS, William L. Berry.
326	PRIORITY SCHEDULING AND INVENTORY CONTROL IN JOB LOT MANUFACTURING SYSTEMS, William L. Berry.
328	THE EXPECTED RATE OF INFIATION BEFORE AND AFTER 1966: A CRITIQUE OF THE ANDERSEN-CARLSON EQUATION, Patric H. Hendershott.
330	A FURTHER PROBLEM IN LEAD-LAG DETECTION, Robert A. Meyer, Jr.
332	THE SMOOTHING HYPOTHESIS: AN ALTERNATIVE TEST, Russell M. Barefield and Eugene E. Comiskey.
333	CONSERVATISM IN GROUP INFORMATION PROCESSING BEHAVIOR UNDER VARYING MANAGEMENT INFORMATION SYSTEMS, Herbert Moskowitz.

Paper No.	Title and Author(s)
334	PRIMACY EFFECTS IN INFORMATION PROCESSING BEHAVIOR - THE INDIVIDUAL VERSUS THE GROUP, Herbert Moskowitz.
336	VEHICLE ROUTING FROM CENTRAL FACILITIES. Brian F. O'Neil and D. Clay Whybark.
339	UNEXPLAINED VARIANCE IN STUDIES OF CONSUMER BEHAVIOR, Frank M. Bass.
340	THE PRODUCTION FUNCTION AS A MODEL OF THE REQUIREMENTS OF THE INFANTRY SERGEANT'S ROLE, Richard C. Roistacher and John J. Sherwood.
341	SELECTING EXPONENTIAL SMOOTHING MODEL PARAMETERS: AN APPLICATION OF PATTERN SEARCH, William L. Berry and Friedhelm W. Bliemel.
342	AN INTEGRATED EXAMINATION OF MEDIA APPROACHES TO MARKET SEGMENTATION, Albert Bruno, Thomas Hustad and Edgar Pessemier.
343	LABORATORY EXPERIMENTATION, Howard L. Fromkin and Siegfried Streufert.
344	REVERSAL OF THE ATTITUDE SIMILARITY-ATTRACTION EFFECT BY UNIQUENESS DERIVATION, Howard L. Fromkin, Robert L. Dipboye and Marilyn Pyle.
345	WILL THE REAL CONSUMER-ACTIVIST PLEASE STAND UP, Thomas P. Hustad and Edgar A. Pessemier.
346	MULTI-ATTRIBUTE MODELS FOR PREDICTING INDIVIDUAL PREFERENCE AND CHOICE, Edgar A. Pessemier.
347	THE VALUE OF INFORMATION IN AGGREGATE PRODUCTION PLANNING - A BEHAVIORAL EXPERIMENT, Herbert Moskowitz.
348	A MEASUREMENT AND COMPOSITION MODEL FOR INDIVIDUAL CHOICE AMONG SOCIAL ALTERNATIVES, Edgar A. Pessemier.
349	THE NEOCLASSICAL THEORY OF INVESTMENT AND ADJUSTMENT COSTS, Akira Takayama.
350	A SURVEY OF FACILITY LOCATION METHODS, D. Clay Whybark and Basheer M. Khumawala.
351	THE LOCUS AND BASIS OF INFLUENCE ON ORGANIZATION DECISIONS, Martin Patchen.
352	A PLEA FOR A FOURTH TRADITION - AND FOR ECONOMICS, Robert V. Horton.
353	EARLY APPLICATIONS OF SPECTRAL METHODS TO ECONOMIC TIME SERIES, Thomas F. Cargill.

Paper No.	Title and Author(s)
354	STUDENT APPLICATIONS IN A PRINCIPLES COURSE OF ECONOMIC ANALYSIS TO SELF-DISCOVERED ITEMS, Robert V. Horton.
355	BRANCH AND BOUND ALGORITHMS FOR LOCATING EMERGENCY SERVICE FACILITIES, Basheer M. Khumawala.
356	BEHAVIORAL SCIENCES LABORATORIES DESIGN FACTORS, Benjamin L. Mays.
357	AN EFFICIENT ALGORITHM FOR CENTRAL FACILITIES LOCATION, Basheer M. Khumawala.
358	AN EXPERIMENTAL STUDY OF ATTITUDE CHANGE, ADVERTISING, and USAGE IN NEW PRODUCT INTRODUCTION, James L. Ginter and Frank M. Bass.
359	DENIAL OF SELF-HELP REPOSSESSION: AN ECONOMIC ANALYSIS, Robert W. Johnson.
360	WAREHOUSE LOCATION WITH CONCAVE COSTS, Basheer M. Khumawala and David L. Kelly.
361	LINEAR AND NONLINEAR ESTIMATION OF PRODUCTION FUNCTIONS, R. A. Meyer and K. R. Kadiyala.
362	QUASI-CONCAVE MINIMIZATION SUBJECT TO LINEAR CONSTRAINTS, Antal Majthay and Andrew Whinston.
363	PRODUCTION FUNCTION THEORY AND THE OPTIMAL DESIGN OF WASTE TREATMENT FACILITIES, James R. Marsden, David E. Pingry and Andrew Whinston.
364	A REGIONAL PLANNING MODEL FOR WATER QUALITY CONTROL, David E. Pingry and Andrew Whinston.
365	ISSUES IN MARKETING'S USE OF MULTI-ATTRIBUTE ATTITUDE MODELS, William L. Wilkie and Edgar A. Pessemier.
366	A SOCIAL PSYCHOLOGICAL ANALYSIS OF ORGANIZATIONAL INTEGRATION, Howard L. Fromkin.
367	ECONOMICS OF WASTEWATER TREATMENT: THE ROLE OF REGRESSION, J. R. Marsden, D. E. Pingry and A. Whinston.
368	THE ROLE OF MODELS IN NEW PRODUCT PLANNING, Edgar A. Pessemier and H. Paul Root.
369	A NOTE ON PREFERENCE ORDERINGS WHICH ARE CONVEX TO THE ORIGIN, James C. Moore.
370	AXIOMATIC CHARACTERIZATIONS OF CONSUMER PREFERENCES AND THE STRUCTURE OF THE CONSUMPTION SET, James C. Moore.
371	BUSINESS POLICY OR STRATEGIC MANAGEMENT: A BROADER VIEW FOR AN EMERGING DISCIPLINE, Dan E. Schendel and Kenneth J. Hatten.

Paper No.	Title and Author(s)
372	MULTI-ATTRIBUTE CHOICE THEORY - A REVIEW AND ANALYSIS, Edgar A. Pessemier and William L. Wilkie.
373	INFORMATION AND DECISION SYSTEMS FOR PRODUCTION PLANNING: A NEED FOR AN INTER-DISCIPLINARY PERSPECTIVE, Herbert Moskowitz and Jeffrey G. Miller.
374	ACCOUNTING FOR THE MAN/INFORMATION INTERFACE IN MANAGEMENT INFORMATION SYSTEMS, Herbert Moskowitz and Richard O. Mason.
375	A COMPETITIVE PARITY APPROACH TO COMPETITION IN A DYNAMIC MARKET MODEL, Randall L. Schultz.
376	BEHAVIORAL MODEL BUILDING, Randall L. Schultz and Dennis P. Slevin.
377	THE HALO EFFECT AND RELATED ISSUES IN MULTI-ATTRIBUTE ATTITUDE MODELS - AN EXPERIMENT, William L. Wilkie and John M. McCann.
378	AN IMPROVED METHOD FOR SOLVING THE SEGREGATED STORAGE PROBLEM, Basheer M. Khumawala and David G. Dannenbring.
379	ON THE PROBABILITY OF WINNING IN COMPETITIVE BIDDING THEORY, Keith C. Brown.
380	COST ALLOCATION FOR RIVER BASIN PLANNING MODELS, E. Loehman, D. Pingry and A. Whinston.
381	FORECASTING DEMAND FOR MEDICAL SUPPLY ITEMS USING EXPONENTIAL AND ADAPTIVE SMOOTHING MODELS, Everett E. Adam, Jr., Willian L. Berry and D. Clay Whybark.
382	SETTING ADVERTISING APPROPRIATIONS: DECISION MODELS AND ECONOMETRIC RESEARCH, Leonard J. Parsons and Randall L. Schultz.
383	ON THE OPTIMAL GROWTH OF THE TWO SECTOR ECONOMY, John Z. Drabicki and Akira Takayama.
384	UNCERTAIN COSTS IN COMPETITIVE BIDDING, Keith C. Brown.
385	EFFECTS OF THE NUMBER AND TYPE OF ATTRIBUTES INCLUDED IN AN ATTITUDE MODEL: MORE IS NOT BETTER, William L. Wilkie and Rolf P. Weinreich.
386	PARETO OPTIMAL ALLOCATIONS AS COMPETITIVE EQUILIBRIA, James C. Moore.
387	A PLANNING AND COST ALLOCATION PROCEDURE FOR COMPUTER SYSTEM MANAGEMENT, J. F. Nunamaker and A. Whinston.
388	PROFESSOR DEBREU'S "MARKET EQUILIBRIUM" THEOREM: AN EXPOSITORY NOTE, James C. Moore.

Paper No.	Title and Author(s)
389	THE ASSIGNMENT OF MEN TO MACHINES: AN APPLICATION OF BRANCH AND BOUND, Jeffrey G. Miller and William L. Berry.
390	THE IMPACT OF HIERARCHY AND GROUP STRUCTURE ON INFORMATION PROCESSING IN DECISION MAKING: APPLICATION OF A NETWORKS/ SYSTEMS APPROACH, David L. Ford, Jr.