

AN ABSTRACT OF THE THESIS OF

JIMMIE LYNN ELLIOTT for the degree of MASTER OF SCIENCE

in Computer Science presented on June 5, 1975

Title: COMPUTER PERFORMANCE AND EVALUATION UTILIZING
THE RESOURCE PLANNING AND MANAGEMENT SYSTEM

Abstract approved: _____

Redacted for privacy

Robert A. Short

A review of current computer performance and evaluation techniques reveals a lack of an acceptable analytic tool for optimal computer system performance and evaluation.

A generalized approach to the formulation of a third generation computer system model is proposed. The approach is used to optimize computer resource utilization and to obtain an upper bound on the system throughput level.

The Stimler computer model is represented as a Resource Planning and Management System (RPMS) network and optimized by Linear programming algorithms. The results are portrayed by an adaptation of the Kiviat graph technique. An application of finite state and context-free grammar theory led to extensions of the current RPMS theory in the form of postulates dealing with compaction, decomposition, and expansion.

Future areas to be exploited include multi-objective goal programming, integer programming, stochastic programming, and functional level analysis.

Computer Performance and Evaluation
Utilizing the Resource Planning
and Management System

by

Jimmie Lynn Elliott

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1976

APPROVED:

Redacted for privacy

Professor and Chairman of Computer Science
in charge of major

Redacted for privacy

Dean of Graduate School

Date thesis is presented May 5, 1975

Typed by Susie Kozlik for Jimmie Lynn Elliott

DEDICATION

Dedicated to my loving wife, Linda Marie,
who always encouraged me to continue
and who already has her Master's.

ACKNOWLEDGEMENTS

I would like to thank the members of my committee:

Dr. R. A. Short, Major Professor

Dr. M. S. Inoue, Minor Professor

Dr. L. C. Hunter, Major Department Representative

Dr. J. F. Engle, Graduate School Representative

With special appreciation to Dr. Short for giving the geometric shape of a circle new meaning, and to Dr. Inoue for doing the same for squares and triangles.

Further, to acknowledge the assistance of the Salem, Oregon, branch office of IBM for supplying the copy of IBM report TR 00.2043, the Gibson Mix, which appears in the Appendix.

Last but not least, to acknowledge the assistance of my typist Mrs. Susie Kozlik for her outstanding job done under a severe time constraint.

TABLE OF CONTENTS

| <u>Chapter</u> | <u>Page</u> |
|--|-------------|
| I INTRODUCTION | 1 |
| Research Objective | 3 |
| Structure of the Thesis | 4 |
| II PRESENT STATUS OF COMPUTER PERFORMANCE AND EVALUATION TECHNIQUES | 6 |
| General Observations | 6 |
| Formulations and Weighting Schemes | 12 |
| Instructional Mixes | 13 |
| Kernels | 15 |
| Benchmark | 15 |
| Synthetic Models | 16 |
| Kiviat Graphs | 18 |
| Monitors | 20 |
| Mathematical Programming | 21 |
| Simulation | 22 |
| Conclusions | 23 |
| III RESOURCE PLANNING AND MANAGEMENT SYSTEM (RPMS) | 24 |
| Linear Programming | 24 |
| Components of RPMS | 28 |
| Postulates of RPMS | 33 |
| RPMS Conventions | 35 |
| Construction of the RPMS Network | 36 |
| Solution of RPMS Network | 36 |
| Solution Procedure for RPMS Networks | 37 |
| IV PROPOSED APPROACH | 42 |
| Developing a Universal CPE Model | 43 |
| V THE RPMS APPROACH APPLIED TO CPE: BATCH VS ON-LINE | 47 |
| Discussion of Results | 53 |
| VI EXTENTIONS OF RPMS THEORY | 59 |
| VII CONCLUSIONS | 70 |
| General and Summary | 70 |
| Future Areas of Research | 72 |

| <u>Chapter</u> | <u>Page</u> |
|--|-------------|
| BIBLIOGRAPHY | 75 |
| APPENDICES | |
| Appendix A - Glossary of Terms | 79 |
| Appendix B - Butler's Formula | 82 |
| Appendix C - Ollivier's Formula | 84 |
| Appendix D - Instruction Category Descriptions | 86 |
| Appendix E - Gibson Mix - IBM Report | 88 |
| Appendix F - Kiviat Graphs | |
| Appendix G - Lee's GOAL Programming for CYBER | 93 |

LIST OF TABLES

| <u>Table</u> | | <u>Page</u> |
|--------------|-------------------------------|-------------|
| 1 | CPE Techniques Summary | 10 |
| 2 | Scientific and Business Mixes | 14 |

LIST OF FIGURES

| <u>Figure</u> | | <u>Page</u> |
|---------------|---|-------------|
| 1-1 | Lucas's Recommended CPE Purposes and Techniques | 11 |
| 2-1 | Morris's Kiviat Graph Axis Labels | 19 |
| 3-1 | RPMS Nodal Conventions | 32 |
| 3-2 | RPMS Basic Flow | 33 |
| 3-3 | RPMS Cause and Effect Diagrams | 34 |
| 3-4 | RPMS Terminal Nodes | 34 |
| 3-5 | RPMS Feasibility and Optimality Conditions | 38 |
| 3-6 | RPMS Network of Example - Step 1 | 39 |
| 3-7 | Example Step 2 | 40 |
| 3-8 | Example Step 3 and 4 | 41 |
| 3-9 | Example Final Solution | 41 |
| 4-1 | Stimler's Model | 45 |
| 4-2 | Expanded Stimler Model | 45 |
| 4-3 | RPMS Diagram of Stimler Subsystem | 46 |
| 5-1 | Basic Applied Model | 50 |
| 5-2 | Double LP | 51 |
| 5-3 | Tripple LP | 52 |
| 5-4 | Kiviat Graphs of Batch vs On-Line | 57 |
| 6-1 | Finite State Machine | 59 |
| 6-2 | RPMS Representation of Finite State | 60 |

| <u>Figure</u> | | <u>Page</u> |
|---------------|---|-------------|
| 6-3 | RPMS S & R Relationships | 61 |
| 6-4 | Concatenated RPMS Finite State Machine | 61 |
| 6-5 | Von Neumann Model | 62 |
| 6-6 | Hellerman Model | 64 |
| 6-7 | Multiple Nodes | 66 |
| 6-8 | Original On-Line Path | 68 |
| 6-9 | Compacted On-Line Path | 68 |
| 7-1 | Time and Cost Comparison of CPE Technique | 71 |

COMPUTER PERFORMANCE AND EVALUATION UTILIZING THE RESOURCE PLANNING AND MANAGEMENT SYSTEM

I. INTRODUCTION

Computer performance and evaluation, abbreviated as CPE, is endowed with potential benefits for the computer community. Theoretically, CPE is capable of providing many urgently needed answers to problems faced today in areas such as procurement, planning, costing, scheduling, designing, and optimization (Highland, 1974). Of these areas, procurement and optimization are shared by both computer manufacturers and systems users. Procurement is defined as the selection and specification of systems components. Optimization refers to the process of maximizing some system objective(s) through judicious allocation of resources.

In the procurement area, currently only 20 to 40 percent of all computer acquisitions are based upon any performance evaluation (Kanter, 1970). Typically such studies benefit only the users. In the area of optimization no practical tool has been available.

Procurement and optimization can both be viewed in quantifiable terms. Two prime terms which are normally calculated are availability and system capability. Availability refers to how much time a system is normally available to do productive work. System capability is an expression of what a system is capable of performing while it is available (Drummond, 1970, p. 4).

Cost and time are two major factors which require that the availability and system capability be evaluated for a computer system (Bell, 1972). In this context, cost is the monetary amount invested in a system while time is the delay caused by the system in performing prescribed tasks. Both of these factors are used as the evaluation criteria.

New systems evolve as the result of recent research and development efforts. Research and development is expensive and time consuming. The manufacturer needs an objective evaluation tool to assess both the strength and weakness of prototype systems and to identify design areas requiring further work. The buyer also needs an objective evaluation tool to predict the performance of his proposed system before he can justify the time and cost involved in the conversion.

Once a system has been selected, its design and performance continue to be challenged. An existing system may perform the originally prescribed tasks satisfactorily, but changes in the amount and mix of tasks to be processed by a system occur frequently. The users are often not aware of the tasks that a computer system is capable of performing until after the system has been operational for some time. Also technological advances provide new alternatives for systems improvements. The recurring questions are (1) whether or not the existing system capacity is adequate for processing the new

job mix, and (2) does the change in job-mix and new technology warrant the investment in time and money required to upgrade the system.

There are several terms used to describe system capability. Throughput is the most popular of these. Throughput is an expression of the processing rate of a system (Drummond, 1973, p. 15).

This study proposes an approach which is helpful to both the manufacturers and users in answering the system capability question in terms of throughput. The approach is based upon the Stimler computer model (Stimler, 1974) represented by the Resource Planning and Management System (Riggs and Inoue, 1975) networks and optimized by Linear programming algorithms.

The examples used during this study indicate that this approach has the potential for becoming a valuable tool in system selection for procurement and system optimization for operation.

Research Objective

The research objective is to develop an approach to answer the computer performance and evaluation questions centered around throughput and capacity. The objective will be considered attained when answers to the following questions are obtainable from the application of the proposed approach:

- * What is the maximum system throughput rate?
- * How much system capacity is currently idle in the existing system?
- * What system component is currently restricting the throughput at its present level?
- * What is the job-mix that will maximize the throughput?
- * What is the minimum equipment characteristic to obtain the desired throughput?
- * What is the marginal value of the system component that is currently restricting the throughput?

Structure of the Thesis

This introductory chapter discussed the need for a practical systematic approach to the performance and evaluation of computer systems.

Chapter II reviews existing computer performance and evaluation techniques.

Chapter III introduces the Resource Planning and Management System, and applies it to the computer performance and evaluation technique of linear programming. The results of this analysis are portrayed by a series of Kiviat graphs.

Chapter IV describes the proposed approach for solving the capacity problem.

Numerical examples are included in Chapter V.

An application of finite state and context-free grammar theory led to extensions of the current RPMS theory in the form of postulates dealing with compaction, decomposition, and expansion is presented in Chapter VI.

Chapter VII discusses the significance of this study and proposes future research possibilities.

II. PRESENT STATUS OF COMPUTER PERFORMANCE AND EVALUATION TECHNIQUES

General Observations

The area of computer performance and evaluation, abbreviated CPE, is a relatively young discipline within the field of computer science. The first major text appeared in 1973 (Drummond), and there is still considerable disagreement as to what CPE purports.

The following definitions are offered to illustrate some current views:

Evaluation is ascertaining the value of a computer system and measurement is ascertaining the extent of a computer system (Drummond, 1973).

To some, computer performance, evaluation and measurement is a tool, a marriage of abstract thought and logic combined with the techniques of statistical and quantitative methods. To others, it is a technique with heavy reliance on modeling and simulation and simultaneously involves features of both classical experimentation and formal analysis (Highland, 1974).

System performance evaluation is used to determine how well a specific system is meeting or may be expected to meet specific diversified processing requirements at specific interfaces (Stimler, 1974).

The following definition summarizes these views and will be used for this study:

Computer performance and evaluation determines computer system's capacity by examining the attainment level of prescribed goals.

Several terms need to be defined before we can proceed further with the study. The following definitions are taken from IFIPS (1971):

A task is a number of steps which are partially or wholly ordered from the point of view of their execution.

A program is a complete specification of one or more tasks that are to be performed on data.

A job is a basic independent unit of work to be carried out by a system.

The following definitions are taken from Drummond (1973), Stimler (1974), and Bell (1972):

Throughput is an expression of the processing rate of a system, given as some unit per time interval (e. g. , jobs/hour).

Relative systems throughput is the relative estimate of the performance of a proposed computing system. $RST = T_b/T_m$ where T_b = throughput rate of a base or standard computing system; T_m = throughput rate of the proposed or new computing system.

Capacity is the maximum average throughput achieved when the system is at 100 percent utilization (Stimler, 1974).

Response time is the average time a user must wait to receive a response from a system once a task has been initiated (Bell, 1972).

A standard set of terms currently does not exist for CPE. The terms presented are given as the most popular and accepted units of measure employed in CPE efforts at the time of this writing. At present, the National Bureau of Standards is working on compiling a

composite bibliography of the most commonly used terms with their definitions.¹

Conceptually, there are two major areas where CPE is critically important. The first of these is in the original selection of a completely new system. This new system can be either an initial system for a new user or the total replacement of an existing system. The second major application of CPE is in replacement which involves the selective changing of individual parts of a system. Replacement is important not only to existing systems, but also is useful in evaluating the alternative selection of parts of a new system.

CPE tends to be practiced by technicians with the results transmitted only to other practitioners. Although CPE efforts are initiated by managerial level personnel, once the assigned CPE effort has entered the realm of the practitioners, the results are seldom fed back to the managerial level in a useful form. This is due to the volume of data that most CPE techniques generate. Simple answers with meaningful results are needed.

To practice CPE, one or more of the techniques discussed in this chapter is employed to attain a specific goal.

For the purpose of this study, we shall accept one or more of

¹ A set of the most commonly used terms and their definitions are given in the Appendix A.

the following as the primary goal(s) of any CPE effort related to procurement or optimization.

1. Minimize the total initial system cost while maintaining some accepted level of system throughput.
2. Maximize utilization of the existing resources in the system.
3. Minimizing the amount of additional resources to be acquired in order to increase the system's capability to meet future job requirements.

These three goals correspond roughly to the statement of feasibility, optimality, and adaptivity of any system. The term "resource" is used to designate a productive component of the computer system that can be assigned for use in completing a task, and thus embodies both the concept of system capacity and availability. I/O channels, line-printers, and CPU time-slices are examples of hardware resources. Operating systems, compilers, and data files are examples of software resources. A resource that is utilized economically increases its marginal value by minimizing its idle-time. Thus, interpreting the CPE goals in terms of resource utilization, we are justified in considering CPE as a problem in optimal resource allocation (Hellerman, 1970). If the availability of the resources and the goals can be expressed in a mathematical form, the CPE problem may be interpreted as a mathematical programming model that maximizes (or minimizes) an objective function or a functional subject to

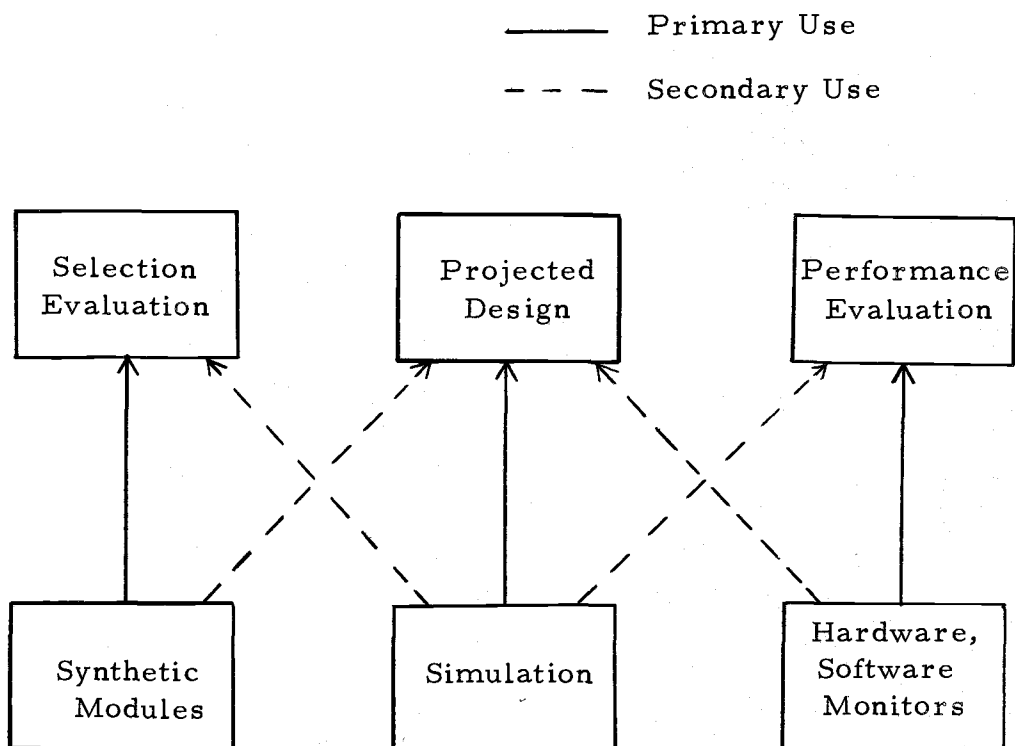
Table 1. CPE Techniques Summary.

| Symbol Description | Total Time Required (Hours) | Total Cost Required (\$) | Applicability to Procurement (Selection) | Amount of Optimization Properties | Shortcomings |
|------------------------------|--------------------------------|-----------------------------|--|---|---|
| (A) Formulation | Low | Low | Poor | Poor | Parallelism not considered |
| (B) Instruction Mixes | Low | Low | Poor | Poor | I/O not considered |
| (C) Kernels | Medium | Medium | Fair | Poor | Identification difficult |
| (D) Benchmarks | Medium | Medium to high | Good | Poor | An approximation or multi programming not considered |
| (E) Synthetic Models | Low | Low | Fair | Poor | Data dependent |
| (K) Kiviat Graphs | Low | Low | Fair | Fair | Supportive tool only |
| (G) Monitors | Medium | Medium to high | Fair | Fair | System dependent or requires system modification |
| (H) Mathematical Programming | High | High | Good | Good | Data difficult to obtain |
| (I) Simulation | High | High | Good | Good | High cost and time delay |

Poor: Gives little aid

Fair: Some aid but inadequate

Good: Satisfactory



a set of resource constraints. When all relationships are approximated linearly, the standard linear programming approach becomes feasible.

Before undertaking the new approach, however, it is imperative that we examine the existing techniques of computer performance and evaluation in use today. A cursory review of the "state of the art" is presented by giving a short description and a statement of the relative shortcomings of each major CPE technique. Table 1 summarizes the CPE techniques.

Formulations and Weighting Schemes

Among the various CPE formulations, the two most prominent studies are those conducted by Knight (Sharp, 1969) and Butler (1970).

These formulations attempted to derive one single term as a numerical measure for the evaluation or selection process. For example, Butler calculates a total price to performance ratio that is the average of the hardware and software price-performance ratios. Butler's complete formulation is given in Appendix B.

An extension of this approach is the concept of weighting schemes which assign different weights to individual factors according to applications. The weights and measures purport to be a refinement upon the formulation technique. One example of weights and measures is given by Ollivier (1970). Ollivier calculates a single figure based

upon hardware characteristics and manufacturer's past performance. Ollivier's complete scheme is included in Appendix C.

The major advantage of all these methods is that an evaluation can be carried out rapidly and at a low total cost. The major disadvantages are that the formulas do not accommodate the parallelism of more advanced computer architecture.

Instructional Mixes

In an attempt to improve the methods of evaluations used for computers, an extension of formulation, called instruction mixes, was conceived. Instruction mixes use the weighted sum of execution times of the individual instructions to arrive at a single numerical value.

Instructional mixes are generally broken down into the two classes: business and scientific mixes (Sharp, 1969). A business mix contains a heavier weight of those instructions which are more frequently used in a business environment. A scientific mix places the emphasis on the instructions which are more computation oriented. Table 2 contains an example of both kinds of mixes (Sharp, 1969). Appendix D contains a detailed description of each instruction category in the two mixes.

Arbuckle made an early attempt at deriving a mix (1966) but the most widely used instruction mix is one attributed to Gibson (1970).

A complete copy of Gibson's original paper is included in Appendix E.

Table 2. Scientific and Business Mixes.

| Instruction Category | Scientific | Business |
|--|------------|-----------|
| 1. Fixed Add/Subtract and compare | 10 | 25 |
| 2. Floating Add/Subtract | 10 | -- |
| 3. Multiply | 06 | 01 |
| 4. Divide | 02 | -- |
| 5. Other manipulation & logic instructions | 72 | 74 |
| | <hr/> 100 | <hr/> 100 |

(All entries in Table 2 are in percent)

The major advantage of this type of evaluation is that the actual execution times are readily available.

The disadvantages are as follows:

1. The mixes do not include any category for Input-Output type of instructions.
2. A subjective approach is used to weigh each class of instructions.
3. Medium and large scale computers are now used for both business and scientific work, thus making the two classifications of mixes meaningless.

Kernels

The kernel approach to CPE was developed from the observation that computer systems run the same application repeatedly. The kernel analysis is performed by running a kernel program and obtaining a figure that reflects the amount of time required. The kernel program is defined as that portion of a program that takes the greatest amount of execution time (Drummond, 1973). Typical kernels include matrix inversion, polynomial evaluation, and square root approximation (Timmerick, 1973).

A kernel analysis is only appropriate when the system executes one task repeatedly for a major portion of the total time that the system is in use. It further assumes that its kernel is easily identified.

The advantage of kernel analysis over the formulation method is that it weighs each group of instructions more objectively.

The disadvantage is that an adequate consideration of I/O and administrative overhead, such as operating systems, is not included.

Benchmark

Benchmark analysis was developed for systems which run one prime program but where the simple kernel approach is inadequate. Benchmark has been defined as a sophisticated kernel (Timmerick,

1973). Benchmark, typically, is a production program that dominates the system or data processing needs of an organization.

The main problem associated with benchmarking is the time required to run the benchmark. A typical benchmark test involves 24 operational hours. Additionally, a benchmark does not evaluate the composite system that includes non-production jobs. A non-production job is taken to mean a task that does not reduce the amount of production jobs the system must run within a time frame. Further, benchmarking requires the coding job be done for each individual system to be evaluated using the same level of programmer expertise.

The major advantage of benchmarking is that it is often the only practical way to check new systems that are to be used for replacement. Benchmarking shows that a system can run a specific job. This is accomplished executing the benchmark job and timing the run.

The major limitation is that only an approximation of the true environment can be obtained. Environment is taken here to mean all conditions related to the use of the system. A benchmark serves best as a before-and-after test to monitor performance in conjunction with system changes (Lucas, 1971).

Synthetic Models

Synthesis is the creation of the whole from its parts. A synthesis model is created by combining all of the subsystems to make the

composite system (Drummond, 1973). Hence, a subsystem is any part of the system that can be taken as a separate system by itself.

Several definitions are needed to clarify the discussion to follow.

CPU is a mnemonic used for the central processing unit of a computer system. I/O includes all functions external to the CPU. Overlap is said to occur when either two or more I/O devices, or the CPU and at least one I/O device are operating concurrently.

Throughput can be determined by using a synthetic model which combines the I/O and the CPU time and considers overlap to yield a total system time. The synthetic model technique has been successful because of the availability of data to build the models and the fact that the interactions between components can be easily understood (Callengaert, 1967).

The true power and flexibility of this technique is based upon how the level of overlap is implemented and how detailed the model is (Drummond, 1973).

The major advantage is that this method is easy to implement because the data are readily available. The disadvantage is that the accuracy of the result is contingent upon the validity of the source data in the user environment.

Kiviat Graphs

A Kiviat graph is a circular graph with polar coordinates that was first proposed by Philip J. Kiviat. It was further developed by the Federal Computer Performance and Evaluation Center (FEDSIM) into an aid to visually display the interrelationships existing between various attributes of a computer system.

In the following description of Kiviat graph methodology, the term tuning is defined as making minor changes to an existing system to increase system efficiency (Bell, 1972). The term local identifies the particular computer system that is being evaluated at the time.

(1) Select an even number of performance indicators, half of which are locally regarded as "good" when they increase due to tuning efforts, and half of which are locally regarded as "bad" when they increase. (2) Divide a circle into as many symmetrical segments as there are performance indicators, beginning with the vertical axis from the circle center to the outmost point on the circle's arc. (3) Number the top vertical axis one and number the rest of the axes sequentially around the circle; (4) Plot the "good" indicators on the odd numbered axes and the "bad" indicators on the even numbered axes (Morris, 1974).

One may question the relevance of an even number of axes. As can be seen on Figure 2-1, the value of a Kiviat graph is derived from the symmetrical star shape created when points on all the axes are connected. The even number of axes provides for the star shape that is the result of alternating "good" and "bad" indicators.

Figure 2-1 gives the axis labels used by Morris (1974).

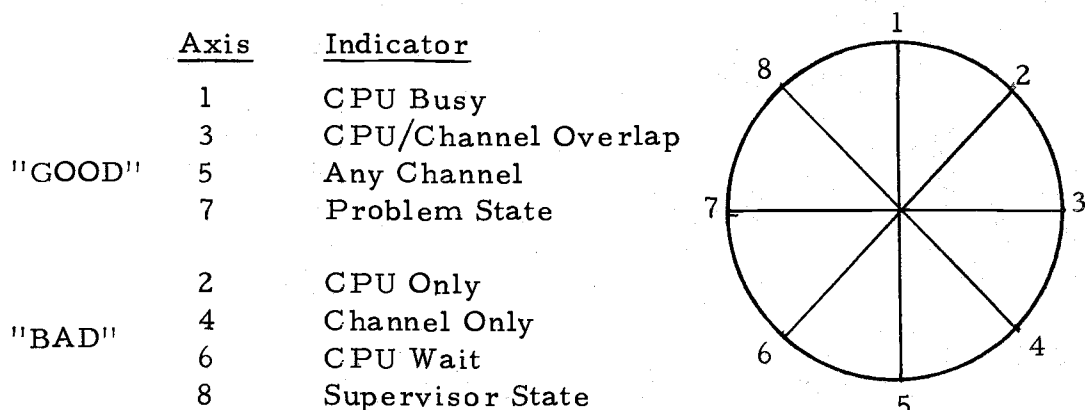


Figure 2-1. Morris's Kiviat Graph Axis Labels.

Currently, Kiviat graphs have been used only to depict a static system. A static system is defined as a view of a system for one instant in time. By contrast, a dynamic system is defined as a system that exists between two points in time. One proposed approach to the problem of portraying dynamic systems is to use multiple Kiviat graphs for the various instances of time. This creates what's known as "stars in a sky" where the sky is the operational environment of the system.

The Kiviat graph is a supportive tool. It gives a pictorial representation of data obtained from other performance and evaluation methods. The graph then says what exists at the time the data was gathered. An example of several different Kiviat graphs is given in Appendix F.

Monitors

Two types of monitors are used in CPE: hardware and software. Monitoring is defined as a method of collecting data on the performance of an existing system (Lucas, 1971).

A hardware monitor is constructed of electronic logic, probes, and a magnetic tape recorder (Stimler, 1974). It often includes a mini-computer and more recently a micro-computer. Hardware monitors make basic measurements that tell: (1) how long a resource has been used during the evaluation period and (2) the number of times an event occurred during the same period (Bell, 1971).

A hardware monitoring involves: (1) an evaluation of a computer system for data or information to be gathered; (2) determination of appropriate spots for connecting monitor probes; (3) running of the monitor for an evaluation period; and (4) post analysis of data gathered (Ibid).

The major advantage of hardware monitors is that the monitor can be connected to any vendor's system to obtain the desired data while producing no load on the existing computer system. The major limitation is that it is often hard to determine the exact spot where the desired data can be gathered. Also, often the modification of the system that is required is a breach of the contractual agreement with the vendor (Stimler, 1974).

A software monitor is a program which is incorporated in the operating system (Stimler, 1974). A software monitor periodically samples the conditions of certain memory locations where the operating system stores operational conditions of the system.

The main advantage of a software monitor is that it involves virtually no system modification. The main disadvantage is one of system dependence; that is, a separate software monitor must be written for each individual manufacturer's system.

One successful application of a software monitor for the IBM 370 was reported by Betz (1973).

In summary, hardware monitors can be used to tell what happened while software monitors will give a better indication as to why it happened (Drummond, 1973).

Mathematical Programming

Some effort has been made to formulate CPE problems by mathematical programming models. The results of previous efforts have shown that an analyst with enough data about the programs which are to be run and the hardware characteristics of the system can formulate his CPE problem in terms of an integer linear programming problem (Sharp, 1969).

An integer linear programming problem is a linear programming

problem where decision variables are not allowed to assume non-integer values.

Sharp (1969) further points out that the problem of capturing all the significant interrelationships between the subsystems is far from a trivial task. The prospects for general use of this method are further hampered by the limited number of CPE practitioners familiar with the level of mathematics involved in applying this optimization technique.

This technique has the advantage of mathematical accuracy while its major disadvantage is the amount of time and cost involved in implementation.

Simulation

Simulation has been considered the most powerful and flexible technique for CPE (Lucas, 1971). Several commercial simulation packages are available. One of these is the "Systems and Computers Evaluation and Review Technique," commonly abbreviated SCERT (Herman, 1967).

Simulation is taken to mean the technique of solving problems by using a mathematical model to follow the changes over time with respect to a dynamic system. Simulation is said to be done when all equations of the model are solved simultaneously with steadily increasing values of time (Gordon, 1969, pg. 17).

• The original General Purpose Systems Simulator (GPSS) language was used as an aid in studying new computer systems (IBM, 1963).

Major limitations of simulation include the time for setting up the simulation, running it, and validating the results. Further, the actual cost and amount of computer time required are other limitations. Personnel is also a problem since a knowledge of simulation languages is required. Simulation has been used successfully for the optimization of performance of time-sharing systems, (Blatny, 1972), and in the solution of hardware allocation problems (Hesser, 1973).

Conclusions

In summary, all of the CPE techniques reviewed were found useful but none was universally practical either as a procurement or as an optimization evaluation tool. This confirmed the generally accepted belief that no one single CPE tool or technique can satisfy all goals or problems which are significant in computer performance and evaluation (Lucas, 1971).

Thus it appears that a combination of various techniques is necessary to accomplish an adequate evaluation. Our proposed study will use a combination of the synthetic modeling and linear programming techniques, Resource Planning and Management System methodology, and Kiviat graphs to accomplish a throughput evaluation of a computer system.

III. RESOURCE PLANNING AND MANAGEMENT SYSTEM (RPMS)

The Resource Planning and Management System (RPMS) represents a mathematical programming problem as a graphical network model. RPMS was first proposed in 1972 as a concatenation of cause and effect diagrams to model linear programming problems and their solutions (Inoue and Riggs, 1972). It has since been extended to cover dynamic programming, quadratic programming, goal programming, and other special cases of mathematical programming models. The most significant improvement is one allowing visual identification of the Kuhn-Tucker conditions (Inoue, 1974).²

The following section is limited to a review of the general linear programming problem. Further, the basic canonical form is modified so as to reflect a property which will be used in the development of the RPMS components section.

Linear Programming

Given any linear programming problem, the problem can be transformed into the following canonical form (Taha, 1971, pg. 25).

$$\text{Maximize } z_x = \sum_{j=1} c_j x_j \quad (3-1)$$

subject to the constraints

²Several of those models are discussed in an introductory OR/MS text (Riggs and Inoue, 1975) that is currently being prepared for publication.

$$\sum_{j=1}^n a_{ij} x_j \leq b_i \quad 1 \leq i \leq m \quad (3-2)$$

$$x_j \geq 0 \quad 1 \leq j \leq n \quad (3-3)$$

where

a_{ij} , b_j , and c_j are constants in a linear programming model

x_j is called a primal variable.

The constants can be separated according to their positive and negative signs,

$$a_{ij} = a_{ij}^+ - a_{ij}^- \quad (3-4)$$

$$b_j = b_j^+ - b_j^- \quad (3-5)$$

$$c_j = c_j^+ - c_j^- \quad (3-6)$$

$$1) \quad a_{ij}^+ = a_{ij} \quad \text{if } a_{ij} > 0 \quad \text{else } a_{ij}^+ = 0 \quad (3-7)$$

$$2) \quad a_{ij}^- = a_{ij} \quad \text{if } a_{ij} < 0 \quad \text{else } a_{ij}^- = 0 \quad (3-8)$$

$$3) \quad (a_{ij}^+) \cdot (a_{ij}^-) = 0 \quad (3-9)$$

$$4) \quad b_i^+ = b_i \quad \text{if } b_i > 0 \quad \text{else } b_i^+ = 0 \quad (3-10)$$

$$5) \quad b_i^- = b_i \quad \text{if } b_i < 0 \quad \text{else } b_i^- = 0 \quad (3-11)$$

$$6) \quad (b_i^+) \cdot (b_i^-) = 0 \quad (3-12)$$

$$7) \quad c_i^+ = c_i \quad \text{if } c_i > 0 \quad (3-13)$$

$$\text{else } c_i^+ = 0 \quad (3-14)$$

$$8) \quad c_i^- = c_i \quad \text{if } c_i < 0 \quad (3-15)$$

$$\text{else } c_i^- = 0 \quad (3-16)$$

$$9) \quad (c_i^+) \cdot (c_i^-) = 0 \quad (3-17)$$

The canonical form can now be written as

$$\text{Maximize } z_x = \sum_{j=1}^n [c_j^+ - c_j^-] x_j \quad (3-18)$$

Subject to the constraints

$$\sum_{j=1}^n [a_{ij}^+ - a_{ij}^-] x_j \leq [b_i^+ - b_i^-] \quad 1 \leq i \leq m \quad (3-19)$$

$$x_j \geq 0 \quad 1 \leq j \leq n \quad (3-3)$$

An expansion of the above primal model gives a primal objective function:

$$\text{Maximize } Z_x = \sum_{j=1}^n c_j^+ x_j - \sum_{j=1}^n c_j^- x_j \quad (3-20)$$

subject to m resource constraints:

$$\sum_{j=1}^n a_{ij} x_j + b_i^+ \geq \sum_{j=1}^n a_{ij}^+ x_j + b_i^- \quad 1 \leq i \leq m \quad (3-21)$$

and non-negativity restrictions:

$$x_j \geq 0; a_{ij}^+ \geq 0; a_{ij}^- \geq 0; b_i^+ \leq 0; b_i^- \geq 0; c_i^+ \geq 0; c_i^- \geq 0 \quad 1 \leq i \leq m \quad (3-22)$$

$$1 \leq j \leq n$$

The RPMS portrays each of the m resource constraints as a cause-and-effect diagram illustrating that the sum of the endogenous $(a_{ij}^- x_j)$ and exogenous (b_i^+) supply of the resource cannot be less than the sum of the endogenous $(a_{ij}^+ x_j)$ and exogenous (b_i^-) demand satisfied by the resource.

Each primal linear programming model (eqn 3-1, 3-2, 3-3) has associated with it a dual model that can be expressed as:

$$\text{Minimize } Z_y = \sum_{i=1}^m b_i y_i \quad (3-23)$$

subject to

$$\sum_{j=1}^n a_{ij} y_i \geq c_j \quad 1 \leq j \leq n \quad (3-24)$$

and

$$y_i \geq 0 \quad 1 \leq i \leq m \quad (3-25)$$

An expansion similar to the primal model transforms the above model (eqn 3-23, 3-24, 3-25) into:

$$\text{Minimize } Z_y = \sum_{i=1}^m b_i^+ y_i - \sum_{i=1}^m b_i^- y_i \quad (3-26)$$

subject to n "process" constraints:

$$\sum_{i=1}^m a_{ij}^+ y_i + c_j^- \geq \sum_{i=1}^m a_{ij}^- y_i + c_j^+ \quad 1 \leq j \leq n \quad (3-27)$$

and

$$y_i \geq 0 \quad 1 \leq i < m \quad (3-25)$$

The n constraints (equation 3-27) are called "process" constraints since they convert endogenous $(\sum_{i=1}^m a_{ij}^+ y_i)$ and exogenous (c_j^-) resource flows into output resource flows $(\sum_{i=1}^m a_{ij}^- y_i + c_j^+)$. As y_i represents the imputed value of the resource i , and c_j^+ and c_j^- represent the per unit benefit and cost of the transformation, each process constraint guarantees that the total value of the input resources and cost of transformation is at least as great as the total value of the output resources and the benefits accrued from this transformation process.

Components of RPMS

In its original form, only three nodal symbols were used with RPMS: circles, squares, and triangles.

A circle node is used to represent a resource, where resource is taken to mean anything that can place a limitation on the attainment of a desired result. As mentioned in Chapter I, the amount of a resource is a function of both its availability and its capacity. A circle node i represents a constraint of the primal model of a linear programming problem: $\sum_{j=1}^n a_{ij}^- x_j + b_i^+ \geq \sum_{j=1}^n a_{ij}^+ x_j + b_i^-$. The circle is subdivided into four parts. The top quarter, y_i , represents the "Shadow price" or value of the Lagrangian multiplier associated with the resource constraint computed by a linear programming simplex algorithm. The residue value, x_i , of the resource is given in the

bottom quarter. This corresponds to the "slack" or "surplus" variable value required to translate an inequality constraint into an equation.

$$x_i = \sum_{j=1}^n a_{ij}^- x_j + b_i^+ - \sum_{j=1}^n a_{ij}^+ x_j - b_i^-$$

The other two quarter sections of the circle are optionally used to tally the input flows,

$$\sum_{j=1}^n a_{ij}^- x_j + b_i^+$$

and the output flows,

$$\sum_{j=1}^n a_{ij}^+ x_j + b_i^-$$

through the node [Figure 3-1(a)] .

A square node is used to represent a process, where process is taken to mean the transformation of available resources to create new resources for further uses. The square represents a primal decision variable in a linear programming problem and a process constraint upon the resource transformation.

Like the circle, the square is divided into four parts. The top quarter, x_j , representing the amount of transformation that is to occur. The bottom quarter, y_j , is reserved for the "opportunity cost" or expected loss value of a variable not becoming basic in the final solution to the linear programming problem.

$$y_i = \sum_{j=1}^m a_{ij}^+ y_i + c_j^- - \sum_{j=1}^m a_{ij}^- y_i - c_j^+$$

The y_j variable can also be interpreted as the Lagrange multiplier associated with the non-negativity constraint imposed upon the primal variable x_j . The users of RPMS may optionally use the other two quarter sections of the square to tally the input flows,

$$\sum_{i=1}^m a_{ij}^+ y_i + c_j^- ,$$

and the output flows,

$$\sum_{i=1}^m a_{ij}^- y_i - c_j^+ .$$

The relationship between a constraint y_i and a primal variable x_j is established by using a directed edge to represent a_{ij} . The circle representing a constraint, y_i , and a square representing a variable, x_j , the interrelation is shown by connecting circles and squares via solid lines with arrowheads. These solid arrows represent the a_{ij} coefficients of the linear programming problem. These endogenous flows show the interrelation between the resources and processes of the linear programming problem.

The circles, squares, and solid arrows combined together are called the internal system.

The triangle is used to represent a terminal node. In linear programming, this corresponds to an objective function. Since each

linear programming model can be represented by a primal and dual model, two terminal nodes are given for each representation of a linear programming problem using RPMS. These terminal nodes are called "Source" and "Sink" depending upon the arrow directions, or "primal" and "dual" depending upon whether dashed arrows connect the node to squares or circles.

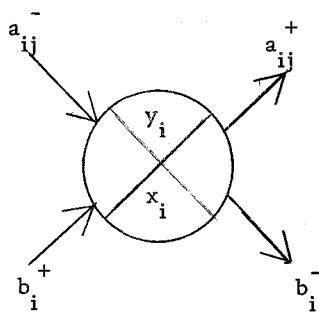
A maximizing primal objective function is shown by connecting a "primal sink" triangle to all process nodes (squares) via dashed arrows. The value of the objective function is entered inside the "sink" triangle.

The corresponding dual objective function is one of minimization. This is represented by connecting a resource (circle) to the "dual source" node terminal and entering the value of the objective function in the source triangle. A minimizing primal problem will have a primal source node and a maximizing dual sink node.

Both triangles are connected to the internal system via dashed arrows. A dashed arrow represents either an endogenous or an exogenous flow with respect to the internal system as designated.

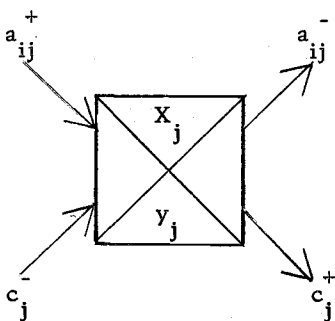
The internal system, combined with dashed arrows and triangles, makes up an RPMS network.

This interconnection of circles, squares, triangles, and arrows is portrayed in Figure 3-2.



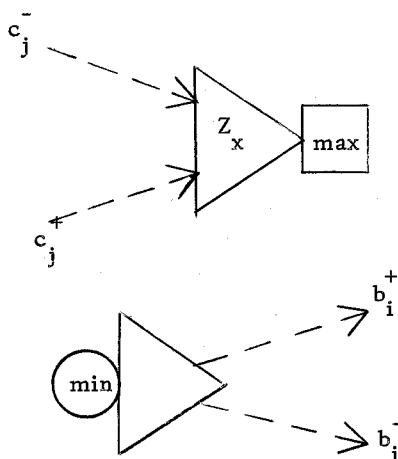
$$\sum_{j=1}^n a_{ij}^- x_j + b_i^+ \geq \sum_{j=1}^n a_{ij}^+ x_j + b_i^-$$

(a) Resource Node



$$\sum_{i=1}^m a_{ij}^+ y_i + c_j^- \geq \sum_{i=1}^m a_{ij}^- y_i + c_j^+$$

(b) Process Node



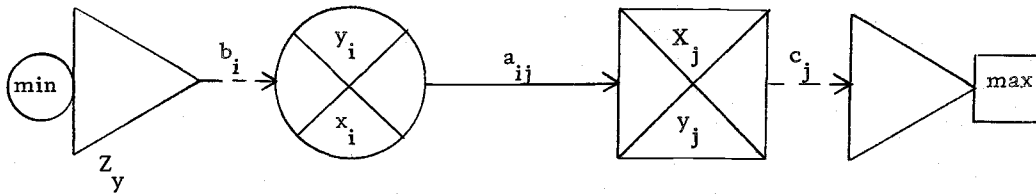
$$\text{Maximize } Z_x = \sum_{j=1}^n c_j^+ x_j - \sum_{j=1}^n c_j^- x_j$$

$$\text{Minimize } Z_y = \sum_{i=1}^m b_i^+ y_i - \sum_{i=1}^m b_i^- y_i$$

circle = i square = j

(c) Maximizing and Minimizing

Figure 3-1. RPMS Nodal Conventions.



$$\text{Min } Z_y = b_i y_i$$

$$\text{Max } Z_x = c_j x_j$$

$$b_i \geq a_{ij} x_j$$

$$a_{ij} y_i \geq c_j$$

$$b_i = a_{ij} x_j + x_i$$

$$a_{ij} y_i = c_j + y_j$$

Figure 3-2. RPMS Basic Flow.

Postulates of RPMS

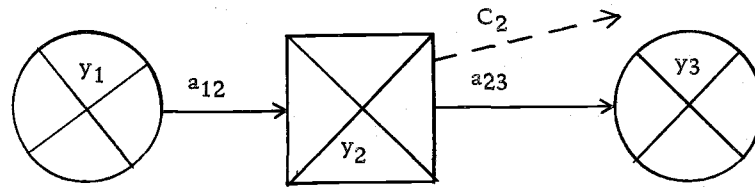
Two characteristics of an RPMS network have been formulated as postulates (Inoue, 1974).

The first postulate addresses the idea of balance around a node within an RPMS network. The second postulate addresses the idea of objective function optimality. These postulates correspond to the revised canonical forms given by equations 3-20, 3-21, 3-26, and 3-27.

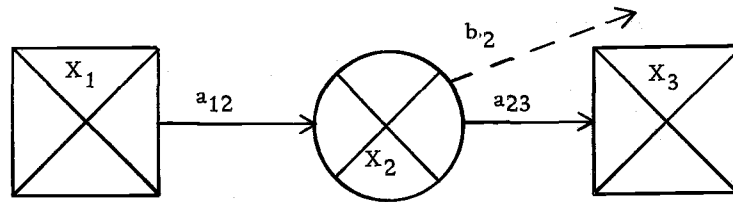
First Postulate of RPMS: The total inflow at a process or a resource node cannot be smaller than the sum of the outflows from the same node.

Total Input \geq Total Output

Total Input = Total Output + Residue



$$a_{12} y_1 \geq a_{23} y_3 + C_2 \quad \text{or} \quad a_{12} y_1 = a_{23} y_3 + C_2 + y_2$$



$$a_{12} X_1 \geq a_{23} X_3 + b_2 \quad \text{or} \quad a_{12} X_1 = a_{23} X_3 + b_2 + X_2$$

Figure 3-3. RPMS Cause and Effect Diagrams.

Second Postulate of RPMS: The productivity of an RPMS network is to be optimized, either by maximizing the net effective endogenous output while holding the exogenous input constant, or by minimizing the exogenous input while maintaining the endogenous output at a given level.

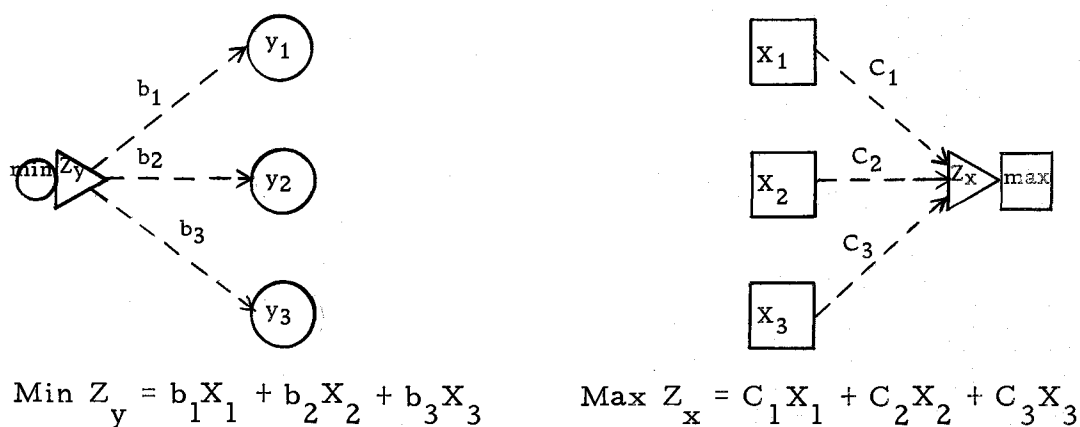


Figure 3-4. RPMS Terminal Nodes.

RPMS Conventions

In addition to the two postulates of RPMS, several conventions exist for the construction of RPMS networks (Riggs and Inoue, 1975).

The main conventions are presented here as a convenience to the reader. These conventions are used throughout this study.

- 1) A circle never connects to a circle and a square never connects to a square directly.
- 2) All squares are explicitly or implicitly connected to one terminal, and all circles are explicitly or implicitly connected to the other terminal.
- 3) The dimension of the solid arrow coefficient is always (Resource Unit/Process Unit).
- 4) Each circle adheres to the logic rules for "inclusive or."
- 5) Each square adheres to the logic rules for "logical product."

Construction of the RPMS Network

The actual construction of an RPMS network is relatively simple once the linear programming problem has been formulated. The construction consists of the following steps.

- 1) Draw a square for each variable in the objective function.
- 2) Identify the optimization type required in the objective function (maximize or minimize) and add the appropriate terminal nodes to the existing diagram (use Postulate Two).
- 3) Draw a circle for each constraint.
- 4) Complete the network using the equations and Postulate One.

The RPMS network is now ready to accept additional information after the linear programming problem has been solved.

Solution of RPMS Network

Though the solution of a linear programming problem is typically handled by a computer program, for linear programming problems, represented by simple RPMS networks, it is often more expedient to solve the linear programming problem without the aid of a computer. The following procedure, coupled with a hand calculator, has been used successfully for solving some problems. For problems which have a high level of interaction or take over one hour of manual computation, most users prefer to solve the LP problems by use of a computer.

Solution Procedure for RPMS Networks

1. Draw the RPMS network using the known values given in the problem statement.
2. Make a forward pass through the RPMS model. Calculate the maximum value of the residue for each resource. That is, calculate X_i if all X_j 's are zero.
3. The resources with the smallest value for X_i is then exhausted. This is done according to the ratio X_i/A_{ij} for each resource. This is like the pivot point calculation of the classical linear programming simplex algorithm.
4. The optimized output worth of the model is then calculated. This value of the sink terminal is then used as the same corresponding value of the source node. With this value known for the input of the model, the remaining coefficients in the model can be calculated using the first postulate. This is the calculation of the shadow prices.
5. Analysis. With the picture of the RPMS complete, the analysis can now take place to indicate where further improvements can be made for the benefit of the total system.

The following figure 3-5 shows conditions which can exist in RPMS networks. This information is used in determining if the current solution is the best that can be achieved, and, if not, where the improvement can be made.

The ideas in this figure were obtained from the paper on Visual Identification of Kuhn-Tucker conditions (Inoue, 1974).









| Condition | Descriptions | RPMS Node | |
|------------------------|--|---|--|
| A Non-Optimal | One or more negative Y_j values |  |  |
| B Non-Feasible | One or more negative X_i values |  |  |
| C Feasible and Optimal | No negative values for X_i or Y_j |  |  |
| D Degeneracy | An alternative solution exists for the same objective function value |  |  |

Figure 3-5. RPMS Feasibility and Optimality Conditions.

The following simple example is used to clarify the RPMS concepts presented in this Chapter.

Example One: Maximize $Z_x = 4X_1 + 3X_2$

Subject to $5X_1 + 3X_2 \leq 18$

$X_1 + X_2 \leq 20$

Step 1: The RPMS network is shown in Figure 3-6.

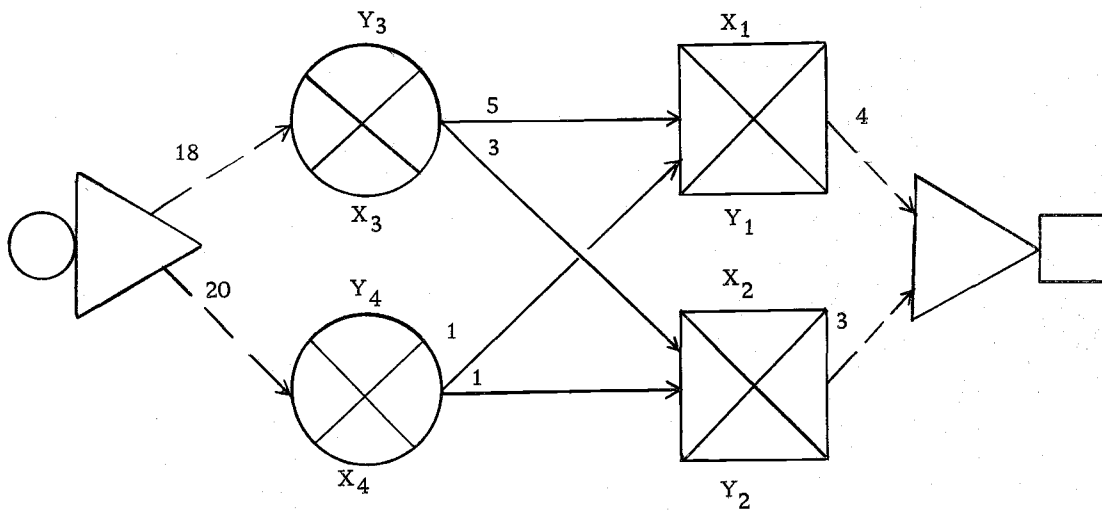


Figure 3-6. RPMS Network of Example, Step 1.

Step 2: Make a forward pass through the network with X_j 's = 0 for each process j . Calculate the value of the residue, X_i , for each resource i . Apply the complementary slackness theorem of Linear Programming (Taha, 1971) which states that $x_i y_i = 0$ for any process or resource node. In our example, $y_3 = 0$ since $x_3 = 18$ and $y_4 = 0$ since $x_4 = 20$. Then the process residues y_1 and y_2 can be computed by taking the difference between the inflows and outflows.

$$y_1 = 0 \times 5 + 0 \times 1 - 4 = -4$$

$$y_2 = 0 \times 3 + 0 \times 1 - 3 = -3$$

$$Z_x = 0 \times 4 + 0 \times 3 = Z_y = 18 \times 0$$

This step is shown in Figure 3-7.

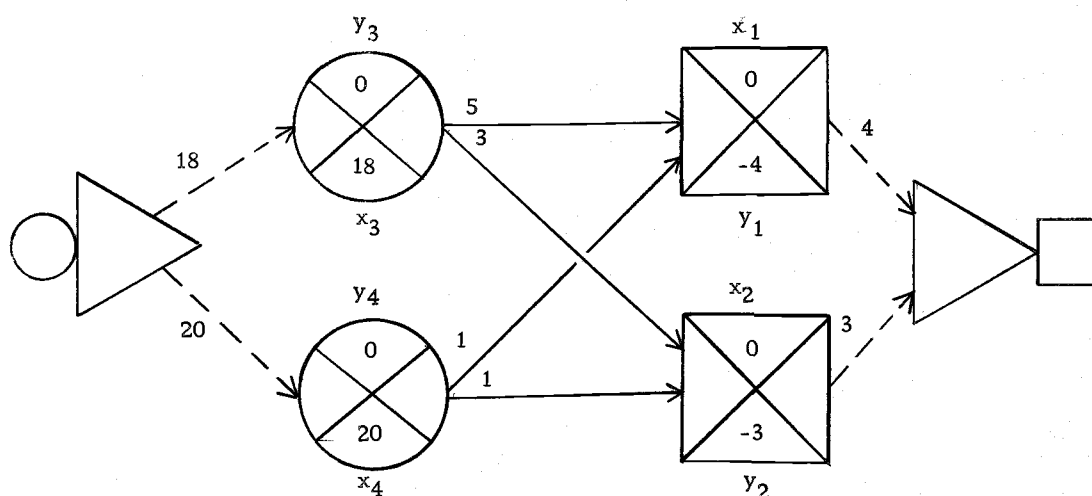


Figure 3-7. Example Step 2.

Step 3: Determine the "pivot point" as in a standard linear programming algorithm. In Figure 3-7, the solution is feasible but non-optimal (condition A in Figure 3-5). The most negative entry occurred at $y_1 = -4$, and our first attempt will try to make $y_1 = 0$ but rendering $x_1 \geq 0$.

The resource 3 with current residue value $x_3 = 18$ will be exhausted first as x_1 is increased. The most we can increase x_1 without creating an infeasible solution [condition (B) of Figure 3-5] is $18/5 = 3.6$.

Step 4: Using the new solution $x_1 = 3.6$ and $x_2 = 0$, the new residues and shadow prices are computed. The result is shown in Figure 3-8.

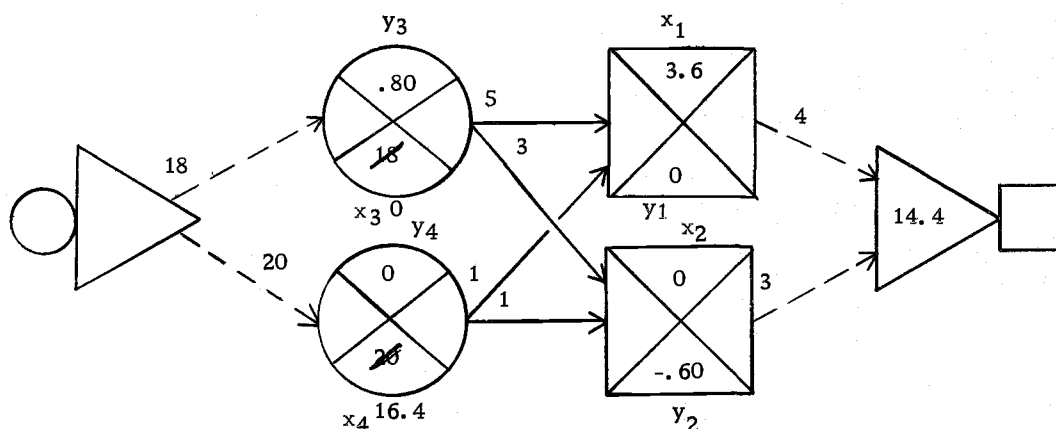


Figure 3-8. Example Step 3 and 4.

Step 5: Inspection of Figure 3-8 shows the solution to be feasible but non-optimal [condition (A) of Figure 3-5] since y_j of X_2 is negative. The objective function value is now increased to $Z_y = 18 \times .80 + 20 \times 0 = 14.4 = 3.6 \times 4 + 0 \times 3 = Z_x$. Step 3 of the next cycle brings X_2 into solution using both the top and the bottom resources. The result of this operation is shown in Figure 3-9. This solution is both feasible and optimal [condition (C) of Figure 3-5].

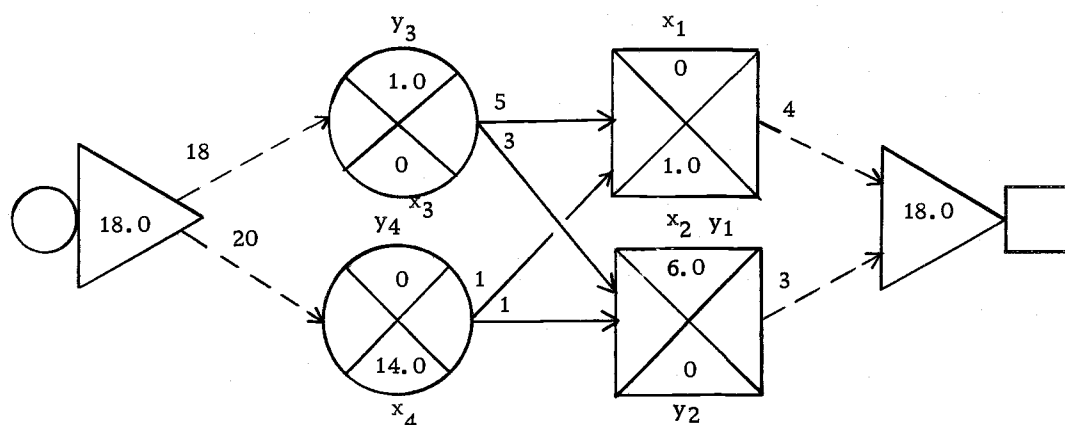


Figure 3-9. Example Final Solution.

This example used the solution procedure for RPMS networks described earlier in this chapter and showed it to be valid for the solution of this simple mathematical programming problem.

IV. PROPOSED APPROACH

The formulation of any mathematical programming problem is difficult, mainly because the constraint and objective equations are so elusive (Aoki, 1971). Stated in RPMS terms this is a problem of resource identification, process identification, and identification of the interconnections between the resources and processes. For CPE, some examples of resources are disk, magnetic tapes, and central processing units. Examples of processes are I/O flows, memory management, program compilation, and execution. Interconnections represent the rates at which resources are produced and consumed by processes.

One approach to CPE uses seven major steps (Bell, 1972).

These steps are as follows:

- 1) Understand the system; 2) Analyze the operations; 3) Formulate the performance improvement hypothesis; 4) Analyze the probable cost-effectiveness of modifications; 5) Test the hypothesis; 6) Implement the appropriate combinations of modifications; 7) Test the effectiveness of the implemented modifications, then go back to step three to repeat the cycle until sufficient improvements have been made.

Our study will use Bell's approach, but with modifications to have the procedure read as follows:

1) Understand the computer system subsystems of hardware and software. 2) Analyze the operations part of the system described in part one, and construct an RPMS model. 3) State the hypothesis as that of improving or maintaining the throughput; optimize the model using the linear programming procedure. 4) Identify the resource which is currently constraining the throughput rate. 5) Tune the system. This means to modify the constraining resource of part 4 while monitoring the throughput rate. 6) Rerun the linear programming problem of the newly tuned system. 7) Analyze the results of the modification made in step 5. Draw Kiviat Graph of resulting system. Go back to step 3 and repeat until sufficient improvement is made.

Developing a Universal CPE Model

When we cannot grasp a system as a whole, we try to find divisions such that we can understand each part separately, and also understand (in the framework) how they interact (Minsky, 1967).

A model is normally defined as a copy of a true situation in as accurate a form as possible (Gordon, 1969). But, all models have inherent problems in that a boundary between the model and its environment must be established somewhere. Thus, if the model results are found to be inadequate the general tendency is just to expand the boundaries of the existing model. This method works until the upper bound of the modeling device is reached.

But, even with expansion, the model can usually use further improvement after the upper bound has been reached.

One then normally chooses to break the system into its subjects and analyze these subparts until full understanding is obtained. Usually, some form of a decomposition algorithm is employed. But the problem of understanding the composite model remains. The problem just moves from that of the system to the understanding of the interconnection of model components.

One useful tool which allows an easy application of decomposition to CPE models is RPMS. The major advantage is that mathematical programming techniques are as applicable to decomposed parts of RPMS as they are to the composite model.

To formulate a universal model for all computer systems, a general conceptual framework needs to exist that can be adapted for each specific system.

One model which accurately depicts current mid 1970's type computer systems contains three subsystems (Stimler, 1974). The subsystems are called the 1) Processor 2) Communications and 3) Terminal. Stimler's model is shown in Figure 4-1.

The functions of each of the component parts is as follows:

Processor: The computational portion of the system. It accomplishes the transformation process from one resource to another resource.

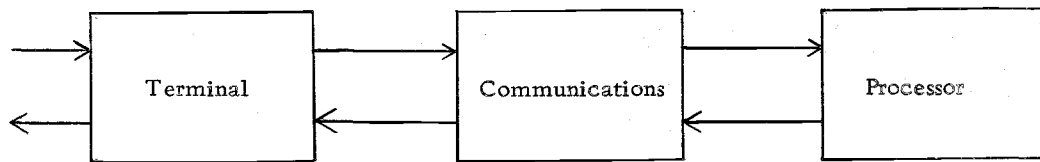


Figure 4-1. Stimler's Model.

Communications: An interface. It matches resources to processes. It can be considered to be transparent.

Terminal: The pool of resources the system has available (Stimler, 1969, p. 6).

An expansion of the basic Stimler Model yields two terminals, two communicators, and one processor. This is shown in Figure 4-2.

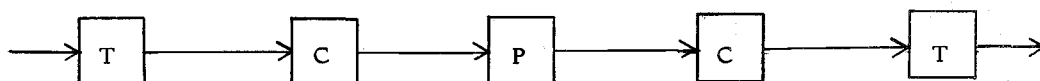


Figure 4-2. Expanded Stimler Model.

RPMS components are directly analogous to these subsystems. The Processor is the square, while the terminal is a circle, and the communications corresponds to the flows.

Accepting these analogies between RPMS components and the Stimler subsystems, Figure 4-2 becomes Figure 4-3.

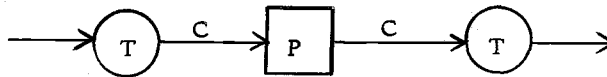


Figure 4-3. RPMS Diagram of Stimler Subsystem.

Stimler's model viewed in RPMS terms will be used in the next section to portray a computer system which will be evaluated by using linear programming and the procedure outlined in the previous section.

The modified Bell's procedure previously stated in this chapter will be followed to obtain analytical results. The resources, processes and their interrelations are formulated as a linear RPMS model and solved as a standard linear programming problem.

The numerical examples in this study were all solved using two methods. The first method followed the RPMS procedure outlined in Chapter III, and the second utilized the *REX linear programming package "a revised product-form, composite, bounded variable, multipracing, simplex, linear programming algorithm" (Scheurman, 1970).

V. THE RPMS APPROACH APPLIED TO CPE: BATCH VS ON-LINE

A numerical example that uses the proposed RPMS approach to evaluate an existing system to determine which resources are being utilized and to what extent the improved throughput contributes to the system's overall profit.

A small system has been set up which gives the user the option of running batch or on-line jobs. A batch job utilizes a card reader input and line printer output. The on-line jobs run with teletype input and can have either teletype output or line printer output. The objective is to maximize the profit of the computer center throughput.

The card reader operates at 1200 cards per minute. The teletypes operate at 120 lines per minute. The line printer operates at 600 lines per minute.

Each batch job has an average of 500 cards of input and 500 lines of output, running in a total CPU usage time of one second, at a profit of \$50 per job.

Each on-line job has an average of 600 lines of input and 600 lines of output, running in a total CPU usage time of two seconds, at a profit of \$120 per job. Let us now assume that a CPE study is to be made for a time quantum of ten minutes.

Thus, the total time allowed for all processing, including CPU, is ten minutes per resource. The system allows spooling of all input

and output devices, and an assumption is made that one card is equal to one line of print and vice versa.

This problem, stated as a mathematical programming problem is as follows:

Maximize $Z = 50 \text{ BUT} + 120 \text{ LUT}$ subject to the following constraints

$$@ \text{ SYST} \quad \text{CTR} \leq 600$$

$$@ \text{ CR} \quad 500 \text{ BIN} - 20 \text{ CTR} \leq 0$$

$$@ \text{ LP} \quad 500 \text{ BUT} + 600 \text{ PRLT} - 10 \text{ CTR} \leq 0$$

$$@ \text{ CPU} \quad 1 \text{ BPR} + 2 \text{ LPR} - \text{CTR} \leq 0$$

$$@ \text{ TTY} \quad 600 \text{ TTYUT} + 600 \text{ TTYIN} - 2 \text{ CTR} \leq 0$$

$$@ \text{ BNQ} \quad \text{BPR} - \text{BIN} \leq 0$$

$$@ \text{ PLNQ} \quad \text{LIN} - \text{TTYIN} \leq 0$$

$$@ \text{ LNQ} \quad \text{LPR} - \text{LIN} \leq 0$$

$$@ \text{ BUQ} \quad \text{BUT} - \text{BPR} \leq 0$$

$$@ \text{ LPLQ} \quad \text{LUT} - \text{PRLT} \leq 0$$

$$@ \text{ LUQ} \quad \text{LUT} - \text{LPR} \leq 0$$

Where, all units are jobs/10 minutes except as noted.

BIN is the process of reading cards from the card reader into a batch queue. BIN is Batch INput.

BNQ is the resource built by BIN. BNQ is the Batch INput Queue.

BPR is the process of doing the actual batch computation.
BPR is Batch PRocessing.

BUQ is the resource built by BPR. BUQ is the Batch oUtput Queue.

BUT is the process of outputing batch jobs. BUT is the Batch outpUT.

CPU is the Central Processing Unit resource.

CR is the Card Reader resource measured in cards.

CTR is the process of operating the computer center.
CTR is the CenTeR.

LIN is the process of accepting on-line input.
LIN is on-Line INput

LNQ is the on-line input queue. LNQ is on-Line iNput Queue.

LP is the line printer Resource. LP is Line Printer measured in lines.

LPLQ is the on-line resource of pre-output queue.
LPLQ is the Line Printer on-Line Queue.

LPR is the processing of on-line jobs. LPR is on-Line PRo-cessing.

LUQ is the resource of on-line output queue. LUQ is the on-Line oUtput Queue.

LUT is the process of producing on-line output.
LUT is on-Line oUTput.

PLNQ is the resource of jobs ready for on-line input.
PLNQ is Pre-on-Line-iNput-Queue.

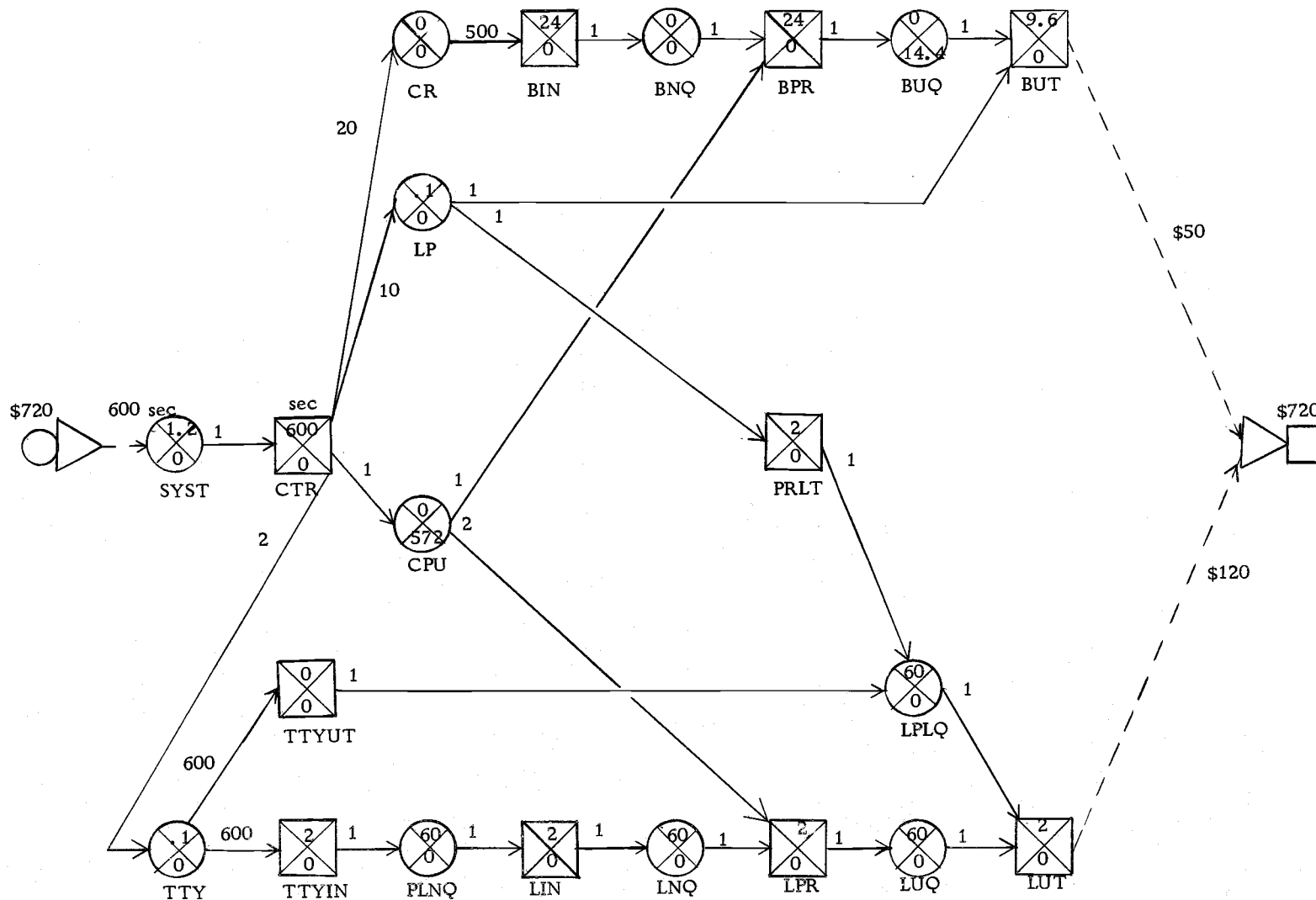


Figure 5-1. Basic Applied Model.

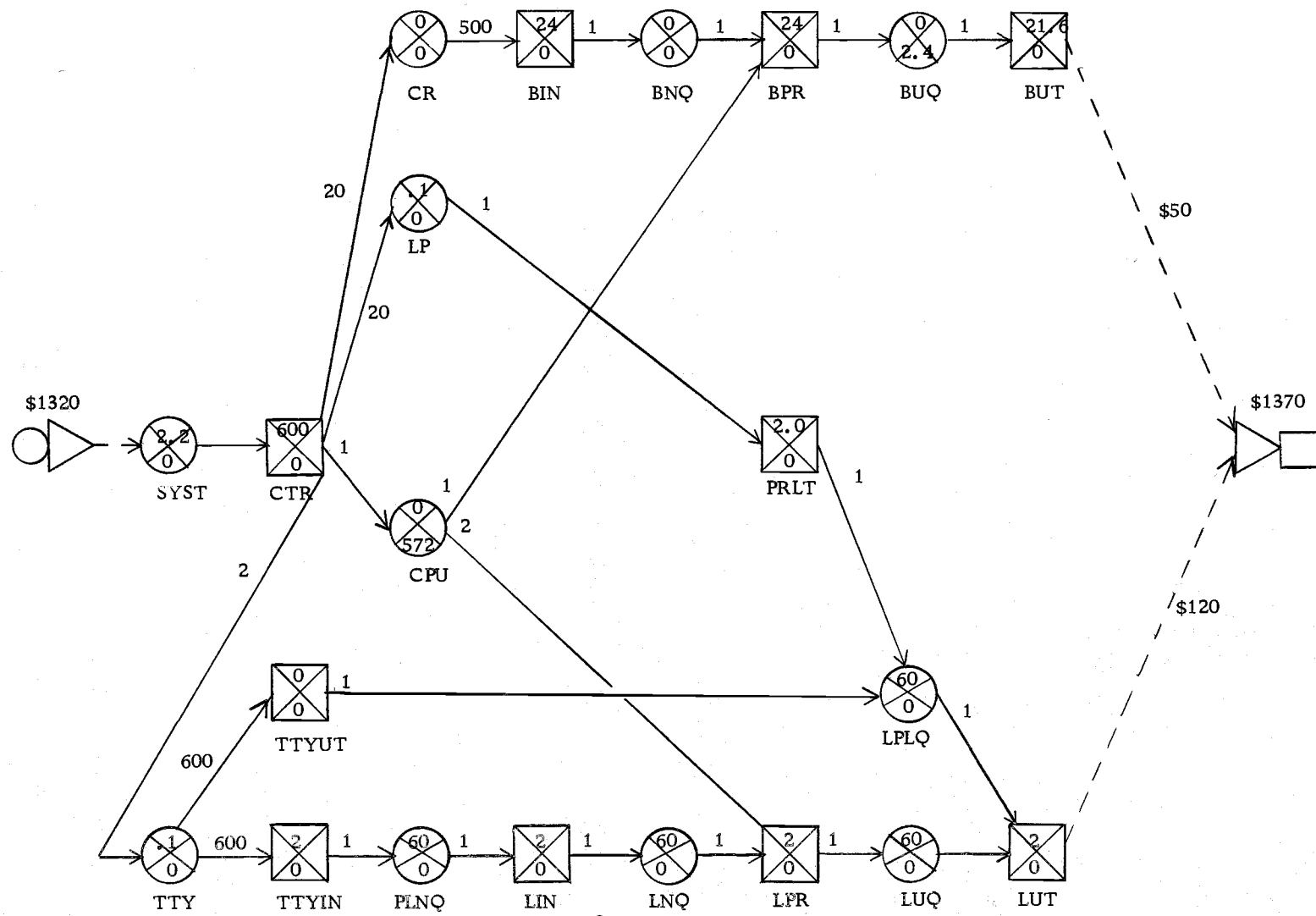


Figure 5-2. Double LP.

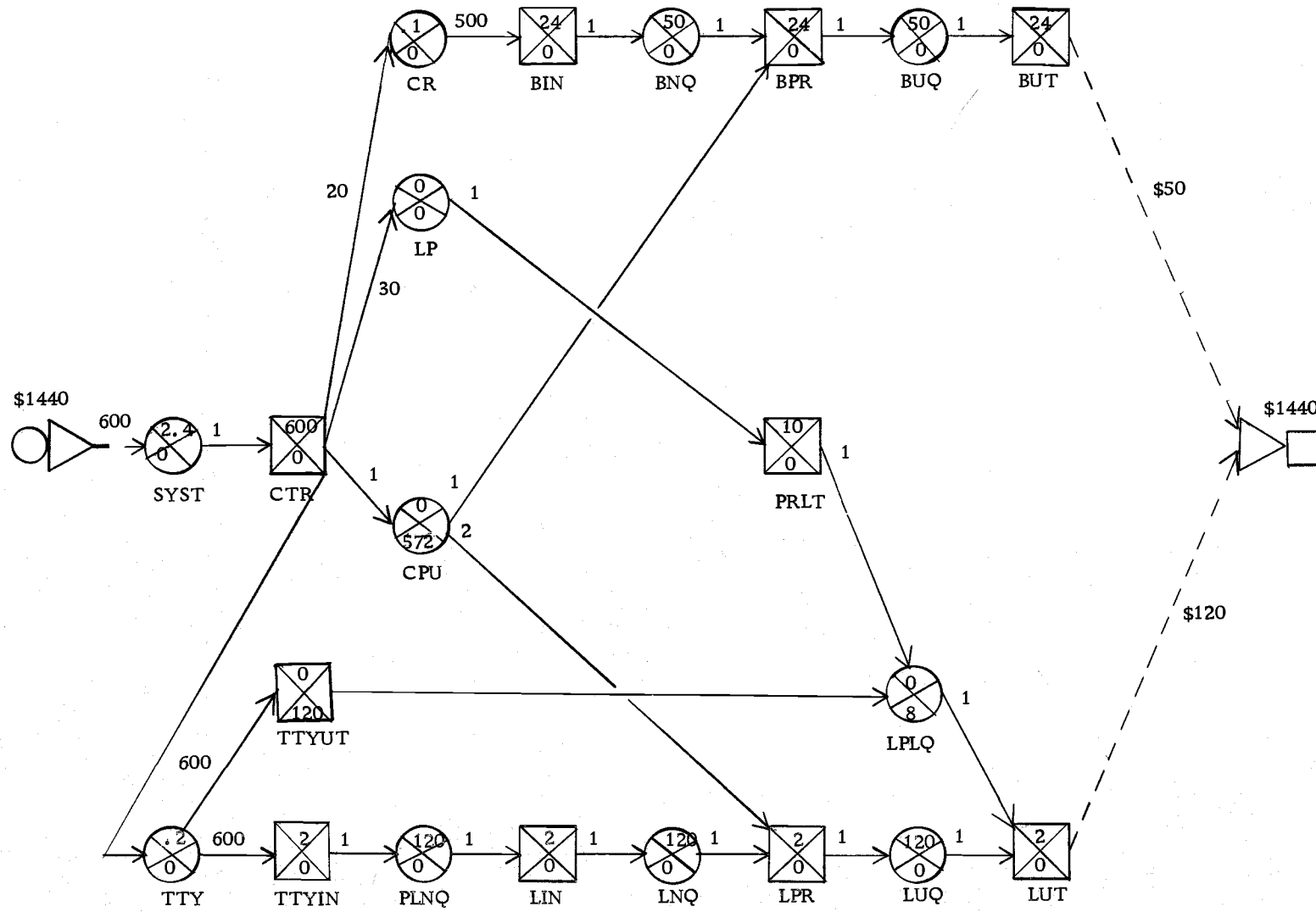


Figure 5-3. Tripple LP.

PRLT is the process of allocating the line printer to the for on-line use. PRLT is PRinter for on-Line output.

SYST is the resource of time. SYST is SYStem Time.

TTY is the resource of teletypes. TTY is TeleTYpes measured in lines.

TTYIN is the process of allocating teletypes for on-line input.

TTYIN is TeleTYpe INput.

TTYUT is the process of allocating teletypes for on-line output.

TTYUT is TeleTYpe outpUT.

These mneumonics are used on the RPMS network portrayal of the problem and in defining *REX variables. The RPMS network of the problem is shown on Figure 5-1.

The effect of doubling the line printer capacity is shown on Figure 5-2.

The effect of tripling the line printer capacity is shown on Figure 5-3.

Discussion of Results

In the solution of the basis problem, several observations can be made from the RPMS network of Figure 5-1.

- * Each second of system time, SYST, is values at \$1.20.

- * Only a total of 28 seconds are consumed for all jobs run

(600-572 at CPU)

- * A total of 24 batch jobs were processed, BPR, but only 9.6 jobs were actually output on the line printer, BUT, while a queue of 14.4 batch jobs remain to be output, BUQ.
- * All on-line jobs used teletype input, TTYIN, and line printer output, PRLT.
- * A total of two on-line jobs were completely run and output, (LIN=LPR=LUT=2), and no on-line jobs were on any queues (Residue part of PLNQ, LNQ, LUQ=0).
- * The current constraining factor for doing more batch jobs is the line printer. LP shadow price value is \$.10.
- * Either the Line Printer, LP, or the Teletypes, TTY, capacity can be increased and a resultant increase of the total profit will increase by \$.10 per line output.

Since these two resources will both contribute the same amount to the total profit, a degenerate condition (condition D, Figure 3-5) exists. Between the two resources, it will be best to increase the resource that contributes the most in terms of flexibility to the system. Thus, since the line printer is a shared resource (that is, it feeds both BUT and PRLT), it is found to be more advantageous to double the line printer capacity.

Figure 5-2 gives information about the system when the line printer capacity is doubled (a_{ij} from CTR to LP = 20).

- * When doubling the line printer capacity, the CPU utilization (CPU), Batch Jobs Processed (BPR), and on-line jobs completed (LUT), remained constant.
- * The total batch jobs completed (BUT) has increased to 21.6, while the batch output queue (BUQ) has decreased to 2.4.
- * The system time (SYST) is now valued at \$2.20 per second.
- * The two constraining resources are the line printer (LP) and the Teletype (TTY). By applying again the same logic that was used on Figure 5-1 to Figure 5-2, it was decided to tripple the line printer capacity from its original capacity.

Figure 5-3 gives information about the system when the line printer capacity is trippled (a_{ij} from CTR to LP = 30).

- * By tripling the line printer capacity, the CPU utilization (CPU), Batch jobs Processed (BPR), and on line jobs completed (LUT) remained constant.
- * The amount of batch jobs completed (BUT) increased to 24, and thus reduced all batch queues, BNQ and BUQ residue values to zero.
- * Each system second (SYST) is now valued at \$2.40.
- * If further improvement is to be made in the system to increase the amount of throughput, then two resources, the card reader (CR) and the teletype (TTY) are candidates. Since the teletype is valued at \$.20 per line produced and the card reader

is valued at \$.10 per card produced, then the teletype is the best candidate for the next capital investment.

In summary, it took tripling the line printer capacity, that is changing a_{ij} from CTR to LP from its original value of 10 to 30, to double the profit of the system and set the batch output queue (BUQ) to zero. Figure 5-4 shows the Kiviat graphs of the results just described on the RPMS networks.

Since this study uses an increase in throughput as being the hypothesis that is to be improved, then the amount of batch output (BUT), on-line output (LUT), and the operational level of the center (CTR) are considered as the good attributes for the Kiviat graph.

Since partially completed jobs constitute no revenue for a system, the batch output queue (BUQ), the line printer queue (LPLQ), and system residue (SYST) were chosen as the bad attributes. The system would have attained a star shape [Figure 5-4(c)] had the optimal LP capacity of 22 been chosen. The Kiviat graphs in Figure 5-4 show that some improvements were made in the system by changing the line printer capacity.

The hardware and software were combined in this RPMS model and this is in keeping with the current trend of integrating the two components (Falk, 1975, pg. 46). The model also assumed the operations time by the operator to be zero. This is in keeping with the

"Good"

"Bad"

Axis

Axis

1 = Batch Output (X BUT)

2 = Batch Output Queue (X BUQ)

3 = On-Line Output (X LUT)

4 = Line Printer On-Line Queue (X LPLQ)

5 = Operate Center (X CTR)

6 = System Time (X SYST)

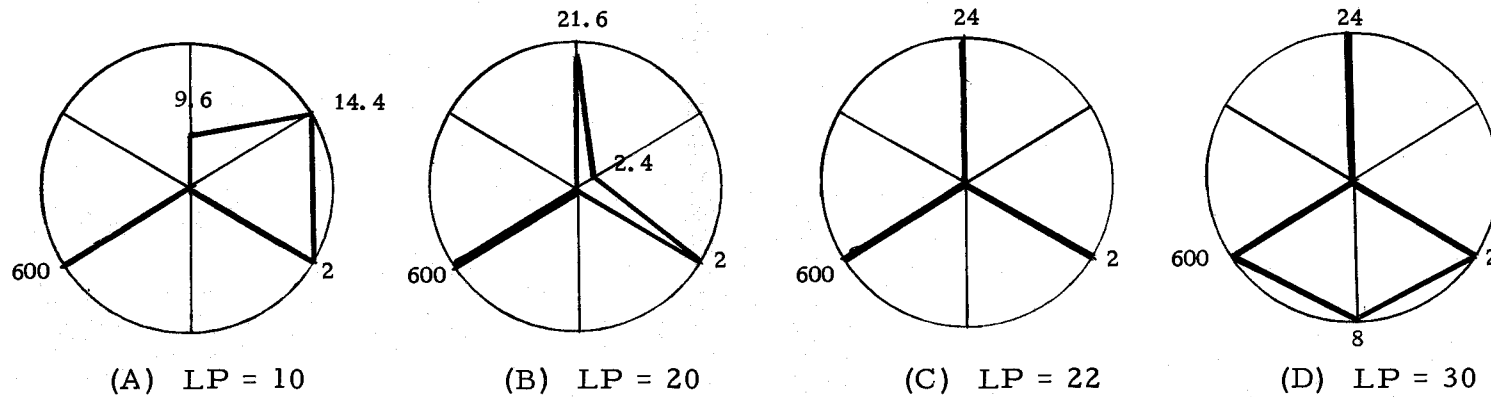


Figure 5-4. Kiviat Graphs of Batch vs On-Line.

idea of minimizing the amount of operator intervention in current computer systems (Stimler, 1969).

With these assumptions made, then the throughput of the system is just the sum of X_{BUT} and X_{LUT} .

Throughput = $\sum_{i \in I} X_i$ where I is a cut set of the process nodes of the network as defining the system throughput.

Though this is a crude upper bound, it gives an indication of the maximum throughput under idealized conditions for the system represented by the RPMS network under study.

VI. EXTENSIONS OF RPMS THEORY

The following extensions of existing RPMS theory are offered to support the observation that many of the concepts embodied in the RPMS methodology already existed in other branches of the field of computer science. By making further use of theories in these fields, the RPMS theory can be extended. It is hoped that these extensions will aid in popularizing the use of the RPMS as a tool for computer performance and evaluation efforts.

The following analogy from finite state machine theory is offered to show that the RPMS is compatible with the basic theories from this discipline.

Each computer system can be viewed as just a black box, M, with an input stimulus, S, and an output response, R (Minsky, 1967, pg. 13).

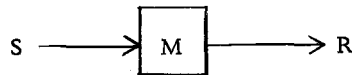


Figure 6-1. Finite State Machine.

This system can be described more formally as follows (Ibid., pg. 16 and 17):

$$M = (K, \Sigma, \delta, S, F)$$

where K = The states of the machine

Σ = The input alphabet of the machine

δ = The transitions from one state to another state

S = The start state of the machine

F = The final states of the machine

$$R(t+1) = F(Q(t), S(t))$$

where $Q \in K$

$S \in \Sigma$

F is a function

$$Q(t+1) = G(Q(t), S(t))$$

$Q \in K$

$S \in \Sigma$

G is a function

Even this simple model of Figure 6-1 yields more information than is obvious. S and R can be considered as resources while M is a transformation process. In RPMS terms, assuming S and R to be resources, and M to be a process, will produce Figure 6-2 which is directly analogous to Figure 6-1. The basic problem then within any

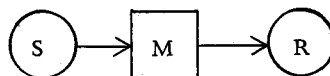


Figure 6-2. RPMS Representation of Finite State Machine.

systems model is the establishment of the boundaries. From an RPMS point of view, the boundaries are more easily formulated if the model starts with a resource and ends with a resource. But all resources come from somewhere and are going somewhere. Since a transformation process is required to change one resource into another resource, a resource must come from a transformation and feed into another transformation. With this in mind, then, each finite state machine is viewed as an RPMS of Figure 6-2. It will have its R value matched to the S value of the next machine and its S value corresponding to the R value of the machine in front of it. This is shown as Figure 6-3.

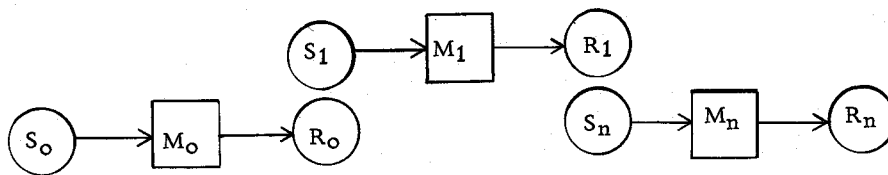


Figure 6-3. RPMS S & R Relationships.

By combining all overlapping S and R resources a concatenated machine (Figure 6-4) will be produced. If we assume the first M to be

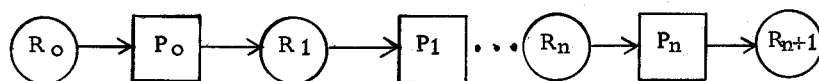


Figure 6-4. Concatenated RPMS Finite State Machine.

a computer system, it can be broken down into subsystems. As seen previously, one classification of subsystems uses the three terms "Terminal," "Communications," and "Processing." These three terms have been shown to be more applicable to modern computer systems than the more widely used Von Neumann or Hellerman models (Stimler, 1974).

The Von Neumann model is also known as the four-block, five-block, or classic model. A diagram of the model is shown in Figure 6-5 (Enslow, 1974, pg. 8).

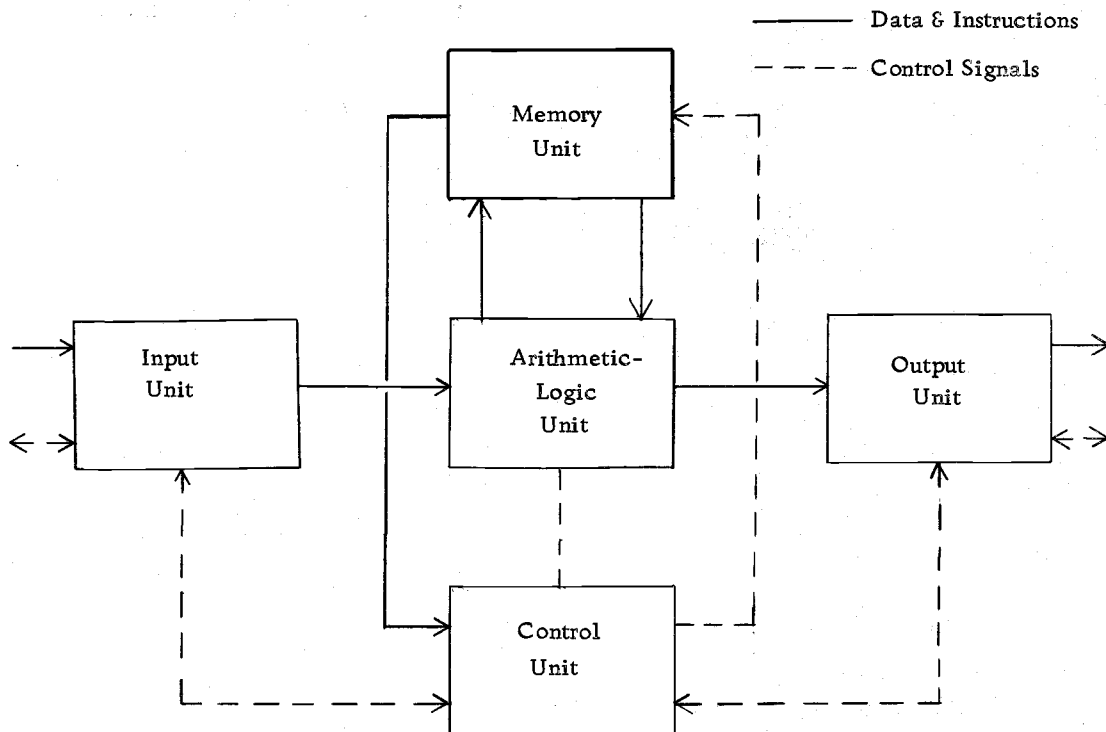


Figure 6-5. Von Neumann Model.

The blocks of the Von Neumann model have the following meanings:

Input/Output is used to transform information from human-consumable form into machine-consumable form and vice-versa.

Arithmetic-Logic performs the arithmetic and logical functions asked for by the control unit.

Control interprets the instructions to be executed and controls there execution.

Memory holds the program and data for the program which is currently being executed.

The major weakness of this system organizational scheme is that all I/O operations are routed through the Arithmetic and Logic unit. This reduces the total number of hardware elements required, but all computation must halt while input/output operations are in progress (Ibid., pg. 7). The Hellerman model is described as by the following and can be graphically shown by Figure 6-6 (Hellerman, 1973, pg. 9-12).

Storage block provides a means for storing a large volume of information from/to storage from/to a single point (register).

Data Flow comprises the switching networks that route information from one part of the computer to another. No unit is permanently connected to any other unit.

Transformation provides the arithmetic and logic circuits used in the data manipulation process.

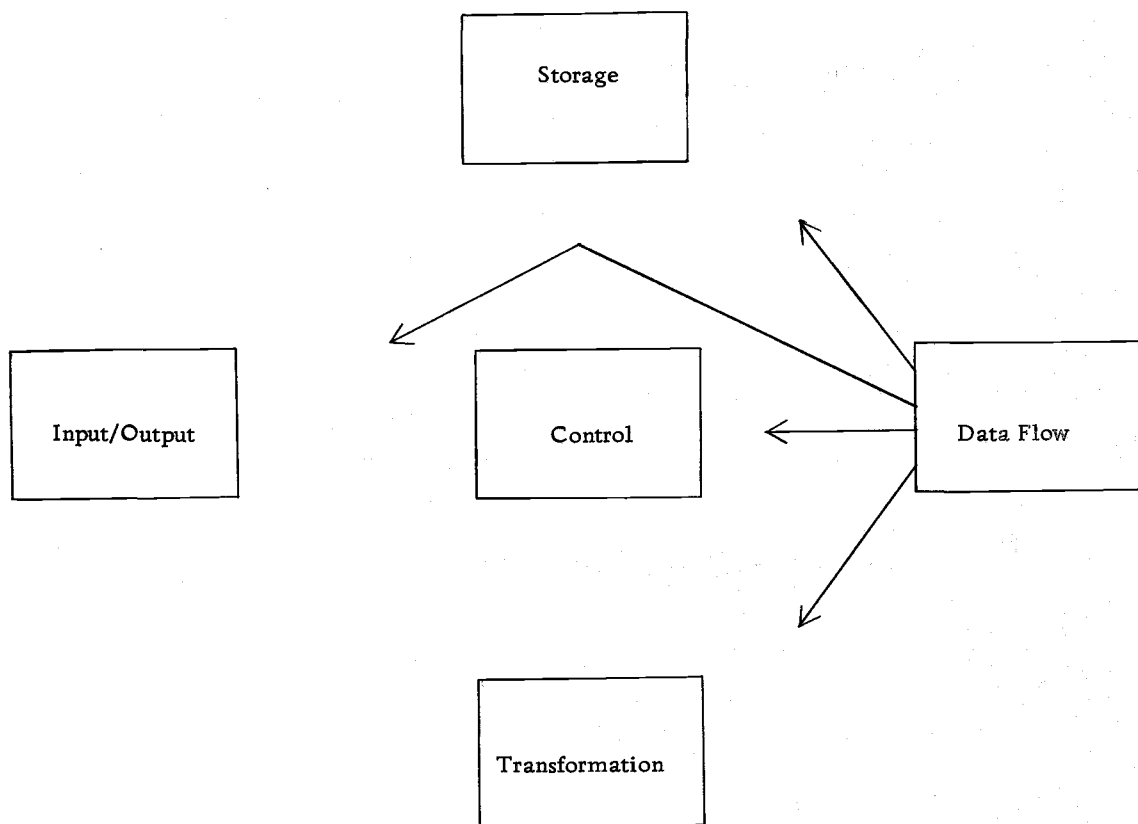


Figure 6-6. Hellerman Model.

Control provides the timing sequences that perform the instruction to be executed. Control exist at many levels in a computer.

Input/Output converts information from human-generated to machine-readable from and vice versa.

Hellerman further points out that the distinction between the Storage and Input/Output function is imprecise. The best distinction being whether or not the output is directly machine-readable.

While this model allows for simultaneous operation of I/O and CPU, it does not provide for the parallism of current computers.

Three divisions of the system which run in parallel are "operations," "software," and "hardware" under a common clock. These three subsystems cannot have time and cost attributes assigned until the structure of the system is known. In keeping with the current trend towards "hard software" or "soft hardware" (Falk, 1975, pg. 46) these two areas tend to merge into "firmware." The "operations" subsystem still exist, but for most modern systems, one goal is to minimize the amount of system dependence upon operators. Thus, the operations subsystem can be said to tend towards zero in looking at the total amount of system time that it consumes.

Let us investigate the processes and resources more closely. Each resource comes from a process and goes to a process, while each process comes from a resource and goes to a resource. Therefore, each of these types of nodes will have at least one line in and

one line out, but it can have multiple lines. These multiple type nodes are shown in Figure 6-7.

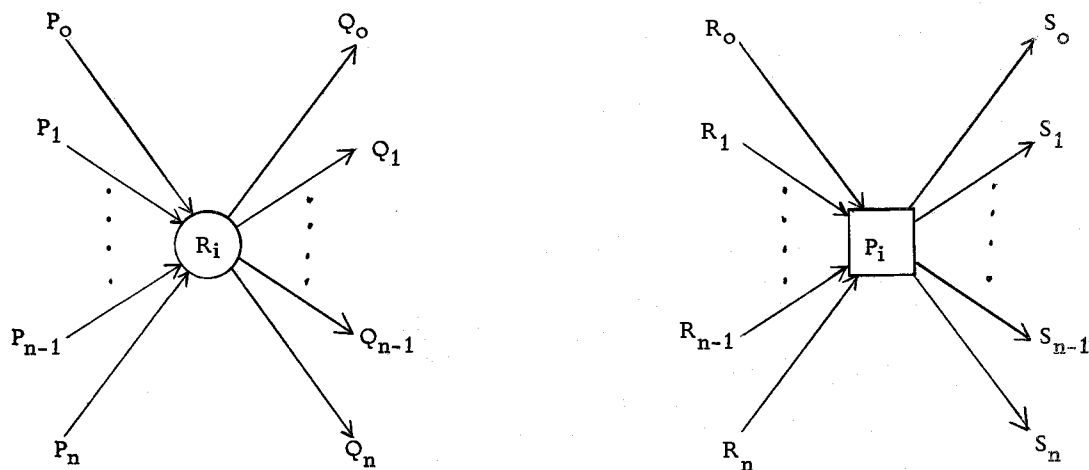


Figure 6-7. Multiple Nodes.

If we consider then the composite system broken into three subsystems we can have multiple resources and multiple processes with multiple lines. This then realizes the composite computer system in a cascade because any one of the resources can connect to any process in any subsystem and vice versa.

The advantage of this technique is that only the subsystems and interconnections of interest can and will be developed as required with respect to the other subsystems.

This finite state approach to RPMS could have been taken to develop the existing RPMS methodology. Given that the RPMS

representation of a computer system can be obtained, it is useful to manipulate the RPMS network.

Due to the size limitations of most mathematical programming packages, it is useful to remove any constraint or variable from the problem which is not a part of the solution. In RPMS terms, this says the removal of a pair of nodes is possible. The following postulate on compaction sums the ideas up in more formal terms.

Postulate of Compaction: Given a single strand of an RPMS network sequence, with resource nodes, $i=1, \dots, n$ and process node, $j=1, \dots, m$, and no other path connected to intermediate nodes, the entire strand can be compressed into a node or a node pair.

This postulate of Compaction was first developed by Chen (1974) as two separate theorems (Ibid., pg. 135, 136). It is presented here as being potentially useful in CPE efforts.

The postulate says that all resource are pushed toward the source node while all processes are pushed toward the sink node.

The example in Chapter V will now be used to illustrate the procedures.

From the basic model, Figure 6-1, the nodes of PLNQ, LIN, and LNQ can be combined into a new node called NINQ, for new INput Queue. The original path involves the nodes: shown in Figure 6-8. The compacted path is shown in Figure 6-9.

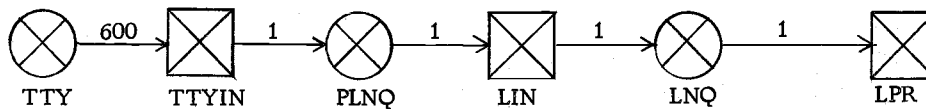


Figure 6-8. Original On-Line Path.

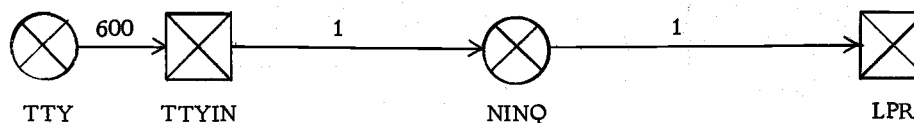


Figure 6-9. Compacted On-Line Path.

When compaction of the RPMS network is not possible, then other methods must be found to allow solving the RPMS network. Calling upon "the block and cut point theorem" from Graph Theory, and expressing it in RPMS terms yields the following postulate.

Postulate on Decomposition: Any RPMS network, which can be broken into multiple networks by cutting at a single node, portrays a mathematical programming problem solvable in sections.

This postulate was originally developed by Inoue and Riggs (1972a).

At times, the existing RPMS network needs to be expanded to include new information and potentially improve upon the answer obtainable from the application of mathematical programming solution techniques. The following postulate describes this expansion process.

Postulate on Expansion: Any RPMS network node can be expanded by replacing that node with a triplet of nodes. This can be expressed in terms of a context-free grammar as follows:

$G = (V_n, V_t, P, S)$ where

V_n = Set composed of P, R, and S

V_t = Set composed of p and r

S = Start symbol

P = Productions as given below,

$S ::= rPr \mid pRp$

$P ::= PRP \mid p$ a_{ij} is assumed equal to one.

$R ::= RPR \mid r$

This postulate will increase the total number of nodes in the network, but since it is a serial expansion, will not change the composite network structure.

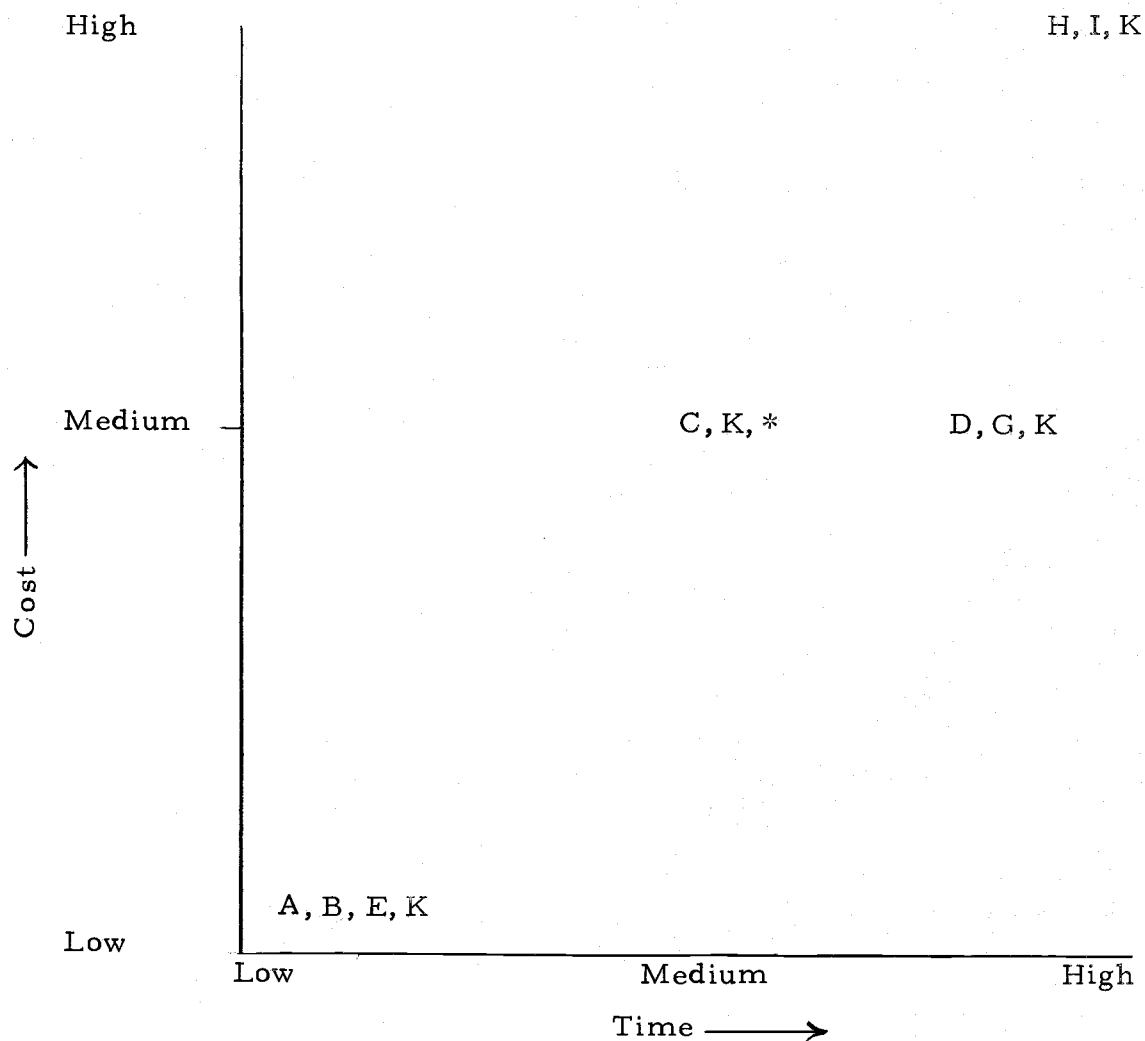
VII. CONCLUSIONS

General and Summary

The proposed approach has been found successful in obtaining an upper-bound on the amount of system throughput of a given computer system for which a RPMS model has been built and solved. Though the bound may be crude, its refinement is proportional to the amount of detail in the model.

Compared with other techniques its major advantage is that the method identifies the optimal allocation of resources in an existing feasible system being subjected to CPE. This is because the model is interpreted as a mathematical programming problem and solved to provide an optimal solution. An adaptivity analysis is possible by considering all information given as a result of the modification of the existing system. In addition, the technique allows for an estimate of throughput from just the vendor supplied system component characteristics.

The major difficulties are in the level of accuracy of the upper-bound. Further, solution of CPE problems using mathematical programming techniques has not met with a great deal of acceptance due to the level of mathematical maturity required and the absence of inexpensive mathematical programming packages.



A = Formulation

B = Instruction Mixes

C = Kernels

D = Benchmarks

E = Synthetic Models

K = Kiviat Graphs

G = Monitors

H = Mathematical Programming

I = Simulation

* = New Approach

Figure 7-1. Time and Cost Comparison of CPE Techniques.

The method proposed combines the properties of mathematical programming, synthetic modeling, and Kiviat graphs. The new approach can be added to Table 1 with the following entries:

| | Total Time <u>Required</u> | Total Cost <u>Required</u> | Selection | Optimization |
|--------------|----------------------------------|----------------------------------|-----------|--------------|
| New Approach | Medium | Medium | Good | Good |

All of the techniques in Table 1, along with the New Approach are shown in Figure 7-1. The new approach is considered to be a combination of the best of each of the techniques it draws upon. It is hoped that this approach will put CPE within the range of all computer system users.

Future Areas of Research

This study has concentrated on developing a versatile tool for analytical work dealing specifically with the computer performance and evaluation field. Though the proposed approach, as applied here, gives useful results, much is left to be done. Some of the major areas are indicated below.

1) Goal Programming: Often a CPE evaluation effort involves satisfying conflicting objectives. For example, it may be desired to increase throughput while minimizing capital expenditures. One modification of linear programming that holds promise for solving problems of this type is goal programming (Lee, 1972).

Developed by A. Charnes and W. W. Copper, Goal programming was further refined into a distinct mathematical programming technique by Ijiri (1965).

The major advantage of goal programming is that it provides a potential solution to problems involving conflicting objectives. Linear programming suffers from a unidimensionality of the objective function, while goal programming allows for multiple objectives.

Lee (1972) points out that since goal programming is relatively new, its true potential is yet to be determined. It appears that the overall potential applicability of goal programming may be at least as wide and far reaching as linear programming, since LP may be considered a subset of goal programming.

Lee's goal programming package was modified to run on the CDC 3300 at Oregon State. But, due to existing system limitations, only small problems involving up to ten resources and ten processes could be handled. Lee's goal programming package was subsequently modified to run on the CEC Cyber 73 under KRONOS 2.1. The program will currently compile for 100 processes, 250 resources and 10 priority levels.

3) Stochastic Programming: Stochastic programming deals with situations where parameters of a problem are random rather than deterministic quantities (Taha, 1971, pg. 649). This form of programming can be used to address real-life problems which are

non-deterministic. Computer Performance and Evaluation efforts involving the optimization of on-line computer systems could potentially benefit from stochastic programming applications for various areas where only the statistical distribution of the level of resource request is known. An example of this is a problem in the memory allocation area.

4) Functional Levels: If RPMS is to be accepted as a universal tool for CPE efforts, then a central common ground among all evaluation methods needs to be found. One proposed approach to the analysis of computer systems as a part of computer networks involves a hierarchical approach call 'Functional Analysis' (Booth, 1973). The functional analysis method takes the approach that a system can be decomposed into its sub-systems. RPMS has been shown to allow for decomposition, adaptation of RPMS to functional analysis could give new insight into CPE. The use of RPMS might allow further development of more levels within the existing functional analysis level structure (Becker, 1973).

Though the Resource Planning and Management System is not a panacea for all things, this study has shown it to be useful in gaining more insight into the solution of computer performance and evaluation problems.

BIBLIOGRAPHY

- Aoki, Masanao. 1971. Introduction to optimization techniques: fundamental applications of non-linear programming. New York, Macmillan. 335 p.
- Arbuckle, R. A. 1966. Computers and automation, "Computer analysis and throughput." Jan., 1966. p. 12-25, 19.
- Becker, Hal B. 1973. Functional analysis of information networks. New York, Wiley. 281 p.
- Bell, T. E. 1971. "Computer performance analysis: measurement objectives and tools." A report prepared for National Aeronautics and Space Administration and United States Air Force Project Rand. Rand Number R-584-NASA/PR. February, 1971. 32 p.
- Bell, T. E., B. W. Boehm, and R. A. Watson. 1972. "Computer performance analysis: framework and its initial phases for a performance improvement effort." A report prepared for United States Air Force Project Rand. Rand Number R-549-1-PR. November, 1972. 55 p.
- Betz, Robert E. 1973. "Use of SMF data for performance analysis and resource accounting on IBM large scale computers." Proceedings of 8th Meeting of Computer Performance and Evaluation Users Group (CPEUG) NBS Special Publication 40. p. 23-32.
- Blatny, J., S. R. Clark, and T. A. Rourke. 1972. CACM. Vol. 15, No. 6. June, 1972. "On the optimization of performance of time-sharing systems by simulation." p. 411-420.
- Booth, Grayce M. 1973. Functional analysis of information processing. New York, Wiley. 269 p.
- Butler, James L. 1970. Instrumentation Technology. "Comparative criteria for minicomputers." Oct., 1970. Vol. 17, No. 10, p. 67-82.
- Calingaert, Peter. 1967. CACM. Vol. 10, No. 1, Jan. 1967. "Systems performance evaluation: survey and appraisal." p. 12-18.

- Chen, Kuei-Lin. 1974. The application of decomposition and condensation algorithms to the logical design of resource planning and management (RPM) networks. Masters thesis, Oregon State University. 205 p.
- Cohen, Leo J. 1973. "Dollar effectiveness evaluation of computer systems." Proceedings of Eighth Meeting of Computer Performance and Evaluation Users Group. NBS Special Publication 401. p. 85-97.
- Drummond, M. E. Jr. 1973. Evaluation and measurement techniques for digital computer systems. New York, Prentice Hall. 338 p.
- Enslow, Philip H. Jr. 1974. Multiprocessors and parallel processing. New York, Wiley. 340 p.
- Falk, Howard. 1975. IEEE spectrum. Vol. 12, No. 4, April, 1975. "Technological forecast-computers II." p. 46-51.
- Gibson, J. C. 1970. IBM Technical Report TROO.2403. June, 1970. "The Gibson Mix." 4 p.
- Gordon, Geoffrey. 1969. System simulation. Englewood Cliffs, N.J., Prentice-Hall. 303 p.
- Gould, I. H. (edr.). 1971. IFIP guide to concepts and terms in data processing. Holland, North Holland Publishing Co. 161 p.
- Hellerman, H. and H. R. Smith, Jr. 1970. Computing surveys. Vol. 2, No. 2, June, 1970. "Throughput analysis of some idealized input, output, and computer overlap configurations." p. 111-118.
- Herman, Donald J. 1967. Datamation. Feb., 1967. "SCERT: A computer evaluation tool." Vol. 13, No. 2. p. 26-28.
- Hesser, W. Andrew. 1973. "The use of simulation in the solution of hardware allocation problems." Proceedings of Eighth Meeting of Computer Performance and Evaluation Users Group. NBS Special Publication 401. p. 73-79.
- Highland, Harold Joseph. 1974. "Preface." Proceedings of Eighth Meeting of Computer Performance and Evaluation Users Group. NBS Special Publication 401. p. v and vi.

- Hellerman, Herbert. 1973. Digital computer system principles. New York, McGraw Hill. 466 p.
- Ijiri, Y. 1965. Management goals and accounting for control. Chicago, Rand-McNally. 191 p.
- Inoue, Michael S. and James L. Riggs. 1972a. "Resource planning and management network." Proceedings of International Symposium on Systems Engineering and Analysis, Vol. 2. p. 187-192. October 23-27 Lafayette Indiana Purdue University, John E. Goldberg et al., eds.
- _____ 1972b. "RPM network." Session TPSi3.4 ORSA - TIMS-AIIE Systems Engineering Joint National Conference, Atlantic City, November 9, 1972.
- Inoue, Michael S. 1974. "Visual identification of Kuhn-Tucker conditions on RPM networks." Paper presented at 2nd Annual Systems Engineering Conference, Minneapolis, Minn. November, 1974.
- Kanter, Jerome. 1970. Management guide to computer system selection and use. Prentice Hall, Englewood Cliffs, N. J. 257 p.
- Knight, Kenneth. 1966. Datamation. September, 1966. "Changes in computer performance." Vol. 12, No. 9. p. 40-54.
- _____ 1968. Datamation. January, 1968. "Evolving computer performance: 1962-1967." Vol. 14, No. 1. p. 31-35.
- Lee, Sang M. 1972. Goal programming for decision analysis. Philadelphia, Auerback. 387 p.
- Lucas, Henry C. Jr. 1971. Computing surveys. Vol. 3, No. 3, September, 1971. "Performance evaluation and monitoring." p. 79-91.
- Minsky, Marvin L. 1967. Computation: finite and infinite machines. Prentice-Hall, Englewood Cliffs, N. J. 317 p.
- Morris, Michael F. 1974. "Kiviat Graphs-conventions and figures of merit." Proceedings of 10th Annual Meeting of Computer Performance and Evaluation Users Group, Columbus, Ohio. October, 1974. (In press)

- Ollivier, Robin T. 1970. Datamation. January, 1970. "A technique for selecting small computers." Vol. 16, No. 1. p. 141-145.
- Riggs, James L. and Michael S. Inoue. 1975. Introduction to operations research and management science - a general systems approach. McGraw Hill, New York. 536 p. (In press)
- Scheurman, H. Lynn. 1970. *REX (Version 1) linear programming system. Corvallis, Oregon, Oregon State University Computer Center. 89 p.
- Sharpe, William R. 1969. The economics of computers. Columbia University Press, New York. 571 p.
- Stimler, Saul. 1969. Real-time data processing systems - a methodology for design and cost/performance analysis. McGraw-Hill, New York. 259 p.
- Stimler, Saul. 1974. Data processing systems: their performance, evaluation, measurement, and improvement. Motivational Learning Programs Inc., Trenton, N. J. 183 p.
- Taha, H. A. 1971. Operations research: an introduction. Macmillan, New York. 703 p.
- Timmereck, E. M. 1973. Computing surveys. Vol. 5, No. 4, December, 1973. "Computer selection methodology." p. 199-222.

APPENDICES

APPENDIX A

GLOSSARY OF TERMS

BATCH PROCESSING: The processing of a program or set of programs, called a batch, under the control of an operating system and without intermediate interaction with the user. Each programmer request, in advance, specific services of the operating system, and then transfers control to the system, which supervises and processes each batch program in a sequence.

BYTE: A small sequence of adjacent binary digits, or bits, which can be treated as a unit (usually either six or eight).

CAPACITY: The maximum achievable average throughput rate regardless of the timeliness of the outputs. It is expressed in units of work (for example, jobs or transactions) successfully completed per hour, minute, or second. Capacity is intended to indicate a theoretical upper processing power limit. It can be valuable for performance evaluation and improvement but usually would not be expected to be achievable in practice. The same hardware and software would have different numerical values for capacity when representative work loads with different characteristics were processed.

CHANNEL: A path along which signals can be sent. In reference to computer systems, this usually refers to a medium that transfers a series of digits or characters between two terminals, or between main memory and peripheral I/O devices, with minimum involvement of the CPU.

CHANNEL CONTENTION: Competition between high-speed I/O devices for one or more channels.

COMPUTER-BOUND: Describes a program for which the time required to perform computation dominates the total time required for program execution. (Also called CPU-bound)

CPU: Central Processing Unit.

CPU UTILIZATION: The ratio of time that the CPU is actively working to the total time that the CPU is available to work. Hardware and software monitors measure CPU utilization by (1) determining the number of cycles that perform work and the total cycles completed over a given time period and (2) computing the ratio. CPU utilization can be calculated from computer accounting data by dividing the total CPU time accounted for over a time interval by the net amount of time that the central processor was operating. (Net time equals the length of the time interval less the amount of time that no work was performed because of hardware or software failure, scheduled maintenance, or insufficient submitted work.)

I/O: An abbreviation for input/output, the transfer of information between a computing device and a peripheral.

I/O Bound: Describes a program, a set of programs, or an entire work-load in which the time required to perform I/O operations dominates the total time required to execute the program.

JOB: One or more programs (often called job steps, activities, or tasks) submitted for processing by a computer system. (2) The unit of work for batch processing systems. (3) A basic independent unit of work to be carried out by a system.

PRODUCTION JOB: A computer run of a checked-out program, often done on a periodic or continuing basis at a computer installation for the purpose of producing specified output.

PROGRAM: A complete specification of one or more task that are to be performed on data.

RELATIVE THROUGHPUT RATE: The dimensionless ratio of the average throughput rate for one set of conditions divided by the average throughput rate for a second set of conditions. The different conditions usually are the processing of the same representative work load in two different systems. For relative throughput rate to be meaningful, it is essential that the same representative work load be processed by each of the systems being compared.

RESPONSE TIME: The average time that a terminal user must wait to receive a response from a time-sharing system. (Note that the definition of a "response" is critical to the magnitude and relevancy of the measure.)

SPOOLING: Queueing input or output on disk or tape. This is done in multiprogramming computer systems so that a program can read input data at a disk or tape speed instead of card-reader speed and can write data at disk or tape speed instead of printer speed. The main reason for spooling is to allow many programs (at one time) to read input data or write output data even though the system may only have one card reader and one or two printers.

TASK: A number of steps which are partially or wholly ordered from the point of view of their execution.

THROUGHPUT: A performance measure for a computer system to indicate how much work is being processed over a given time period (e.g., jobs processed per hour). (2) Total data processing work successfully completed during an evaluation period.

THROUGHPUT RATE: The data processing work successfully completed per unit of time.

TIME-SHARING: A technique of system operation in which each of several programs receives a short quantum of the CPU's time. (They share the processor.) Time-sharing systems often include facilities to move programs, or parts of programs, between memory and other storage devices (e.g., disk or drum)

TRANSACTION: The unit of work for a real time systems.

TUNING: Making relatively minor modifications to a computer system's hardware, software, operational procedures, or any other facet of the operation of a computer installation for the purpose of increasing efficiency of operation.

APPENDIX B

BUTLER'S FORMULA

$$P = \frac{P_h + P_8}{2}$$

$$P_h = \text{Basic system cost (\$)} \div 0.1M \left(1 - \frac{W-F}{2W} \right) + \frac{20}{T} (A_h + L_h + I_h) + 100N + 50R$$

$$P_8 = \frac{\text{Basic system cost (\$)}}{500(D+B+L) + 1,000A + 2,000C + 50S}$$

- M = Core memory storage capacity of a basic machine, total bits
 F = Number of bits in the address field of single word instructions
 W = Word length, bits
 R = Number of general purpose registers
 T = Core-memory read-write cycle time
 N = Number of "extras" in the basic cost of the machine, including:
 . Real-time clock
 . Power failure protection
 . Automatic restart after power failure
 . Memory parity checking
 . Memory protect
 A_h = A number proportional to the arithmetic capability of the computer--with a range of 0 to 100:
 0 No arithmetic capability
 25 Hardware add and complement
 50 Hardware add and subtract; software multiply and divide (fixed point, slow)
 75 Hardware add and subtract; hardware multiply and divide (fixed point, fast)
 90 Hardware add and subtract; hardware multiple and divide (fixed point); software floating point arithmetic
 100 Hardware fixed point and floating point arithmetic
 L_h = A number proportional to the logic capability of the computer--range of 0 to 100:
 0 No logic capability
 25 "And" and "or" hardware
 50 "And," "or," and "exclusive or"

- 75 All of the above, also word test and conditional branch instructions
 - 90 All of the above, also bit test and bit manipulation instructions
 - 100 All of above, also arithmetic rational test instructions
- I_h = A number proportional to the I/O capability of the computer--range of 0 to 100:
- 0 No I/O
 - 25 Programmed I/O through internal registers only
 - 75 Same as above, also multiple I/O processors
- D = Off-line diagnostic routines supplied:
 NO = 0 YES = 1
- B = Debugging routines supplied:
 NO = 0 YES = 1
- L = Loader routines supplied:
 NO = 0 YES = 1
- A = Number of assembler
- C = Number of compilers
- S = Power of on-line operating system (range of 0 to 100)

APPENDIX C

OLLIVIER'S FORMULA

Weighting Schemes (Manufacturer Criteria)

| Factor | Weight | Scoring Bases |
|--------------------------|--------|--|
| Delivery time | 7 | 4, 3: Less than 45 days ARO; 2, 1: 45-75 days ARO; 0: Over 75 days ARO |
| Past performance | 4 | 4-2: Many reports of on-time delivery and good service; 1-0: Known for late delivery, poor service |
| Maintenance | 3 | 4-2: 24-hour turnaround on cpu, on-call maintenance; 2-0: No experience, remote or difficult corporate interface |
| Location | 2 | 4: Southern California 2: Within 500 miles 0: Distant |
| Alternative sites | 1 | 4: Same computer installed at JPL; 3-1: Locally available; 0: No alternative site |
| Number installed | 4 | 4: Over 100; 3-1: 10-100 installed; 0: Less than 10 in field |
| Documentation & training | 5 | 4: Excellent hardware and software manuals, or training provided; 3-1: Adequate inter- face and programming manuals; 0: Little or no documentation |

Weighting Schemes (Computer Criteria) cont.

| Factor | Weight | Scoring Bases |
|------------------------|--------|---|
| Word size | 10 | 4: 16 bits or more; 2: 12 bits; 0: 8 bits or less |
| Cycle time | 6 | 4: 1 μ sec; 3-1: 1-2 μ sec 0: 2 μ sec |
| Instruction set | 5 | 4, 3: Extensive; 2: Adequate; 1-0: Primitive |
| Arithmetic | 2 | 4: Hardware multiply/divide; double precision and floating point options; good precision 3-1: Adequate capability; hardware mul/div or fast subroutines 0: Very little arithmetic capability |
| Addressing | 4 | 4-0: Score one for each of the following: indirect, relative, indexed, direct to greater than 4096, or by addressing |
| Programmable registers | 6 | 4: Many; 3-1: More than one; 0: one |
| Interrupts | 7 | 4: 3 or more priority, no identification necessary; 3-1: Adequate for 3 devices 0: None quoted |
| Input/Output | 8 | 4: 2 or more automatic channels at rates to 1.3 megabits/sec; 3-1: At least one 1.0 megabits/ sec with good accumulator I/O; 0: Marginal I/O capability |
| Physical size | 1 | 4-0: Subtract one point for each 5 inches over 11 inches |
| Console | 3 | 4-0: Sense switches, displays, debugging aids |

APPENDIX D

INSTRUCTION CATEGORY DESCRIPTIONS

Weights for a Scientific Mix and a Commercial Mix

| Instruction Category* | Scientific Weight | Commercial Weight |
|--|-------------------|-------------------|
| 1. Fixed add (subtract) and compare instructions | 0.10 | 0.25 |
| 2. Floating add (subtract) instructions | .10 | 0 |
| 3. Multiply instructions | .06 | .01 |
| 4. Divide instructions | .02 | 0 |
| 5. Other manipulation and logic instructions | <u>.72</u> | <u>.74</u> |
| | 1.00 | 1.00 |

Source: Kenneth E. Knight, "A Study of Technological Innovation-- The Evolution of Digital Computers." doctoral dissertation, Carnegie Institute of Technology, November, 1963, pp. IV-5, IV-6, IV-7.

*Category descriptions:

1. "These instructions are the fixed additions, subtractions and compare operations performed. We may obtain the fixed add time for each system from the computing literature.
2. "The floating point add time is given in the computing literature for machines with built-in floating-point arithmetic. For other machines the figure can be approximated by multiplying the fixed-point add time by 10... (the mean value for six computing systems considered)."
3. "We have included only one multiply category since the operating times for these two operations on systems capable of both floating and fixed-point arithmetic are approximately equal. The multiplication time is a characteristic available in the computing literature."
4. "The fixed- and floating-point operations were combined... the divide time represents a characteristic of each system published in the computing literature."

5. "This category combines a large number of branch, shift, logic and load-register instructions... For computers with parallel arithmetic, the time... is the shortest of... add time or... 2(times) the memory access time for one word... For computers with serial arithmetic, the... time equals the shortest of (1) add time or (2) (the time required to access an instruction, slightly modified)."

APPENDIX E

June 18, 1970

TR 00. 2043

The Gibson Mix

by

Jack C. Gibson

ABSTRACT

The Gibson Mix is a set of weights developed by the author to evaluate the speed of a central processor. Developed in 1959 for use with the IBM 704 vocabulary, the method became widely known and used throughout the industry at a time when there was a dearth of such tools. Because of a renewed current interest in what was considered to be an obsolete tool, the Gibson Mix is published here for the first time. The author's critique of usefulness of the mix as a computer performance evaluation tool is included.

Computer Evaluation
07 Computers

IBM

International Business Machines Corporation
Systems Development Division, Poughkeepsie, New York

INTRODUCTION

In spite of no published material on the method and no intent to extend to external use the Gibson Mix developed at IBM, the method became widely known and used throughout the computer world during the early 1960's. The tool was used to plan and design new computers, to estimate the worth of a computer to a user, and to plan data processing systems.

Today, for no specific reasons that can be determined, there is a strong and widespread resurgence of interest in what was considered to be an obsolete tool. Granted the name with its suggestion of a well-known recipe makes for easy recall, there must be more basic reasons for the renewed interest. For that reason -- belated but for what value it has now for the student and scientist -- here is the first published recipe for: the Gibson Mix.

WHAT IS IT?

The Gibson Mix is a set of weights developed for 13 different classes of instructions, by which to evaluate the speed of a central processing unit in performing scientific-type problems. It was developed by the author in 1959. At that time, the computer world was basing its estimates of CPU speed on storage cycle time, add time, or from an average of add and multiply times. The Gibson Mix with its weighted average of 13 different instruction times promised a more precise evaluation of CPU speed.

The mix was based primarily on the operation codes in the IBM 704 data processing system vocabulary. These codes fell neatly into 12 classes by function. Execution times for the codes within each class tended to vary little from one another. A 13th class was artificially devised to treat the indexing of an address as if it were a separate instruction. A percentage by which to weight the established instruction times of a computer then was derived for each instruction class. Here, then, is the Gibson Mix recipe:

The Gibson Mix

| | |
|--------------------------------------|-------------|
| 1. Loads and Store | 31.2 |
| 2. Fixed Point Add and Subtract | 6.1 |
| 3. Compares | 3.8 |
| 4. Branches | 16.6 |
| 5. Floating Add and Subtract | 6.9 |
| 6. Floating Multiply | 3.8 |
| 7. Floating Divide | 1.5 |
| 8. Fixed Point Multiply | 0.6 |
| 9. Fixed Point Divide | 0.2 |
| 10. Shifting | 4.4 |
| 11. Logical, And, Or, etc. | 1.6 |
| 12. Instructions Not Using Registers | 5.3 |
| 13. Indexing | <u>18.0</u> |
| | 100.0 |

APPLYING THE MIX

To apply the mix to evaluating the CPU speed of a computer. Computer X say, it is necessary only to obtain the average time for X to execute instructions in each of the 13 classes and to take a weighted average of these times using the indicated weights. The implication is that the result of using the mix is closely dependent on the estimated instruction time for each class.

ESTIMATING CLASS AVERAGES

Ground rules must first be established by which the averages are to be estimated. The ground rules should take into consideration typical field sizes, frequency of zero as a factor, distribution of zero digits in a multiplier, average number of positions of pre-shift in floating point add, the number of times a conditional branch is taken, rather than bypassed, etc.

It must also be decided in which class each instruction belongs, and how frequently the slowest instruction in the class is used compared to the fastest. It should be noted that Class 11, Logical, includes all bit-manipulating and address-computing operation codes; Class 12, Instructions Not Using Registers, includes a miscellany such as Start, Stop, I/O instructions (instruction interpret only), and unconditional branch.

The mix was expected to be applied to single-address machines only, as the choice of classes reflects. To apply the mix to other machines, say a three-address machine performing $A+B=C$ in one instruction, the instruction must be analyzed to determine what part of the time should be apportioned to Class 1, Load and Store, and what part to the appropriate Add Class.

SOURCE OF THE WEIGHTS

Each weight is the relative frequency of execution of all operation codes in a class, during actual processing. Seven jobs were run on the IBM 704, and 5.7 million instruction executions were traced. For each execution, a record was written on magnetic tape. Later, these records were counted by instruction type to determine how often each type was used. All jobs were scientific in nature, including several utilizing matrix algebra.

To spread the application base somewhat and to reduce the influence of the IBM 704 architecture on the weights, similar traces of ten small IBM 650 data processing system jobs (3.0 million executions) were also distributed by instruction type. Each weight in the Gibson Mix is the average obtained by weighting the IBM 704 count by 88.6%, and the IBM 650 count by 11.4%. Although the 650 jobs were of a commercial type, the Gibson Mix is predominantly scientific.

INTERPRETING THE RESULTS

It is important to keep in mind that the Gibson Mix is a tool to predict the speed of a CPU on jobs such as those traced on the IBM 704 and 650. On such jobs, it is representative; on other jobs, there is no certainty the mix will be representative. Other constraints on the results include the

insensitivity of the mix to: variations in programming; operating system overhead; compiler execution time; and I/O system effects, for example the interference between CPU and I/O for primary storage cycles.

When the mix is applied to Computer X, the result obtained is a weighted average instruction time. Should the mix be used to estimate the run time of a benchmark program? Probably not, because the mix is too imprecise. Should the average microseconds per instruction be converted to millions of instructions per second (MIPS)? Perhaps for comparative purposes. First, however, a computer with which one is familiar, call it Computer B, should be evaluated as a basis for comparison. By comparing the average instruction time for Computer X with that of Computer B, or by comparing their MIPS rates, Computer X speed can be declared to be a certain percentage of Computer B speed. This is a conservative way to use the mix but it does help in mitigating some of its imprecision.

CRITIQUE

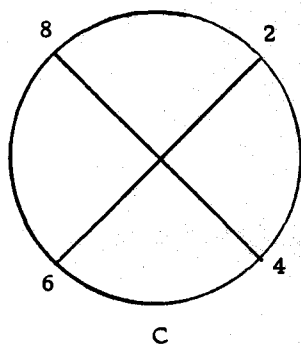
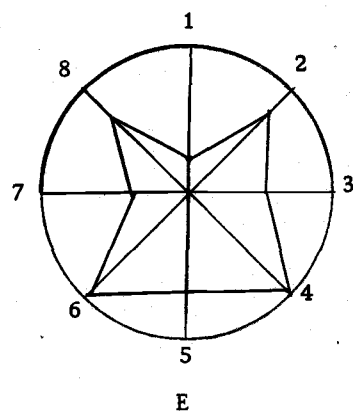
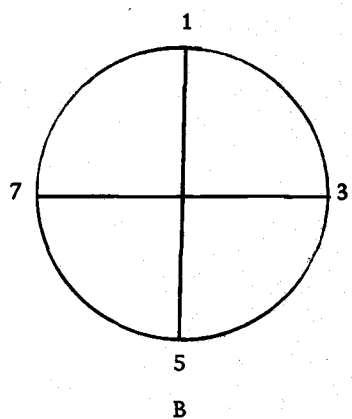
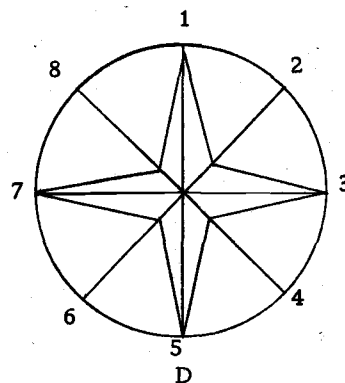
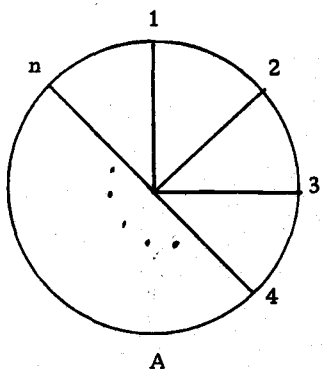
With due attention given to the representativeness and precision discussed, the Gibson Mix has been a relatively easy tool to apply -- particularly when a dearth of such tools existed. The mix still could be used today to compare relative hardware speed potentials. Nonetheless, the mix is obsolete. The same job is being done better today using such techniques as simulation, trace times and measurement monitoring.

A mix generated today could be constructed on a wider job base and on hundreds of millions of instruction executions. Validation could be a controlled scientific experiment. Instruction pairs, triplets, etc. could be traced. Mixes based on source language statements could be developed, as well as compiler mixes. I/O and operation system effects could be isolated.

Undoubtedly, there is much in the field of developing mixes to interest the computer scientist and student. Indeed, it is perhaps from the university that the recent interest in the Gibson Mix has surged.

APPENDIX F

KIVIAT GRAPHS



A = General Case
 B = Best Case Star
 C = Worst Case Star
 D = "Good" Star
 E = "Bad" Star

APPENDIX G. Lee's GOAL Programming for CYBER
PROGRAM JLEGOAL 73/74 OPT=1

FTN 4.3+74353

```

1      PROGRAM JLEGOAL (INPUT,OUTPUT,TAPE60=INPUT,TAPE61=OUTPUT)
      C *****
      C
      C      THIS IS A GOAL PROGRAMMING PACKAGE MODIFIED FOR
5      C      THE CDC CYBER AT OREGON STATE UNIVERSITY
      C
      C      DATE MODIFIED...MAY 20,1975
      C *****
10     C      NOTE.....THE FOLLOWING ARRAYS SHOULD BE SET
      C      TO THE VALUES GIVEN FOR A FULL RUN WITH THE ORIGINAL
      C      LIMITATIONS OF LEES GOAL PROGRAMMING PACKAGE GOALP1.
      C      RVLX (10,250)
      C      VALX (10,250)
15     C      C (100,250)
      C      D (100,250)
      C      VALY (100,10)
      C *****
      COMMON C(100,250)
20     COMMON D(100,250)
      COMMON RVLX(10,250)
      COMMON VALX(10,250)
      COMMON VALY(100,10)
      DIMENSION X(250)
25     DIMENSION Y(100)
      DIMENSION IY(100)
      DIMENSION AMT(100)
      DIMENSION OOD(100)
      DIMENSION OUD(250)
30     DIMENSION KEPT(100)
      DIMENSION PRDT(100)
      DIMENSION RHS1(100)
      DIMENSION ZVAL(10)
      15 FORMAT(I3,F12.2)
35     12 FORMAT(10F12.3)
      13 FORMAT(8F9.0)
      313 FORMAT(I3,10X,F20.5)
      CALL START(N,M,L,PRDT,RHS1,KPCK,KEPT,TEST)
      DO 21 J=1,M
40     21 X(J)=J
      DO 20 I=1,N
      20 Y(I)=I
      DO 25 K=1,L
      DO 25 I=1,N
45     VALY(I,K)=VALX(K,I)
      25 CONTINUE
      ITAB=0
      C *****
      C      BRING IN NEW VARIABLES
50     C      CALCULATE NET CONTRIBUTION OF EACH VARIABLE (RVLX(K,J))
      C *****
      ITER=0
      L1=0
      32 K3=L-L1
55     IF(K3-1) 800, 40, 40
      40 DO 60 K=1,K3
      DO 60 J=1,M

```

PROGRAM JLEGOAL 73/74 OPT=1

FTN 4.3+74353

```

      SUMP=0.
      DO 50 I=1,N
60      P=VALY(I,K)*C(I,J)
      SUMP=SUMP+P
      50 CONTINUE
      RVLX(K,J)=SUMP-VALX(K,J)
      60 CONTINUE
65      C*****C
      C      BRING IN X(K2)
      ZMAX=0.
      DO 90 J=1,M
      IF(K3-L) 92,70,70
70      92 K4=K3+1
      DO 91 K=K4,L
      IF(RVLX(K,J)) 90,91,91
      91 CONTINUE
      70 IF(RVLX(K3,J)-ZMAX) 90,90,80
75      80 ZMAX=RVLX(K3,J)
      K2=J
      90 CONTINUE
      IF(K3-1+ZMAX) 95,1211,95
      95 IF(ZMAX) 790,790,100
80      C*****C
      C      WHICH VARIABLE IS REMOVED FROM THE BASIS
      C      CALCULATE LIMITING AMT FOR EACH BASIS VARIABLE
      C*****C
      100 DO 150 I=1,N
      IF(PROT(I)) 110,120,120
85      110 WRITE (61,13) PROT(I)
      GO TO 830
      120 IF(C(I,K2)) 130,130,140
      130 AMT(I)=-1.
90      GO TO 150
      140 AMT(I)=PROT(I)/C(I,K2)
      150 CONTINUE
      C*****C
      C      SELECT SMALLEST POSITIVE LIMITING AMT
95      I=1
      160 IF(AMT(I)) 170,210,210
      170 I=I+1
      IF(I-N) 160,160,180
      180 WRITE (61,13) AMT(N)
      GO TO 830
100      210 ZMIN=AMT(I)
      K1=I
      220 I=I+1
      IF(I-N) 230,230,300
105      230 IF(AMT(I)) 220,240,240
      240 IF(ZMIN-AMT(I)) 220,220,210
      C*****C
      C      REMOVE Y(K1)
      300 Y(K1)=X(K2)
110      DO 310 K=1,L
      VALY(K1,K)=VALX(K,K2)
      310 CCNTINUE
      C*****C
      C      CALCULATE NEW RIGHT-HAND SIDES

```

PROGRAM JLEGAL 73/74 OPT=1

FTN 4.3+74353

```

115      DO 400 I=1,N
          PROT(I)=PROT(I)-ZMIN*C(I,K2)
          400 CONTINUE
          PROT(K1)=ZMIN
C*****
120      C      CALCULATE NEW SUBSTITUTION RATES
C*****
C      THIS IS THE ORIGINAL CODE FROM LEE
C      THESE MODIFICATIONS ARE FOR TEST PURPOSES
C      TO TRY AND CUT OUT THE D ARRAY AND SAVE 25K
125      C*****
          DO 500 J=1,M
          DO 500 I=1,N
              D(I,J)=C(I,J)-C(K1,J)* (C(I,K2)/C(K1,K2))
          500 CONTINUE
130      DO 510 J=1,M
          D(K1,J)=C(K1,J)/C(K1,K2)
          510 CONTINUE
          DO 520 J=1,M
          DO 520 I=1,N
135      C(I,J)=D(I,J)
          IF (ABS(C(I,J)).LE.0.00001) C(I,J)=0.
          520 CONTINUE
C*****
          ITER=ITER+1
140      C*****C
          C      WRITE ALL TABLES OR JUST OPTIMAL TABLE
          IF (ITER-1) 200,200,1210
          200      WRITE (61,1220)
145      1220      FORMAT(//,*, THE INITIAL ZJ-CJ MATRIX*,//)
          DO 1230 KK=1,L
              MX=L+1-KK
              IF (TEST.EQ.0.0) GO TO 1229
              MX=L-KK
              IF (MX.NE.0) GO TO 1229
150      WRITE (61,1231)
          1231      FORMAT(* ARTIFICIAL*)
          GO TO 1230
          1229      WRITE (61,5007) MX
          1230      WRITE (61,12) (RVLX(KK,JJ),JJ=1,M)
155      IF (ITAB) 40,40,600
          1210      IF (ITAB) 40,40,6813
          1211      IF (ITAB) 790,790,6813
          6813      WRITE (61,5003)
          DO 6814 K=1,L
160      MM=L+1-K
          WRITE (61,5007) MM
          WRITE (61,12) (RVLX(K,J), J=1,M)
          6814      CONTINUE
          IF (ZMAX) 790,790,600
165      C*****C
          C      WRITE EACH TABLE
          600      WRITE (61,6001) ITER
          6001      FORMAT(1H1,10X,13HITERATIONS = ,I3,//)
          WRITE (61,5001)
170      DO 610 I=1,N
          IV(I)=V(I)

```

PROGRAM JLEGOAL 73/74 OPT=1

FTN 4.3+74353

```

        WRITE (61,313) IY(I),PRDT(I)
175      610 CONTINUE
        WRITE (61,5002)
        DO 620 I=1,N
        WRITE (61,5006) I
        WRITE (61,12) (C(I,J),J=1,M)
        620 CONTINUE
        GO TO 40
180      C*****C
        C      MOVE TO NEXT LOWER PRIORITY LEVEL
        790 L1=L1+1
        GO TO 32
        C*****C
185      C      WRITE FINAL RESULTS
        800 WRITE (61,1015)
        1015 FORMAT(1H1)
        WRITE (61,1014) ITER
        1014 FORMAT(10X,*,ITERATIONS,*,I5)
190      WRITE (61,5000)
        5000 FORMAT(55X,*,THE SIMPLEX SOLUTION*,25X,*,PAGE 05*,//)
        WRITE (61,5001)
        5001 FORMAT(//,*,THE RIGHT HAND SIDE*,//)
195      DO 810 I=1,N
        IY(I)=Y(I)
        WRITE (61,313) IY(I),PRDT(I)
        810 CCNTINUE
        WRITE (61,5002)
        5002 FORMAT(//,*,THE SUBSTITUTION RATES*,//)
200      DO 812 I=1,N
        WRITE (61,5006) I
        5006 FORMAT(1X,3HROW,I5)
        WRITE (61,12) (C(I,J),J=1,M)
        812 CONTINUE
205      WRITE (61,5003)
        5003 FORMAT(//,*,THE ZJ-CJ MATRIX*,//)
        DO 814 K=1,L
        MM=L+1-K
        IF (TEST.EQ.0.0) GO TO 5008
210      MM=L-K
        IF (MM.NE.0) GO TO 5008
        WRITE (61,1231)
        GO TO 5009
        5008 WRITE (61,5007) MM
215      5007 FORMAT(1X,3HPRIORITY,I5)
        5009 WRITE (61,12) (RVLX(K,J), J=1,M)
        814 CONTINUE
        C*****C
        C      EVALUATE OBJECTIVE FUNCTION
220      C*****C
        DO 820 K=1,L
        ZVAL(K)=0.
        DO 820 I=1,N
        ZVAL(K)=ZVAL(K)+PRDT(I)*VALY(I,K)
225      820 CONTINUE
        WRITE (61,5004)
        5004 FORMAT(*,AN EVALUATION OF THE OBJECTIVE FUNCTION*)
        DO 821 K=1,L
        KK=L-K
230      IF (TEST.EQ.1.0) GO TO 89
        KK=KK+1
        89 WRITE (61,15) KK,ZVAL(K)
        821 CCNTINUE
        CALL FINISH(RHS1,PRDT,L,KPCK,Y,N,KEPT,TEST)
235      830 CONTINUE
        STOP
        END

```

SUBROUTINE START 73/74 OPT=1 FTN 4.3+74353

```

1      C*****
      C
      C      AND NOW FOR THE SUBROUTINES
      C*****
5      C*****
      C      THE START SUBROUTINE IS DESIGNED TO TAKE INFORMATION IN A SPED
      C      IFIED FORMAT AND TRANSFORM IT INTO A SERIES OF USABLE MATRICIES
      C.....
10     SUBROUTINE START(NROWS,NVAR,NPRT,RHS,RHS1,KPCK,KEPT,TEST)
      REAL KEPT
      INTEGER POS
      INTEGER NEG
      INTEGER DATEA
      INTEGER OBJ
15     INTEGER PROB
      INTEGER L
      INTEGER RGHT
      INTEGER B
      INTEGER E
20     INTEGER G
      C*****
      C      THESE ADDED INTEGER
      C      VALUES ARE FOR OS3
      C*****
25     INTEGER ANAME
      INTEGER EQUALS
      C*****
      C      NOTE...THE FOLLOWING ARRAYS SHOULD BE RESET TO THE
      C      INDICATED VALUES TO MAKE THIS PROGRAM COMPATABLE
30     C      WITH THE ORIGINAL LIMITATIONS OF LEES GOALP1
      C      NV=250 AND NR=100
      C      C (100,250)
      C      VALX (10,250)
      C      RVLX (10,250)
35     C      VALY (100,10)
      C*****
      COMMON C(100,250)
      COMMON RVLX(10,250)
40     COMMON VALX(10,250)
      COMMON VALY(100,10)
      DIMENSION EQUALS(100)
      DIMENSION RHS(100)
      DIMENSION KEPT(100)
45     DIMENSION RHS1(100)
      DATA (POS=4HPOS )
      DATA (NEG=4HNEG )
      DATA (DATEA=4HDATA)
      DATA (OBJ=4HOBJ )
50     DATA (PROB=4HPROB)
      DATA (B=1HB)
      DATA (E=1HE)
      DATA (G=1HG)
      DATA (L=1HL)
55     DATA (RGHT=4HRGHT)
      NV=250
      NR=100

```

SUBROUTINE START 73/74 OPT=1

FTN 4.3+74353

```

C*****
60  C    READ THE PROBLEM CARD
C*****
1    FORMAT(A4,3I3)
    TEST=0.0
    READ (60,1) ANAME,NROWS,NVAR,NPRT
65  LISP=NPRT+1
    IF(NVAR.LE.0) GO TO 1020
    IF(NPRT.LE.0) GO TO 1020
    IF(NROWS.LE.0) GO TO 1020
    IF(ANAME.NE.PROB) GO TO 901
70  C*****
C    READ THE SIGN CARD.
C    IT WILL CONTAIN ONE OF THE FOLLOWING:  ERS FOR EACH ROW
C    FOR EQUALS                               E
75  C    FOR LESS THAN OR EQUAL TO           L
C    FOR GREATER THAN OR EQUAL TO           G
C    FOR BOTH DEVIATIONS                     B
C*****
    READ (60,11) (EQUALS(I),I=1,NROWS)
80  11 FORMAT(80A1)
C*****
C    COUNT THE NUMBER OF POSITIVE SLACK VARIABLES
C*****
85  NART=0
    NFLDS=0
    DO 12 I=1,NROWS
    IF(EQUALS(I).EQ.B)NFLDS=NFLDS+1
    12 IF(EQUALS(I).EQ.G)NFLDS=NFLDS+1
90  C*****
C
C    TEST FOR SIZE
C*****
    NSIZE=NFLDS+NROWS+NVAR
95  IF(NROWS.GT.NR) GO TO 911
    IF(NSIZE.GT.NV) GO TO 911
C*****
C
C
100 C    CLEAR ALL MATRICIES
C*****
    IF (NPRT.GT.NROWS) GO TO 1105
    NUM = NROWS
    GO TO 1106
105 1105 NUM = NPRT
    1106 KUD = NPRT + 1
    DO 1059 KX=1,NSIZE
1059 VALX(KUD,KX)=0.0
    DO 16 J=1,NSIZE
110  DO 16 I = 1, NUM
    KEPT(I)=0
    IF(I.GT.KUD) GO TO 17
    K=I
    RVLX(K,J)=0.0

```

SUBROUTINE START

73/74 OPT=1

FTN 4.3+74353

```

115      VALX(K,J)=0.0
      17 IF(I.EQ.J) C(I,J)=1.0
      VALY(I,K)=0.0
      IF(I.NE.J) C(I,J)=0.0
120      16 CONTINUE
      KPCK=0
      K=KDUO
C*****
C
C      ADJUST THE SLACK VARIABLES AND OBJECTIVE FUNCTION TO MEET THE
125      C      REQUIREMENTS OF THE SIGN
C*****
      DO 13 I=1,NROWS
      IF(EQUALS(I).EQ.E) GO TO 14
      IF(EQUALS(I).EQ.G) GO TO 15
130      IF(EQUALS(I).EQ.L) GO TO 13
      IF(EQUALS(I).EQ.B) GO TO 18
      GO TO 910
      14 J=I
      VALX(K,J)=1.0
135      NART=NART+1
      TEST=1.0
      GO TO 13
      15 KPCK=KPCK+1
      J=NROWS+KPCK
140      C(I,J)=-1.0
      KEPT(I)=J
      J=I
      VALX(K,J)=1.
      NART=NART+1
145      TEST=1.0
      GO TO 13
      18 KPCK=KPCK+1
      J=KPCK+NROWS
      C(I,J)=-1.0
150      KEPT(I)=J
      13 CONTINUE
C*****
C
C      READ THE OBJECTIVE FUNCTION
155      C*****
      READ (60,21) ANAME
      I=0
      IF(ANAME.NE.OBJ) GO TO 920
      20 READ (60,21) ANAME,I,M,TEMP
160      IF(ANAME.EQ.DATEA) GO TO 30
      IF(M.LE.0) GO TO 1022
      K=LISP-M
      21 FORMAT(A4,2I5,F16.0)
      IF(J.LE.0) GO TO 1022
165      IF(K.GT.NPRT) GO TO 1024
      IF(ANAME.EQ.NEG) GO TO 26
      IF(ANAME.EQ.POS) GO TO 25
      GO TO 27
      26 J=I
170      IF(EQUALS(I).EQ.G.OR.EQUALS(I).EQ.E) GO TO 1055
      VALX(K,J)=TEMP

```

SUBROUTINE START

73/74 OPT=1

FTN 4.3+74353

```

      GO TO 20
25  J=KEPT(I)
      IF (KEPT(I).EQ.0) GO TO 1026
175  VALX(K,J)=TEMP
      GO TO 20
27  IF(TEMP)926,20,926
C*****
C
180  C      READ THE DATA MATRIX IN
C*****
30  READ (60,21) ANAME,I,J,TEMP
      IF (ANAME.EQ.RGHT) GO TO 40
      IF (I.LE.0) GO TO 1090
185  IF (J.EQ.0) GO TO 1090
      J=KPCK+NROWS+J
      C(I,J)=TEMP
      GO TO 30
C*****
190  C
C      READ THE RIGHT HAND SIDE
40  READ (60,44) (RHS(I),I=1,NROWS)
44  FORMAT(8F10.0)
      DO 48 I=1,NROWS,
195  IF (RHS(I).LT.0.0) GO TO 941
48  CONTINUE
C*****
C
C      WRITE THE ABOVE RESULTS
200  C*****
      WRITE (61,5015)
5015 FORMAT(55X,*,THE RIGHT HAND SIDE-INPUT*,33X,*,PAGE 01*)
      DO 41 I=1,NROWS
      IF (RHS(I))941,42,43
205  42  RHS(I)=.00001
      43  RHS1(I)=RHS(I)
      WRITE (61,1111) I,RHS(I)
1111 FORMAT(10X,I3,2X,F15.5 )
      41  CONTINUE
210  WRITE (61,620)
      620  FORMAT(1H1)
      WRITE (61,5016)
5016 FORMAT(55X,*,THE SUBSTITUTION RATES-INPUT*,18X,*,PAGE 02*)
      DO 1112 I=1,NROWS
215  WRITE (61,2519) I
2519 FORMAT(1X,*,ROW#,I5)
1112 WRITE (61,1113) (C(I,J),J=1,NSIZE)
1113 FORMAT(10F12.3)
      WRITE (61,620)
220  WRITE (61,5017)
5017 FORMAT(55X,*,THE OBJECTIVE FUNCTION-INPUT*,19X,*,PAGE 03*)
      DO 1114 K=1,NPRT
      M=LISP-K
      WRITE (61,2150) M
225  2150 FORMAT(* PRIORITY*,I5)
1114 WRITE (61,1113) (VALX(K,J),J=1,NSIZE)
      WRITE (61,620)
      WRITE (61,5018)

```


SUBROUTINE START

73/74 OPT=1

FTN 4.3+74353

```

5018 FORMAT(55X,#SUMMARY OF INPUT INFORMATION #,19X,#PAGE#,# 04#)
230      NVAR=NSIZE
        WRITE (61,2017) NR0WS,NVAR,NPRT,NART
2017 FORMAT(10X,#NUMBER OF ROWS.....#,15,/,10X,#NUMBER OF VARIABLES
        #.....#,15,/,10X,#NUMBER OF PRIORITIES!..#,15,/,10X,#ADDED PRIOR
235      2ITIES.....#,15)
        IF(NART.GT.0) NPRT=NPRT+1
        RETURN
910 WRITE (61,914)
9140 FORMAT(*PROGRAM CONTAINS AN ERROR EITHER IN THE NUMBER OF ROWS PUN
1CHEC OR IN THE SIGN CARD.THE VALUE IS SOMETHING OTHER THAN (E,(G(
240      2,CRLI#)
        GO TO 999
1090 WRITE (61,1091)
1091 FORMAT(* IMPROPER DATA COLUMN OR ROW DEFINITION#)
        GO TO 999
245      920 WRITE (61,921)
921 FCRMAT(*OBJECTIVE CARD WITH VALUE#,F16.3,#WAS FCUND#,
1#BUT DEVIATION WAS OMITTED. EXAMINE INPUT DATA)#)
        GO TO 999
1020 WRITE (61,1021)
250      1021 FORMAT(* NUMBER OF ROWS, VARIABLES, OR PRIORITIES CANNOT BE EQUA
1L TO ZERO UNDER ANY CIRCUMSTANCES#)
        GO TO 999
1022 WRITE (61,1023)
255      1023 FORMAT(* COLUMN VALUE OR PRIORITY VALUE IS EQUAL TO OR LESS THAN
1ZERO #)
        GO TO 999
911 WRITE (61,912)
912 FORMAT(* NUMBER OF VARIABLES EXCEEDS CURRENT NV VALUE.#
1#MODIFY SOURCE CODE DECK FOR GOAL PROGRAMMING#)
260      GO TO 999
1026 WRITE (61,1027) EQUALS(I),ANAME,I,M,TEMP
1027 FORMAT(* ATTEMPT IS MADE TO MINIMIZE NON EXISTANT POSITIVE DEVI
1TION#/,# THE SIGN IS #,A1,/,# THE OBJECTIVE FUNCTION DATA CARD IS
2#/,1X,A4,2I5,F20.6)
265      GO TO 999
1024 WRITE (61,1025)
1025 FCRMAT(* OBJECTIVE FUNCTION PRIORITY EXCEEDS STATED NUMBER OF PRI
1ORITIES#)
        GO TO 999
270      901 WRITE (61,902)
902 FORMAT(* PROBLEM CARD MISSING OR MISPUNCHED#)
        GO TO 999
926 WRITE (61,927)
275      927 FORMAT(* OBJECTIVE FUNCTION AND DEVIATION ARE DEFINED BUT#
1# SIGN OF DEVIATION IS OMITTED#)
        GO TO 999
941 WRITE (61,942) RHS(I)
942 FORMAT(* NEGATIVE VALUES ARE NOT ALLOWED ON THE RIGHT HAND SIDE.
1 CORRECT PROBLEM BY MULTIPLYING ENTIRE CONSTRAINT THROUGH BY MINU
2S ONE.#/,# THE RIGHT HAND VALUE IS#,2X,F20.6)
280      GO TO 999
1055 WRITE (61,1056) EQUALS(I),ANAME,I,M,TEMP
1056 FORMAT(* ATTEMPT IS MADE TO MINIMIZE A NON EXISTANT NEGATIVE DEVI
1TION#/,# THE SIGN IS #,A1,/,# THE OBJECTIVE FUNCTION DATA CARD I
285      2S#/,1X,A4,2I5,F16.5)

```

SUBROUTINE START

73/74 OPT=1

FTN 4.3+74353

```

999      CCNTINUE
        RETURN
        END

```

SUBROUTINE FINISH 73/74 OPT=1

FTN 4.3+74353

```

1      C*****
      C    AND NOW FOR THE #FINISH# SUBROUTINE#
      C*****
5      C    NCTE...SEE COMMENT AT BEGINNING OF THE PROGRAM.
      C    THE ARRAY LISTED BELOW SHOULD BE AS INDICATED.
      C    VALY (100,10)
      C*****
10     SUBROUTINE FINISH(RHS1,RHS,NPRT,KPCK,Y,NROWS,KEPT,TEST)
      REAL NEGLK
      COMMON VALY(100,10)
      DIMENSION ZVAL(10)
      DIMENSION RHS(100)
15     DIMENSION KEPT(100)
      DIMENSION Y(100),RHS1(100)
      C*****
      C    RHS1 IS THE RESERVED VECTOR OF RHS VALUES FROM THE BEGINNING.
      C    THE ENDING RHS VALUES ARE SUBTRACTED FROM THE BEGINNING ONES
      C    AND THE RESULT IS PLACED INTO THE APPROPRIATE SLACK COLUMN.
20     C    THE REMAINDER OF THE VALUES ARE PRINTED ON PAGE TWO OF THE RE-
      C    SULTS.
      C
      C
      C    SLACK ANALYSIS
25     C*****
      WRITE (61,21)
21     FORMAT(1H1,120X,#PAGE 06###,50X,#SLACK ANALYSIS#)
      1 FORMAT(////)
      WRITE (61,1)
30     WRITE (61,8)
      8 FORMAT(10X,#ROW#,6X,#AVAILABLE#,12X,#POS-SLK#,12X,#NEG-SLK#)
      WRITE (61,1)
      DO 19 I=1,NROWS
      NEGLK=0.0
35     POSSLK=0.0
      DO 11 J=1,NROWS
      M=Y(J)
      IF(I-M) 9,10,9
      9 IF(M-KEPT(I)) 11,12,11
40     11 CCNTINJE
      GO TO 13
      10 NEGLK=RHS(J)
      GO TO 13
      12 POSSLK=RHS(J)
45     13 WRITE (61,14) I,RHS1(I),POSSLK,NEGLK
      14 FORMAT(10X,I3,3F20.5)
      19 CONTINUE
      43 FORMAT(10X,I3,3X,F15.5)
      C*****
50     C    VARIABLE AMOUNTS
      C*****
      WRITE (61,44)
      44 FORMAT(1H1,120X,#PAGE 07###,50X,#VARIABLE ANALYSIS#)
      WRITE (61,45)
55     45 FORMAT(////,7X,#VARIABLE          AMOUNT#,//)
      DO 41 I=1,NROWS
      NCHCK=Y(I)-KPCK-NROWS

```

SUBROUTINE FINISH 73774 OPT=1

FTN 4.3+74353

```

        IF(NCHCK)41,41,42
60      42 WRITE (61,43)NCHCK,RHS(I)
        41 CONTINUE
        WRITE (61,72)
        72 FORMAT(1H1)
        WRITE (61,50)
65      50 FORMAT(//,55X,*,ANALYSIS OF THE OBJECTIVE*,23X,*,PAGE 8*,//,50X,*,P
        IORITY*,10X,*,UNDER-ACHIEVEMENT*,/)
        DO 52 K=1,NPRT
        ZVAL(K)=0.0
        DO 51 I=1,NROWS
70      51 ZVAL(K)=ZVAL(K) +VALY(I,K)*RHS(I)
        LISP=NPRT+1
        KK=LISP-K
        IF(TEST.EQ.0.0) GO TO 52
        KK=NPRT-K
        IF(KK.GT.0) GO TO 52
75      KK=NPRT-K
        IF(KK.GT.0) GO TO 52
        WRITE (61,78) ZVAL(K)
        78 FORMAT(/,45X,*,ARTIFICIAL*,5X,F20.5)
        GO TO 77
80      52 WRITE (61,53) KK,ZVAL(K)
        53 FORMAT(1H0,52X,I2,5X,F20.5)
        77 CONTINUE
        RETURN
        END

```