

N O T I C E

THIS DOCUMENT HAS BEEN REPRODUCED FROM
MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT
CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED
IN THE INTEREST OF MAKING AVAILABLE AS MUCH
INFORMATION AS POSSIBLE

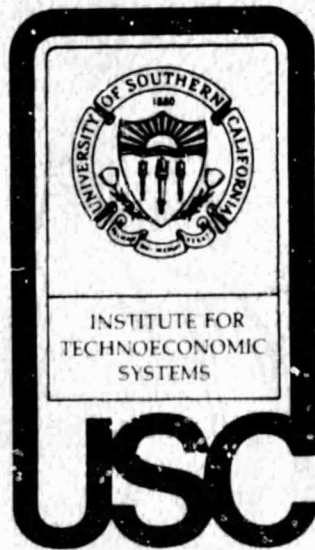
9950-645

(NASA-CR-168652) A STUDY OF INTERACTIVE
CONTROL SCHEDULING AND ECONOMIC ASSESSMENT
FOR ROBOTIC SYSTEMS Final Report
(University of Southern California) 23 p
HC A11/MF A01

N82-20878

Unclas
16815

CSSL 05H G3/54



INSTITUTE FOR
TECHNOECONOMIC SYSTEMS

UNIVERSITY OF SOUTHERN CALIFORNIA
UNIVERSITY PARK, LOS ANGELES, CALIFORNIA 90007

FINAL REPORT
for
CONTRACT #955332
JET PROPULSION LABORATORY

A STUDY OF INTERACTIVE CONTROL SCHEDULING
AND ECONOMIC ASSESSMENT FOR
ROBOTIC SYSTEMS

29 JANUARY 1982

TECHNICAL REPORT

ITEC-10-81

This work was performed for the Jet Propulsion Laboratory,
California Institute of Technology sponsored by the National
Aeronautics and Space Administration under Contract NAS7-100.

INSTITUTE FOR TECHNOECONOMIC SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007

ABSTRACT

A class of interactive control systems is derived in Part I by generalizing interactive manipulator control systems. Tasks of interactive control systems can be represented as a network of a finite set of actions which have specific operational characteristics and specific resource requirements, and which are of limited duration. This has enabled the decomposition of the overall control algorithm into simultaneously and asynchronously.

The general objective for development of Part II is to evaluate the performance benefits of sensor-referenced and computer-aided control of manipulators in a complex environment. This report represents the first phase of the CURV Arm Control System (CACS) software development, and gives the basic features of the control algorithms and their software implementation.

Part III investigates the problem of finding an optimal solution for a production scheduling problem that will be easy to implement in practical situations. The results show that the optimal solution is very easy to implement in real life problems as the jobs have to be arranged according to monotonic increasing processing times.

Part IV is an initial investigation and is the first in a series leading to a fully-developed model, ROBECON, which may be used for specifying the economic consequences of robot systems acquisitions. The model will be computer-based and user interactive

CONTENTS

- I. Asynchronous Interactive Control Systems
- II. Computer Modeling and Evaluation of Sensor-Aided Semiautomated Operations
- III. Optimal Production Scheduling for a Linear Flow Shop
- IV. ROBECON. A generalized methodology for assessing the economic consequences of acquiring robots for repetitive operations

PART I

August 31, 1981

ASYNCHRONOUS INTERACTIVE CONTROL SYSTEMS

by

M. I. Vuskovic

INSTITUTE FOR TECHNOECONOMIC SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007

M. I. Vušković
Senior Research Associate
Institute for Technoeconomic Systems
Department of Industrial & Systems Engineering
University of Southern California
Los Angeles, California 90007

ABSTRACT

A class of interactive control systems is derived by generalizing interactive manipulator control systems. The general structural properties of such systems are discussed and an appropriate general software implementation is proposed. This is based on the fact that tasks of interactive control systems can be represented as a network of a finite set of actions which have specific operational characteristics and specific resource requirements, and which are of limited duration. This has enabled the decomposition of the overall control algorithm into a set of subalgorithms, called subcontrollers, which can operate simultaneously and asynchronously. Coordinate transformations of sensor feedback data and actuator set-points have enabled the further simplification of the subcontrollers and have reduced their conflicting resource requirements. The modules of the decomposed control system are implemented as parallel processes with disjoint memory space communicating only by I/O. The synchronization mechanisms for dynamic resource allocation among subcontrollers and other synchronization mechanisms are also discussed in this paper. Such a software organization is suitable for the general form of multiprocessing using computer networks with distributed storage.

INTRODUCTION

In recent years, the emerging requirements for interactive computer-aided control of systems became progressively more evident. The more obvious areas where such requirements appear are space, undersea exploration, nuclear energy-producing facilities, and the like, where the work space is inaccessible or dangerous for man to operate in, yet where complex tasks need to be accomplished. Many of these tasks are generally not susceptible to performance by completely automated systems at the present state of automated decision-making technology. It is therefore required to develop systems that project certain human capabilities, such as sensing and handling, into the work space, thereby enabling remote operations and process control.

Remote operations, frequently also called tele-operations, benefit tremendously from computer interactive control where low-level control and decision-making functions are done by the computer, while the higher level decision functions are performed by the human operator. The efficient allocation of functions between man and machine in remote operations has been the central subject of intensive research activities at various institutions during the past decade. For handling and assembly in space, the development requirements and the state of advanced technology for the control of remote manipulators have been outlined in Ref 1.

With this background in mind, we build on and extend in this paper some of the developments for asynchronous control of manipulators in Refs 2 and 3 and construct a more general framework for asynchronous control of operations and processes requiring the scheduling of many single-actuator controllers by a "supervisory" control computer. However, it should be recognized here that a manipulator system can serve as a convenient, yet sufficiently complex, frame of reference for the discussion of more general systems.

Asynchronous control differs from synchronous control in that initiation of a single-actuator controller does not depend on cyclic interrupts of equal time intervals but, rather, on interrupts based on overall process requirements. This enables tight scheduling with minimal idle time for each actuator.

The implementation of asynchronous control is based on the decomposition of the overall control activity into a number of control subactivities which affect different actuators or groups of actuators. In the interactive control process, these actuators or groups of actuators are active only for a limited duration. The control subactivities are generally performed simultaneously but mutually time-independently, i.e., asynchronously.

In this paper, we first discuss the general physical structure of the interactive control system. Then, we study the control functions to derive their decomposition into control subfunctions. And, finally, we propose a general software implementation of the decomposed system. It is shown that all subsystem components can be made as parallel, asynchronous processes which are disjoint in the address space and which communicate between each other only by I/O. This approach builds on the works of Hoare (Ref 4) and Brinch-Hansen (Ref 5), and is oriented toward the use of a general computer environment, such as multiprocessor networks with distributed storage. Accepting degradation of reliability and efficiency, the same approach can be used in multiprocessor networks with common storage or in a single-processor environment.

GENERAL SYSTEM DESCRIPTION

Since our concern here is primarily with the control computer in Fig 1, we abstract the system as shown in Fig 2. Only units communicating directly with the control computer are shown with the corresponding incoming and outgoing information flows.

Command Device

The command unit receives commands from the human operator and translates these into a control computer-acceptable form. Commands have characteristics of discrete events and are generally issued through keyboards, voice, switches, pushbuttons, and the like. Here, we consider commands in a simplified form as a vector of Boolean variables $\underline{c} = (c_1, c_2, \dots, c_n)$.

ORIGINAL PAGE IS
OF POOR QUALITY

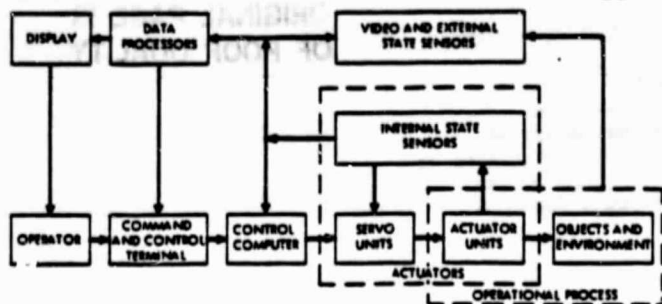


Fig. 1 Functional subsystems and information flow for interactive control of operational processes

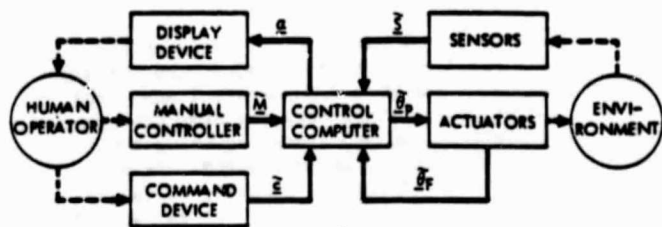


Fig. 2 General configuration of an interactive control system

which represents 2^{nc} possible instructions regarding the process by which tasks are to be executed. In Fig 2 the symbols which represent inputs and outputs of the devices have wave sign that indicates "raw" values of data. This sign will be ignored in this section.

Display Device

Generally, the display device receives, from the control computer, all information required by the human operator to make appropriate decisions for override control. This may include video data and audio signals as well as force and position data, all of which are transformed into a human operator-convenient form for display and sensing purposes.

In simpler versions of display units, one has only a fixed number of discrete messages, such as warning and alarm signals. These messages can be represented by the character string $\underline{a} = a_1 a_2 a_3 a_4 \dots$ which is generated by the control computer.

Manual Controller

The manual controller consists of one or more human-factored devices by which the human operator generates control signals that affect the execution of the tasks. The human operator acts as a supervisor by overriding and/or instructing the control computer as required. The generated signals are converted into an appropriate standardized form and sent to the control computer as a vector $\underline{M} = (M_1, M_2, \dots, M_{nm})$ of the values of the nm simultaneous signals sampled within a particular time interval.

Sensors

The sensors represent a set of sensor devices which measure the effects of the actuators on their environment and the deviation of a control process from a prescribed state. The sensor data are the essential element of all automatically supported, i.e., human operator-independent operations, that appear as a part of the overall interactive control process. The sensor signals are converted into a standardized form and are sent to the control computer in form of a vector

$\underline{S} = (S_1, S_2, \dots, S_{ns})$ of the ns simultaneous signals sampled within a particular time interval.

Actuators

The actuators are a collection of N devices each consisting of actuator units, servo units, and internal state sensors. The internal state is described by the vector of physical state variables $\underline{\theta} = (\theta_1, \theta_2, \dots, \theta_N)$. Each actuator executes set-point values or simply "set-points" received from the control computer as a vector $\underline{\theta}_p = (\theta_{p1}, \theta_{p2}, \dots, \theta_{pN})$. The physical-state variables are measured and sent back to the control computer as a sampled vector $\underline{\theta}_F = (\theta_{F1}, \theta_{F2}, \dots, \theta_{FN})$. The actuator can execute any admissible set-point in a finite time due to its own internal feedback servo control.

Control Computer

The inputs of the control computer are sensor data \underline{S} , manual controls \underline{M} , commands \underline{c} , and feedback state variables $\underline{\theta}_F$. The outputs are messages \underline{a} and set-points $\underline{\theta}_p$.

The critical control outputs are the set-points $\underline{\theta}_p$. These outputs are generated iteratively by complex algorithms based on continuous and discrete internal variables and functions. Generally, the output values $\underline{\theta}_p$ do not only depend on the present values of the inputs mentioned above, but also on their past values, i.e., on the complete history of the control process. To avoid handling of infinitely many current and past values of the input variables, a well-known practice is to introduce state variables and transition functions. These state variables can be grouped into a numeric state vector $\underline{r} = (r_1, r_2, \dots, r_R)$. Because the algorithms of the control computer also deal with logic variables, it is necessary to introduce a logic state vector $\underline{l} = (l_1, l_2, \dots, l_L)$ which is a vector of Boolean variables. Generation of the iterative sequence of the set-points $\{\theta_p(k)\}$ ($k = 0, 1, 2, 3, \dots$), where k represents the iteration index, can now be described by a recurrence relation in the following general form

$$\begin{aligned} \underline{\theta}_p(k) &= \underline{f}(\underline{M}(k), \underline{S}(k), \underline{c}(k), \underline{\theta}_F(k), \underline{r}(k), \underline{l}(k)), \\ \underline{r}(k+1) &= \underline{g}(\underline{M}(k), \underline{S}(k), \underline{c}(k), \underline{\theta}_F(k), \underline{r}(k), \underline{l}(k)), \\ \underline{l}(k+1) &= \underline{h}(\underline{M}(k), \underline{S}(k), \underline{c}(k), \underline{\theta}_F(k), \underline{r}(k), \underline{l}(k)), \\ &k=0, 1, 2, 3, \dots \\ \underline{r}(0) &= \underline{r}_0, \\ \underline{l}(0) &= \underline{l}_0. \end{aligned} \quad (1)$$

Here \underline{r}_0 and \underline{l}_0 are initial conditions of the numeric and logic state vector respectively, while \underline{f} , \underline{g} , and \underline{h} are general vector functions that represent the control algorithms implemented on the control computer. The last two functions are often called transition functions.

The mathematical construct of Eq 1 represents a sequential machine, the output of which can also be represented in the operator form

$$\underline{\theta}_p = \underline{\delta}(\underline{M}, \underline{S}, \underline{c}, \underline{\theta}_F, \underline{r}_0, \underline{l}_0), \quad (2)$$

where $\underline{\delta}$ is a vector operator defined by Eq 1. In the following, the operator $\underline{\delta}$ will be called "control algorithm" or simply "controller".

The remainder of the paper will be devoted primarily to the internal structure and functions of the controller.

DECOMPOSITION OF THE CONTROLLER

The controller is usually implemented as a monolithic software package, i.e., as a single complex sequential program. Another approach to structuring the controller is an appropriate decomposition into a set of subalgorithms or subcontrollers $\delta_1, \delta_2, \dots, \delta_n$. This second approach is discussed here. Compared to the monolithic approach, it appears to offer less complexity and better portability, flexibility, and maintainability. Moreover, this approach offers the possibility to use more attractive processing alternatives, such as multiprocessing, which could be the only solution when time constraints become critical.

Control Tasks and Actions

To decompose the controller, it is useful to consider the structure of the system tasks that have to be performed by the actuators. The actions of the actuators have varying degrees of complexity. For manipulator systems they have been grouped in Ref 2 and 3 into three categories: (1) a finite set of primitive actions (2) composite actions consisting of a sequence of primitive actions, and (3) complex actions which are networks of composite actions that are executed sequentially and/or in parallel. An aggregate of complex actions makes up a system task. Here we shall not be concerned with the hierarchy of actions. The system task will simply be considered a network of actions which, in themselves, are defined by specific control algorithms and which have specific I/O requirements.

Initiation and termination of an action during task execution depends on the state of the system which is defined by the relative position of the actuator in its environment and by other events that describe the degree of task completion. For this purpose, the event-status vector $\underline{e} = (e_1, e_2, \dots, e_n)$, has been introduced. It is a Boolean vector whose components indicate occurrences or absences of the corresponding events such as "end effector is well-aligned with the plane", or "the object is grasped", etc.

The initiation of actions is a chain-reaction-like process, where one action initiates the following action. These action initiations propagate through the action network until the system task is completed. Transitions from one action to another action are defined by the set of Boolean functions $\tau_{ij}(\underline{c}, \underline{e})$, $i, j = 1, 2, \dots, n$, $i \neq j$, of the command vector \underline{c} and the event-status vector \underline{e} . These are called transition conditions. If an action, say i , is in execution, and if the transition condition τ_{ij} becomes true, then the action j will immediately be initiated. The termination of an action is defined by a similar Boolean function $\tau_{ii}(\underline{c}, \underline{e})$, $i = 1, 2, \dots, n$. If the action i is in execution and if τ_{ii} becomes false, this action will be terminated immediately. If $\tau_{ij} = \text{true}$ implies $\tau_{ii} = \text{false}$, then the action i will be terminated simultaneously with the initiation of the action j . Otherwise the two actions will remain in parallel execution.

Subcontrollers

The subcontrollers are designed so that each one corresponds to a single action. Therefore, they will generally be executed as asynchronous, sequential programs which can be independently coded, tested, and integrated with the system by satisfying certain subcontroller interface requirements to be discussed later.

A subcontroller δ_j associated with an action j , generates the set-points for those actuators which used to be active for the execution of the action. The corresponding operator equation can be represented as follows

$$\underline{\theta}_{pj} = \delta_j(\underline{M}_j, \underline{S}_j, \underline{\theta}_{pj}, \underline{E}_{j0}, \underline{I}_{j0}) \quad (3)$$

where the numeric and logic state vectors \underline{E}_j and \underline{I}_j and their respective initial conditions \underline{E}_{j0} and \underline{I}_{j0} are derived by appropriate decomposition of the state vectors \underline{E} and \underline{I} . The variables $\underline{\theta}_{pj}$, \underline{M}_j , \underline{S}_j , and $\underline{\theta}_{pj}$ are subvectors of $\underline{\theta}_p$, \underline{M} , \underline{S} and $\underline{\theta}_p$ respectively.

The subvectors have the same dimensions as the corresponding vector in which the components of no interest for the action under execution are ignored in the computations. Two subvectors are mutually disjoint if they do not have nonignorable components in common. Therefore, the arguments of δ_j include only those components required for the computation of the set-points associated with the action j . In other words, for an action j the subcontroller δ_j generally does not need to compute the set-points for all actuators and does not need to use all sensors, the manual controls, and the actuator feedback.

It should be noted, that some actions are fully automatic and do not depend on the human operator. These actions do not call for manual inputs, i.e., the subvector \underline{M}_j is empty.

As seen, Eq 3 does not include elements of the command code vector \underline{c} as an input argument. As shown above, this vector is an argument of the transition conditions $\tau_{ij}(\underline{c}, \underline{e})$ which coordinate the activities of the subcontroller.

Generally, two different subcontrollers δ_i and δ_j can use the same components of the input vectors \underline{M} and \underline{S} , and can generate the set-points of the same actuators. If such subcontrollers work simultaneously, they interfere with each other. Therefore, subcontrollers competing for the same actuators should be coordinated so that their outputs $\underline{\theta}_{pi}$ and $\underline{\theta}_{pj}$ are mutually disjoint subvectors. However, this can considerably reduce the number of actions that can be performed simultaneously. In order to solve this problem, we introduce a coordinate transformation

$$\underline{Q} = \underline{\phi}_0(\underline{\theta}) \quad (4)$$

which minimizes the coordinate interactions for the majority of actions. Note that this transformation must have an inverse

$$\underline{\theta} = \underline{\phi}_0^{-1}(\underline{Q}) \quad (5)$$

The transformed set-points $\underline{Q}_p = (Q_{p1}, Q_{p2}, \dots, Q_{pn})$ now represent the new space called "controller space". The coordinate transformation (Eq 4) also simplifies the subcontrollers, because the controller space is more convenient for constructing the algorithms by which the actions are implemented than the previously used actuator space. For example, the controller space in the case of manipulator control systems, Ref 2, can be the hand coordinate system, while the actuator space is the joint space of the manipulator. The change of only one coordinate of the hand coordinate system, say the front distance of the end-effector from the object being manipulated, can cause simultaneous adjustments of several joint coordinates. The coordinate transformation (Eq 4) in the case of nonredundant manipulators is defined by the manipulator geometry equations.

In order to make further simplifications of the subcontrollers, a similar transformation for sensors data is introduced

$$\underline{Y} = \underline{\phi}_s(\underline{S}) \quad (6)$$

where $\underline{Y} = (Y_1, Y_2, \dots, Y_{ny})$ is the vector of the transformed sensor data. Examples are outputs of strain gauges placed in the wrist of the manipulator arm, which are transformed into orthogonal forces and

torque defined in the hand coordinate system. Transformation (Eq 6) does not necessarily need to have an inverse, i.e., $NY \leq ns$. The manual controls \underline{M} do not have to be transformed, because the manual controller can be designed so that it directly generates signals that correspond to the controller space. Because of the iterative nature of the subcontrollers and some formal conveniences, the outputs of the subcontrollers will be represented by the increments of the set-points $\underline{U}(k) = \underline{Q}_P(k) - \underline{Q}_P(k-1)$. The new operators of the subcontrollers can now be written in the form

$$\underline{U}_j = \Delta_j(\underline{M}_j, \underline{Y}_j, \underline{E}_{j0}, \underline{E}_{j0}'). \quad (7)$$

The operators Δ_j are expected to be simpler than the operators of Eq 3 because of transformations (Eq 4 and Eq 6). These transformations can be centrally executed for all subcontrollers and can be performed within another functional block to be discussed later.

As seen, the subvector $\underline{\theta}_{Y_j}$ is omitted in Eq 7. It is not essential to the controllers defined in the controller space. As will be shown later, $\underline{\theta}_Y$ is used within the actuator block (Eqs 13 thru 17). In fact, the $\underline{\theta}_{Y_j}$ could be generally retained as an argument of Δ_j , because some algorithms of subcontrollers might be based on the information of the actuator states, but it will be omitted here for the sake of the simplicity.

Channels

A triple of input and output subvectors $(\underline{M}_j, \underline{Y}_j, \underline{U}_j)$ is associated with every subcontroller Δ_j . In the following discussions this triple will be called "channel". Two channels $(\underline{M}_1, \underline{Y}_1, \underline{U}_1)$ and $(\underline{M}_2, \underline{Y}_2, \underline{U}_2)$ are mutually noninterfering if:

- 1) Inputs \underline{M}_1 do not affect the outputs \underline{U}_2 , and the inputs \underline{M}_2 do not affect the outputs \underline{U}_1 .
- 2) Changes of the environment due to the execution of \underline{U}_1 will not cause changes of \underline{Y}_2 , and changes due to the execution of \underline{U}_2 will not cause changes of \underline{Y}_1 .

It follows that the necessary condition that two channels are mutually noninterfering is that the corresponding output vectors \underline{U}_1 and \underline{U}_2 are mutually disjoint subvectors.

If two subcontrollers work on mutually interfering channels, they can interfere with each other. Because they work asynchronously, i.e., time independently, it is practically impossible to determine and to control the interference between the subcontrollers. Their algorithms must therefore be designed under the assumption that they are completely independent of each other. Consequently, only for subcontrollers working simultaneously on mutually noninterfering channels can stability be guaranteed, i.e., the convergence of the corresponding algorithms. Therefore, the system must provide the channel management that will enable the coordinated allocation and deallocation of mutually interfering channels to simultaneously active subcontrollers.

Because some actions can be more urgent than others, as is with actions invoked in emergency situations, priorities should be assigned to the actions. These priorities will resolve conflicts when two or more subcontrollers are competing for mutually interfering channels. The matter of priority assignments will be discussed later.

Having done the decomposition of the controller, the functional block diagram of the control computer can be represented as shown in Fig 3. There are six functional blocks, five of them having assigned I/O functions and coordinate transformations and one with

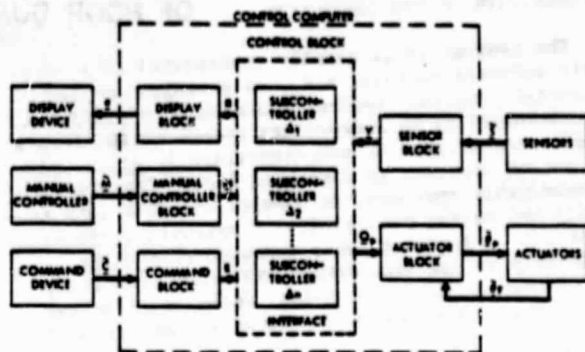


Fig. 3 Distribution of functions of the control computer

control functions. The control block consists of n subcontrollers and the necessary interfaces which enable data communication between subcontrollers internally and between subcontrollers and peripheral blocks externally. The interfaces also have the responsibility of channel management. These functional blocks and interfaces are discussed in the next two sections.

MAIN SYSTEM COMPONENTS

The functional blocks of Fig 3 will be implemented as parallel and asynchronous processes. In accordance with the system structure discussed so far, we define five peripheral processes and n processes which support subcontrollers Δ_j . The latter are called action processes. The peripheral processes support the corresponding peripheral functional blocks and are called command process, display process, manual controller, sensor process, and actuator process. These processes will now be discussed in greater detail. As will be seen later, these processes are not the only system components. There are additional components which appear as a result of the decomposition, and which support the communication between the processes. These components will be considered in the next section.

A more detailed and formal description of the system is given in the Appendix, using the notation developed by Dijkstra (Ref 6) and Hoare (Ref 4). The reader not familiar with this notation can skip the Appendix without loss of the ideas presented here.

Command Process

Assuming, for the sake of simplicity, that the command device is an array of nc on/off switches, then the command process cyclically interrogates these switches, whose positions are defined by the raw command vector $\underline{c} = (\underline{c}_1, \underline{c}_2, \dots, \underline{c}_{nc})$. It transforms this vector into the command vector \underline{c} and checks if there is any change in the commands. Checking is done by comparing \underline{c} with its value \underline{c}_{old} from the previous cycle. If $\underline{c} \neq \underline{c}_{old}$, then the new value \underline{c} will be sent to the action processes for further computations.

Display Process

We assume again the simpler case that the display device is intended for discrete messages such as warning and alarm signals. In this case, the display process receives, from the control block, a message code in the form of a Boolean vector $\underline{a} = (a_1, a_2, \dots, a_{nd})$, whenever it has a new value. It produces, then, the character string $\underline{a} = a_1 a_2 a_3 a_4 \dots$, and displays it to the human operator. The character string is generated by the program CONTEXT, which for each state of a responds by a fixed, predefined verbal message. The details of

the program are not important for the following discussion and will not be pursued.

Manual Controller

This process is performing three operations. First, it cyclically reads the raw value of the manual controls \underline{M} from the A/D converters which are attached to the controller devices. Second, it calibrates raw values in accordance with the relation

$$\underline{M} = \psi_M(\bar{M}), \quad (8)$$

where the vector function ψ_M represents all operations necessary to transform \bar{M} into standardized signals, thus providing a good interface between the human operator and the control computer (linearization, compensation for unwanted effects due to nonlinear characteristics of the controller devices, introduction of saturation and dead-band, etc.). Third, the process checks if the new manual controls \underline{M} are different from those generated in the previous cycle \underline{M}_{old} by checking the inequality

$$|\underline{M} - \underline{M}_{old}| \leq \underline{\epsilon}_M, \quad (9)$$

where $\underline{\epsilon}_M$ is a given tolerance vector. If Eq 9 does not hold, the new value \underline{M} will be sent to the action processes.

Sensor Process

This process cyclically reads raw values of sensor data \underline{S} from the A/D converters attached to the sensors, and then calibrates the data by the relation

$$\underline{S} = \psi_S(\bar{S}). \quad (10)$$

The vector function ψ_S operates similar to ψ_M , but can be more complicated due to the complexity of some sensor systems. For example, proximity sensors with fiber optics must be calibrated for environmental conditions, such as color and texture of the manipulated objects, etc. Therefore, the calibration process may need a dynamic definition of the calibration parameters. The function ψ_S may also include a filtering capability to reduce noise.

After calibration, the process checks if the new value \underline{S} is changed with respect to the value \underline{S}_{old} from the previous cycle by the inequality

$$|\underline{S} - \underline{S}_{old}| \leq \underline{\epsilon}_S, \quad (11)$$

where $\underline{\epsilon}_S$ is a given tolerance vector. If Eq 11 is true, the process will read new raw sensor data from the A/D converters. Otherwise, it will first perform the coordinate transformation

$$\underline{Y} = \phi_S(\underline{S}), \quad (12)$$

which has been discussed in the previous section, and it will then read new raw sensor data. The transformed sensor data \underline{Y} is sent to the action processes.

Actuator Process

This process has an input and output part. The input part cyclically reads raw feedback values of the actuator state variables \underline{Q}_F and performs the data calibration

$$\underline{\theta}_F = \psi_Q(\bar{\theta}_F). \quad (13)$$

It then transforms the vector $\underline{\theta}_F$ from the actuator to the controller space by a mapping

$$\underline{Q}_F = \phi_Q(\underline{\theta}_F). \quad (14)$$

which was discussed in the previous section. Finally, it defines the Boolean vector $\text{ready} = (\text{ready}_1, \text{ready}_2, \dots, \text{ready}_N)$, the components of which define the set-points of the actuator process that have been already executed, i.e.,

$$\text{ready}_i = \begin{cases} \text{true} & \text{if } |Q_{P_i} - Q_{F_i}| \leq \epsilon_{Q_i} \\ \text{false} & \text{otherwise} \end{cases} \quad i=1,2,\dots,N, \quad (15)$$

or in shorter notation

$$\text{ready} = (|Q_P - Q_F| \leq \epsilon_Q), \quad (16)$$

where Q_P is the vector of the actual set-points, and ϵ_Q is a given tolerance vector. The vector ready is sent to the action processes in each process cycle.

In the output part the actuator process receives the total increment (if any) of the set-points (this vector is composed of the partial increments \underline{U}_j generated by the different subcontrollers working simultaneously) and forms the new set-points using the current positions Q_P

$$Q_P = Q_F + \underline{U}_{tot}. \quad (17)$$

The set-points in controller space are now transformed into the actuator space by the inverse mapping

$$\underline{\theta}_P = \phi_Q^{-1}(Q_P), \quad (18)$$

and converted into the raw output value

$$\bar{\theta}_P = \psi_Q^{-1}(\underline{\theta}_P). \quad (19)$$

This is then written on the D/A converters of the physical actuator.

Execution of the input and output part of the actuator process is not essentially synchronous and alternative, but the software implementation must ensure fairness to both parts. Also, to avoid delay effects in the control loops sensor-controller-actuator, the cycle period of the actuator process must be long enough in comparison with the cycle period of the sensor process.

Action Processes

The action processes perform the actions, i.e., they execute the subcontrollers A_j ($j = 1, 2, \dots, n$) and provide the necessary administration. They all have a similar structure, with the exception that they employ a different algorithms for the subcontrollers. Therefore, the action processes will be represented as an array of processes with the subscripted names $A(1), A(2), \dots, A(n)$. Every action process can have a blocked and an active state. In the blocked state, the process is waiting to be activated by some other action process. In the active state, it cyclically checks the relevant transition conditions and executes the subcontroller, if it has the right for further existence.

Checking the transition conditions for an action process, say $A(j)$, means evaluating and checking the Boolean functions $\tau_{ij}(\underline{c}, \underline{e})$ ($i = 1, 2, \dots, n, i \neq j$), which were discussed in the previous section. If some of these expressions turn out to be true, the process $A(j)$ will notify the system which will then send awake signals to the corresponding action processes that must be activated. If $\tau_{ij}(\underline{c}, \underline{e})$ becomes false, the process $A(j)$ will immediately put itself in the blocked state. Otherwise, it will execute the subcontroller, i.e., it will invoke the procedure with the name SUB-CONTROLLER_j which is specific to the corresponding

action. Numeric and logic state vectors r_j and l_j of the subcontroller are local variables of the process $A(j)$.

Concerning the action processes and their subcontrollers, there are two important issues mentioned in the previous section. These are channel allocation and controller synchronization.

When two subcontrollers are competing for the same channel or for channels that are mutually interfering, only one subcontroller can be assigned to the channel. The other must wait until the first terminates and releases its channel. Immediately after being activated, an action process requests from the system the channel required by its subcontroller and then waits. As soon as the requested channel becomes available, the action process will be notified and activated. It will then iteratively execute the subcontroller until the corresponding action has been completed. When the subcontroller terminates, the action process notifies the system that the channel can be released. The channel can now be used by another waiting action process.

It has been noted that some actions may be more urgent than others. For instances, emergency actions must be performed immediately to prevent system failure, incorrect performance, or collision. In this case, the emergency action should be able to get the channel even if it has been allocated to another action process. In order to administer this, the priority vector $p=(p_1, p_2, \dots, p_n)$ is introduced, where the priorities p_j ($j=1, 2, \dots, n$) are associated with the processes $A(j)$. If the process $A(k)$ has a higher priority than $A(j)$ which is in progress, ($p_k > p_j$), then $A(k)$ will be granted immediate access to the channel while $A(j)$ will be temporarily blocked. Here, we only consider static priorities. More general capabilities of the system would be with dynamic priorities which can be changed during task execution by the human operator or by some algorithm.

The synchronization of a subcontroller consists of the following. The execution of the subcontroller \underline{u}_j within the action process $A(j)$ is done in an iterative manner, i.e., the subcontroller generates the sequence of the set-point increments $\{u_j(k)\}$ ($k=0, 1, 2, 3, \dots$). Here, any increment $u_j(k)$ cannot be sent to the actuator process before the previous increment $u_j(k-1)$ has been executed. In other words, when $A(j)$ wants to generate and send the $u_j(k)$, it must wait until the actuator process first executes $u_j(k-1)$, and then notifies $A(j)$ by the signal "go" that it can proceed. Only then can $A(j)$ ask for, and receive, the new values $M(k)$ and $Y(k)$ to compute the new value $u_j(k)$ and then send it to the actuator process. This synchronization mechanism, and the previously discussed channel allocation mechanisms, are external to the action processes. They will be discussed in the next section in more detail.

INTERFACES AND SYSTEM INTEGRATION

In the previous section, the main components were discussed. In order to provide a simple and reliable implementation, the processes defined should not communicate directly with each other, but they should communicate through some interfaces. For this purpose we introduce two additional software components which are a status monitor and a channel monitor. These monitors can be built as passive components with a data structure and associated operations (monitor procedures) as was proposed in the previous work (Ref 3) which was based on a monitor concept by Brinch-Hansen (Ref 7) and Hoare (Ref 8). The limitation of this approach is that the processes accessing the monitor must share common memory space. However, a more attractive possibility is if the processes are made to be mutually disjoint in memory space. Then, the system

can be implemented on a computer network with distributed storage. Solutions to this problem have been also proposed by Hoare (Ref 4) and by Brinch-Hansen (Ref 5), where the monitors are implemented as processes which communicate with other processes by I/O. These ideas are applied in the following paragraphs.

Status Monitor

The status monitor supports (a) data communication between all peripheral processes (except the actuator process) and the action processes $A(j)$ ($j=1, \dots, n$), (b) evaluation of the event-status vector e and the message-code vector u , and (c) activation of the action processes.

The data communication is organized so that the status monitor examines other processes to determine if they are ready to receive or to send data. The command process, manual controller, and sensor process send data to the status monitor whenever they are ready, and the status monitor will receive these data and will copy them into its local variables. When the action processes need these data, they ask for them by sending the appropriate signals to the status monitor (the signal "status" for e and u , and the signal "input" for M and Y). The status monitor will immediately respond by sending the corresponding data to the corresponding process. The action processes must ask for the data, because the status monitor serves more than one process, and it must respond to their frequent requests and therefore should not be delayed for longer periods.

When the status monitor receives sensor data Y it will always invoke the two procedures EVENTUPDATE and MESSAGECODE. The first procedure defines and updates the new value of the event-status vector e , using the most recent sensor data and possibly their last values. Having Y , u , and e , the procedure MESSAGECODE defines and updates the message code u . If there is a change of this vector with respect to the old value, i.e., if $e \neq e_{old}$, status monitor will send the new value u to the display process.

The mechanism of activating an action process, say $A(k)$, by another action process, say $A(j)$, is performed as follows. If $A(j)$ finds that the transition condition $r_{jk}(e, u)$ is true, it will send the integer k to the status monitor. This integer identifies the action process that must be activated. As soon as status monitor receives k , it will send the "awake" signal to the process $A(k)$. This process, if in the blocked state, is waiting for the "awake" signal from the status monitor and will change its state to the active one. The "awake" signal has no effect if the process $A(k)$ is already in the active state.

Channel Monitor

The channel monitor supports three functions: (a) data communication between action processes and actuator process, (b) channel allocation, and (c) channel synchronization.

The first function is based on the same principles as for the status monitor. The monitor receives from the actuator process the Boolean vector ready and sends the vector of set-point increments data to the actuator process. The vector data is discussed below.

The channel allocation, i.e., servicing the request/release requirements of the action processes is done by two procedures: REQUEST and RELEASE. The first procedure is invoked whenever a "request" signal is sent from the action process. The input parameter of the procedure is an integer j that identifies the requesting process. The procedures have access to the $N \times n$ Boolean matrix $m = [m_{ij}]$ ($i=1, 2, \dots, N$; $j=1, 2, \dots, n$) which defines the channel requirements for all action processes. For example, $m_{ij} = \text{true}$ indicates that the i -th input of the actuator process is required by the subcontroller Δ_j . The j -th column

$m_j = (m_{1j}, m_{2j}, \dots, m_{Nj})$ of the matrix m defines the complete channel requirement of the subcontroller A_j . The procedure also has access to the priority vector p . The matrix m and the vector p are constants local to the channel monitor which are defined at the time of system installation. When a request is made by the $A(j)$, the procedure REQUEST first checks if the true pattern of the m_j matches the pattern of the channels already allocated. If not, the channel will be immediately granted to the $A(j)$ and the signal "allocated" will be sent to it. If yes, the procedure will put the process index j into the waiting queue, which is also a data structure local to the channel monitor. If the requesting process has a priority greater than the processes holding the channel, then the monitor procedure will perform channel reallocation.

The procedure RELEASE is invoked whenever a "release" signal is sent to the channel monitor. The input parameter of this procedure is also the integer j , which identifies the process releasing the channel. The procedure examines the waiting queue for a process waiting for the channel. If such process is found (priorities are taken into account), an "allocated" signal will be sent to it, and its identification index will be removed from the waiting queue.

The third function of the monitor, the channel synchronization, is done by the procedure CHECKCHANNEL. This procedure is invoked whenever the channel monitor receives the vector ready from the actuator process. It will then check for channels which are still busy, or which are ready to accept new set-points. The check is made by comparing the vector ready with all columns of the matrix m , which correspond to the active channels. Active channels which meet the true pattern of the vector ready are ready to accept new data. These channels are defined by the index set $J_r = \{j | m_j \wedge \text{ready} = m_j \text{ and } j \in J_A\}$ where J_A is the index set of all active channels. The procedure CHECKCHANNEL will send a "go" signal to all processes $A(j)$, $j \in J_r$.

We now turn back to the vector U_{tot} . The channel monitor plays the role of a buffer that collects different portions of U_{tot} which are sent as subvectors U_j from the different processes $A(j)$. The collection of these portions is defined by:

$$U_{tot} = \sum_{j \in J_A} U_j \quad (20)$$

where U_j are "cleared" versions of the subvector U_j , defined by

$$U'_{ij} = \begin{cases} U_{ij} & \text{if } m_{ij} = \text{true} \\ 0 & \text{if } m_{ij} = \text{false} \end{cases} \quad (21)$$

System Data Flow Diagram

The system components can be connected into a system shown by the data flow diagram in Fig 4. The processes are represented by circles and the peripheral devices by squared boxes. The data and the signals are represented by arrows indicating their sources and their destinations. To indicate the difference between data and signals, the latter are shown by dashed arrows.

As seen, the diagram in Fig 4 has the form of a double star. Both stars have the monitor processes in the center. The action processes constitute the common branches of the stars. The free branches belong to peripheral processes which have access to the external physical devices.

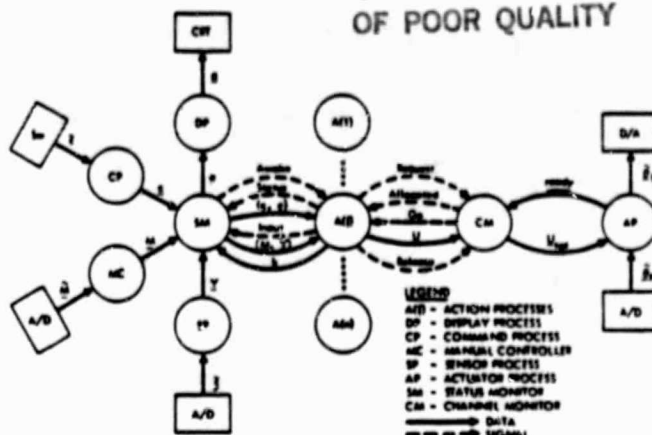


Fig. 4 Data flow diagram of the integrated system

CONCLUSION

The proposed software structure of the interactive control system is highly modularized and is suitable to be implemented on a computer network with distributed storage. The modularization is done in accordance with the varied I/O and control functions of the system. Because of this, the system can be implemented gradually and with the possibility of subsequent refinement and improvement. All modules can be designed, coded, and tested separately and almost independently of each other. The second characteristic is attractive because of the time constraints and the system reliability which become more and more important factors of such systems. Also, the trend of computer technology justifies such an orientation.

The intention here is not to give a definite and detailed implementation of the software, but to identify certain structural properties of the interactive control systems and to give hints for the application of modern concepts of real-time programming proposed by Dijkstra, Hoare, Brinch-Hansen and others.

For the sake of simplicity we ignored, in this paper: (1) the parameterization of the commands c and messages a , (2) the dependency of the subcontrollers (Eq 7) and sensor transformations (Eq 12) on the actuator-state feedback by , and (3) the dynamic assignment of channel priorities to the subcontrollers. These issues will be considered in the further work and through practical realizations.

ACKNOWLEDGEMENTS

The research described in this paper was carried out, in part, at the Jet Propulsion Laboratory, California Institute of Technology, Under NASA Contract NAS7-100. The authors also wish to acknowledge valuable discussions on the subject with Dr. A. K. Bejczy.

REFERENCES

- [1] Heer, E. and Bejczy, A. K., "Control of Robot Manipulation for Handling and Assembly in Space," 2nd IFAC/IFIP Symposium on Information Control Problems in Manufacturing Technology, Stuttgart, Germany, 22-24 Oct 1979.
- [2] Bejczy, A. K. and Vukobratović, M. I., "An Interactive Manipulator Control System," Proceedings of the

2nd International Symposium on Mini- and Micro-computers in Control, December 10-11, 1979.

- [3] Vučković, M. I. and Zawacki, R. L., "Structural Approach to the Design and Implementation of Computer Control of Manipulators," Proceedings of the 11th International Symposium on Mini- and Microcomputers, Pacific Grove, California, January 30 - February 1, 1980.
- [4] Hoare, C. A. R., "Communicating Sequential Processes," Communication of the ACM, Vol. 21, No. 8, August 1978.
- [5] Brinch-Hansen, P., "Distributed Processes: A Concurrent Programming Concept," Communication of the ACM, Vol. 21, No. 11, November 1978.
- [6] Dijkstra, E. W., "Guarded Commands Nondeterminacy and Formal Derivation of Programs," Communication of the ACM, Vol. 18, No. 8, August 1975.
- [7] Brinch-Hansen, P., "Operating Systems Principles," Prentice Hall, Englewood Cliffs, N.J., 1973.
- [8] Hoare, C. A. R., "Monitors: An Operating System Structuring Concept," Communication of the ACM, Vol. 17, No. 10, October 1974.

APPENDIX

The following statements should not be considered as a program, but as a pseudocode used to describe algorithms. Therefore, the mathematical symbols for variables and functions are used rather than symbolic names. The detailed explanation for the syntax of the notation is given in Ref 4. In this section the following nonstandard data types are used: "Boolean-vector", "realvector" and "characterstring". These are arrays of corresponding standard data types, where the dimensions correspond to the dimensions given through the paper. Variables cm , c_s and q_0 are constant vectors with corresponding dimensions.

- [DP :: DISPLAYPROCESS
- MC :: MANUALCONTROLLER
- CP :: COMMANDPROCESS
- SP :: SENSORPROCESS
- AP :: ACTUATORPROCESS
- A(j:1..n) :: ACTIONPROCESS
- CM :: CHANNELMONITOR
- SM :: STATUSMONITOR]

DISPLAYPROCESS =

a: booleanvector; u: characterstring;
*[SM?a + CONTEXT; CRT!a]

MANUALCONTROLLER =

M, M_{old}, \bar{M} : realvector;
*[ADC?M + M := $\psi_M(\bar{M})$;
b: boolean; b := (|M - M_{old}| $\leq c_M$);
[b + skip $\square \neg b$ + SM!M; M_{old} := M]]

COMMANDPROCESS

c, c_{old}, \bar{c} : booleanvector; c_{old} := false;
*[switches?c +
c := $\psi_c(\bar{c})$; b: boolean; b := (c = c_{old});
[b + skip $\square \neg b$ + SM!c; c_{old} := c]]

SENSORPROCESS =

S, S_{old}, \bar{S} : realvector;
Y: realvector;
*[ADC?S +
S := $\psi_S(\bar{S})$;
b: boolean; b := (|S - S_{old}| $\leq c_S$);
[b + skip $\square \neg b$ + Y := $\phi_S(S)$; SM!Y]]

ACTUATORPROCESS =

$\theta_F, \bar{\theta}_F, \theta_P, \bar{\theta}_P$: realvector;
 Q_F, Q_P, U_{tot} : realvector;
 $Q_P := Q_{P0}$; $Q_F := Q_P$;
ready: boolean;
*[ADC? $\bar{\theta}_F$ +
 $\theta_F := \psi_{\theta}(\bar{\theta}_F)$;
 $Q_F := \phi_{\theta}(\theta_F)$;
ready := (|Q_P - Q_F| $\leq c_Q$);
CM!ready
 \square CM?U_{tot} +
Q_P := Q_F + U_{tot};
 $\theta_P := c_{\theta}^{-1}(Q_P)$;
 $\bar{\theta}_P := \psi_{\theta}^{-1}(\theta_P)$;
DAC! $\bar{\theta}_P$]

ACTIONPROCESS =

blocked, ca, cr: boolean;
blocked := true; ca := false; cr := false;
*[blocked + SM?awake(); blocked := false
 $\square \neg$ blocked +
[SM?awake() + skip
 $\square \neg$ ca +
[\neg cr + CM!request(); cr := true
ocr + CM?allocated();
r := r₀; i := i₀; ca := true
oca + CM?go(); PERFORMACTION]]

PERFORMACTION =

c: booleanvector;
e: booleanvector;
b: boolean;
SM!status(); SM?(c,e);
i: integer; i := 0;
*[i < n +
i := i + 1;
[i = j + skip
oi \neq j + b := $\tau_{ij}(c,e)$;
[b + SM! $\square \neg b$ + skip]]]
b := $\tau_{jj}(c,e)$;
[b + SM!input();
M, Y: realvector; SM?(M, Y);
SUBCONTROLLER_j;
CM!U
 $\square \neg b$ + blocked := true; CM!release()]

CHANNELMONITOR =

U, U_{tot}: realvector; U_{tot} := 0;
empty: boolean; empty := true;
ready: boolean;
m: (1..n, 1..N) boolean;
*[(j:1..n)A(j)?request() +

```

REQUEST(...if channel free then
  A(j)!allocated( )...)
C(j:1..n)A(j)?release( ) →
RELEASE(...if this channel requested by A(m) then
  A(m)!allocated( )...)
CAP?ready →
CHECKCHANNEL(...if channel k not busy then
  A(k)!go( )...)
C(j:1..n)A(j)?U →
i: integer; i:=0;
*{i<N →
  i:=i+1;
  [~m[i,j] → skip
  om[i,j] → Utot[i]:=Utot[i]+U[i]];
empty:=false
crempty → AP!Utot; Utot:=0; empty:=true]

```

STATUSMONITOR =

```

c: booleanvector;
e: booleanvector;
a,aold: booleanvector; aold:=false;
k: integer;
M: realvector
Y: realvector;
*{(j:1..n)A(j)?status( ) → A(j):(c,e)
C(j:1..n)A(j)?input( ) → A(j):(M,Y)
OCP?c → skip
CMC?M → skip
OSP?Y → EVENTUPDATE; MESSAGECODE;
  b: boolean; b:=(a = aold);
  [b → skip
  [~b → aold := a; DP!a]
C(j:1..n)A(j)?k → A(k)!awake( )

```

ORDER PAGE 19
OF FOUR QUALITY

PART II

August 31, 1981

COMPUTER MODELING AND EVALUATION
OF SENSOR-AIDED SEMIAUTOMATED OPERATIONS

BY

M. I. Vuskovic

INSTITUTE FOR TECHNOECONOMIC SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007

ABSTRACT

The CURV Arm Control System (CACS) is a computer aided control system for interactive computer-aided control of the six-degree-of-freedom manipulator of the JPL teleoperator laboratory. The manipulator is equipped with proximity and force-torque sensors. In order to perform complex tasks like tracking, capturing and stopping of slowly moving heavy objects, the human operator commands are supported by automatic control algorithms based on sensory feedback data. The general objective of this development project is to evaluate the performance benefits of sensor-referenced and computer-aided control of manipulators in a complex environment. This progress report represents the first phase of the CACS software development, and gives the basic features of the control algorithms and their software implementation. The control structure development is based on three concepts: incremental motion synthesis, basic control routines, and parallelism of algorithms. Incremental motion synthesis consists of generating a series of motion increments instead of generating endpoint values. This enables a unified handling of position and rate control modes of the manipulator, and uses simpler coordinate transformations based on linearization. The basic control routines represent the set of elementary algorithms for generating different kinds of motion increments shared by all algorithms of higher levels of the control hierarch. The parallelism of algorithms is a natural consequence of considering manipulator activities as integral components of complex manipulator tasks. Because of the relatively complex control structure and its inherent parallelism, special attention has been paid to its software implementation. Therefore, modern concepts of monitors and concurrent processes are applied in this work.

ABBREVIATIONS

- CACS - CURV Arm Control System
- TO - Target object
- TP - Tracking plane
- EE - End effector
- CAL - Common Assembler Language
- WCS - World Coordinate System
- JCS - Joint Coordinate System

CONTENTS

I.	FUNCTIONAL DESCRIPTION -----	1-1
A.	INTRODUCTION -----	1-1
B.	OBJECTIVES -----	1-1
II.	FUNCTIONAL REQUIREMENTS -----	2-1
III.	OPERATIONAL DESCRIPTION -----	3-1
A.	GENERAL SYSTEM DESCRIPTION -----	3-1
1.	Manipulator -----	3-1
2.	Operator Control Console -----	3-3
3.	Control Computer -----	3-6
4.	Computer Console -----	3-6
5.	Alarm Display -----	3-8
6.	Computer Interface -----	3-8
B.	CONTROL ALGORITHMS -----	3-10
1.	Incremental Motion Synthesis -----	3-10
2.	Application of the Incremental Motion Synthesis in the CACS -----	3-13
3.	Basic Control Routines -----	3-18
4.	Dynamic Response of Manipulator and Delay -----	3-26
5.	Parallel Processing Concept -----	3-29
6.	Application of the Parallel Processing Concept in the CACS -----	3-33
C.	SOFTWARE IMPLEMENTATION -----	3-39
1.	General Software Architecture -----	3-39
2.	Parameter Editor Subsystem -----	3-39

CONTENTS (contd)

3.	Manipulator Testing Subsystem -----	3-43
4.	Monitor Concept -----	3-43
5.	Basic Prerequisites and Assumptions -----	3-46
6.	Implementation of Monitors and Processes -----	3-48
7.	Scheduler -----	3-52
8.	Program Documentation -----	3-59
IV.	CONCLUSIONS AND PLANS -----	4-1
V.	REFERENCES -----	5-1
APPENDIXES		
A.	GRAPHICAL REPRESENTATION OF DATA AND STANDARD OPERATION -----	A-1
B.	TBD -----	B-1
<u>Figures</u>		
2-1	Manual/Automatic Tracking/Grasping Operation Sequence -----	2-2
3-1	Data Flow Diagram of CACS -----	3-3
3-2	Principle of Incremental Motion Synthesis -----	3-11
3-3	Example of Combined Motion -----	3-14
3-4	General Data Flow Diagram of CACS -----	3-15
3-5	Translatory Motion of the Arm in World Coordinate System --	3-20
3-6	Rotational Motion of the Hand in World Coordinate System --	3-21
3-7	Translatory Motion of the Arm in World Coordinate System --	3-22
3-8	Constrained Translatory Motion of the Arm in Tracking Plane -----	3-23
3-9	Expansion and Contraction of the Jaw -----	3-24
3-10	Asymmetric Expansion and Contraction of the Jaw (Fixed Left Side) -----	3-27

CONTENTS (contd)

3-11 Asymmetric Expansion and Contraction of the Jaw (Fixed Right Side) -----	3-29
3-12 Dynamic Response of the Manipulator -----	3-31
3-13 Process Precedence Chart -----	3-33
3-14 General Outline of Temporary Active Processes -----	3-36
3-15 Process Precedence Chart of the OPER Subsystem -----	3-38
3-16 System Block Diagram of Parameter Editor Subsystem -----	3-41
3-17 Example of Parameter Display -----	3-43
3-18 General Outline of Buffer Monitors (in Pascal Language) ---	3-48
3-19 General Structure of the Module -----	3-52
3-20 Access Graph of Subsystem OPER -----	3-54
3-21 Access Right Cross-Reference Table -----	3-55
3-22 Scheduler - Main Program -----	3-56
3-23 Scheduler - Process Multiplexing -----	3-57
3-24 Process Synchronization Monitor MPRQ -----	3-59
3-25 -----	3-62
3-26 -----	3-63
3-27 -----	3-64
A-1 Graphical Representations of Data and Standard Operations--	A-3

Tables

3-1 Functional Switches -----	3-5
3-2 Pushbuttons -----	3-7
3-3 Alarms -----	3-9
3-4 Basic Control Routines -----	3-19
3-5 CACS Processes -----	3-34
3-6 Event Flags -----	3-38
3-7 CACS Monitors -----	3-50

SECTION I

FUNCTIONAL DESCRIPTION

A. INTRODUCTION

The CURV Arm Control System (CACS) is a computer aided system which provides interactive human operator and computer control of the manipulator. The system generally consists of two parts: hardware components and software. The hardware components include computer hardware, the JPL/CURV arm equipped with sensor systems and an operator control console designed as a universal control panel. These components are part of the JPL Teleoperator Laboratory and are described in Refs. 1-3.

The CACS software is a new system component intended to support a class of real-time manipulator control activities such as tracking, grasping, and stopping of slowly moving objects.

B. OBJECTIVES

The objectives which are pursued in this project are given in the proposal "Develop Experimental Modeling and Evaluation of Sensor-Aided Manipulator Control" which was submitted to JPL in November 1978. The objectives will be briefly reviewed here.

The general objective of the project is to demonstrate and determine experimentally the impact of sensor and computer aided manipulator control on overall task performance. The experimental nature of this general objective implies two major points:

- (1) The sensor/computer aids are tools in the hand of a human operator, and consequently, a major concern is to provide a proper interface between sensor/computer tools and the human operator.
- (2) The sensor/computer aids are real-time tools, and consequently, their performance properties should match the versatility of a real-time control environment.

The specific objective is to develop real-time computer control programs for the JPL/CURV manipulator referenced to proximity and force-torque sensors.

The specific objective includes also that a well-designed software basis shall be provided for further research on the following issues:

- (1) Development, testing and improvement of new control strategies and algorithms.
- (2) Investigation of the impact of design parameters including the physical characteristics of proximity and force-torque sensors and other relevant system components on overall control capabilities and on overall system performance.
- (3) Investigation of the impact of real environment (irregularities of objects, noise, component imperfections, etc.) on control capabilities.
- (4) Study of software approaches to the solution of manipulator control problems.

This report is the first quarterly progress report in which the basic concepts of control algorithms and their software implementation are considered. The programs developed thus far are described in the program documentation given in Appendix B of this report, which has been issued as a separate volume.

Before starting the system description, the functional requirements of the CACS software will be reviewed.

SECTION II

FUNCTIONAL REQUIREMENTS

The class of manipulator tasks, which the CACS software has to support, can be defined by the following control functions:

- (1) Unconstrained control of the arm, hand and jaw in Cartesian coordinates (world space).
- (2) Constrained control of the manipulator, i.e., moving the hand with the end-effector (EE) at a constant distance above a fixed tracking plane (TP).
- (3) Tracking a target object (TO) which is slowly moving on the TP with constant speed, with arbitrary orientation and with straight line trajectory.
- (4) Grasping and stopping the TO which is stationary or moving on the TP.

These functions must be performed interactively from the operator's control console (universal control panel) which is specially designed for this purpose.

To facilitate the interactive manipulator control, the following automatic operations should be supported by CACS software. These operations are as follows:

- (1) Roll and pitch alignment of the EE to TP.
- (2) Tracking (identification) of TO speed.
- (3) Yaw alignment of the EE to the TO.
- (4) Centering the EE to the TO for best grasping.
- (5) Grasping and stopping of the TO with compliance to its motion dynamics.

All manipulator control functions should be capable of being performed independently, or to be imbedded in one continuous sequence of operations. Such a sequence is shown in Fig. 2-1, where an idealized ordering of system states and the corresponding transition operations are depicted. This sequence pattern exists if all operations are successfully accomplished and all system states are stably attained. If some algorithm fails or system disturbances occur, the corresponding transition operation must be repeated, and the sequence pattern becomes more complicated.

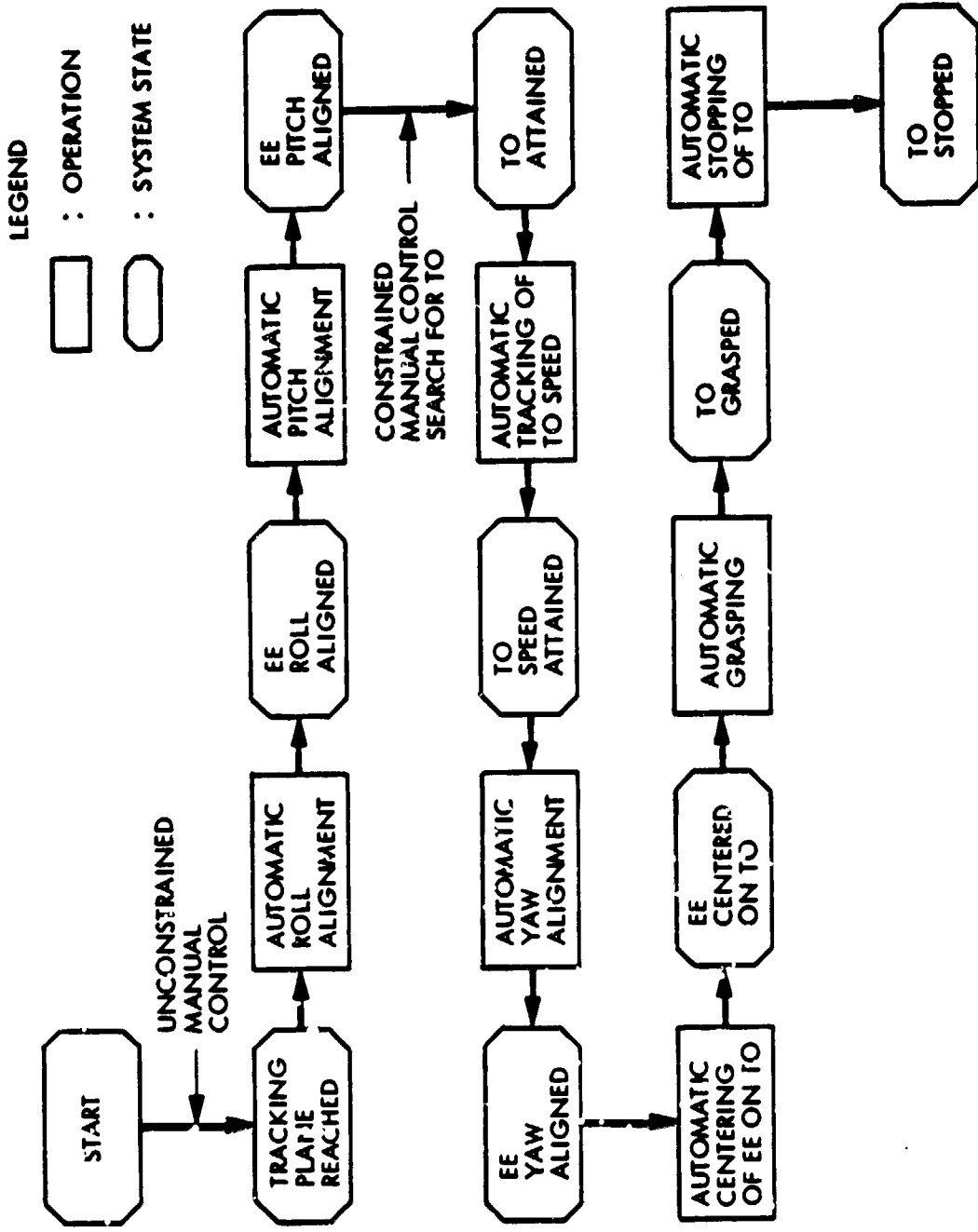


Fig. 2-1. Manual/Automatic Tracking/Grasping Operations Sequence

As seen in Fig. 2-1, all operations are sequential. This means that they are executed one after another, bringing the system from state to state. However, there is a need for some operations that can support the state transition process and maintain the stability of attained states. These operations are not shown in Fig. 2-1, but should be performed simultaneously with the sequential operations. Examples are:

- (1) Maintenance of constant distance of EE from TP.
- (2) Maintenance of constant distance of EE from TO.

The first operation starts immediately as EE is roll and pitch aligned to TP, i.e., when the manipulator begins the constrained motion over the TP, and terminates when TO is stopped. The second operation starts when the speed of TO is identified, and terminates when EE starts the grasping procedure.

Automatic operations help the operator to carry out manipulator control tasks. However, situations can occur when automatic operations require operator's assistance. For example, the yaw alignment algorithm can have difficulties caused by an inconvenient angle of TO trajectory and/or speed. In that case, it should be allowed for the operator to control the hand angles directly from his control panel. There are many similar examples. Therefore, to facilitate or speed up the automatic operations, two new operations are added to the operation list given above:

- (1) Operator's manual assistance.
- (2) Operator's emergency stop.

The first operation enables the operator to issue to the manipulator commands which can be executed simultaneously with the ongoing automatic operations, without interrupting them. The second operation allows the operator to put the system in a hold state for a certain time period to do off-line interventions without aborting the whole control process. Both operations are parallel to other operations.

The operations discussed thus far define the basic functional requirements of the CACS software related to the basic interactive control of the manipulator. However, there are also other capabilities which are provided through this project. With reference to the objectives given in Section I Introduction, the CACS is considered a research and development system rather than a final product. This imposes additional requirements which extend the control features. The most important additional capabilities are the following:

- (1) Centralized parameter maintenance.
- (2) Automated system testing.

The first capability provides a fast, easy and reliable setting and modification of system parameters, while the second provides a fast and comprehensive system testing after hardware/software changes. Both features are implemented as an integral part of the CACS software.

SECTION III
OPERATIONAL DESCRIPTION

A. GENERAL SYSTEM DESCRIPTION

An overview of the CACS is given in Fig. 3-1. As seen, the system employs the following hardware units:

- (1) Manipulator.
- (2) Operator control console.
- (3) Control computer.
- (4) Computer console.
- (5) Alarm display.
- (5) Computer interface.

A brief description of these units is given in the following six paragraphs.

1. Manipulator

The manipulator comprises the JURV linkage arm with EE, actuators, servo-potentiometers and sensor systems. The latter are force-torque and proximity sensors supported by corresponding electronics. Details of these components are given in Refs. 1-3.

The outputs of the manipulator can be divided into two groups of data: sensor data and joint position feedback data. The sensor data are represented by the vectors $\bar{w} = (\bar{w}_1, \bar{w}_2, \dots, \bar{w}_8)$ and $\bar{s} = (\bar{s}_1, \bar{s}_2, \dots, \bar{s}_4)$ which correspond to force-torque and proximity data respectively. The wavy sign over the variables denotes their "raw" values which must be converted into their corresponding mathematical values by a calibration procedure. Ordering of proximity sensor data is as follows:

- s_1 - front left
- s_2 - front right
- s_3 - lower left
- s_4 - lower right

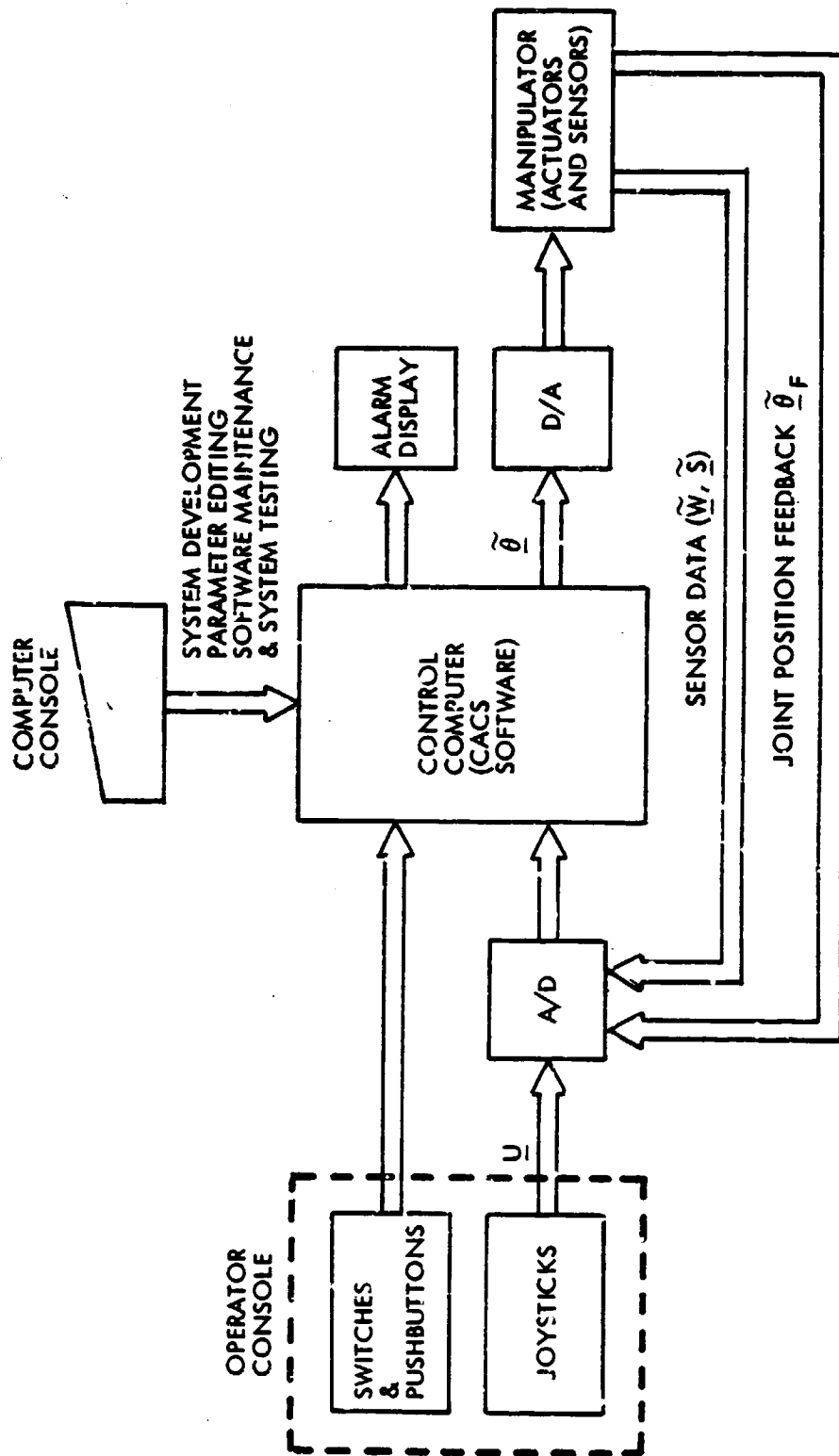


Fig. 3-1. Data Flow Diagram of CACS

The joint position feedback vector $\bar{\theta}_F = (\bar{\theta}_{F1}, \bar{\theta}_{F2}, \dots, \bar{\theta}_{F7})$ represents the servo-potentiometer readings of joint angles:

- θ_1 - arm azimuth
- θ_2 - arm elevation
- θ_3 - arm extension
- θ_4 - hand azimuth
- θ_5 - hand elevation
- θ_6 - hand twist
- θ_7 - gripper opening

2. Operator Control Console

The operator control console is an interface between operator and control computer, which enables the interactive control of the manipulator. The main features of the control console are two joysticks for arm and hand control, potentiometer for EE control (opening/closing the jaw), and a group of functional switches and pushbuttons.

The joysticks provide easy position or rate control of three independent space coordinates where the magnitude of position/rate is proportional to stick deflection. Details of joysticks, potentiometer for EE control and other features of the console are described in Refs. 1-3.

Joysticks and potentiometer for EE control generate an output vector \underline{u} which has two versions:

$$\underline{u} = (\Delta x_c, \Delta y_c, \Delta z_c, \Delta \alpha_c, \Delta \beta_c, \Delta \gamma_c, \Delta g_c) \quad - \quad \text{for position control}$$

$$\underline{u} = (\dot{x}_c, \dot{y}_c, \dot{z}_c, \dot{\alpha}_c, \dot{\beta}_c, \dot{\gamma}_c, \dot{g}_c) \quad - \quad \text{for rate control}$$

where $\Delta x_c, \Delta y_c, \Delta z_c$, and $\Delta \alpha_c, \Delta \beta_c, \Delta \gamma_c$ are commanded translational displacements of the arm and rotational displacements of the hand respectively, both in world coordinates. Variables $\dot{x}_c, \dot{y}_c, \dot{z}_c$ and $\dot{\alpha}_c, \dot{\beta}_c, \dot{\gamma}_c$ are translational and rotational speeds of the arm and hand in the same coordinate system. Values Δg_c and \dot{g}_c represent commanded increments and commanded speeds of the jaw opening, respectively.

Functional switches and pushbuttons are new features which are peculiar to CACS, and which have to be added to the operator console. During the development phase, these switches will be implemented on the computer panel. The list of all switches is given in Table 3-1. The following comments will help the understanding of this table.

If the switch SMC is in the "off" position, the operator has full manual control of the manipulator. The commanded position/rate values will result in arm/hand movement and jaw opening/closing, which is unconstrained within the manipulator's motion envelope. Choice of position or rate mode of control is made by switch SRT.

If the switch SMC is in the "on" position, the manual control of the manipulator becomes bounded to a fixed plane which has been defined previously (default value is x-y plane in world coordinate system). In this case, the control system will ignore the commanded values Δz_c or z_c , and the resulting motion of the arm will be constrained to the given plane. For instance, the distance of the arm from a given plane will automatically be maintained at a constant value. Commanded values Δx_c , Δy_c or x_c , y_c will be taken as displacements or speeds in the new coordinate system defined by the plane.

Switches SAT, SAY, SAC and SAG define the corresponding automatic operations if the required conditions are met. For example, if the switch SAT is set, the control system will automatically take the control over from the operator when the front proximity sensors register a preselected "proximity distance" from the TO.

By switch SSC the operator can start an automatic search for the TO by scanning with the EE in the work space. Parameters of this operation must be inputted previously from the computer console noting the parameter editing procedure.

Switch SOS enables operator interference during automatic operations. For example, the operator can adjust the hand angles simultaneously with automatic tracking of TO to accelerate this action or, to support this action completely. Without setting this switch, no joystick commands will be acknowledged during automatic actions.

By switch SIN the operator can reinitialize the system, i.e., to put the system in the initial state as it was in the beginning of the manipulator operation sequence. After reinitialization, the system starts from the beginning.

Setting the switch SES, the operator immediately stops the movement of the arm and puts it in the hold state. The arm can again be released only by resetting this switch.

Table 3-1. Functional Switches

Symbolic Name	Functional Description
SIN	Initialization of the system
SMC	Manual constrained control (searching for T0 in tracking plane)
SAT	Automatic tracking
SAY	Automatic yaw alignment
SAC	automatic centering
SAG	Automatic grasping and stopping
SRT	Rate control
SSC	Automatic searching by scanning
SOS	Operator's manual assistance
SES	Operator's emergency stop

The list of pushbuttons is given in Table 3-2.

By pressing the pushbutton TRE, the operator can release the arm from the blocked position ("hold state"). Namely, during manual control of the manipulator, the computer control system automatically puts the arm in the hold state when the proximity sensors detect the proximity of the TP or the TO. In this case, an alarm signal will warn the operator who must command an appropriate motion to the arm to avoid collision with TP and/or TO. Before issuing corrective commands, the operator must release the arm by pressing pushbutton TRE.

If the pushbutton TPC is pressed, the computer control system will immediately memorize the current values of the hand coordinates in order to define the orientation of the tracking plane for subsequent constrained control. This action will always be taken when TPC is pressed. This means that the TP coordinate setting can be done more than once.

By pressing the pushbutton TCA, the operator can turn off all alarms displayed at that time.

By pressing pushbutton TPA, the operator invokes a parameter updating procedure. Namely, all parameters are stored in a particular redundant storage area which is accessible by the parameter editor subsystem EDIT (see Subsection C paragraph 1) concurrently with manipulator operation. After updating the parameters by EDIT, the new parameter values must be passed to the corresponding parameter locations of the control subsystem (OPER). This is automatically performed by special transfer procedures which are executed immediately after pressing the TPA pushbutton.

3. Control Computer

The control computer is the heart of the CACS. It consists of an INTERDATA M70 minicomputer and the CACS software package which runs under OS/16 MT2 real-time operating system. The CACS software contains three subsystems: testing subsystem (TEST), parameter editing subsystem (EDIT), and manipulator control subsystem (OPER). The former two subsystems are described in Subsection C paragraph 1 of this report. The latter subsystem is the main part of the CACS software. All three subsystems are implemented as different tasks.

4. Computer Console

Communication with the operating system and with the CACS software is through the computer console. It is a teletype unit, but can be

Table 3-2. Pushbuttons

Symbolic Name	Functional Description
TRE	Release of the arm from the hold state
TPC	Setting the TP coordinates
TCA	Clearing of all alarms
TPA	Parameter transfer to the control subsystem

any other suitable peripheral device. Communication with the TEST and EDIT subsystems is only possible through the computer console, while communication with the OPER subsystem is through the operator augmented control console.

5. Alarm Display

The purpose of the alarm display is to warn the operator in case of system irregularities, system abort, or whenever his assistance is needed. The alarm display can be implemented as a mosaic field of labeled lights or as alphanumeric messages on the CRT display. During the development phase the latter will be used. The complete list of alarms is given in Table 3-3.

6. Computer Interface

Computer interface consists of analog to digital (A/D) and digital to analog (D/A) converters. The input to the A/D converter is the compound vector $(\tilde{q}_p, \tilde{u}, \tilde{w}, \tilde{s})$, while the input to the D/A converter is the joint position vector $\tilde{\theta} = (\tilde{\theta}_1, \tilde{\theta}_2, \dots, \tilde{\theta}_7)$.

Table 3-3. Alarms

Symbolic Name	Cause/Action to be Taken by Operator
ALIM	Arm/hand on the boundary envelope.
ATPP	EE in the proximity of TP. System is in HOLD state and operator has to release it by pressing TRE in order to move arm away from TP. (Alarm ignored if SMC set.)
ATOP	EE in proximity of TO. System is in HOLD state and operator has to release it by pressing TRE in order to move EE away from TO. (Alarm ignored if SAT set, or SMC reset.)
AYAW	Angle between motion vectors of EE and TO is greater than 90 degrees. Automatic yaw alignment not possible. Operator assistance is needed.
ATOL	TO lost, control is given back to manual constrained searching.
ARPL	Roll or pitch alignment lost. Operator assistance required.
ATOG	TO too large, cannot be grasped.
ASTP	TO grasped, but cannot be stopped by given force-torque limit.
AGRT	Grasping terminated. Procedure must be repeated.
ASCT	Search by scanning has been terminated.
ASCC	Search by scanning has been completed successfully.
ACOL	EE in collision with TP or TO. Automatic action of moving EE one step from TP or TO has been taken. Operator's assistance may be needed.

B. CONTROL ALGORITHMS

In order to develop the system described in Section III with the requirements in Section II, a variety of control algorithms has been and will be developed. The complexity of the system requires a structural approach to the development of the control algorithms to provide conditions for an easy functional decomposition of the system and to assure easy and stable system integration. This is especially important in a laboratory development environment in which a stepwise development/refinement is required, as is the case with CACS.

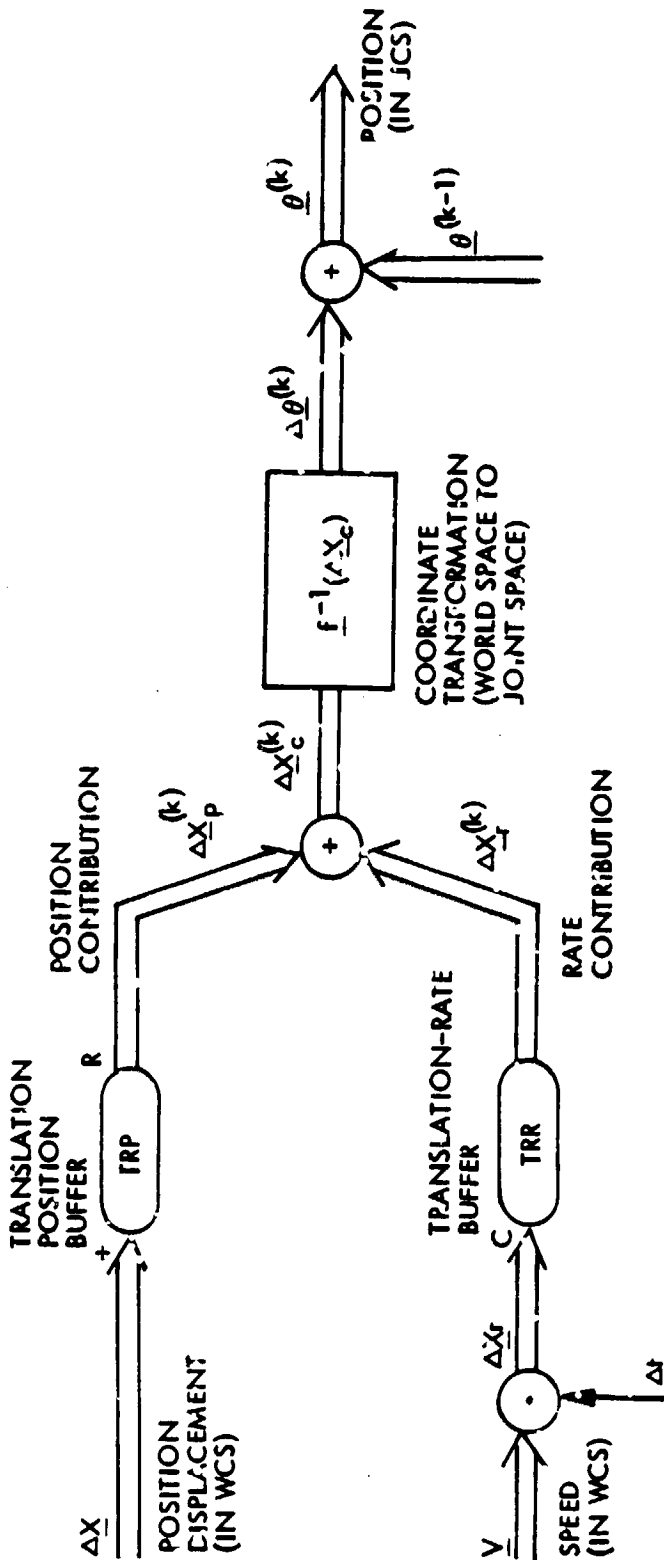
As guidelines for a structural approach to the control algorithms development, the following three concepts are introduced:

- (1) Incremental motion synthesis.
- (2) Basic control routines.
- (3) Parallel process concept.

All these guidelines and their application will be considered in the next six subsections.

1. Incremental Motion Synthesis

The idea of incremental motion synthesis requires the production of increments of motion rather than "endpoint" values in each system iteration cycle. The principle is depicted in Fig. 3-2 (graphical symbols used in this picture are explained in appendix A). Vectors $\Delta \underline{x}$ and \underline{v} represent incremental translatory displacement and translatory speed which must be performed by the manipulator EE. These quantities are effected by the control algorithms to perform the required automatic operations initiated by manually issued commands. The position increment is added to the content of the buffer TRP (translatory-position). The speed vector is converted to a corresponding rate increment $\Delta \underline{x}_r$, by multiplying with the clock interval Δt , and stored in the buffer TRR (translation-rate). In each cycle the contents of both buffers are read and summed. The resulting increment $\Delta \underline{x}^{(k)}$ represents a composed motion increment which must be transformed into a corresponding increment in the joint space coordinate system $\Delta \underline{\theta}^{(k)}$. This value will be added to joint position vector $\underline{\theta}^{(k-1)}$ which has been generated in the previous iteration cycle. The new vector $\underline{\theta}^{(k)}$ will be used in the current iteration cycle. The essential point here is that the position buffer TRP must be reset after reading, while the content of the rate buffer TRR remains unchanged. This will cause arm/hand movement with constant speed \underline{v} and simultaneous arm/hand displacement $\Delta \underline{x}$. The latter will not occur in the next iteration cycle, if the position buffer is not filled



$$C: \Delta x_{r_i} \leq \Delta x_{r_{max}} \quad (i = 1, 2, 3)$$

Fig. 3-2. Principle of Incremental Motion Synthesis

again. In order to execute displacement $\Delta \underline{x}$ in one iteration cycle, the magnitude of this vector must be less than the fixed value determined by the iteration cycle and by the manipulator dynamic performance. If this is satisfied, the position displacement of the manipulator can be considered as an instantaneous displacement. If the position increment does not satisfy this requirement, the corresponding operation must be delayed. This will be discussed latter.

The incremental approach enables easy coordinate transformation. Usually the equation of joint variables are given in the following form:

$$\underline{x} = \underline{f}(\underline{\theta}) \quad (1)$$

where $\underline{f}(\underline{\theta})$ represents a vector mapping of joint coordinates into the world (Cartesian) coordinate system. Using the incremental form of variables, (1) can be written:

$$\underline{x}^{(k)} = \underline{x}^{(k-1)} + \Delta \underline{x}^{(k)} = \underline{f}(\underline{\theta}^{(k-1)} + \Delta \underline{\theta}^{(k)}) \quad (2)$$

For sufficiently small increments $\Delta \underline{\theta}^{(k)}$, the Taylor expansion of (2) gives:

$$\Delta \underline{x}^{(k)} \cong \underline{H}^{(k)} \Delta \underline{\theta}^{(k)} \quad (3)$$

or:

$$\Delta \underline{\theta}^{(k)} \cong \underline{H}^{(k)-1} \Delta \underline{x}^{(k)} \quad (4)$$

where:

$$\underline{H}^{(k)} = \underline{H}(\underline{\theta}^{(k)}) = \left. \frac{\partial \underline{f}}{\partial \underline{\theta}} \right|_{\underline{\theta} = \underline{\theta}^{(k)}} \quad (5)$$

represents the Jacobian of the transformation (1).

Having this in mind, as well as the description given above, the advantages of the incremental motion synthesis principle can be summarized as follows:

- (a) It fits the nature of cyclic processes and, therefore, enables a more convenient implementation than the endpoint approach.
- (b) It allows unified handling of position and rate control modes.
- (c) It enables easy synthesis of composed motions of both types: position-position and position-rate. (An example of a combined motion of the position-rate type is given in Fig. 3-3, where the EE which tracks the target object with the same speed must correct its relative position Δn .)
- (d) It provides easy coordinate transformations.

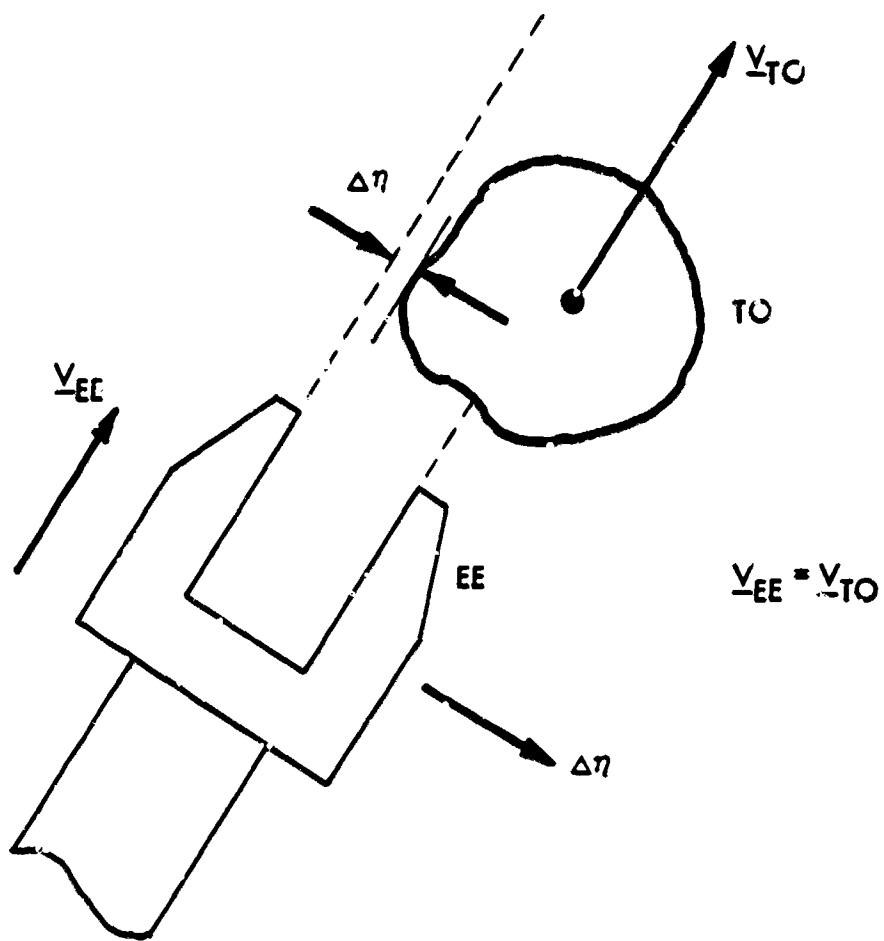
2. Application of the Incremental Motion Synthesis in the CACS

The principle of incremental motion synthesis described in the previous subsection is applied to the CACS. The main features of the application are given in Fig. 3-4. This diagram is derived from a basic property of the CURV arm: hand orientation is independent from arm elevation and extension. This is due to the double parallelogram mechanism added to the linkage. Therefore, the coordinates of the arm and hand can be handled separately, i.e., the position displacement and speed composition shown in Fig. 3-2 is now directly applied to the arm position coordinates $\underline{x} = (x, y, z)$ and to the hand orientation angles $\underline{\alpha} = (\alpha, \beta, \gamma)$. For that reason two new buffers are introduced: ROP (rotation-position) and ROR (rotation-rate). The jaw operation is considered only as a matter of position control. Therefore, only one buffer JAW is introduced for this coordinate.

Fig. 3-4 will now be explained step by step. Raw values read from A/D buffer are grouped into four vectors which are already described in Section IIIA. All these variables are calibrated separately. Calibration of joint position feedback is done by the following linear equations:

$$\theta_{Fi} = \tilde{\theta}_{Fi} \cdot a_i - b_i, \quad i = 1, 2, \dots, 7, \quad (6)$$

where a_i are scale factors of dimension rad/Volt, and b_i are zero offsets. These coefficients will be considered as system parameters.



$\Delta\eta$ POSITION DISPLACEMENT CONTROL

\underline{V}_{EE} VELOCITY VECTOR CONTROL

Fig. 3-3. Example of Combined Motion

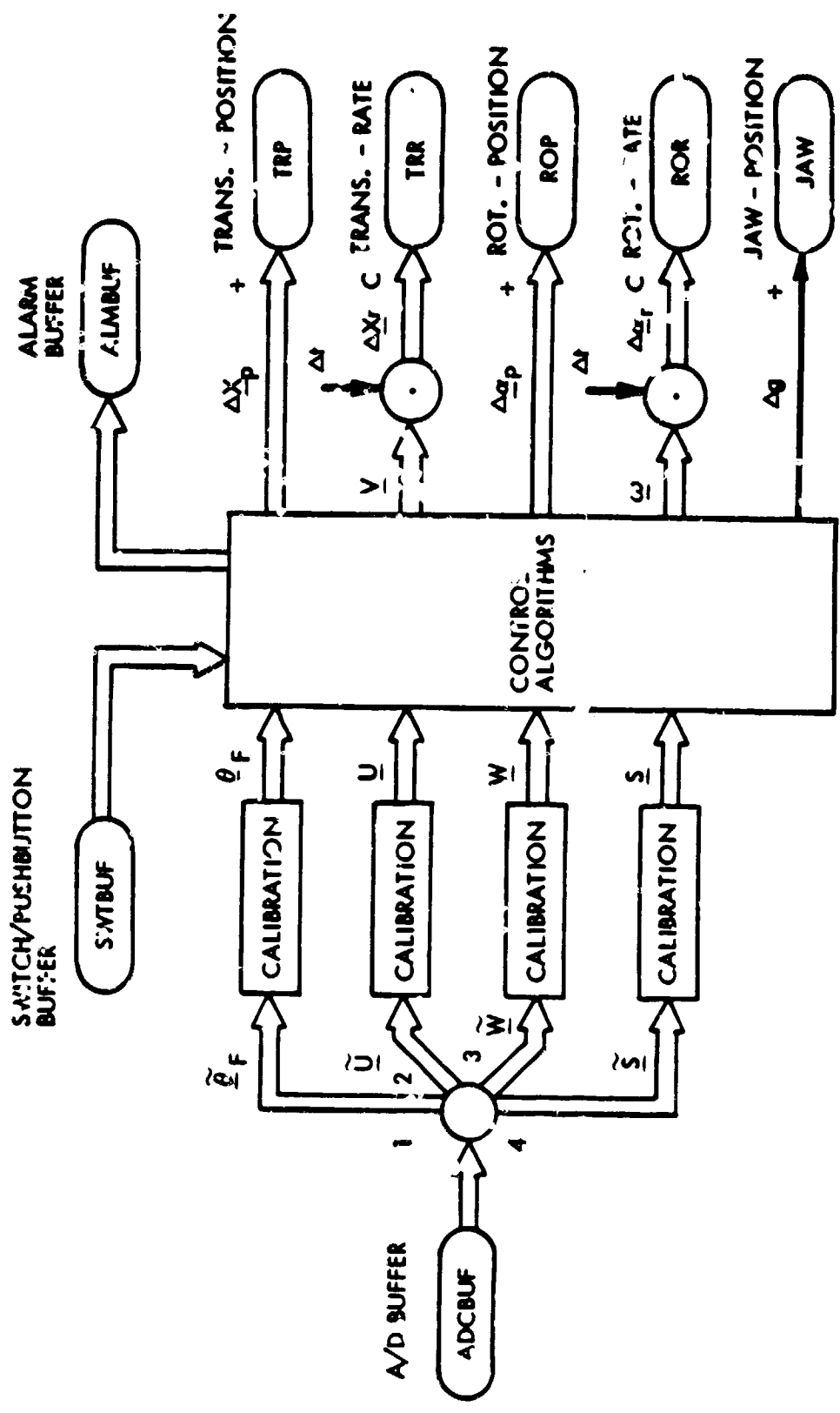


Fig. 3-4. General Data Flow Diagram of CACS

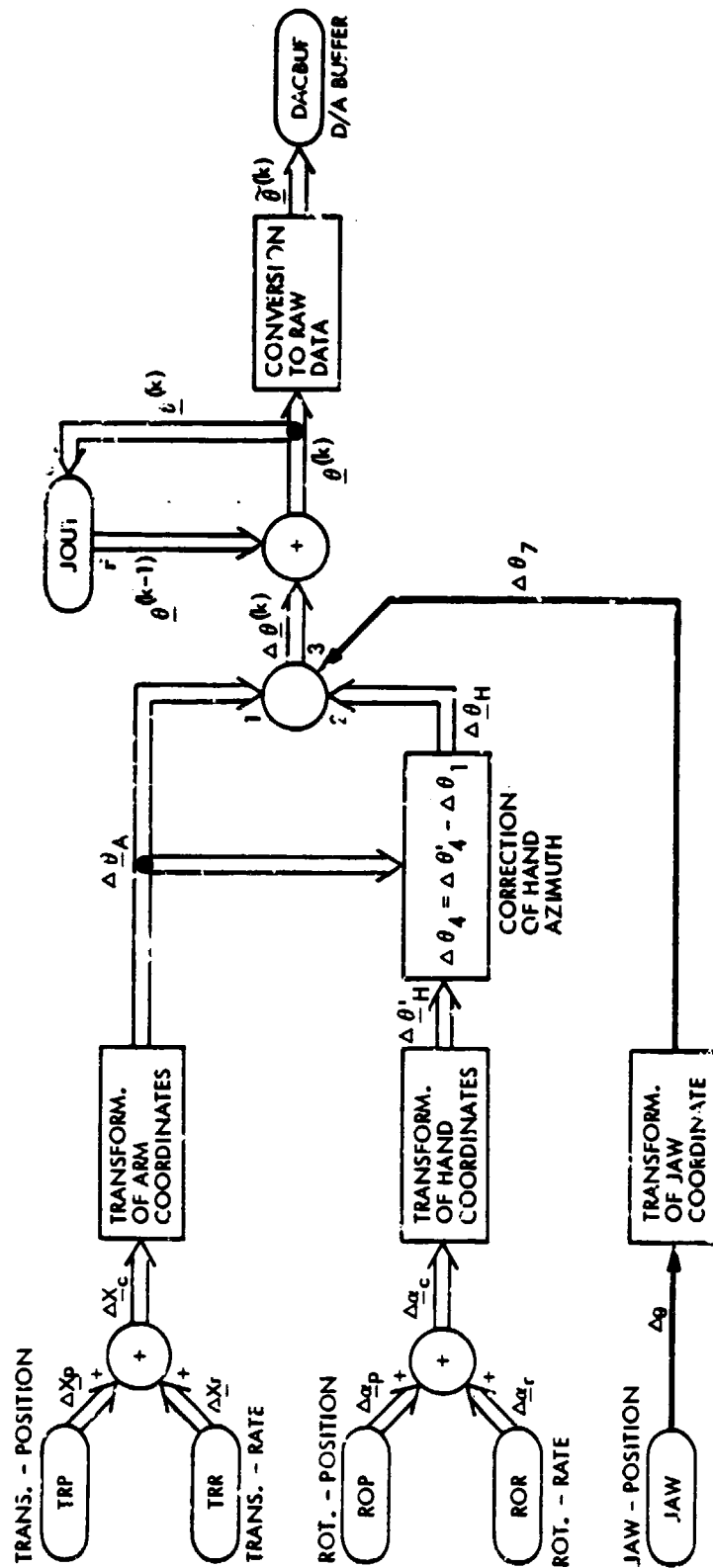


Fig. 3-4. General Data Flow Diagram of CACS (Continued)

A similar calibration procedure is applied to vector \underline{u} . Sensor data are calibrated by special table look-up procedures which will not be considered here.

Calibrated variables are used by control algorithms to produce displacement increments $\Delta \underline{x}_p = (\Delta x_p, \Delta y_p, \Delta z_p)$, $\Delta \underline{\alpha}_p = (\Delta \alpha_p, \Delta \beta_p, \Delta \gamma_p)$ and Δg , and velocities $\underline{v} = (u, v, w)$ and $\underline{\omega} = (\dot{\alpha}, \dot{\beta}, \dot{\gamma})$, i.e., the corresponding rate increments $\Delta \underline{x}_r = (\Delta x_r, \Delta y_r, \Delta z_r)$ and $\Delta \underline{\alpha}_r = (\Delta \alpha_r, \Delta \beta_r, \Delta \gamma_r)$. These values are stored in the buffers TRP through JAW.

Because of the finite dynamic response characteristics of the manipulator, the rate increments must be limited. Therefore, the following value will be stored in the buffer TRR:

$$\Delta \underline{\alpha}_r \quad \text{if} \quad \max \{ \Delta x_r, \Delta y_r, \Delta z_r \} \leq r_{\max} \quad (7)$$

$$\lambda \cdot \Delta \underline{\alpha}_r \quad \text{otherwise}$$

where:

$$\lambda = r_{\max} / \{ \max \{ \Delta x_r, \Delta y_r, \Delta z_r \} \}, \quad (8)$$

and r_{\max} is a given parameter and represents the maximum possible value of any displacement that can be achieved in one clock cycle. This is the condition for the rate buffer input operation, which prevents getting the manipulator into unpredictable working conditions. Similarly, the rotation rate increments are also limited. However, they are limited component by component, because in the case of rotation it is not important to keep the proportionality of the vector components.

In each iteration cycle all buffers are read, the composed values $\Delta \underline{x}_c$ and $\Delta \underline{\alpha}_c$ are formed and brought to the coordinate transformation block.

Coordinate transformations are given by the following equations:

$$\Delta \underline{\theta}_A = T_A(\theta_A) \cdot \Delta \underline{x}_c, \quad (9)$$

$$\Delta \underline{\theta}_H = T_H(\theta_H) \cdot \Delta \underline{\alpha}_c,$$

and

$$\theta_7 = k_j \cdot \Delta g \quad , \quad (10)$$

where: $\underline{\theta}_A = (\theta_1, \theta_2, \theta_3)$ and $\underline{\theta}_H = (\theta_4, \theta_5, \theta_6)$ are arm and hand joint variables, respectively.

Transformation matrices T_A and T_H are inverse Jacobians of joint to world space transformations given in Ref. 3. These matrices have the following elements:

$$T_A = \begin{bmatrix} -\frac{s_1}{P} & \frac{c_1}{P} & 0 \\ \frac{c_1 c_2}{r_3} & \frac{s_1 c_2}{r_3} & \frac{s_2}{r_3} \\ \frac{c_1 s_2}{Dc_3} & \frac{s_1 s_2}{Dc_3} & \frac{-c_2}{Dc_3} \end{bmatrix} \quad , \quad (11)$$

$$T_H = I,$$

where: $P = a_1 + r_3 s_2$, $r_3 = 2Ds_3 + d$, $s_1 = \sin \theta_1$, $c_1 = \cos \theta_1$, $s_2 = \sin \theta_2$, $c_2 = \cos \theta_2$, $s_3 = \sin (\theta_3/2)$, $c_3 = \cos (\theta_3/2)$, D , d , k_j and a_1 are mechanical parameters of the manipulator, and I is the unit matrix. The most recent feedback values can be used as the current values of joint variables θ_1, θ_2 and θ_3 .

The arm and hand coordinates are not completely independent. It can be seen from the equation for the hand azimuth (Ref. 3):

$$\alpha = \theta_1 + \theta_4 - \frac{\pi}{2} \quad . \quad (12)$$

Therefore, the following correction must be done for $\Delta\theta_4$, after the coordinate transformation has been completed:

$$\Delta\theta_4 = \Delta\theta'_4 - \Delta\theta_1 \quad . \quad (13)$$

Finally, the transformed and corrected joint variable increments are joined into one vector $\Delta\theta^{(k)}$, where the index k denotes the current iteration cycle. This vector is added to the joint position vector $\theta^{(k-1)}$ from the last iteration cycle to form the new joint position vector $\theta^{(k)}$. This vector $\theta^{(k)}$ is then converted to "raw" values and is brought to the D/A buffer DACBUF. The conversion to raw values is an inverse to the calibration procedure defined by equation (6).

3. Basic Control Routines

The algorithms which perform the operations of roll/pitch/yaw alignments, tracking, centering, grasping and stopping of TO as well as other interactive operations specified in Section II, are based on a variety of specific motions of the arm, hand and/or jaw. A look-ahead study of the whole CACS has shown that all foreseen operations can successfully be carried out by a unique and finite set of elementary manipulator actions. These actions will be implemented as a set of common routines which can be used by all CACS algorithms. The procedures related to these actions are listed in Table 3-4. Their explanations are given in Figs. 3-5 - 3-9. The following comments will supplement the explanations.

Rotational motions (changing the yaw, pitch and roll angle) are effected by three independent routines, YAW, PITCH and ROLL. This is for the sake of simplicity of algorithm implementation, since different kinds of angle changes are used in the various algorithms.

Motions referenced to the hand coordinate system (Fig. 3-7) require additional coordinate transformations. The rotation matrix which transforms the hand coordinate system into the world coordinate system can be represented in a block form as follows:

$$\left[\begin{array}{c|c|c} \underline{A}_1^T & \underline{A}_2^T & \underline{A}_3^T \\ \hline \underline{A}_1 & \underline{A}_2 & \underline{A}_3 \end{array} \right] \quad (14)$$

Table 3-4. Basic Control Routines

Routine Identifier	Description of the Operation
SHIFT	Translatory displacement of the arm one step in the world coordinate system $\Delta \underline{x} = (\Delta x, \Delta y, \Delta z)$.
SHIFTC	Constrained translatory displacement of the arm one step in the tracking plane $\Delta \underline{x}_t = (\Delta x_t, \Delta y_t)$.
YAW	Rotational displacement of the hand one step in the world coordinate system $\Delta \alpha$ (changing the yaw angle).
PITCH	Rotational displacement of the hand one step in the world coordinate system $\Delta \beta$ (changing the pitch angle).
ROLL	Rotational displacement of the hand one step in the world coordinate system $\Delta \gamma$ (changing the roll angle).
MOVEK	Translational displacement of the arm one step in the hand coordinate system $\Delta \xi$ (longitudinal motion of the hand).
MOVEE	Translational displacement of the arm one step in the hand coordinate system $\Delta \eta$ (lateral hand motion of the hand).
MOVEZ	Translational displacement of the arm one step in the hand coordinate system $\Delta \zeta$ (vertical motion of the hand).
EXPND	Expansion (opening) of the jaw for one increment Δg .
CONTR	Contraction (closing) of the jaw for one increment Δg .
RUN	Motion of the arm with the constant speed $\underline{v} = (u, v, w)$ in world coordinate system.
RUNC	Motion of the arm with constant speed $\underline{v}_t = (u_t, v_t)$ in tracking plane.
ROT	Rotational motion of the hand with constant angular speed $\underline{\omega} = (\dot{\alpha}, \dot{\beta}, \dot{\gamma})$ in the world coordinate system.

Note: All translational motion of the arm assumes unchanged hand angles, and all rotational motion of the hand assumes unchanged arm coordinates.

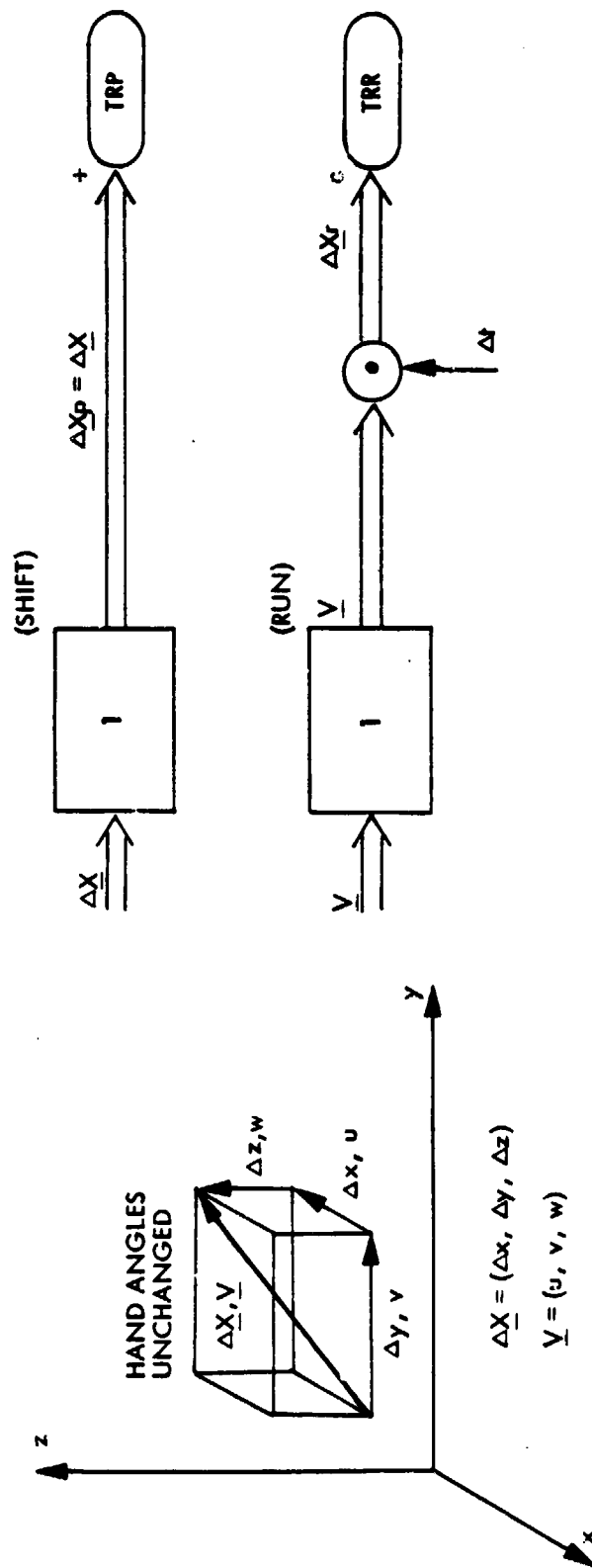


Fig. 3-5. Translatory Motion of the Arm in World Coordinate System

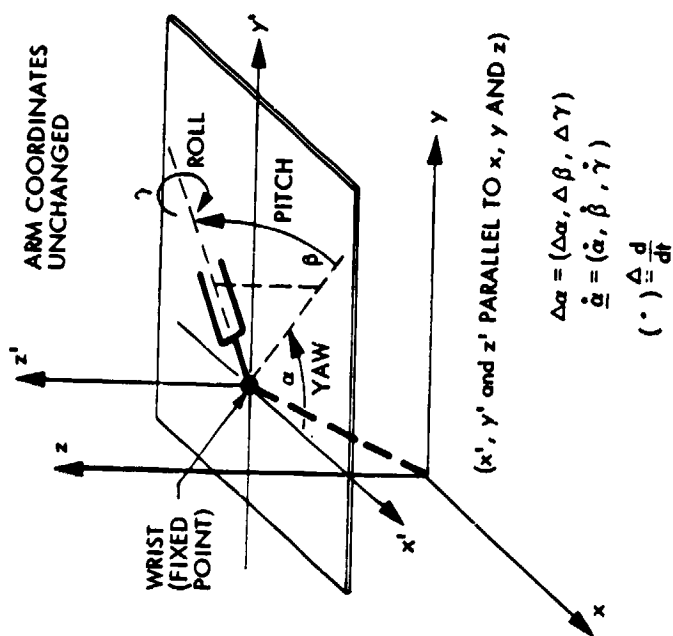
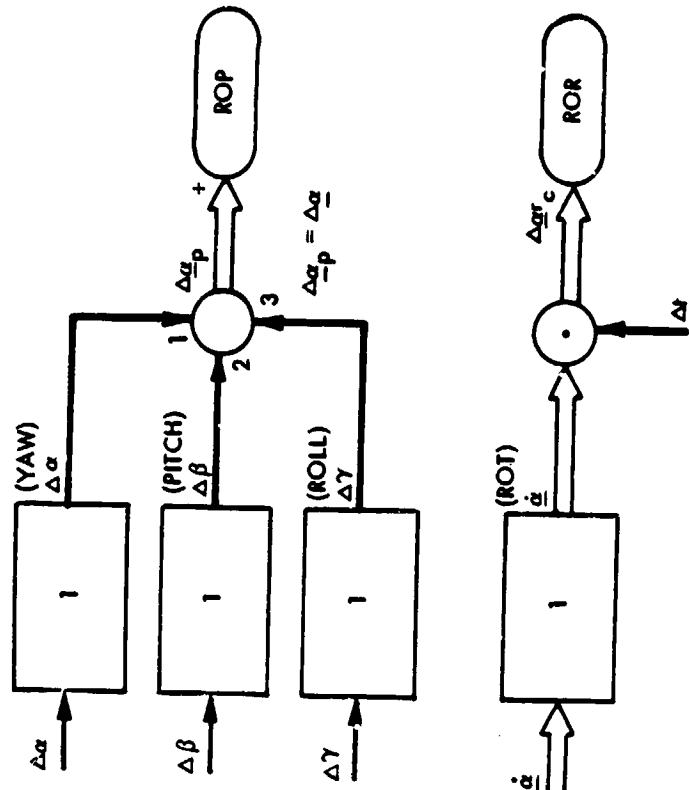
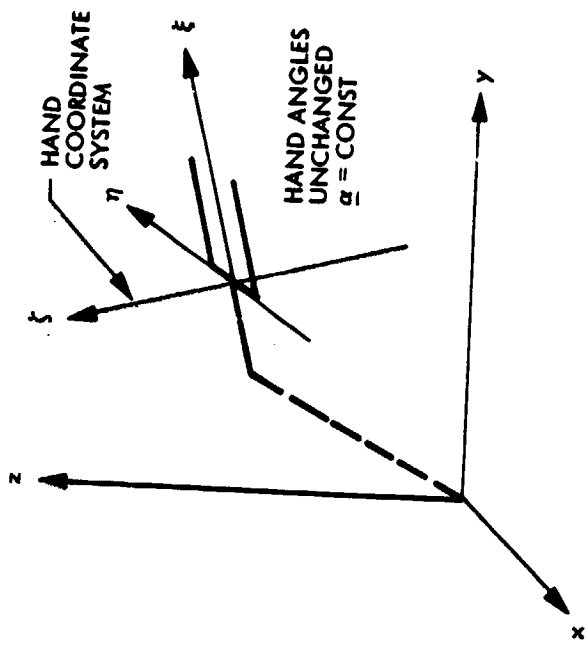
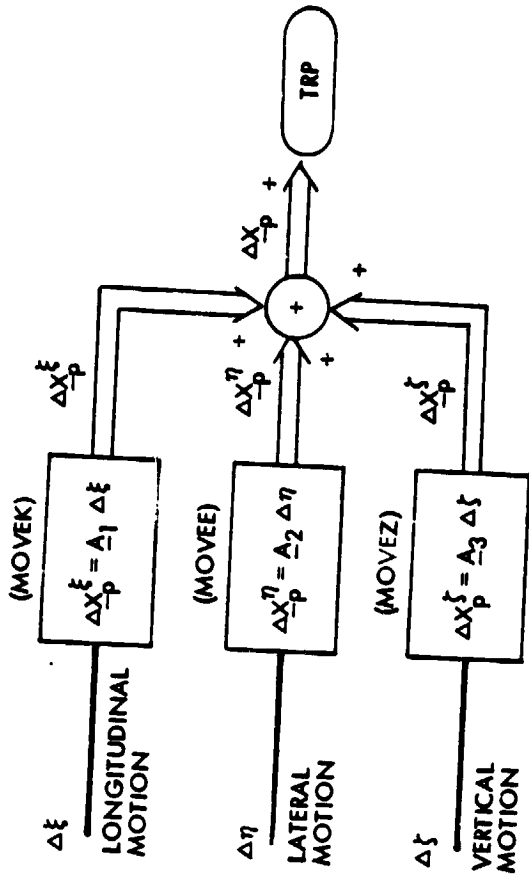


Fig. 3-6. Rotational Motion of the Hand in World Coordinate System



$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} = \underline{A}(\underline{\alpha}) - \text{ROTATION MATRIX}$$

Fig. 3-7. Translatory Motion of the Arm in Hand Coordinate System

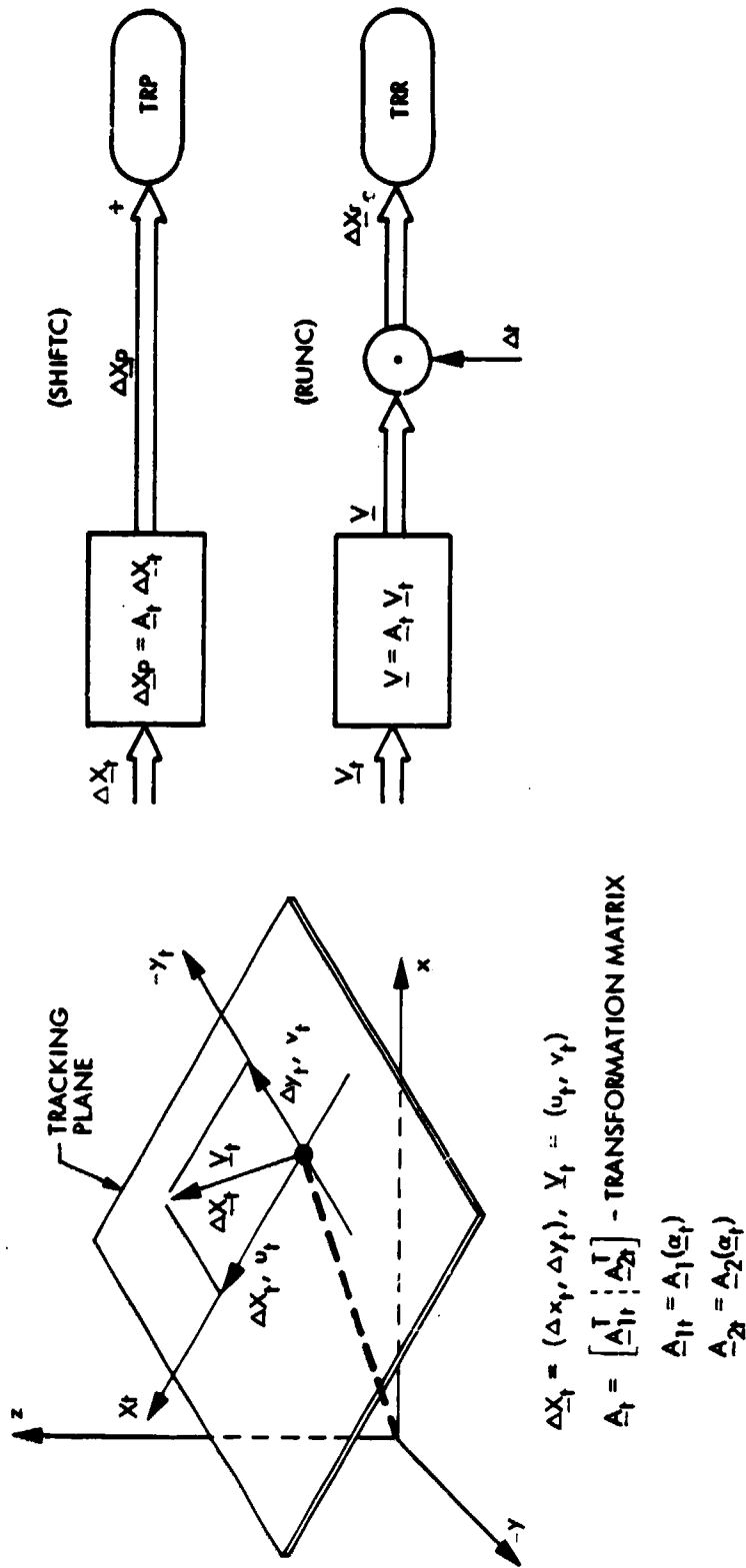


Fig. 3-8. Constrained Translatory Motion of the Arm in Tracking Plane

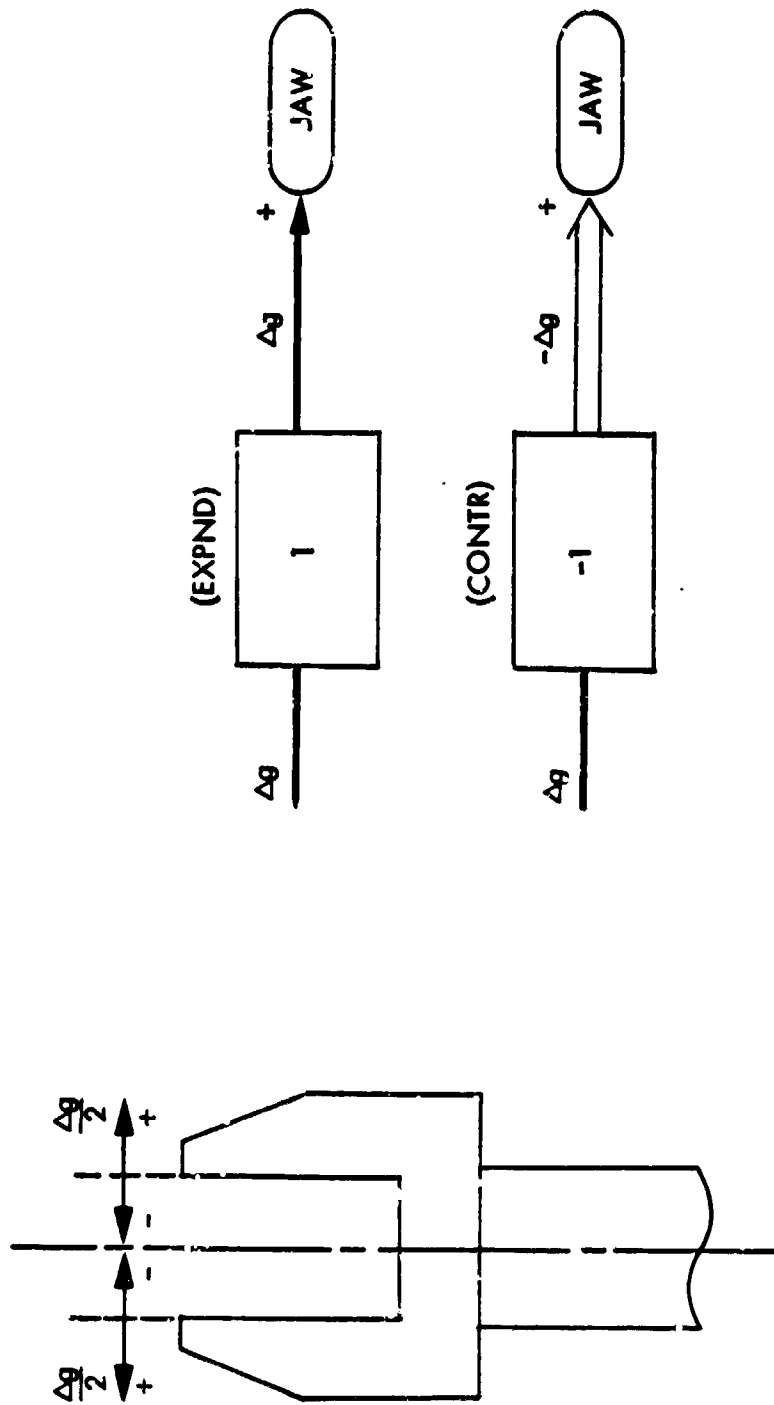


Fig. 3-9. Expansion and Contraction of the Jaw

where the submatrices A_i^T ($i = 1, 2, 3$) are the following column vectors:

$$\begin{aligned}
 \underline{A}_1^T &= \begin{bmatrix} \cos\beta \cos\alpha \\ -\sin\gamma \sin\beta \cos\alpha + \cos\gamma \sin\alpha \\ \cos\gamma \sin\beta \cos\alpha + \sin\gamma \sin\alpha \end{bmatrix} \\
 \underline{A}_2^T &= \begin{bmatrix} -\cos\beta \sin\alpha \\ \sin\gamma \sin\beta \sin\alpha + \cos\gamma \cos\alpha \\ -\cos\gamma \sin\beta \sin\alpha + \sin\gamma \cos\alpha \end{bmatrix} \\
 \underline{A}_3^T &= \begin{bmatrix} -\sin\beta \\ -\sin\gamma \cos\beta \\ \cos\gamma \cos\beta \end{bmatrix}
 \end{aligned} \tag{15}$$

The angles α , β and γ are the current values of the hand angles in the world coordinate system expressed by joint coordinates:

$$\begin{aligned}
 \alpha &= \theta_1 + \theta_4 - \frac{\pi}{2} \\
 \beta &= \theta_5 - \frac{\pi}{2} \\
 \gamma &= \theta_6
 \end{aligned} \tag{16}$$

The most recent feedback values can be used as actual values of joint coordinates.

Constrained translatory motion in the tracking plane also needs additional coordinate transformations. Here, the transformation matrix has the form:

$$\underline{A}_t = \begin{bmatrix} \underline{A}_{1t}^T & | & \underline{A}_{2t}^T \end{bmatrix} \quad (17)$$

with

$$\begin{aligned} \underline{A}_{1t} &= \underline{A}_1(\alpha_t) \\ \underline{A}_{2t} &= \underline{A}_2(\gamma_t) \end{aligned} \quad (18)$$

where $\alpha_t = (\alpha_t, \beta_t, \gamma_t)$ represents tracking plane angles, i.e., the hand angles taken in the TP-coordinate setup procedure (invoked by pressing pushbutton TPC). As will be shown in Section III C, elements of matrix \underline{A}_t are calculated during the coordinate setup procedure and memorized in the special buffer (AT) for subsequent use.

Using the Basic Control Routines, an extended set of new routines can be derived for more complex manipulator operations. For example, the centering algorithm can be simplified considerably if asymmetric extension/contraction jaw operations are introduced. Therefore, four new routines are added to the list in Table 3-4:

- EXPNDR - Expansion of the jaw for one increment Δg to the right.
- CONTRR - Contraction of the jaw for one increment Δg from the right.
- EXPNDL - Expansion of the jaw for one increment Δg to the left.
- CONTPL - Contraction of the jaw one increment Δg from the left.

The first two routines assume opening/closing of the jaw with fixed left side and right side moving Δg out/in. The last two routines are opposite to the operations implied by the first two routines. These routines are shown in Figs. 3-10 - 3-11.

4. Dynamic Response of Manipulator and Delay

Incremental manipulator displacements, commanded by CAC software, are realized by relatively complex electro-hydraulic servo systems.

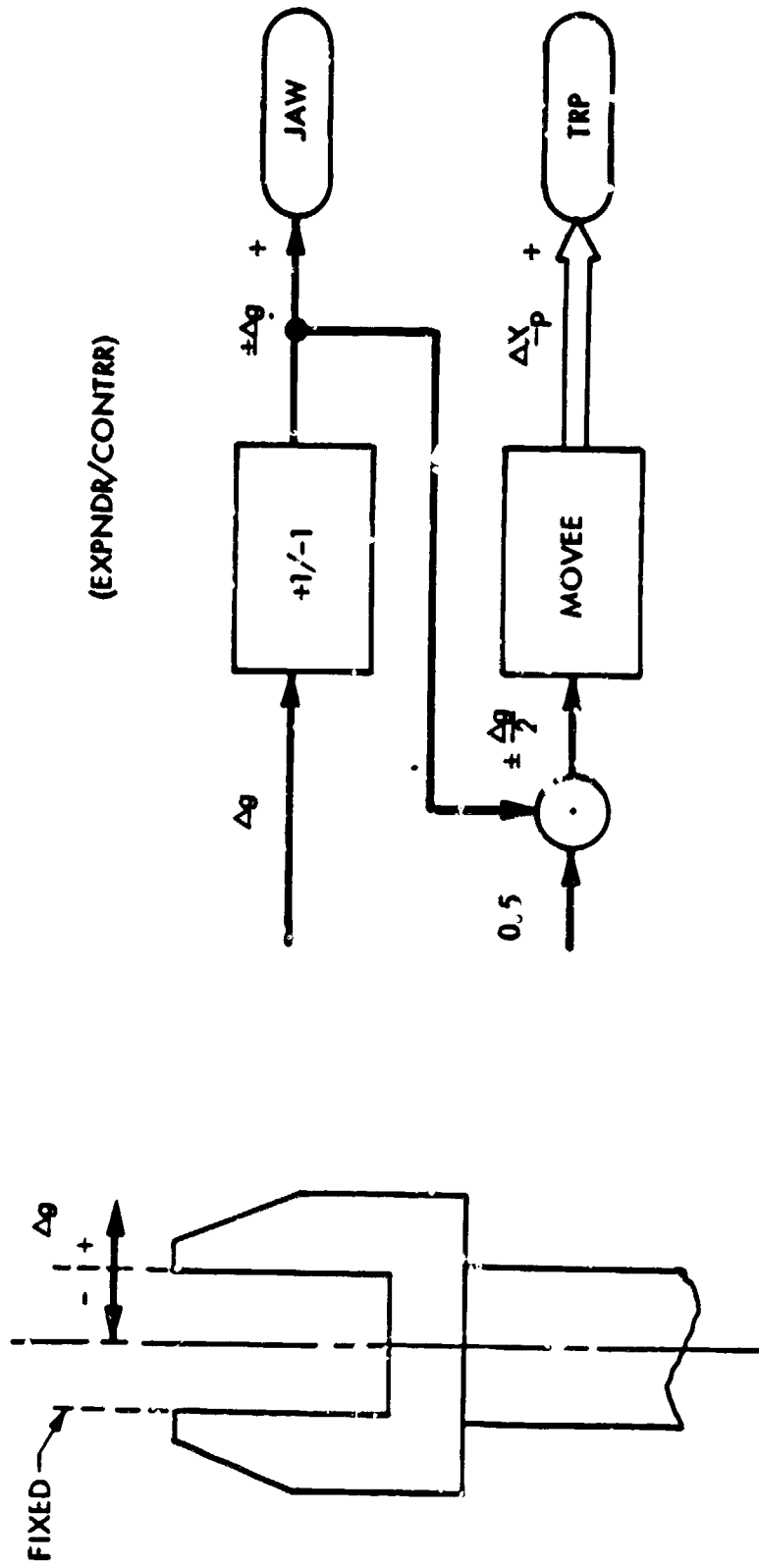
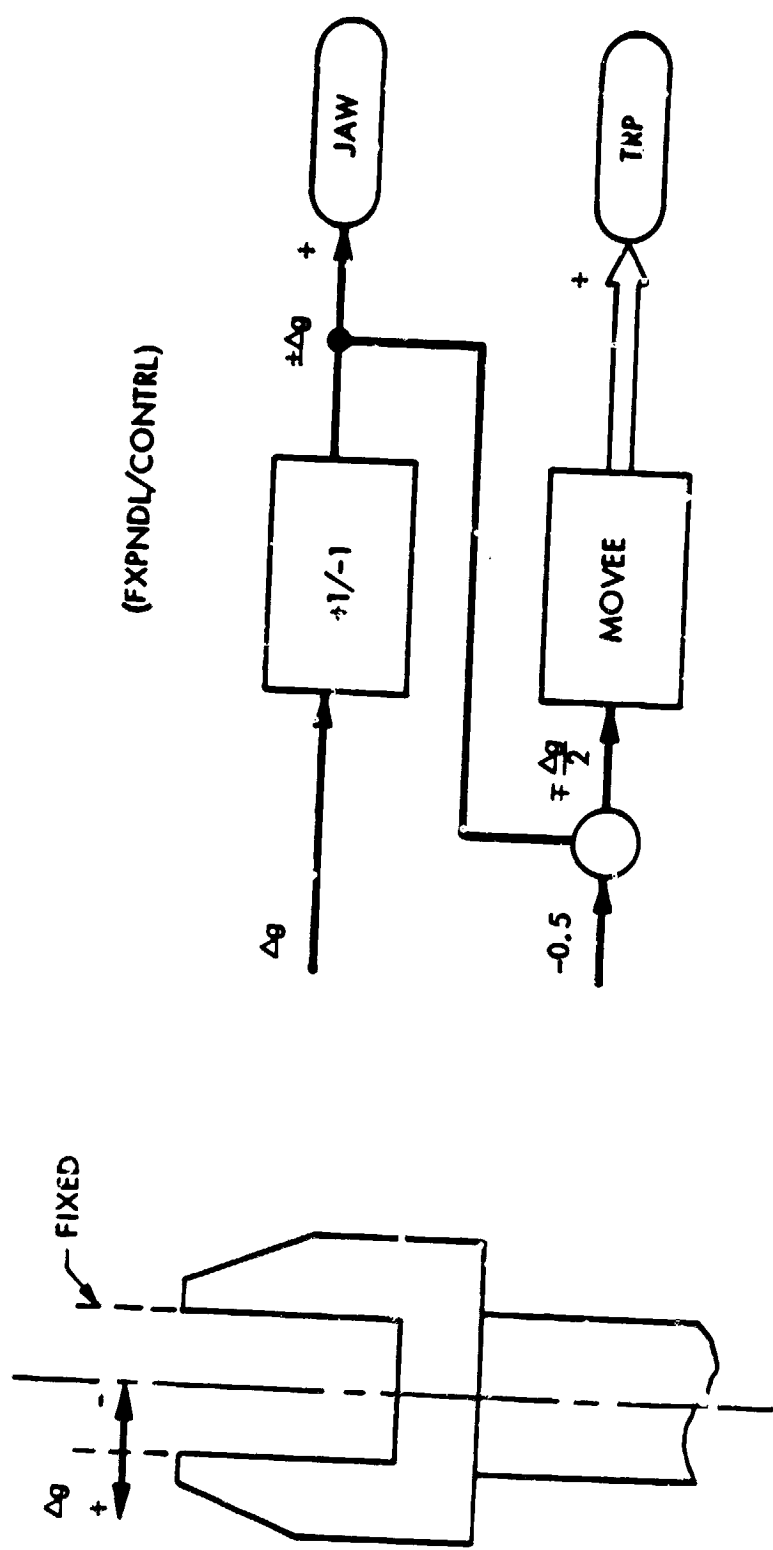


Fig. 3-10. Asymmetric Expansion and Contraction of the Jaw (Fixed Left Side)



(FXPNDL/CONTRL)

Fig. 3-11 Asymmetric Expansion and Contraction of the Jaw (Fixed Right Side)

Because of limited dynamic performances of these servo systems, the adjustments of joints, i.e., the realization of corresponding displacements in the world coordinate system, requires some finite time. Therefore, the displacement magnitudes must be in accordance with the manipulator servo time constants and with the iteration frequency of the computer system. This means that the commanded displacements must be sufficiently small so that they can be realized in one iteration cycle. Let Δt be the time interval of one iteration cycle, and Δx_{\max} and $\Delta \alpha_{\max}$ the maximal values of translatory and rotational displacements of the manipulator which can be realized within the time Δt . Then all components of the vectors $\Delta \underline{x}_c = (\Delta x_c, \Delta y_c, \Delta z_c)$ and $\Delta \underline{\alpha}_c = (\Delta \alpha_c, \Delta \beta_c, \Delta \gamma_c)$ must be less than or equal to the values Δx_{\max} and $\Delta \alpha_{\max}$, respectively. Of course, this is a rough consideration since displacements executed in a given time interval can differ from case to case, depending on the current state of the manipulator and on the attached load. But it is acceptable here as an initial step which can be refined later.

As will be seen, in some cases it will be necessary to realize displacements greater than the limiting values defined above. In such cases, delay mechanism must be introduced. If, for example, the value $\Delta x > \Delta x_{\max}$ has been issued by one algorithm, it must wait for a next decision until the displacement Δx is completely realized by the manipulator. This situation is illustrated in Fig. 3-12. As seen, the displacement will be completed during the third iteration counting from the moment the command has been issued. It means that the algorithm must be delayed two iteration cycles. In general, the number of delayed cycles n_D can be determined by the following approximation:

$$n_D = \left\lceil \frac{\max \{ \Delta x_c, \Delta y_c, \Delta z_c \}}{\Delta x_{\max}} \right\rceil - 1, \quad (19)$$

where $\lceil a \rceil$ denotes upper integer value of "a", and Δx_{\max} is a given parameter and represents the maximum possible value of any displacement that can be achieved in one clock cycle. The same formula is used for the rotational displacement $\Delta \underline{\alpha}_c = (\Delta \alpha_c, \Delta \beta_c, \Delta \gamma_c)$. This formula is derived under the assumption of a linear shape of the dynamic response of the manipulator. Implementation of the delay mechanism will be explained in Section IIIC.

5. Parallel Processing Concept

As shown in Section II, the entire process of searching, tracking, grasping and stopping of the target object can be broken down into a sequence of small units of activities or operations. Some of the unit

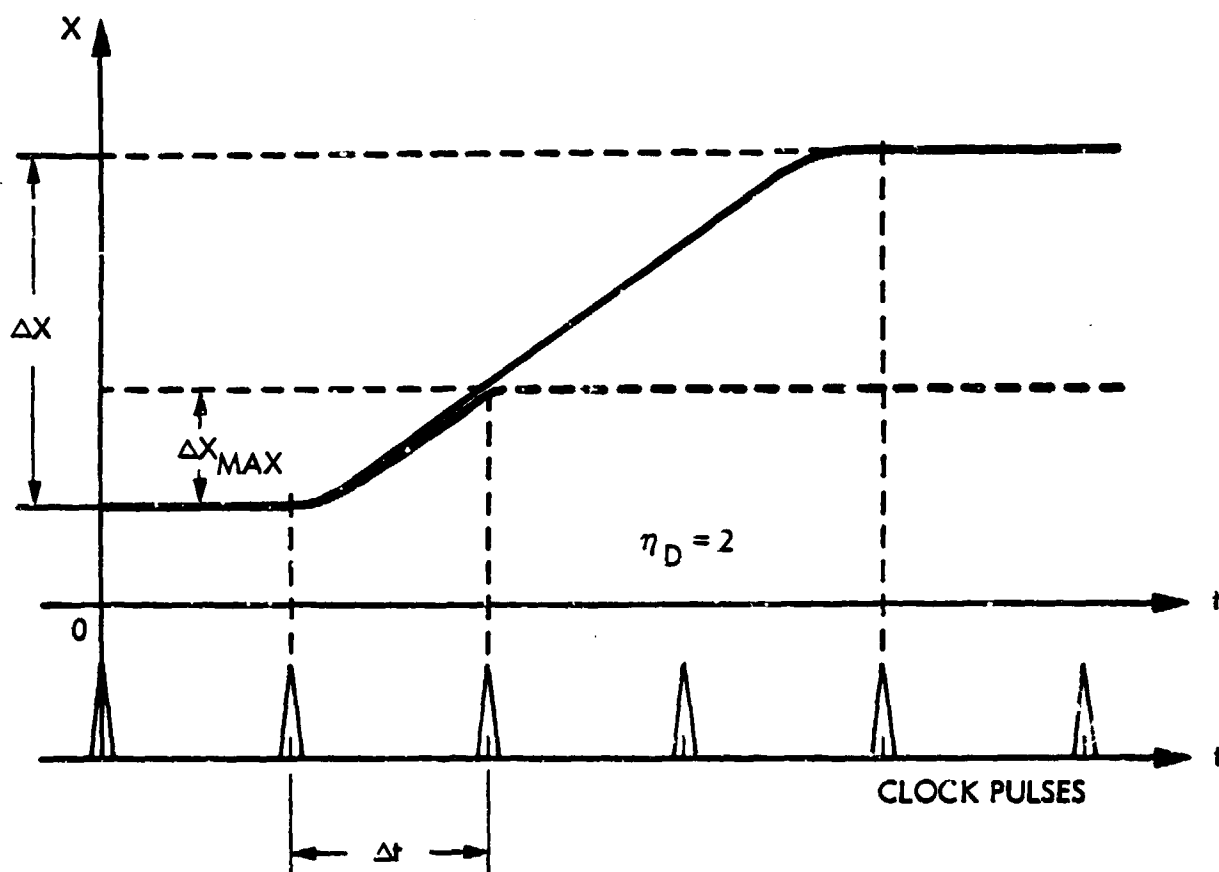


Fig. 3-12. Dynamic Response of the Manipulator

activities are carried out simultaneously, i.e., they are executed in parallel and asynchronously. This is the case with the following activities: operator manual assistance, operator emergency stop, input-output operation, automatic maintenance of the distance between EE and TP, and automatic maintenance of the distance between EE and TO. These activities are performed independently from and simultaneously with other activities which are being executed sequentially.

The structural approach to the algorithm design and the software implementation of the CACS requires that all activities be considered as separate programming modules. Furthermore, the parallelism of the activities requires the concurrent execution of these programming modules. An additional benefit of the concurrent programming approach is that more than one processor can be allocated to the system to increase execution speed and reliability. This is especially important in the case of real-time environments expected for future versions of manipulator control systems. Of course, the concurrent programming system can be realized with only one processor which is multiplexed among concurrent programs. The scheduling of one or more processors for concurrent programs will be achieved through system software not discussed here. The structure of the control algorithms and their software implementation considered here are invariant to the number of processors which can be ignored in further considerations.

The program which is being developed to accomplish the specified activities will be called process. A process can generally have two states: active and blocked. Active state means that the process is in execution. (The active state has two substates: running when the process is being executed by the processor; and ready when the process is waiting to be allocated to the processor. But, this is not important for the following discussions.) In the blocked state the process is waiting to be resumed, or it is waiting for a specified time interval by which it has been delayed. Putting the process in active or blocked state is done by special synchronization signals. By means of these signals a process can be waiting or delayed, or it can resume another activity if some specific conditions occur.

For a concurrent programming system it is important to describe the interprocess relations in terms of synchronization signals. This can be shown graphically by process precedence charts which are introduced here to simplify the discussion. On these charts, Fig. 3-13, processes are depicted by circles which are connected by arrows representing synchronization signals (no data flow!). Besides the arrows, the synchronization conditions are indicated as boolean variables or expressions. In the case of regular arrows, signals will occur if the condition indicated is true. In the case of dotted arrows, synchronization signals will occur if the condition is not satisfied. Dotted arrows will be used to emphasize "backward signalling", that is, when the



P_i RESUMES P_i IF THE CONDITION c IS SATISFIED (c IS TRUE).



SAME AS 1). IN ADDITION P_i IS WAITING.



P_i RESUMES P_i IF THE CONDITION c IS NO MORE SATISFIED (c IS FALSE).



SAME AS 3). IN ADDITION P_i IS WAITING. (P_i LOSES ACTIVE STATUS BECAUSE c IS NO MORE SATISFIED).



P_i IS WAITING IF CONDITION c IS SATISFIED (c IS TRUE).

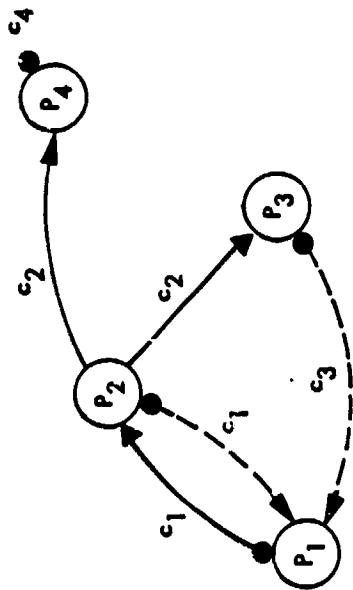


Fig. 3-13. Process Precedence Chart

process loses the active state because the required condition is not satisfied anymore. In order to preserve system reliability, the following general rule will be accepted here: A process can be made to wait or delayed only by itself. Therefore, only process resumption is presented by arrows.

At the right side in Fig. 3-13 an example of a process precedence chart is given. It shows the process P_1 being resumed first. If during its execution condition c_1 occurs, the process P_2 will be resumed and P_1 will be waiting immediately. If this condition is changed to false, process P_2 will be waiting and P_1 resumed again. If P_2 is active and condition c_2 occurs, two processes will be resumed at the same time: P_3 and P_4 . These processes are executed concurrently, together with process P_2 , until the condition c_3 holds. If condition c_3 changes to false, P_3 will be waiting and P_1 will be resumed again. P_4 , once resumed, remains active until c_4 becomes false.

Besides the synchronization signals, data is another reason for interprocess communication. Some processes produce data while others use data. The data used (accessed) by several processes are called shared data. In order to preserve system reliability and to facilitate data communication, special attention must be given to this question. Today's software techniques offer special mechanisms, called monitors, for handling both, shared data and synchronization signals (Refs. 4-7). By means of such monitors it is possible to control the access to shared data and to enforce various access right policies. Monitors and their implementation, as well as process scheduling, will be discussed in Section IIIC paragraph 6.

6. Application of the Parallel Processing Concept in the CACS

The list of all processes of the manipulator operating subsystem OPER is given in Table 3-5. It has been made using Fig. 2-1, Section II. As seen, five new processes AMP, AMO, IOP, OMA and OES have been added. The first three processes are unconditionally active when the OPER subsystem is active. Therefore, two groups of processes can be distinguished: permanent active, and temporary active. A new rule will be added to the one mentioned in the preceding paragraph: permanent active processes cannot be waiting or delayed. This rule will be enforced by the scheduler as will be shown in Section IIIC paragraph 7.

The process synchronization conditions will be handled by two sets of boolean variables: functional switches and event flags. The former is already described in Section IIIA paragraph 2, while the latter is given in Table 3-6. The event flags describe the state of the manipulator. They are updated by an input-output process which is a permanent active process. More details about event flag generation are given in the program documentation in Appendix B.

Table 3-5. CACS Processes

Identifier	Description
IOP*	Input-output process
OMA*	Operator manual assistance
OES*	Operator emergency stop
MUC	Unconstrained manual control
ARA	Automatic roll alignment
APA	Automatic pitch alignment
MCS	Constrained manual control (search for TO)
AMP	Automatic maintenance of distance from TP
ATS	Automatic tracking of TO speed
AMO	Automatic maintenance of distance from TO
AYA	Automatic yaw alignment
ACO	Automatic centering of EE on TO
AGO	Automatic grasping
ASO	Automatic stopping of TO

* permanent active processes.

The general outline of all temporary active processes is given in Fig. 3-14. As seen, the processes are cyclic. If a process is active, the switches and activity conditions are checked in every iteration cycle. These conditions are represented by boolean expressions of functional switches and event flags, depending on the particular process. Synchronization signalling is performed by two primitive procedures: signal and wait. (Signalling procedures put in parentheses are not applied in all processes.) Completion of the activity to which the process is dedicated is also determined by event flags. If the activity is completed, special state variables must be updated. These variables will be discussed later, together with the algorithms implemented by the processes.

The interprocess relation is described by the process precedence chart in Fig. 3-15 which comprises all temporary active processes. This chart is self explanatory by looking up the condition variables listed in Tables 3-1 and 3-6. Hyphenated event variables represent a logical condition or operation. For example, ef-t replaces the logical expression eft or efrt.

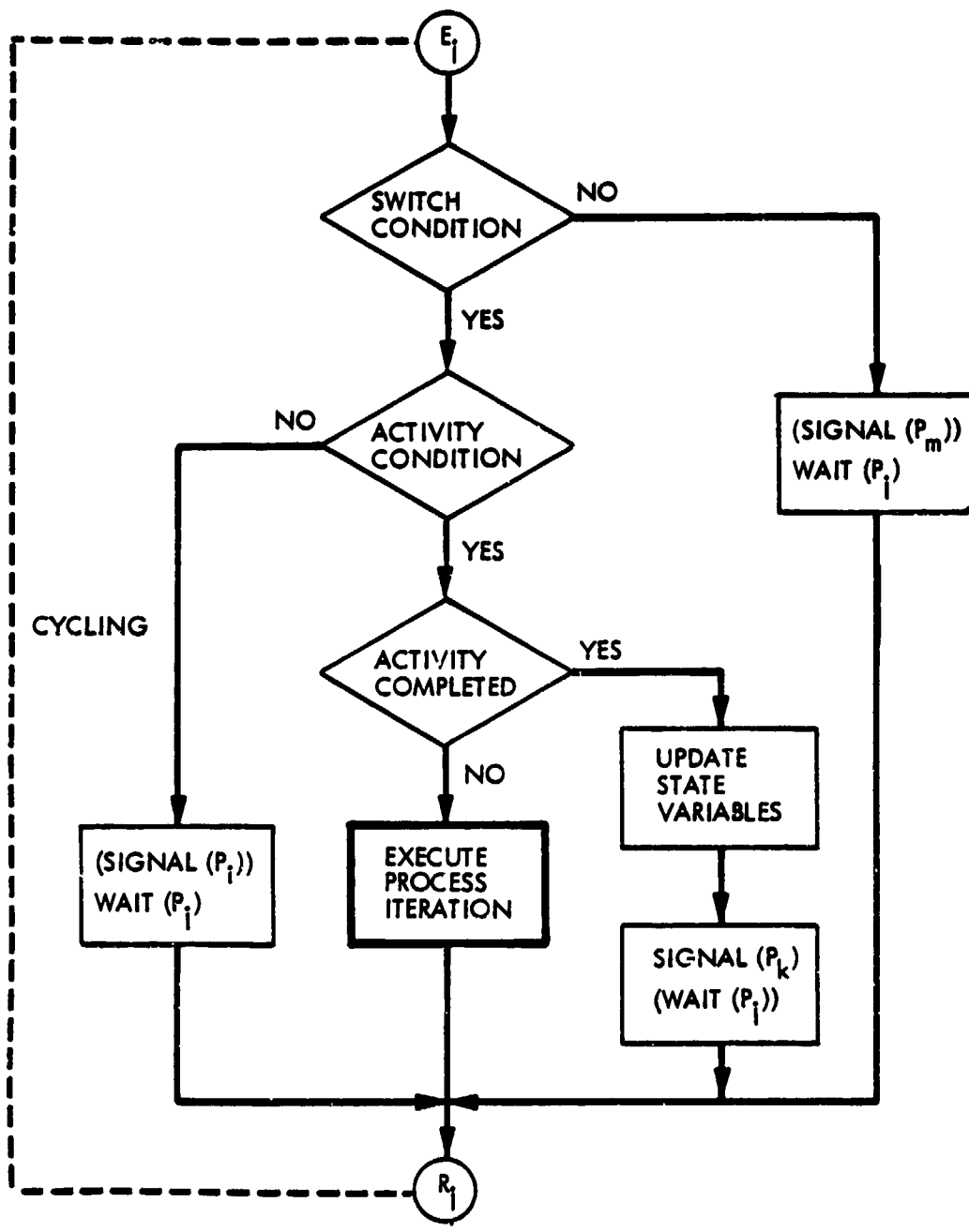
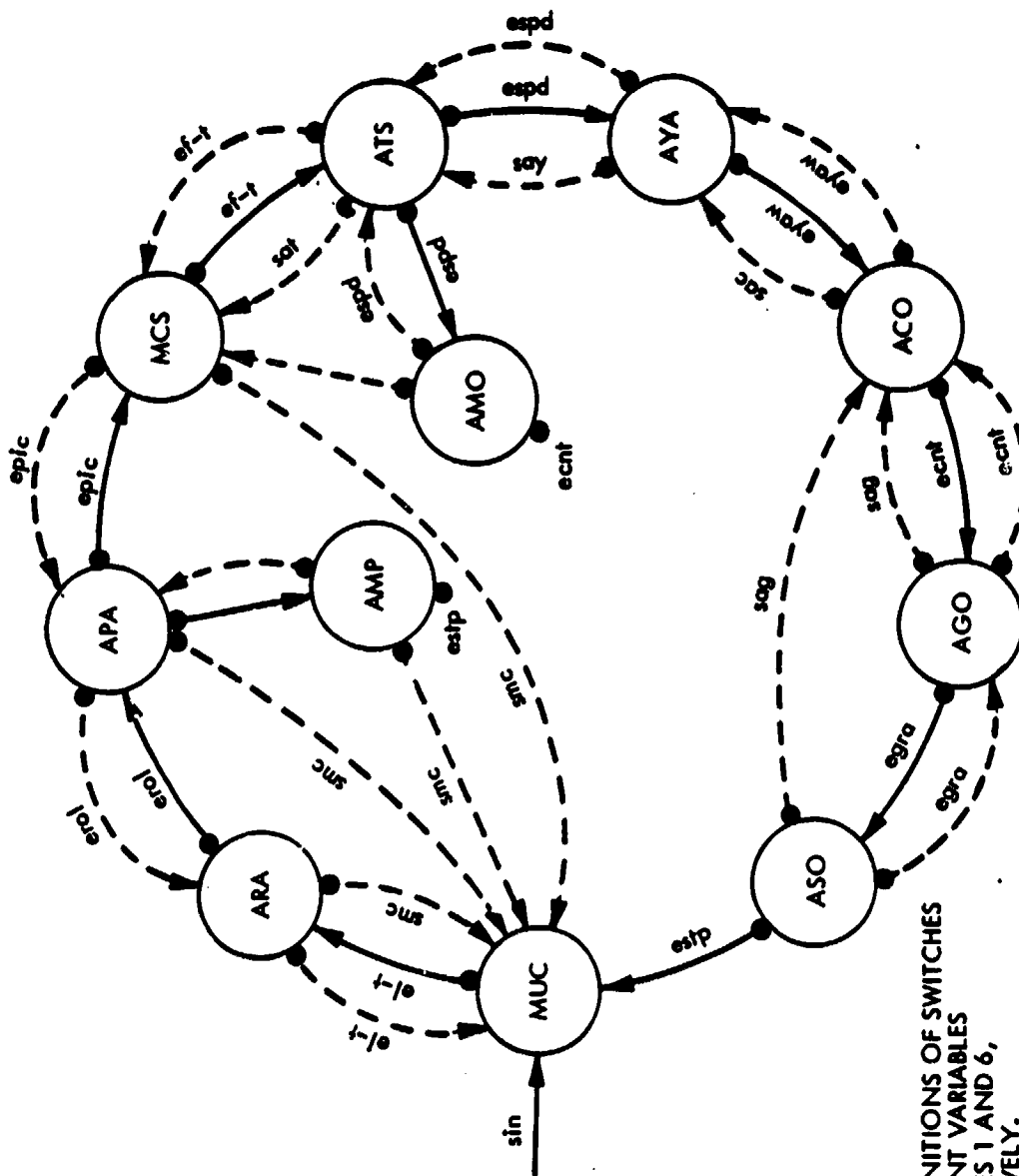


Fig. 3-14. General Outline of Temporary Active Processes



NOTE: FOR DEFINITIONS OF SWITCHES AND EVENT VARIABLES SEE TABLES 1 AND 6, RESPECTIVELY.

Fig. 3-15. Process Precedence Chart of the OPER Subsystem

Table 3-6. Event Flags

Identifier	Description*
EALM	Arm has reached boundary envelope (joint variables θ_1, θ_2 and/or θ_3 in end position).
EHLM	Hand has reached boundary envelope (joint variables θ_4, θ_5 and/or θ_6 in end position).
EFLP/EFRP	Front-left/right proximity sensor indicates proximity distance.
EFLT/EFRT	Front-left/right proximity sensor indicates tracking distance.
EFLC/EFRC	Front-left/right proximity sensor indicates collision distance.
ELLF/ELRP	Lower-left/right proximity sensor indicates proximity distance.
ELLT/ELRT	Lower-left/right proximity sensor indicates tracking distance.
ELLC/ELRC	Lower-left/right proximity sensor indicates collision distance.
EROL	EE is roll aligned to TP.
EPIC	EE is pitch aligned to TP.
ESPD	Speed of TO is attained.
EYAW	EE is yaw aligned to TO.
ECNT	EE is centered on TO.
EGRA	TO is grasped.
ESTP	TO is stopped.
EJCT	Jaw closed to the tracking aperture.
EJCL	Jaw closed.
EJOP	Jaw open.

* description holds for flag true.

C. SOFTWARE IMPLEMENTATION

In previous subsections the CACS has been described from functional, hardware and algorithmic points of view. Now, the software implementation of the system is discussed. The general software architecture, the basic software components and the scheduling of the processes are considered in the first seven subsections. The processes themselves are not designed at the time of writing this report and will not be considered here. The description of the program documentation is given in paragraph 8 of this section.

1. General Software Architecture

As already mentioned in Section III A paragraph 4, the CACS software consists of three parts:

- (a) Manipulator control subsystem (OPER).
- (b) Parameter editing subsystem (EDIT).
- (c) Manipulator testing subsystem (TEST).

These three subsystems will be implemented as three independent tasks under the OS/16 MT2 real-time operating system.

The first subsystem is the main part of the CACS software which supports interactive control of the manipulator from the operator console. The preceding sections as well as the major part of this section are devoted to this subsystem. The later two subsystems are not part of this report and will only briefly be discussed in the next two subsections.

2. Parameter Editor Subsystem

The EDIT subsystem is intended to support fast and easy editing of all CACS parameters and constants. This is extremely important in a laboratory environment and for the purpose of experimentation as emphasized in Section 2. The relatively large number of parameters and the frequent need for their readjustments make this problem nontrivial. The following needs must be taken care of:

- (a) Provide facilities for easy parameter changes before or during manipulator operation.
- (b) Preserve system integrity and reliability.

Both problems can be solved simultaneously by means of a centralized and redundant parameter file which is maintained independently of the manipulator control software. This is shown in Fig. 3-16, where four routines are outlined.

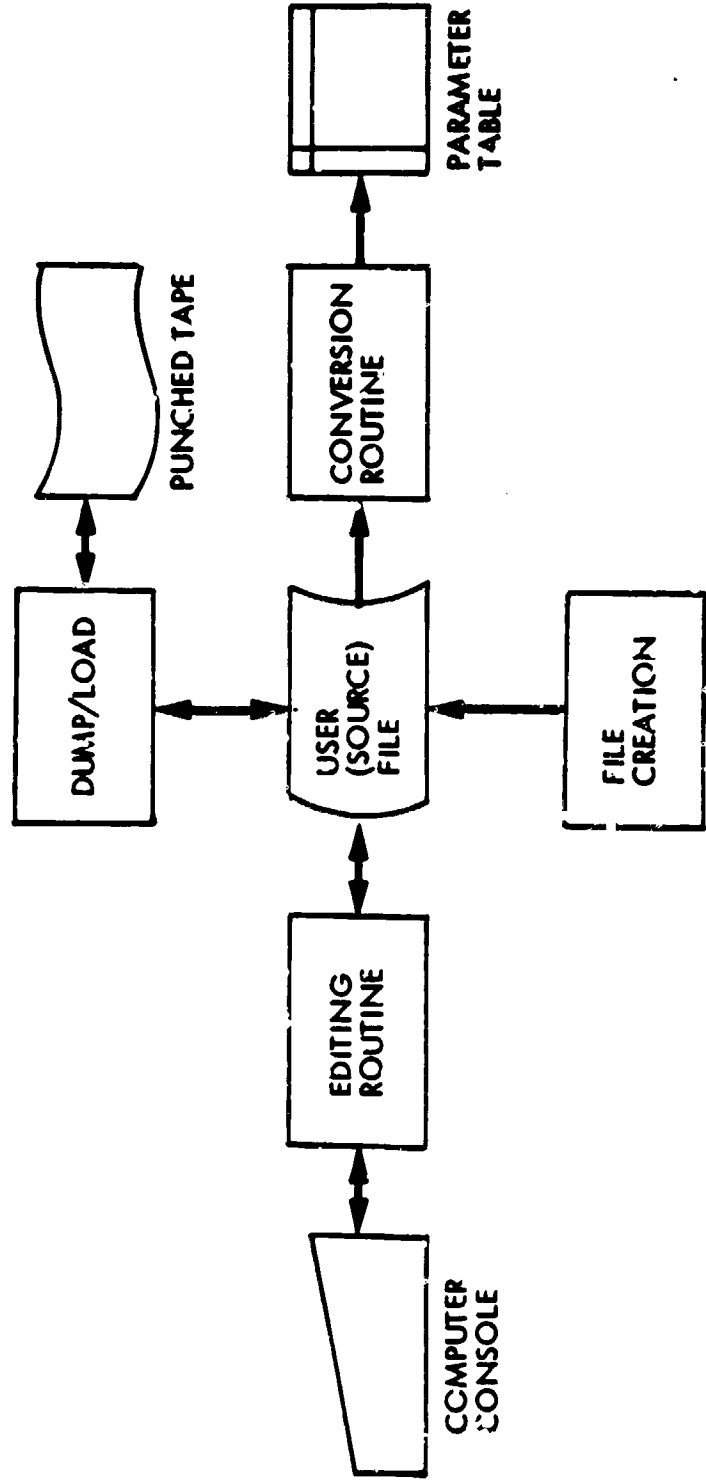


Fig. 3-16. System Block Diagram of Parameter Editor Subsystem

The first routine parameter editing provides an on-line update and display of the user (source) parameter file. This file contains one record for each parameter, where each record contains at least the following data fields:

- (a) Parameter identifier (its symbolic name used in OPER programs).
- (b) Index number for case of parameter arrays (every vector component represents one parameter entity).
- (c) Description of the parameter limited to fixed number of characters.
- (d) Unit of measurement.
- (e) Parameter value.
- (f) Date of the last parameter update.

The updating procedures should comprise addition, deletion and modification of entire records and of particular record fields. To facilitate the experiments, two values can be associated with each parameter: the "actual value" and the "try value". The former represents the value established in previous experiments, while the latter represents the new value ready for use in the current experiment. The try value can always be returned to the actual value if the user is not satisfied with its effect on the system, or it can be fixed as a new actual value if it gives better results. In the last case the date of the last parameter update must also be changed. The user parameter file can be displayed by a CRT or by console printouts as shown by example in Fig. 3-17.

The parameter conversion routine automatically generates the parameter table after the editing procedure is complete. This table contains parameters and their combinations in the "object" form used by the OPER procedures. These values are not necessarily redundant, and cannot be accessed by the user. Examples of parameter conversion are the following expressions:

$$AR = (H + DLOWER(1)) * SIN(GAMINC)$$

$$BR = -DBS/2 * (1 - COS(GAMINC))$$

$$CR = DBS/2 * SIN(GAMINC)$$

$$DR = (H + DLOWER(1)) * (1 - COS(GAMINC))$$

where AR, BR, CR and DR are "objective" parameter values used directly in the roll alignment algorithm, while H and DBS are constructive constants of the CURV arm, and DLOWER(1) and GAMINC are algorithmic parameters, all representing redundant source parameters in user form and maintained by the user.

PAR NO.	IDENTIFIER	INDEX	PARAMETER DESCRIPTION	UNIT	ACTUAL VALUE	DATE	TRY VALUE
001	PROCNO	1	NUMBER OF OPER PROCESSES	1	14	3/15/79	14
002	INTRVL	1	TIME INTERVAL OF CLOCK CYCLE	MILLISEC	20	3/15/79	20
003	ZOTETA	1	JOINT CALIBRATION ZERO OFFSET	32K. RAD/PI	0	3/03/79	0
004	ZOTETA	2	JOINT CALIBRATION ZERO OFFSET	32K. RAD/PI	-16384	3/03/79	-16384
005	ZOTETA	3	JOINT CALIBRATION ZERO OFFSET	32K. RAD/PI	-14274	3/03/79	-14274
009	SFTETA	1	JOINT CALIBRATION SCALA FACT.	32K. RAD/PI	-8554	3/03/79	-8554
010	SFTETA	2	JOINT CALIBRATION SCALA FACT.	/VOLT	6371	3/03/79	6371
011	SFTETA	3	JOINT CALIBRATION SCALA FACT.	/VOLT	7981	3/03/79	7981
015	MXOPEN	1	MAX. JAW OPENING INCREMENT	1	150	3/05/79	150
016	MXSTEP	1	MAX. TRANSLAT-POSITION INCR.	1	30	3/05/79	30
020	DFRONT	1	PROXIMITY DIST. OF FRNT SEN.	MM	125	3/06/79	135
021	DFRONT	2	TRACKING DIST. OF FS (HIGH)	MM	60	3/06/79	60
022	DFRONT	3	TRACKING DIST. OF FS (LOW)	MM	50	3/06/79	50
023	DFRONT	4	COLLISION DIST. OF FRNT SEN.	MM	5	3/06/79	5
024	DLOWER	1	PROXIMITY DIST. OF LOWER SN.	MM	100	3/06/79	100
025	DLOWER	2	TRACKING DIST. OF LS (HIGH)	MM	55	3/06/79	55

Fig. 3-17. Example of Parameter Display

The file creation routine is used only once, immediately after establishing the EDIT task. This routine is trivial if the sequential organization of the user parameter file is accepted.

The save routine is a system utility routine which provides copying of the user file onto the output media as punched cards, paper tape, or magnetic tape. This enables the keeping and maintaining of more than one set of parameters. For example, each set of parameters can be associated with different sensor calibrations or with different experimental runs. Before starting the manipulator operations, a particular set of parameters must be chosen and the corresponding save file must be loaded into the user file. The user file can be a direct access auxiliary storage (disk for example) or it can be directly in the core. The storage required for a user file is about 12 K bytes, and the storage required for the EDIT code is not expected to be large, so that both can be core resident during the execution of the EDIT task. The parameter table, which is an output of the parameter conversion routine, contains only one set of parameter values requiring about 300 bytes of computer storage. This table must be transferred from EDIT to OPER task, where it will be placed in a shared data buffer. This is discussed in paragraph 6.

3. Manipulator Testing Subsystem

The TEST subsystem is intended as a maintenance tool for checking the OPER subsystem after every hardware change and/or after system reinstallation caused by changes in the software. It is obvious that the dynamic environment of the CACS will require frequent modifications of the software to improve the existing programs or to extend the system by new features and capabilities. This means that the CACS software must be considered as a "living" part of the system, subject to changes and continuous growth. To maintain system integrity and efficiency of maintenance, it is strongly recommended that considerable attention be paid to the manipulator testing subsystem. It must be developed as a sequence of testing procedures which will automatically and systematically check all modules of the subsystem, their mutual interactions and their interactions with the I/O devices. The sequential order must be designed in a way so that all possible errors in the hardware and software can be located fast and, eventually, without operator assistance. The major part of the TEST subsystem can probably be synthesized from testing programs which are an outcome of the overall program development process. This must be kept in mind when creating the testing programs which must be flexible and suitable for future retailoring.

4. Monitor Concept

The basic software components of the OPER subsystem are processes and monitors. Processes are already discussed in the preceding Section (paragraphs 5 and 6), and their implementation will be considered in paragraph 6 of this Section. Monitors are introduced by Brinch Hansen,

Refs. 6-7, and by Hoare, Ref. 8, as a new concept for a hierarchical structuring of multiprogramming operating systems. This concept is adopted to develop the CACS software. It will be discussed briefly in this subsection to complete the discussion of Section III B paragraph 5. A more detailed description of monitors and their implementation is given in Refs. 6, 7, 9, 11 and elsewhere.

Processes communicate with each other by sending or receiving data. As already mentioned before, these data are called shared data. To preserve system integrity, the processes should not operate directly on these data. An example of direct operations on data is common data structures which enable unrestricted data accessibility by minimal system overhead. However, the reliability of complex systems, especially systems with concurrent programs, is highly sensitive to incorrect data communication. Therefore, the accessibility of data, i.e., the access rights of the processes, should be carefully controlled. This can be done by allowing access to data only by special procedures. Procedures provide much safer interface mechanisms than common data structures. This is the basic assumption underlying the modern approach to structural design of concurrent programming, Ref. 11. It will also be employed in the design of the CACS software.

The shared data and the procedures which can access them, are grouped within an abstract object called monitor. These objects are usually high level language constructs designed for concurrent programming, Ref. 9-10. They also can be used as a powerful concept for building hierarchically structured systems in sequential languages. This is the case for the CACS software.

Regardless of the interpretation of monitors, they define the following four entities:

- (a) shared data structure,
- (b) monitor procedures,
- (c) initial operations,
- (d) access rights

The shared data structure includes data transmitted among processes, as well as other data necessary for the correct functioning of the monitor in the context of a multiprogramming environment. Monitor procedures explicitly define all operations which the processes can perform on shared data. Initial operations define all operations which must be executed in time of creation of the monitor. Finally, access rights define all connections of the monitor to the rest of the system hierarchy. This is achieved by giving an explicit list of all processes or other monitors which can enter the monitor.

Besides the control of operations on shared data and their accessibility, another feature concerning the monitors is essential: mutual exclusion. It must be ensured that only one process can operate on shared data at the same time. This can be accomplished by allowing that only one procedure within the same monitor can be called by processes or other monitors at the same time. If some process is executing a monitor procedure and another process is trying to call this or another procedure within the same monitor, the latter must be delayed until the first process leaves the monitor. Simultaneous monitor calls are scheduled outside the monitor by special procedures grouped within one program called kernel. The kernel is a basic part of the system software which implements exclusive access rights of processes and scheduling of CPU's and other physical resources among concurrent processes. For its work, the kernel uses interrupt mechanisms and low level communication facilities which are implemented in hardware or lowest level machine software. Therefore, the kernel is highly machine dependent, i.e., it can be considered as an extension of the machine which hides its details from the user who builds the concurrent programming system. This is an essential point of the monitor concept.

As mentioned before, monitors are not only intended for transmission of shared data. They are also used for process synchronization and for scheduling of physical resources. Although, monitors have a structure essentially identical for all purposes, two general types of monitors can be distinguished: buffer monitors and resource monitors.

The buffer monitors are designated for shared data transmission among concurrent processes. Their data structure includes three parts: shared data buffer, full-empty indicator, and two single process queues. Buffers can be designed to handle one (single buffers) or more (multiple buffers) data portions. Multiple buffers are usually designed as linked circular lists, which impose three more data items in the monitor data structure: current buffer length, and two pointers, one for the head and one for the tail of the list. (In CACS the simpler single buffers are used.) The sizes of the buffers correspond to the amount of data to be handled. The data can be arrays, records or sets. (In CACS only data arrays will be used.)

A full-empty indicator is a boolean variable which tells the processes whether or not the buffer contains data. If the buffer is full no data can be transmitted to the buffer, but the data can be taken from the buffer. If the buffer is empty, data can not be taken from the buffer but can be transmitted to the buffer. Single process queues are usually integers which represent process waiting to send data to the buffer and process waiting to receive data from the buffer. These integers are basic elements of process synchronization by the kernel which performs the blocking (delaying) or the activating of the process.

A typical single buffer monitor is shown in Fig. 3-18. (In this section the Pascal language, Ref. 14, is used as a more precise and more comfortable means for program description than program flowcharts.) As seen, two monitor procedures are associated with the monitor: SEND for transmitting data to the buffer and RECEIVE for taking the data from the buffer. Procedures BLOCK and CONTINUE are kernel procedures which block and activate the processes in a pairwise manner.

Resource monitors do not have buffers. They use multiprocess queues to schedule the resource among processes. This kind of monitor will not be used in CACS in its usual form, and it will not be considered here.

5. Basic Prerequisites and Assumptions

The implementation of monitors and processes depends on computing capabilities assigned to the project. Other limiting factors are the attitude of the research personnel about the utilization of capabilities and directives derived from the experience during the foregoing work. In the case of the present project, the essential prerequisites and assumptions can be summarized as follows:

- (a) Only one processor is assigned to the CACS project.
- (b) The whole OPER subsystem should be implemented as a single partition user program that is imbedded in the OS/16 MT2 real-time operating system.
- (c) The clocked I/O version for CACS software-to-manipulator interface is suggested, Ref. 12.

The first prerequisite is due to current limited hardware resources. However, the eventual possibility of utilizing more than one processor should not be rejected. Current trends in microprocessor technology development make this perspective realistic, and specific suggestions have already been made, Ref. 12. Hence, software design, compatible with a multiprocessor environment, is advocated.

The second prerequisite is made for two reasons. First, the characteristics of the MT2 real-time operating system are not ideally adjusted for this kind of real-time environment. It is designed to fit more on-line interactive information systems than systems such as manipulator control. Second, the top-level operation of MT2 requires a fair amount of knowledge about its operation. Only with a single partition user program can the operator be unburdened from many details required by the operation of the MT2, Ref. 13. It should be mentioned that no suitable high level languages, especially for concurrent programming, are available at the present time for this machine.

```

type vector : array [1..bufdim] of integer;
var buffid: vector;           {shared data.}
var full  : boolean;         {Buffer full-empty indicator.}
var sender, receiver: integer; { process queues.}

procedure SEND (argvec: vector);
begin
  if full then BLOCK(sender) else
    begin
      { put argvec into buffer }
      full:=true;
      CONTINUE(receiver)
    end
  end;

procedure RECEIVE(var argvec : vector);
begin
  if not full then BLOCK(receiver) else
    begin
      { get content from the buffer into argvec }
      full:=false;
      CONTINUE(sender)
    end
  end;

```

Fig. 3-18. General Outline of Buffer Monitors
(in Pascal language)

Finally, the clocked I/O is the most suitable vehicle for communicating between the digital computer as the controller and the continuous dynamic system as the controlled object.

6. Implementation of Monitors and Processes

Taking into account the assumptions made in the preceding subsection, it is obvious that the problems of mutual exclusion, processor switching, and scheduling of other physical resources do not exist any more. Therefore, the implementation of monitors and processes becomes trivial. The processes can be developed as simple subroutines and their switching to the processor as simple subroutine calls. Also, the producer-consumer relations between processes are not asynchronous due to periodic I/O operations. It means that in every iteration cycle, all input buffers are first filled by an input process (DOIO), and then they are read by the internal processes. The same is true with the output buffers and the other internal buffers for shared data handling. In addition, most of the internal buffers are not of the producer-consumer type. As shown in Subsection 4.2, the condition for putting a new value into the buffer is not its emptiness, and the condition for getting the content of the buffer out is not its fullness. This considerably simplifies the monitor procedures and their data structures, because the buffer full-empty indicator as well as the receiver and sender queues are not needed anymore. Therefore, the data structures in the buffer monitors are reduced to the buffer content itself. Consequently, the kernel is dramatically simplified and reduces to a single monitor of the synchronization type and to one short program.

Now, the justification of the monitor concept in this project might be appropriate based on the following reasons:

- (a) The complexity of CACS software and its dynamic environment demand a highly structured organization. Concurrent processes and the monitor concepts are powerful methods for such software structuring, even if the processes and monitor are simulated. The natural parallelism of manipulator control activities reinforces this reason considerably.
- (b) The simplified versions of the monitors used out of a language context can dramatically reduce the hazard of destroying the system integrity. The system overhead (in time and space), which can be considered a procedural drawback for data communication, is still negligible in comparison to other computations in the system.
- (c) Because the monitor calls and process switchings are reduced to simple subroutine calls, the kernel run-time overhead is practically reduced to zero, and therefore, cannot be used as an argument against the monitor concept.

- (d) If the system is built based on the monitor concept it can be easily transferred to a multiprocessor environment. As mentioned earlier, multiprocessor control of a manipulator has a real future.
- (e) Even in the single processor case more complex scheduling strategies must be considered if the execution time of CACS programs exceeds the maximum allowable value of the cycle time. Preemptive scheduling strategies and processor switching will not disturb the general system structure if it is based on the monitor concept. Only the monitor procedures must be extended by queuing features, and a new version of the multiprocessor case.

Let us now consider the implementation of the monitors in the OPER subsystem. The list of all OPER monitors is given in Table 3-7.

All monitors listed, except the last three, are of the buffer type. However, the majority of the input/output buffer operations, as required by the control algorithms, are not of the simple send/receive type. They are dependent on some specific conditions, or they have to provide some nontrivial data conversion. For example, calibration and decalibration (MPXS, MJIN, MJOU), computation of trigonometric functions (MSCT), computation of matrix coefficients, and coordinate transformations (MSCH, MAT). These operations could be done outside of the monitors, but for structural reasons they are kept inside as an integral part of the data transmission. A detailed specification and description of the monitor procedures is given in the program documentation (Appendix B).

Monitors MREL and MHL D are used as locking flags for controlling some arm manipulations, such as putting it in the hold state or releasing it from the hold state by operator intervention. These monitors and related locking mechanisms will be explained together with the processes that use them. Monitor MPRQ is intended for process scheduling and synchronization. It is discussed in the next subsection.

The implementation of the monitors and processes will be done in an identical manner in accordance with the features of the INTERDATA Common Assembler Language (CAL). Thus, every monitor or process will be coded as separate assembly block called module. The structure of a module is given in Fig. 3-19. As seen, it contains three parts: list of entry-point and external symbols, program code (procedures), and data declarations. Programs within one module can be hierarchically structured, and they cannot be accessed by procedures defined in other modules unless they are listed in the entry-point list. The same is true with data. If their identifiers are not included in the entry-point list, they remain private variables of the monitor and cannot

Table 3-7. CACS Monitors

Identifier	Description
MIOB	Input-output buffers
MJIN	Calibrated joint variables (feedback values)
MJOU	Calibrated joint variables (set-point values)
MSCT	Trigonometric functions of joint variables (feedback value)
MEST	Event status table
MAST	Alarm status table
MSST	Switch status table
MTRP	Translation-positional increment buffer
MTRR	Translation-rate buffer
MROP	Rotation-positional increment buffer
MROR	Rotation-rate buffer
MJAW	Jaw manipulation increment buffer
MSCH	Trigonometric functions of hand angles (world coordinate system)
MAT	Tracking plane rotation matrix
MPXS	Calibrated proximity sensors data
MFTS	Calibrated force-torque sensors data
MPAR	CACS parameters and constants
MHLD	Hold lock
MREL	Release lock
MPRQ	Process queue and delay semaphore

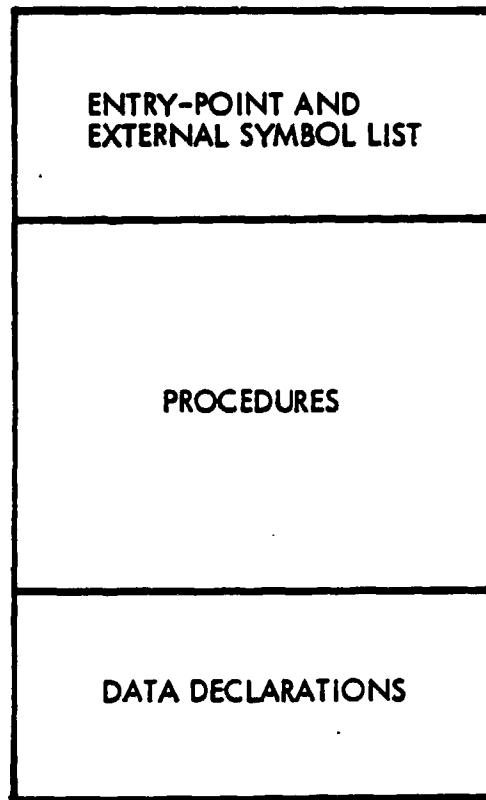


Fig. 3-19. General Structure of the Module

be accessed from other modules. The monitor procedures called by a process must be in the external list of the process. The same holds for nested monitor calls. If the monitor procedure calls a procedure from another monitor, the latter must be in the external list of the former module. The entry-point and external symbol features (ENTRY and EXTRN pseudo-instructions) are the only means for access right control in the CACS.

It is accepted here that all shared variables are implemented as private variables of the modules, i.e., they must not appear in any entry/external list. Furthermore, all private variables of processes which must be initialized are implemented as global variables declared as entry-points.

In order to describe the hierarchical structure of the system, the graphical technique called access graph has been used, Ref. 9. By access graph the monitors and processes are depicted as circles and the access right by arrows (the latter should not be interpreted as data flow!). The access graph of the OPER subsystem is given in Fig. 3-20. In this figure all internal processes are represented by one circle to simplify the diagram.

A more precise definition of the access rights can be done by a cross-reference table of the form shown in Fig. 3-21. The columns of this table represent processes and monitors, while the rows represent the monitor procedures which are called by processes and/or monitors indicated at the head of the columns. The access right is checked by "X". For example, "X" in the first column and the second row of the table means that process P₁ has access to procedure S₁₂ of the monitor M₁. As a consequence, the module representing the process P₁ must have a pseudo-instruction EXTRN S₁₂, while the module representing monitor M₁ must have a pseudo-instruction ENTRY S₁₂.

7. Scheduler

Scheduling of I/O operations and internal processes is provided by the program called scheduler (SCHED) and by one monitor (MPRQ) called process queue. These two system components will be discussed in the present subsection. The scheduler is the main program of the subsystem OPER, which is a clock interrupt driven cyclic program. The Pascal program of the scheduler is given in Figs. 3-22 - 3-23.

I/O operations and cycle control is done by subroutine DOIO, which reads A/D converter and switches/pushbuttons and writes D/A converter and alarm display. These operations are performed through special clock interrupt service routines. After executing service routines, the control is given to the first instruction following DOIO call. The rest of the main program is executed within a clock interval. After execution, the processor is trapped in a "busy wait" loop (1 goto 1), until the next clock interrupt.

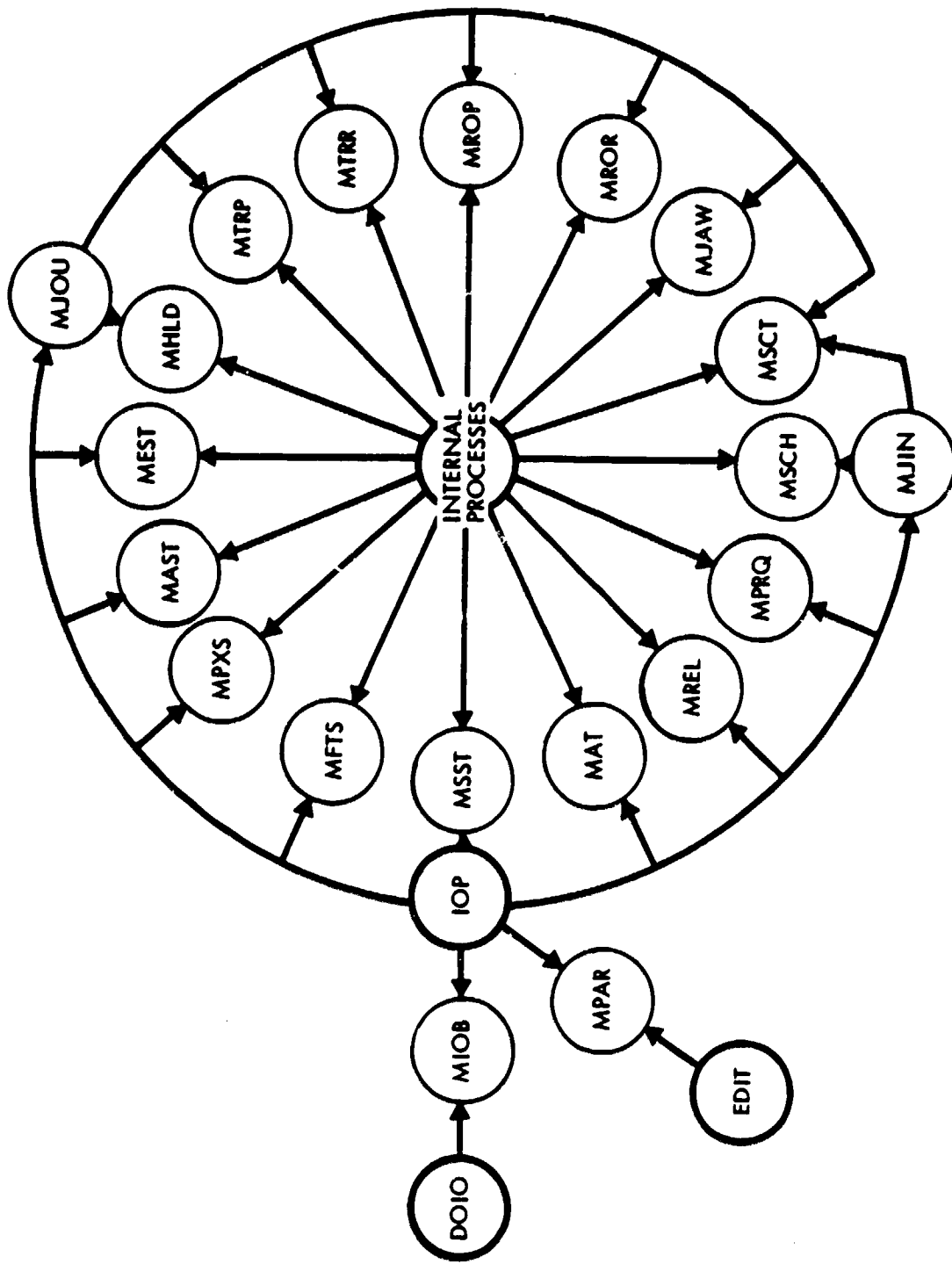


Fig. 3-20. Access Graph of Subsystem OPER

CALLED MONITORS	CALLED MONITOR PROCEDURES	PROCESSES						MONITORS						
		P ₁	P ₂	P ₃	P _{n-1}	P _n	M ₁	M ₂	M ₃	M _{m-1}	M _m	
M ₁	S ₁₁	X	X				X							
	S ₁₂	X	X										X	
	S ₁₃		X	X			X						X	
	⋮													
	S _{1k₁}	X	X										X	
M ₂	S ₂₁	X											X	
	S ₂₂						X						X	
	S ₂₃	X											X	
	⋮													
	S _{2k₂}	X											X	
⋮	⋮													
M _m	S _{m1}	X											X	
	S _{m2}	X											X	
	S _{m3}	X					X		X				X	
	⋮													
	S _{mk_m}	X	X	X										

Fig. 3-21. Access Right Cross-Reference Table

```

program SCHED;
var first : boolean,
    sin : boolean;                                {Initialization switch.}
label 1;
begin
    first:=true;
    DOIO;                                         {Start clock, perform I/O operations on
                                                    ADC, DAC, switches/pushbuttons and
                                                    alarm display. Return address of Clock
                                                    Interrupt Service follows this statement.}

    INITSW(sin);                                  {Get sin from SWTBUF.}

    if not sin then first:=true else
    begin
        if first then
        begin
            first:=false;
            INIT;                                  {Initialize OPER.}
        end
        MUPLEX;                                   {Multiplex internal processes.}
    end
    | goto 1 |                                   {Busy wait until clock interrupt.}
end;

```

Fig. 3-22. Scheduler - Main Program

```

procedure MUPLEX;
var pindex : integer,           {Process ID index.}
    procno : integer,           {Number of processes (parameter).}
    nc     : integer;           {Number of skipped clock cycles and
                                resume indicator.}

begin
    pindex:=0
    while pindex < procno do
    begin
        pindex:=pindex + 1;
        RESUME(pindex, nc);      {Check if process is for resumption
                                (if nc ≥ 0).}

        if nc ≥ 0 then
            begin {Branch to process with index pindex} end
        end
    end;

```

Fig. 3-23. Scheduler - Process Multiplexing

In every iteration cycle the switch SIN is interrogated. If SIN is set, the subroutine INIT is called. This subroutine performs initialization of all pertinent shared variables (for example: process queue, alarm status table, event status table, etc.). Initializations are done by executing corresponding monitor procedures. It is obvious that initialization of the system is possible at any moment without terminating the program SCHED. This feature is helpful for experimental work with a manipulator.

Scheduling of internal processes is accomplished by subroutine MUPLEX, which is shown in Fig. 3-23. This subroutine scans during every iteration cycle all internal processes represented by their index (pin- dex), and it examines if they are for execution or not. This examina- tion is done by subroutine RESUME which is a monitor procedure of the synchronization monitor, MPRQ. The scanned internal process will be resumed, i.e., the corresponding subroutine will be called if the out- put variable nc of the RESUME is nonnegative. If it is negative, the next process will be examined. If it is positive, the variable nc represents the number of skipped iteration cycles in the case of delay- ing the process. This number is important information for the process which must know the exact time passed since the last execution in order to provide time dependent computations. The variable nc is set to a negative value if the process is blocked or delayed.

Figure 3-24 shows the monitor MPRQ which is the basic synchroni- zation component of subsystem OPER. The data structure of the monitor consists of five vectors: pqueue, pqinit, permpr, delays, and dcount. Vector pqueue is a process queue consisting of a bit map pointing to all processes waiting for execution. The process identification is pro- vided by a vector index, so that pqueue [pindex] = true, means that the process indexed by pindex is ready for execution. The vector pqinit is an initial value of the pqueue, while the vector permpr is a bit map of the same size as pqueue and pqinit, defining permanently active proc- esses. Permanently active processes cannot be delayed. Integer vector delays contain information about delay, i.e., the number of iteration cycles which have to be skipped by the scheduler. For example, delays [pindex] = 6 means that the process pindex will not be executed for the next six iteration cycles. During every iteration, this number is decremented by one until delays [pindex] becomes zero. The vector dcount is of the same size as delays, and it contains information about the number of iteration cycles being skipped since the last delay opera- tion. It is used to form the output variable nc of the procedure RESUME.

Monitor MPRQ contains six procedures: INITPQ, INITDS, SIGNAL, WAIT, DELAY and RESUME. The first two procedures are called by sub- routine INIT, Fig 3-22, in order to initialize the vectors pqueue, delays, and dcount. Procedures SIGNAL and WAIT are used by all internal processes for synchronization purposes. The first procedure makes the process active, while the second one blocks the process. As seen, it is ensured that permanent active processes cannot be blocked. These procedures are tools for implementing the process precedence chart given in Fig. 3-15.

```

const maxpno = 32;                {Maximal number of processes.}
type integv : array [1.. maxpno] of integer;
type boolve : array [1.. maxpno] of boolean;
var pqueue : boolve;              {Process queue.}
var pqinit : boolve;              {Initial value of process queue.}
var pempr : boolve;               {Definition of permanent active processes.}
var delays : integv;              {Delay semaphore.}
var dcount : integv;              {Delay count.}

procedure INTPQ;                  {Initialize process queue.}
begin
    pqueue:=pqinit
end;

procedure INITDS;                 {Initialize delay semaphore.}
begin
    delays:=0;
    dcount:=0
end;

procedure SIGNAL (pindex : integer); {Signal process pindex.}
begin
    pqueue [pindex] :=true
end;

procedure WAIT (pindex : integer); {Wait process pindex.}
begin
    if not pempr [pindex] then pqueue [pindex] :=false
end;

```

Fig. 3-24. Process Synchronization Monitor MPRO

```

procedure DELAY (pindex, nd : integer); {Delay process pindex nd clock cycles.}
begin
  if not pempr [pindex] then
    begin
      if delays [pindex] < nd then delays [pindex] := nd
    end
  end
en.

procedure RESUME (pindex : integer; var nc: integer);
                                                    {Examine if process pindex is for resumption.}

  begin
    nc := -1;
                                                    {Initialization of skipped clock cycle number nc.}
                                                    {(if nc < 0 process is not for resumption).}

    if not pqueue [pindex] then
      begin
        delays [pindex] := 0;
        dcount [pindex] := 0
      end else
      begin
        if delays [pindex] > 0 then
          begin
            delays [pindex] := delays [pindex] - 1;
            dcount [pindex] := dcount [pindex] + 1
          end else
          begin
            nc := dcount [pindex];
            dcount [pindex] := 0
          end
        end
      end
    end;

```

Fig. 3-24. Process Synchronization Monitor MPRQ (Continued)

The DELAY procedure causes blocking of process for a limited period of time, defined by the number of iteration cycles to be skipped by the scheduler. If during the same iteration cycle and within the same process two delay operations are called, the one with the greater delay will have priority. The permanently active processes also cannot be delayed.

The procedure RESUME is called by the scheduler. Its purpose is to determine the variable *nc* based on the state of the vector *pqueues* and *delays*. It updates also the vector *delays* and *dcount*.

8. Program Documentation

The programs under design at the time of writing this report will be presented in Appendix B. To facilitate the program design phase and future program maintenance, the program documentation is elaborated systematically by the use of forms which are specifically developed for this purpose. There are three types of forms with the following titles: "Module Definition," "Procedure Definition," and "Data Definition." All of these forms have identical upper right corners and bottom parts. The upper right corner contains the identifier of the object (module, procedure, or data entity). The bottom part includes identifiers of system, subsystem, designers, date, and page number for the case when the form is continued. This uniquely defines each form regardless of the time, system or project. For easy look-up, these forms can be sorted alphabetically by the object identifier, where different object types are separated. Examples of compiled forms are given in Figs. 3-25 - 3-27. Entries in the forms are self-explanatory through the corresponding headers, and the following comments will help their understanding.

The procedure hierarchy of the module is given by the line indentation, rather than by a hierarchical diagram. Procedures are represented by their identifiers together with identifiers of their I/O parameters which are enclosed in parentheses.

The dimensions of data arrays (number of data items in the case of records) are enclosed in square brackets. The procedure identifiers which appear in the entry-point list are underlined, while the identifiers of procedures which belong to another module, i.e., which appear in the external symbol list, are enclosed in the brackets. The same is true for data identifiers. Explanatory comments are given in narrative form on the right side. Every part of the form has a check-box which has to be checked by an "X" if that part of the form must be continued. For this purpose, a new form must be attached to the first one and page numbers must be given.

A procedure definition form is created as HIPO (Hierarchical Input-Process-Output) form. Column "I/O Method" defines the way of

MODULE DEFINITION				IDENTIFIER: MJOUT	
PURPOSE/DESCRIPTION:					
monitor, supports access to calibrated output joint variables.					
PROCEDURE HIERARCHY (ENTRY-POINT SYMBOLS UNDERLINED, EXTERNAL SYMBOLS ENCLOSED IN BRACKETS):					
<u>UPDEJ</u>					
[GETHLD(HLDF [1])]	Monitor MHLD				
DJARM(VP [3])					
[GETTRP(VP [3])]	Monitor MTRP				
[GETTRR (VR [3])]	Monitor MTRR				
[GETSCT(SICOTF [12])]	Monitor MSCTF				
DJHND(VR [3])					
[GETROP(VP [3])]	Monitor MROP				
[GETROR(VR [3])]	Monitor MROR				
DJJAW(VF [1])					
[GETJAW(VJ [1])]	Monitor MJAW				
<u>CPJRAW(argument [7])</u>					
DATA STRUCTURE (ENTRY-POINT SYMBOLS UNDERLINED, EXTERNAL SYMBOLS ENCLOSED IN BRACKETS):					
HLDF [1]	Hold flag (HLD)				
VP [3]	Auxiliary variable				
VR [3]	Auxiliary variable				
VJ [1]	Auxiliary variable				
SICOTF [12]	Trigon. fun. of joint. var.				
JOUT [7]	Calib. joint var. (monitor)				
[ZOTETA [7]]	Zero offsets (parameter)				
[SFTETA [7]]	Scaling factor (parameter)				
SYSTEM:	SUBSYSTEM:	DESIGNER:	DATE:	PAGE:	OF:
CACs	OPER	M. V.	3/15/79	1	1

Fig. 3-25

PROCEDURE DEFINITION					IDENTIFIER: DJARM
	IDENTIFIER	DIM.	TYPE	DESCRIPTION	I/O METHOD
INPUT					
	VP	3	H	increments of arm-joint variables $\Delta \underline{\theta}_A = (\Delta \theta_1, \Delta \theta_2, \Delta \theta_3)$	ad = link + 2
OUTPUT					
FUNCTIONS	<ol style="list-style-type: none"> 1. Takes $\Delta \underline{x}_p$ from buffer TRP by GETTRP. 2. Takes $\Delta \underline{x}_r$ from buffer TRR by GETTRR. 3. Computes $\Delta \underline{x}_c = \Delta \underline{x}_p + \Delta \underline{x}_r$. 4. Takes $\sin(\theta_{iF})$ and $\cos(\theta_{iF})$ ($i = 1, 2, 3$) from buffer SCT by GETSCT. 5. Transforms coordinates from world to joint space: $\Delta \underline{\theta}_A = T_A(\underline{\theta}_A) \cdot \Delta \underline{x}_c, \text{ where } \underline{\theta}_A = (\theta_{1F}, \theta_{2F}, \theta_{3F}).$ 				
SYSTEM:	SUBSYSTEM:	DESIGNER:	DATE:	PAGE:	OF:
CACS	OPER	M. V.	3/15/79	1	1

Fig. 3-26

DATA DEFINITION				IDENTIFIER: JOUT
COMP. INDEX	COMPONENT IDENTIFIER	INITIAL VALUE	MATHEM. SYMBOL	DESCRIPTION
1			θ_1	Arm azimuth
2			θ_2	Arm elevation
3			θ_3	Arm extension
4			θ_4	Hand azimuth
5			θ_5	Hand elevation
6			θ_6	Hand twist
7			θ_7	Gripper opening
<p><u>Notices:</u></p> <p>(1) All values represent calibrated output values (set-points).</p> <p>(2) Angles scaled rad x 2^{15} (Horn form.)</p> <p>(3) Access to JOUT controlled by monitor MJOUT.</p>				
SYSTEM: CACs	SUBSYSTEM: OPER	DESIGNER: M. V.	DATE: 3/15/79	PAGE: OF: 1 1

Fig. 3-27

passing the parameters to/from the procedure. In case of CAL the following notation will be used for different methods:

- (a) $ad = link + m$ Argument passage by argument address located immediately after calling instruction (m is relative position from return address).
- (b) $ad = R_1$ Argument passage by argument address which is contained in the register R_1 .
- (c) R_1 Direct argument passage, as a content of the register R_1 .
- (d) local Argument passage through local variable of the module.
- (e) - No arguments are passed to/from the procedure.

The description of procedure functions can be done in a narrative form or by using flowcharts, pseudo-codes or higher level languages (Pascal, for example).

Data definition forms are intended for simple data structures, such as arrays or simple records.

SECTION IV

CONCLUSIONS AND PLANS

The functional and operational description of the CACS software is given in this quarterly progress report. The operational description comprises the basic description of CACS hardware, the structure of control algorithms, and the principles of the system software implementation. In this document the project objectives and framework have been clarified, the method of approach has been established, and the design of basic system software components, monitors and basic control routines, have been completed.

The next phase of development will consist of three activities: software implementation, additional control algorithms development and real-time experimentation.

The software implementation will be performed in two steps: implementation of the monitors and implementation of the processes. Since the monitors are the basic system components, containing procedures which are common to all processes, they will be implemented separately, out of the top-down development line. Therefore, the set of special off-line testing programs will be elaborated. These programs will be implemented in a high level language, in this case Fortran V. Their purpose is to test extensively all monitor procedures, in order to provide a reliable and secure programming base for further process implementation. This approach of program development is usually called "bottom-up development."

The implementation of the processes will be carried out by top-down step-by-step refinements. The development will start with the main program of OPER subsystem, the scheduler, and will continue with the development of the processes. First the permanent active processes (IOP, OES, OMA) will be implemented, and then other internal processes, starting with MUC and ending with ASO, according to the natural sequence of manipulator operations. The processes will also be implemented by a top-down technique, as far as it offers practical benefits. As a consequence of this approach, no special testing programs are needed, except the procedure stubs for simulation of as yet unimplemented lower level procedures.

The control algorithms will be developed within the frame of the proposed principles, and the basic control routines and other relevant monitor procedures designed in this report. It should be noted that almost all CACS algorithms have been analyzed and evaluated by a look-ahead design which has preceded this report. The final version of the algorithms and related parameters will be established by experimental procedures.

The real-time experiments will be carried out using the full capacity of the CACS hardware under real operational conditions. The purpose of this activity is to check out the system interface and to adjust and review algorithmic parameters, calibration constants, and other pertinent data. In addition, the system performance and limitations as well as the impact of the real operational conditions on system functioning will be examined systematically in order to be able to improve control performance through algorithmic modifications.

The next progress report will describe the development activities outlined above. The first part of the report will be devoted to testing problems. The second part will consider processes, i.e., the related algorithms and their software implementation.

PRECEDING PAGE BLANK NOT FILMED

SECTION V

REFERENCES

1. Bejczy, A. K., Environment-Sensitive Manipulator Control, IEEE Conference on Decision and Control and 13th Symposium on Adaptive Processes, Phoenix, AZ, Nov. 20-22, 1974.
2. Bejczy, A. K., Issues in Advanced Automation for Manipulator Control, Joint Automatic Control Conference, Purdue University, W. Lafayette, IN, July 27-30, 1976.
3. Bejczy, A. K., "Effect of Hand-Based Sensors on Manipulator Control Performance," Mechanism and Machine Theory, vol. 12, pp. 547-567, 1977.
4. Dijkstra, E. W., "Cooperating Sequential Processes," In Programming Languages, P. Genuys (ed.), Academic Press, New York, NY, 1968.
5. Wirth, N., "On Multiprogramming, Machine Coding and Computer Organization," Comm. ACM, vol. 12, no. 9, pp. 489-498, Sept. 1969.
6. Brinch Hansen, P., "Structured Multiprogramming," Comm. ACM, vol. 15, no. 7, pp. 574-578, July 1972.
7. Brinch Hansen, P., "Operating System Principles," Prentice-Hall, Inc., Englewood Cliffs, NJ, July 1973.
8. Hoare, C. A. R., "Monitors: an Operating System Structuring Concept," Comm. ACM, vol. 17, no. 10, pp. 549-557, Oct. 1974.
9. Brinch Hansen, P., "The Programming Language Concurrent Pascal," IEEE Transactions, vol. SE-1, no. 2, pp. 199-207, June 1975.
10. Wirth, N., "Modula: a Programming Language for Modular Programming," Software Practice and Experience, vol. 7, no. 2, March 1977.
11. Brinch Hansen, P., "The Architecture of Concurrent Programs," Prentice-Hall, Inc., Englewood Cliffs, NJ, 1977.
12. Zawacki, R. L., "FORTRAN-TO-CURV Arm Interface I/O Drivers," Interoffice memo 343-76-794, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Dec. 1976.
13. Raibert, M. H., "Preliminary Proposal for Teleoperator Software Architecture," Interoffice memo 343-78-101, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, Jan. 1978.
14. Jensen, K. and Wirth, N., "Pascal, User Manual and Report," Springer-Verlag, New York, NY, 1975.

APPENDIX A
GRAPHICAL REPRESENTATION OF DATA AND STANDARD OPERATION

In order to simplify the system description, a graphical technique is employed in this report. This technique consists of graphical symbols which express both the data and the operations on the data.

Two categories of data will be considered here: scalar and vector data. The former represent single data values with which a symbolic name is associated and indicated on the diagram. Vectors represent one-dimensional data arrays, which are ordered sets of scalar data under one symbolic name.

Operations on data can also be considered as two general categories: elementary algebraic operations and data buffering. Elementary algebraic operations include summation and multiplication of scalar and/or vector data, as well as elementary arrangements/rearrangements of data arrays (join and disjoin operations). For operations which are more general special symbols are used. This will require additional explanations.

Data buffering are specific kinds of storage operations which play an important role in CACS software. Data buffers are memory locations with dimensions corresponding to the dimensions of the data arrays indicated on the diagram. These locations are accessible through special procedures which ensure mutual exclusiveness and access right control. The length of all buffers will be the same. This means that all buffers can accept only one scalar/vector data entity.



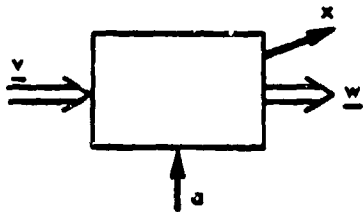
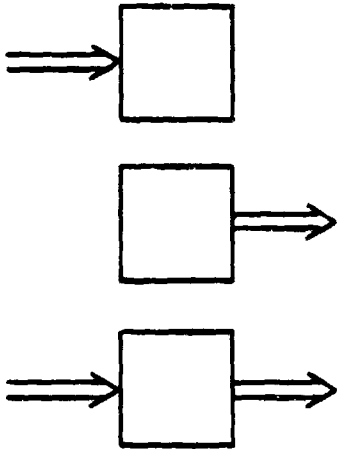
	<p>Scalar data</p>
	<p>Vector data</p> $\underline{v} = (v_1, v_2, \dots, v_N)$ <p>(v_1 is scalar data)</p>
	<p>General data processing</p>
	<p>Output device</p> <p>Input device</p> <p>Input-output device</p>

Fig. A-1. Graphical Representations of Data and Standard Operations

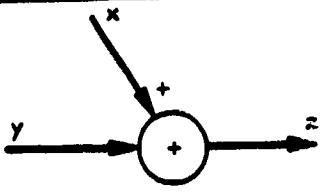
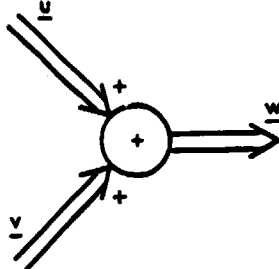
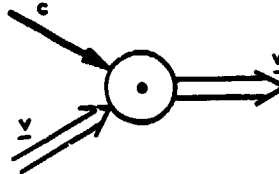
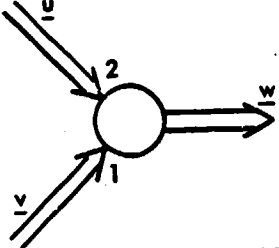
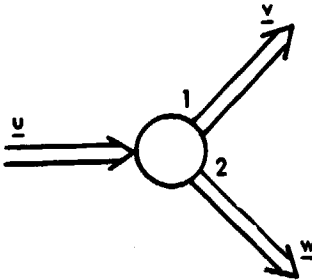
	<p>Algebraic summation</p> $z = x - y$
	<p>Vector summation</p> $\underline{w} = \underline{u} + \underline{v}$ $\underline{u} = (u_1, u_2, \dots, u_N)$ $\underline{v} = (v_1, v_2, \dots, v_N)$ $\underline{w} = (u_1 + v_1, u_2 + v_2, \dots, u_N + v_N)$
	<p>Scalar multiplication</p> $\underline{w} = (c \cdot v_1, c \cdot v_2, \dots, c \cdot v_N)$
	<p>Joining the vectors</p> $\underline{u} = (u_1, u_2, \dots, u_M)$ $\underline{v} = (v_1, v_2, \dots, v_N)$ $\underline{w} = (v_1, v_2, \dots, v_N, u_1, u_2, \dots, u_M)$
	<p>Disjoining the vectors</p> $\underline{u} = (u_1, u_2, \dots, u_M)$ $\underline{v} = (u_1, u_2, \dots, u_N) \quad M > N$ $\underline{w} = (u_{N+1}, u_{N+2}, \dots, u_M)$ <p>Note: Numbers besides the data symbols define the order of joining/disjoining</p>

Fig. A-1 (Continued)






	<p>Shared data buffer (bid is buffer identifier)</p> <p>Note: The size of buffer corresponds to the dimension of input/output data.</p>
INPUT BUFFER OPERATION	
	<p>Unconditional replacement of buffer content y by value x</p>
	<p>Conditional replacement of buffer content y by value x (Condition must be defined separately.) Examples:</p> <ol style="list-style-type: none"> 1. Replace if x is less than y. 2. Replace if x is less than m. Otherwise replace y by m.)
	<p>Add value x to buffer content y.</p>
	<p>Store x in buffer if it is empty. If it is full, the store operation will be delayed until the buffer becomes empty.</p>

Fig. A-1. (Continued)


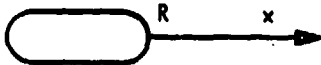
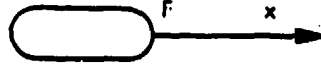
OUTPUT BUFFER OPERATION	
	<p>Read buffer without changing its content.</p>
	<p>Read and reset buffer.</p>
	<p>Read and empty buffer if it is full. If it is empty, the read operation will be delayed until the buffer becomes full.</p>

Fig. A-1. (Continued)

PART III
August 31, 1981
OPTIMAL PRODUCTION SCHEDULING
FOR A LINEAR FLOW SHOP

by

Gad Vitner

INSTITUTE FOR TECHNOECONOMIC SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007

ABSTRACT

This study investigates a flow shop scheduling problem which is defined as the linear flow shop problem (LFP). The time to perform the jobs in the shop is a linear function of the batch size. The results of the research show that the shortest processing time (SPT) sequencing is the optimal solution for the mean lateness, mean flow time and waiting time. A typical scheduling problem that can fit the definition of the linear flow shop problem is an assembly line.

The main objective of this research is to find an optimal solution for a production scheduling problem that will be easy to implement in practical situations. The results show that the optimal solution is very easy to implement in real life problems as the jobs have to be arranged according to a monotonic increasing processing times.

The proofs for mean lateness and mean flow time use the idea of pairwise interchange of adjacent jobs in the sequence. The method of mathematical induction is used to prove that the theorems hold for the case of m jobs in the sequence. A numerical example is presented to

explain how the theory is implemented. A sensitivity test is conducted on the numerical results to show how the shortest processing time sequencing yields the optimal solutions.

TABLE OF CONTENTS

	Page
ABSTRACT	i
TABLE OF CONTENTS	iii
LIST OF FIGURES	v
LIST OF SYMBOLS	vii
Chapter	
1 INTRODUCTION TO SEQUENCING AND SCHEDULING	1
1.1 Definition and Classification of a Sequencing/Scheduling Problem	1
1.2 Types of Scheduling Models	2
1.3 Performance Measurements of a Production Schedule	3
1.4 Research Objectives	5
2 FLOW SHOP SCHEDULING - BASIC CONCEPTS	6
2.1 Definitions and Terminology	6
2.2 Characteristics of a Flow Shop	8
3 REVIEW OF PAST WORK	10
4 FORMULATION OF THE PROBLEM	17
4.1 The General Linear Flow Shop Problem	18
	iii

	Page
4.2 Time to Perform All Jobs Through All Machines	19
4.3 Machining Time Required to Perform The Jobs	20
4.4 Idle Time Involved in Linear Flow Shop	21
4.5 Makespan (MS) in Linear Flow Shop	28
4.6 Bounds for Performance Time in a Linear Flow Shop	29
5 OPTIMALITY IN LINEAR FLOW SHOP SCHEDULING	31
5.1 Lateness in a Linear Flow Shop	31
5.2 Computational Results	58
5.3 Flow Time in a Linear Flow Shop	98
5.4 Waiting Time in a Linear Flow Shop	102
6 SUMMARY AND CONCLUSIONS	104
BIBLIOGRAPHY	108

LIST OF FIGURES

Figure		Page
2.1	A "Pure" Flow Shop	7
2.2	A General Flow Shop	7
3.1	Graph of a Two Job Flow Shop Schedule	13
4.1	Idle Time in a Two Machine Linear Flow Shop	23
4.2	Idle Time in a Three Machine Linear Flow Shop	26
5.1	A Pairwise Interchange of Adjacent Jobs	33
5.2	A Two Job Linear Flow Shop	41
5.3	A Three Job Linear Flow Shop	44
5.4	A Four Job Linear Flow Shop	50
5.5	A Block Diagram that Summarizes the Steps of the Proof	57
5.6	Flowchart that Describes the Sensitivity Test	60
5.7	Results of Sensitivity Test for Sequence 1	61
5.8	Results of Sensitivity Test for Sequence 2	62
5.9	The Optimal Sequence	63
5.10	Results of Sensitivity Test for Sequence 4	63
5.11	Results of Sensitivity Test for Sequence 5	64
5.12	Results of Sensitivity Test for Sequence 6	65
5.13	Results of Sensitivity Test for Sequence 7	66
5.14	Results of Sensitivity Test for Sequence 8	67
5.15	Results of Sensitivity Test for Sequence 9	68

	Page
5.16 Results of Sensitivity Test for Sequence 10	69
5.17 Results of Sensitivity Test for Sequence 11	70
5.18 Results of Sensitivity Test for Sequence 12	71
5.19 Results of Sensitivity Test for Sequence 13	72
5.20 Results of Sensitivity Test for Sequence 14	73
5.21 Results of Sensitivity Test for Sequence 15	74
5.22 Results of Sensitivity Test for Sequence 16	75
5.23 Results of Sensitivity Test for Sequence 17	78
5.24 Results of Sensitivity Test for Sequence 18	81
5.25 Results of Sensitivity Test for Sequence 19	82
5.26 Results of Sensitivity Test for Sequence 20	83
5.27 Results of Sensitivity Test for Sequence 21	86
5.28 Results of Sensitivity Test for Sequence 22	89
5.29 Results of Sensitivity Test for Sequence 23	92
5.30 Results of Sensitivity Test for Sequence 24	95
5.31 Two Machine Linear Flow Shop	99

LIST OF SYMBOLS

b_j	Processing time per part on machine j , p.17
C_i	Completion time of job i , p.4
C_A	Completion time of the last job in set A, p.34
d_i	Due date of job i , p.3
F	Mean flow time, p.4
F_i	Flow time of job i , p.4
F_A	The sum of the flow times of all the jobs in set "A", p.100
F_B	The sum of the flow times of all the jobs in set "B", p.100
I	Index set of jobs, p.18
I_{ij}	Idle time of machine j , immediately before the i^{th} job starts on machine j , p.22
IT_j	Idle time of machine j , p.4
J	Index set of machines, p.18
\underline{L}	Lower bound for performance time in a linear flow shop, p.29
\bar{L}	Mean lateness, p.5
L_i	Lateness of job i , p.4
L_A	Total lateness of the jobs in set "A", p.35
L_B	Total lateness of the jobs in set "B", p.35
M	Total machining time over all n machines, p.21
MS	Makespan, p.28

M_j	Time that machine j is busy, p.20
n_i	number of parts in job, i , p.17
P	$\max\{T_A + t_{i,1}, C_A\}$, p.36
Q	$\max\{T_A + t_{i+1,1}, C_A\}$, p.36
r_i	ready time, p.3
R	$\max\{T_A + t_{i,1} + t_{i+1,1}, C_i(S)\}$, p.37
S_i	A slope index for ordering jobs, p.16
SL	Scheduling line of a two job flow shop, p.13
t_{ij}	Performance time of job i on machine j , p.17
T	Total performance time of all m jobs over all n machines, p.19
T_A	The point in time at which job i starts in sequence S and at which job $i+1$ starts in sequence S' , p.32
T_i	Time to perform job i over all n machines, p.19
TR_i	Tardiness of job i , p.4
U	Upper bound for performance time in a linear flow shop, p.49
V	$\max\{T_A + t_{i+1,1} + t_{i,1}, C_{i+1}(S')\}$, p.37
X	$\max\{(1+b)t_{11}, t_{11} + t_{21}\}$, p.47
Y	$\max\{X + bt_{21}, t_{11} + t_{21} + t_{31}\}$, p.48
Y_1	$Y + \max\{Y + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\}$, p.54
Z	$\max\{(1+b)t_{21} + bt_{11}, t_{21} + t_{11} + t_{31}\}$, p.48
Z_1	$Z + \max\{Z + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\}$, p.54

CHAPTER 1
INTRODUCTION TO SEQUENCING AND SCHEDULING

**1.1 Definition and Classification of a Sequencing/
Scheduling Problem**

A sequencing problem is defined as one that determines the relative order or sequence in which given jobs or tasks are to be performed by the available facilities or resources. A scheduling problem is defined as one that assigns the actual starting time for each task on various resources.

In general, there are two kinds of scheduling problems. The static problem handles a set of tasks that is available for scheduling before the scheduling process starts. In this case the set of jobs is fixed and does not change over time. The dynamic problem deals with a scheduling process where in addition to a given fixed set of tasks, new jobs arrive to the system after the scheduling process starts. The dynamic problem presents a real life production process where the set of tasks to be considered for scheduling contains two subsets of tasks. The first subset includes tasks that the system has from orders on hand and the second subsystem includes tasks from orders that will arrive to the system in the future.

1.2 Types of Scheduling Models

To classify the major scheduling models it is necessary to characterize the configuration of resources and the behavior of the tasks. Each model can be used in a static or a dynamic approach

- (a) **Single Machine:** One machine is continuously available and is never kept idle while work is waiting
- (b) **Parallel Machines:** Several identical machines are available for scheduling the tasks. Tasks are performed starting at time $t=0$, consecutively, so that as soon as a task is completed another task can put on the machine that is freed.
- (c) **Flow Shop:** Jobs to be scheduled follow a fixed routing and the routing is the same for all jobs.
- (d) **Job Shop:** Routing for all jobs is fixed but each job may have a different routing.

The solutions to the various models are achieved mainly by two kinds of techniques; optimal techniques and heuristic techniques. The optimal techniques use mathematical programming such as linear programming, integer programming, dynamic programming and enumeration techniques like branch and bound. Mathematical approaches become very complex for practical cases as computational requirements will be severe for large problems. Even for relatively small problems,

there is no guarantee that the solution can be obtained quickly. Heuristic techniques usually assign a priority, or set of priority rules to sequence the jobs that are ready to be performed. Computational problems are avoided and the solutions to large problems can be obtained with limited computational effort. The problem with heuristic methods is that they do not guarantee optimality. The goal in practical problems is to get a "good" (not optimal) solution by using a simple sequencing rule, in a relatively short time. In special cases the results of using heuristic priority rules will result in an optimal or near optimal solution.

1.3 Performance Measurements of a Production Schedule

Quantitative measures for evaluating schedules are very important to determine whether a specific schedule is efficient. Performance measurements are functions of variables that define a scheduling process. The basic variables are:

Ready Time (r_i). The point in time at which job i is available for processing. In a static model where all the jobs are available for processing at time zero, $r_i=0$.

Processing Time (t_i). The amount of processing required by job i

Due Date (d_i). The point in time at which the processing of job i is due to be completed.

Completion Time (C_i). The time at which the processing of job i is finished.

The major quantitative measures for evaluation are:

Flow Time (F_i). The amount of time job i spends in the system:

$$F_i = C_i - r_i$$

$$F_i = C_i \text{ if } r_i = 0.$$

Makespan (MS). The amount of time from zero until all jobs are completed or the completion time of the last job.

Lateness (L_i). The amount of time by which the completion time of job i exceeds its due date: $L_i = C_i - d_i$.

Tardiness (TR). The lateness of job i if it fails to meet its due date, or zero otherwise: $TR_i = \max\{0, L_i\}$

Idle Time (IT_j). The amount of time that machine j is not productive.

Schedules are generally evaluated by aggregate quantities that involve information about all jobs, resulting in one dimensional performance measures. Suppose that m jobs are to be scheduled on n machines then aggregate performance measures are:

Mean Flow Time:
$$\bar{F} = \frac{1}{m} \sum_{i=1}^m F_i$$

Maximum flow time:
$$F_{\max} = \max_{i \in I} \{F_i\}$$

Mean Lateness:
$$L = \frac{1}{m} \sum_{i=1}^m L_i$$

Maximum Tardiness:
$$TR_{\max} = \max_{i \in I} \{TR_i\}$$

1.4 Research Objectives

This research investigates a linear flow shop model where the time to perform the jobs on the machines is a linear function of the batch sizes. The results of the research show that the shortest processing time (SPT) sequencing is the optimal solution for the mean lateness, mean flow time and waiting time.

The main objective of this investigation is to find an optimal solution for a production scheduling problem that will be easy to implement in practical situations. The results show that the optimal solution is very easy to implement in real life problem as the jobs have to be arranged according to a monotonic increasing processing times. The results can be implemented in processes as assembly lines where the jobs are following the same routing.

The proof for the minimum mean lateness and the minimum mean flow time is carried out by using mathematical induction for the case of m jobs in the system. An analysis of a numerical example shows how the results are implemented in a specific case.

CHAPTER 2
FLOW SHOP SCHEDULING - BASIC CONCEPTS

2.1 Definitions and Terminology

A flow shop scheduling system is defined as a process where the jobs to be scheduled follow a fixed routing and the routing is the same for all jobs.

The shop contains n different machines and each job consists of n operations, one for each machine as illustrated in Figure 2.1 for a "Pure" n machine flow shop. The machines in a flow shop are numbered so that the j^{th} operation of any job precedes its k^{th} operation, then the machine required by the j^{th} operation has a lower number than the machine required by the k^{th} operation. The machines in a flow shop are numbered $1, 2, \dots, n$ and the operations of job i are numbered $(i,1), (i,2), \dots, (i,n)$. It is not required that every job have an operation on each machine in the shop as in the case of a general flow shop illustrated in Figure 2.2. Jobs must not enter the shop on a single machine, or leave from a single machine. In a general flow shop, each job is treated as if it had exactly n operations, for in cases where fewer operations exist, the corresponding processing times are taken to be zero.

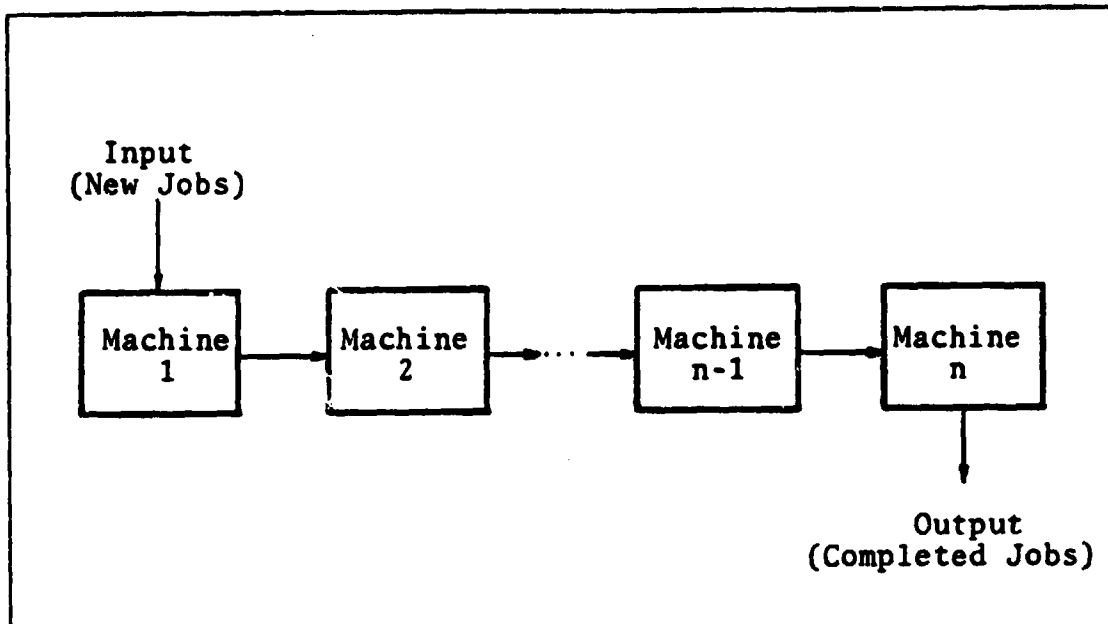


Figure 2.1. A "Pure" Flow Shop

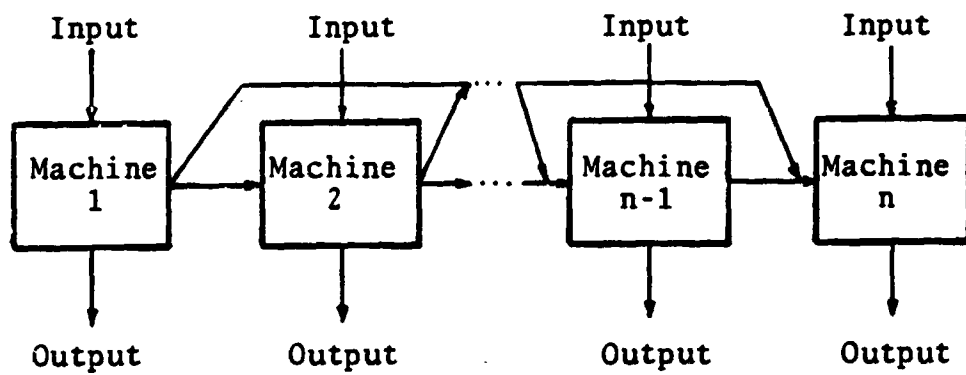


Figure 2.2. A General Flow Shop

The only requirement is that all movements between machines within the shop be in uniform direction from machine j to $j+1$ and from machine $j+1$ to machine $j+2$ etc.

An example of such a shop is an assembly line, where the workers or work stations represent the machines. However, a group of machines served by a unidirectional, noncyclic conveyor would be considered a flow shop.

2.2 Characteristics of a Flow Shop

A basic flow shop problem is characterized by these conditions.

- (a) A set of m multiple-operation jobs is available for processing at time zero.
 - (b) n different machines are continuously available, without consideration of temporary unavailability for causes such as breakdown or maintenance.
 - (c) Each operation can be performed by only one machine in the shop.
 - (d) Setup times for the operations are sequence-independent and are included in processing times.
 - (e) Job descriptors are known in advance.
 - (f) Individual operations are not preemptable, once an operation is started on a machine, it must be completed before another operation can begin on that machine.
- Only a single interval (b,c) is to be assigned to each

operation with $(c-b)$ equal to the processing time of the operation.

- (g) Each machine can handle at most one operation at a time. Consider the interval (b_x, c_x) , the assignment of operation x to a particular machine. For every other assignment (b_y, c_y) to that machine, either $b_x \geq c_y$ or $c_x \leq b_y$.

CHAPTER 3

REVIEW OF PAST WORK

The literature on flow shop scheduling contains many papers that introduce general ideas, algorithms to solve specific problems and analysis of different performance measures. This literature survey presents the typical papers in the area, papers that present general ideas or general techniques to solve the problem.

The earliest optimal results for the two machine flow shop problem were obtained by Johnson (1954). The objective of Johnson was to minimize makespan or minimize the maximum flow time. Johnson proved that in an optimal sequence, job i precedes job j if: $\text{MIN}\{t_{i1}, t_{j2}\} \leq \text{MIN}\{t_{i2}, t_{j1}\}$ where t_{i1} is the time to perform job i on machine 1 etc. Johnson extended this algorithm for the case of three machines. The problem loses some of the nice structure of the two machine case. The problem is formulated, however, and for the special cases where $\text{MIN}\{t_{i1}\} \geq \text{MAX}\{t_{j2}\}$ or $\text{MIN}\{t_{i3}\} \geq \text{MAX}\{t_{j2}\}$ the complete solution is found analogously to the two machine problem.

Many papers have been written about other performance measurements but no one has found an optimal algorithm,

other than enumeration, for the general flow shop problem. Conway, Maxwell and Miller (1967) state in their book, Theory of Scheduling, "Even for the two machine flow shop the optimization of mean flow time is a very difficult problem... Johnson's procedure is not optimal with respect to this criterion and, in general, it is not even very good" (p. 89).

Ignal and Schrage (1965) applied a branch and bound technique to the three machine flow shop problem. They observe that the three machine maximum flow time problem is easier for the "branch and Bound" procedure than the two machine flow time problem, in that a higher proportion of the job sets were solved with the minimum number of nodes.

Jackson (1956) considers a case in which the m jobs have a common machine for their first operation and a common machine for their last (third) operation, but in which the second operation of each job is performed on a different machine. There are thus $m+2$ machines, m of them corresponding to a second machine in a flow shop which can process many jobs simultaneously. The algorithm is similar to Johnson's three machine method.

Dudek and Teuton (1964) have proposed an algorithm to minimize the maximum flow time in a three machine flow shop, which is also applicable to larger shops if one arbitrarily limits consideration to permutation schedules. The algorithm suggests a method for selecting the job to be placed

first in the sequence, the job to be selected from the remaining $m-1$ to be placed second, etc.

Wagner and Story (1963) have used integer programming to formulate and solve the problem of three machine flow shop, to minimize maximum flow time.

Gupta and Dudek (1971) examine various optimization criteria and investigate the interaction of several cost factors on optimal schedules. Based on the results of a sensitivity analysis performed to study the interactions of several cost factors on the optimal schedule, they suggest the adoption of minimization of total opportunity cost as the optimization criterion for the flow shop schedules.

Akers (1956) has a graphical solution for a special flow shop problem. There are only two jobs to be scheduled through a flow shop of any number of machines. Suppose that the machine are numbered $1, 2, \dots, n$ in the order in which they process the jobs, so that the processing times are given by:

$$\begin{array}{ll} t_{1,1}, t_{1,2}, \dots, t_{1,n} & \text{For job 1,} \\ t_{2,1}, t_{2,2}, \dots, t_{2,n} & \text{For job 2.} \end{array}$$

These times can be marked off on axes for the two jobs as shown in Figure 3.1. A schedule can be represented by any line.

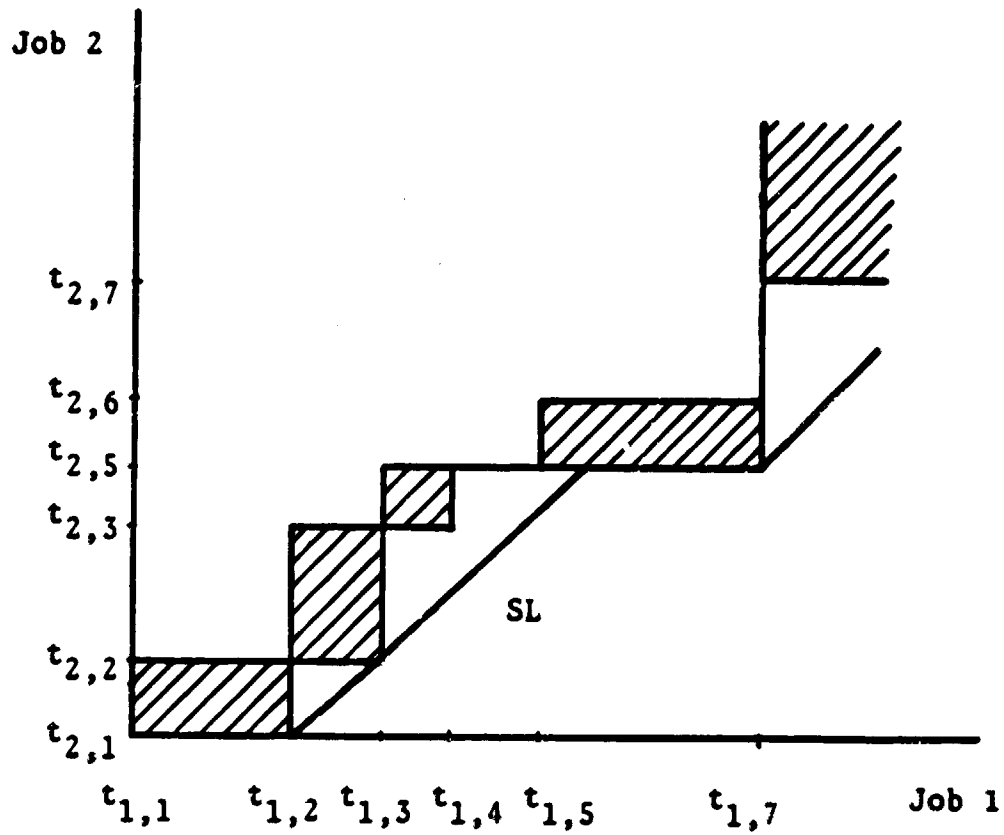


Figure 3.1. Graph of a Two Job Flow Shop Schedule

1. From $(0,0)$ to $(\sum_{j=1}^n t_{1,j}, \sum_{j=1}^n t_{2,j})$;
2. That is composed of horizontal (work on job 1 only), vertical (work on job 2 only) and 45° (work on both jobs) line segments;
3. That does not enter the interior of any of the shaded regions (which would imply one machine working on both jobs simultaneously).

Line SL in Figure 3.1 is such a "schedule line."

Mathematical approaches such as linear programming, integer programming, dynamic programming and branch and bound to the flow shop schedule become very complex for practical problems. The formulation of specific problems in mathematical models is time consuming and need experts that are not available in every production facility. In practical problems many variables are involved in the model and computational requirements will be severe. Even for relatively small problems, there is no guarantee that the solution can be obtained quickly. In dynamic organizations where schedules change every day for example, solutions that are the output of a mathematical model cannot be used efficiently as it takes some times hours to get the results, even if one uses a fast computer. There are cases where in the production plant there are not people who can read and understand the computer output and then implement it in the

shop. Using heuristic algorithms, computational problems are avoided. These techniques obtain solutions to large problems with limited computational effort. Experts are not needed to develop mathematical models and to understand the output of the models. Heuristic methods usually assign a priority, or set of priority rules to sequence the jobs that are ready to be performed in the shop. Priority rules as first come first serve, shortest processing time and minimum slack time are easy to implement in the shop floor and it is very easy to run a production facility following these simple priority rules. In many cases priority rules are used in production plants as a result of experience of years and they turn to yield good results. The problem with heuristic methods is that they do not guarantee optimality. However, the goal in practical problems is to get a "good" (not optimal) solution by using a simple sequencing rule, in a relatively short time without the need of experts. In special cases the results of using heuristic priority rules will result in an optimal or near optimal solutions. Heuristic rules of assigning jobs in a scheduling process can be modified after using it in the shop and getting some results about the level of efficiency.

Palmer (1965) developed a heuristic algorithm for the makespan problem. The algorithm gives priority to jobs having the strongest tendency to progress from short times

to long times in the sequence of operations. He proposes the calculation of a slope index S_i for each job.

$$S_i = \frac{(n-1)t_{in} + (n-3)t_{i,n-1} + (n-5)t_{i,n-2} + \dots + (n-3)t_{i,2} + (n-1)t_{i,1}}{(n-1)t_{i,1}}$$

where:

n is the number of machines in the shop.

$t_{i,n}$ is the time to process job i on machine n .

Then a permutation schedule is constructed using the job ordering $S_1 \geq S_2 \geq \dots \geq S_m$

CHAPTER 4
FORMULATION OF THE PROBLEM

The machine scheduling problem considered for this research is for a flow shop facility engaged in the batch production of parts. Each job consists of a known number of parts (n). The parts are produced, in order, on a number of machines (j). The goal is to sequence all of the parts production jobs (i) so that the entire production schedule is completed in the minimum amount of time. The feature that makes the problem both attractive and practical is that the processing time on each machine is a linear function of the number of parts in that job.

A common representation of machine time is by

$$t_{ij} = b_j n_i$$

where t_{ij} = the time of job i on machine j ,
 b_j = processing time per part of machine j ,
 n_i = the number of parts in job i .

This problem is a typical example of an assembly line problem where one has to produce different types or models. In an automobile assembly for example, you will produce n_1 cars of Model A, n_2 cars of Model B, etc.

The value of b_j is either computed from historical data or estimated from machine operating characteristics.

Periodically the value of b_j is updated in order to keep it as a good representative of the real machine performance.

The conditions that characterize a linear flow shop problem (LFP) are:

- a. A set of m multiple-operation jobs is available for processing at time zero.
- b. Each job requires n operations and each operation requires a different machine.
- c. Set up times for the operations are sequence independent and are included in processing times.
- d. Job descriptors are known in advance.
- e. n different machines are continuously available.
- f. Individual operations are not preemptable.

4.1 The General Linear Flow Shop Problem

The General LFP Problem considers the case of m jobs processed on n machines where the performance time t_{ij} is a linear function of the batch sizes

$$t_{ij} = b_j n_i, \quad i \in I, j \in J \quad (4.1)$$

$$n_i = \frac{t_{ij}}{b_j}, \quad j \in J, i \in I \quad (4.2)$$

Substituting equation (4.2) in equation (4.1) yields the following linear relations between the time to perform a batch on a specific machine and the times to perform the same batch on all other machines.

$$t_{ij} = b_j \frac{t_{ik}}{b_k} = \frac{t_{ik} b_j}{b_k}, \quad j, k \in J, i \in I \quad (4.3)$$

4.2 Time to Perform All Jobs Through All Machines

Let T_i be the time to perform job i ($i \in I$) over all n machines that the shop contains.

$$T_i = \sum_{j=1}^n t_{ij} \quad (4.4)$$

Substituting equation (4.3) into (4.4) yields:

$$T_i = \sum_{j=1}^n \frac{t_{ik}}{b_k} b_j = \frac{t_{ik}}{b_k} \sum_{j=1}^n b_j, \quad i \in I, k \in J \quad (4.5)$$

Let T be the total performance time of all m jobs over all n machines, then:

$$T = \sum_{i=1}^m T_i \quad (4.6)$$

$$\sum_{i=1}^m T_i = \sum_{i=1}^m \left[\frac{t_{ik}}{b_k} \sum_{j=1}^n b_j \right], \quad k \in J \quad (4.7)$$

$$T = \sum_{i=1}^m T_i = \sum_{i=1}^m \sum_{j=1}^n \frac{t_{ik}}{b_k} b_j, \quad k \in J \quad (4.8)$$

Substituting equation (4.2) into (4.5) yields:

$$T_i = \sum_{j=1}^n n_i b_j = n_i \sum_{j=1}^n b_j \quad (4.9)$$

The total performance time T is:

$$T = \sum_{i=1}^m T_i = \sum_{i=1}^m \left[n_i \sum_{j=1}^n b_j \right] \quad (4.10)$$

$$T = \sum_{i=1}^m \sum_{j=1}^n n_i b_j \quad (4.11)$$

4.3 Machining Time Required to Perform the Jobs

Let M_j be the time that machine j is busy, then:

$$M_j = \sum_{i=1}^m t_{ij}, \quad j \in J \quad (4.12)$$

Given that:

$$t_{ij} = \frac{t_{ik}}{b_k} b_j, \quad k \in J$$

The general term for machining time of machine j is:

$$M_j = \sum_{i=1}^m t_{ij} = \sum_{i=1}^m \frac{t_{ik}}{b_k} b_j, \quad k \in J \quad (4.13)$$

$$M_j = \frac{b_j}{b_k} \sum_{i=1}^m t_{ik}, \quad k \in J \quad (4.14)$$

Let M be the total machining time over all n machines, then:

$$M = \sum_{j=1}^n M_j \quad (4.15)$$

$$\sum_{j=1}^n M_j = \sum_{j=1}^n \left[\frac{b_j}{b_k} \sum_{i=1}^m t_{ik} \right], \quad k \in J \quad (4.16)$$

$$M = \sum_{j=1}^n M_j = \sum_{j=1}^n \sum_{i=1}^m \frac{b_j}{b_k} t_{ik}, \quad k \in J \quad (4.17)$$

The term for M can be rewritten. Substituting equation (4.2) into (4.14) yields:

$$M_j = \sum_{i=1}^m n_i b_j, \quad j \in J \quad (4.18)$$

Then

$$M = \sum_{j=1}^n M_j = \sum_{j=1}^n \sum_{i=1}^m n_i b_j \quad (4.19)$$

4.4 Idle Time Involved in Linear Flow Shop

Idle time is defined as the time in which machines or work centers are not producing any parts. In any production plant the purpose is to minimize the idle time of machines in order to increase the total efficiency of the process. In case this idle time of the machines can be controlled and it is known in advance when the machines are not busy,

it is possible to take advantage of this fact and do some preventive maintenance or any other planned maintenance to keep these machines running. Let's define I_{ij} as the idle time of machine j , immediately before the i^{th} job starts on machine j . There is no idle time on the first machine.

4.4.1 Two Machine LFP

Figure 4.1 illustrates the case of a two machine linear flow shop. In developing the terms for the idle time of machines in the shop, the linear relations constituted by the definition of the LFP problem are used. The idle time immediately before job 1 starts on machine 2 is:

$$I_{12} = t_{11} \quad (4.20)$$

$$I_{22} = \max\{t_{11} + t_{21} - \frac{b_2}{b_1} t_{11} - I_{12}, 0\} \quad (4.21)$$

The sum of (4.20) and (4.21) yields:

$$I_{12} + I_{22} = \max\{t_{11} + t_{21} - \frac{b_2}{b_1} t_{11}, t_{11}\} \quad (4.22)$$

$$I_{32} = \max\{\sum_{i=1}^3 t_{i1} - \frac{b_2}{b_1} \sum_{i=1}^2 t_{i1} - \sum_{i=1}^2 I_{i2}, 0\} \quad (4.23)$$

The sum of (4.22) and (4.23) yields:

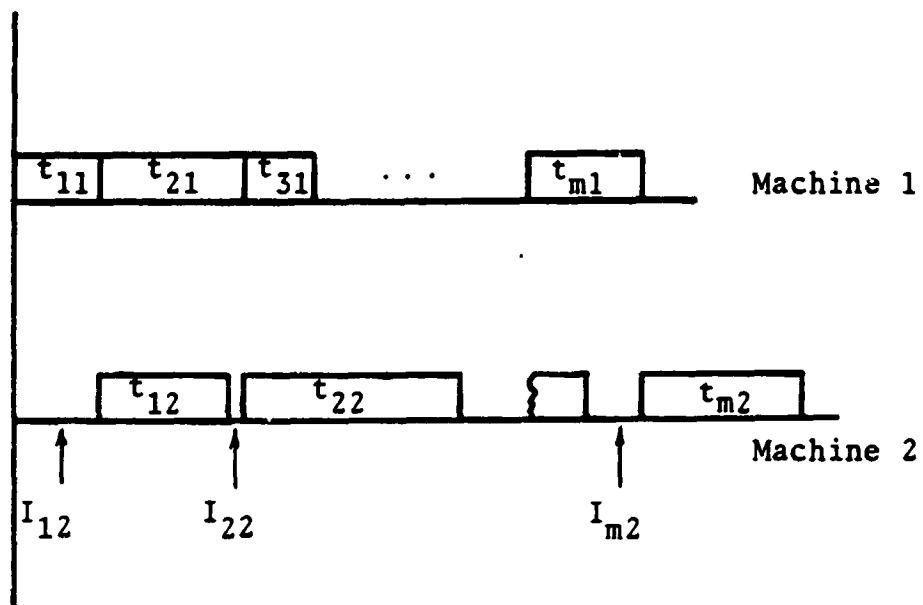


Figure 4.1. Idle Time in a Two Machine Linear Flow Shop

$$\begin{aligned}
I_{12} + I_{22} + I_{32} &= \max\left\{\sum_{i=1}^3 t_{i1} - \frac{b_2}{b_1} \sum_{i=1}^2 t_{i1}, \sum_{i=1}^2 I_{i2}\right\} \\
&= \sum_{i=1}^3 I_{i2} = \max\left\{\sum_{i=1}^3 t_{i1} - \frac{b_2}{b_1} \sum_{i=1}^2 t_{i1}, \sum_{i=1}^2 t_{i1} - \frac{b_2}{b_1} t_{11}, t_{11}\right\}
\end{aligned} \tag{4.24}$$

The idle time immediately before the last job starts on machine 2 is:

$$I_{m2} = \max\left\{\sum_{i=1}^m t_{i1} - \frac{b_2}{b_1} \sum_{i=1}^{m-1} t_{i1} - \sum_{i=1}^{m-1} I_{i2}, 0\right\} \tag{4.25}$$

The total idle time on machine 2 is:

$$\begin{aligned}
\sum_{i=1}^m I_{i2} &= \max\left\{\sum_{i=1}^m t_{i1} - \frac{b_2}{b_1} \sum_{i=1}^{m-1} t_{i1}, \sum_{i=1}^{m-1} t_{i1} - \frac{b_2}{b_1} \sum_{i=1}^{m-2} t_{i1}, \dots, \sum_{i=1}^2 t_{i1} - \frac{b_2}{b_1} t_{11}, t_{11}\right\}
\end{aligned} \tag{4.26}$$

In general

$$\sum_{i=1}^m I_{i2} = \max_{1 \leq u \leq m} K_u \tag{4.27}$$

where

$$K_u = \sum_{i=1}^u t_{i1} - \frac{b_2}{b_1} \sum_{i=1}^{u-1} t_{i1}.$$

4.4.2 Three Machine LFP

Figure 4.2 illustrates the use of a three machine linear flow shop. The idle time immediately before job 1 starts on machine 3 is:

$$I_{13} = t_{11} + t_{12} = \left[1 + \frac{b_2}{b_1}\right]t_{11} = \sum_{j=1}^2 t_{1j} \quad (4.28)$$

$$I_{23} = \max\left\{\sum_{i=1}^2 t_{i2} + \sum_{i=1}^2 I_{i2} - I_{13} - \frac{b_3}{b_1} t_{11}, 0\right\} \quad (4.29)$$

The sum of (4.28) and (4.29) yields:

$$I_{13} + I_{23} = \max\left\{\sum_{i=1}^2 t_{i2} + \sum_{i=1}^2 I_{i2} - \frac{b_3}{b_1} t_{11}, \sum_{j=1}^2 t_{1j}\right\} \quad (4.30)$$

$$I_{33} = \max\left\{\sum_{i=1}^3 t_{i2} + \sum_{i=1}^3 I_{i2} - \frac{b_3}{b_1} \sum_{i=1}^2 t_{i1} - \sum_{i=1}^2 I_{i3}, 0\right\} \quad (4.31)$$

The sum of (4.30) and (4.31) yields:

$$\begin{aligned} \sum_{i=1}^3 I_{i3} &= \max\left\{\sum_{i=1}^3 t_{i2} + \sum_{i=1}^3 I_{i2} - \frac{b_3}{b_1} \sum_{i=1}^2 t_{i1}, \sum_{i=1}^2 I_{i3}\right\} \\ &= \max\left\{\sum_{i=1}^3 t_{i2} + \sum_{i=1}^3 I_{i2} - \frac{b_3}{b_1} \sum_{i=1}^2 t_{i1}, \sum_{i=1}^2 t_{i2} + \right. \\ &\quad \left. \sum_{i=1}^2 I_{i2} - \frac{b_3}{b_1} t_{11}, \sum_{j=1}^2 t_{1j}\right\} \end{aligned} \quad (4.32)$$

The idle time immediately before the last job starts on machine 3 is:

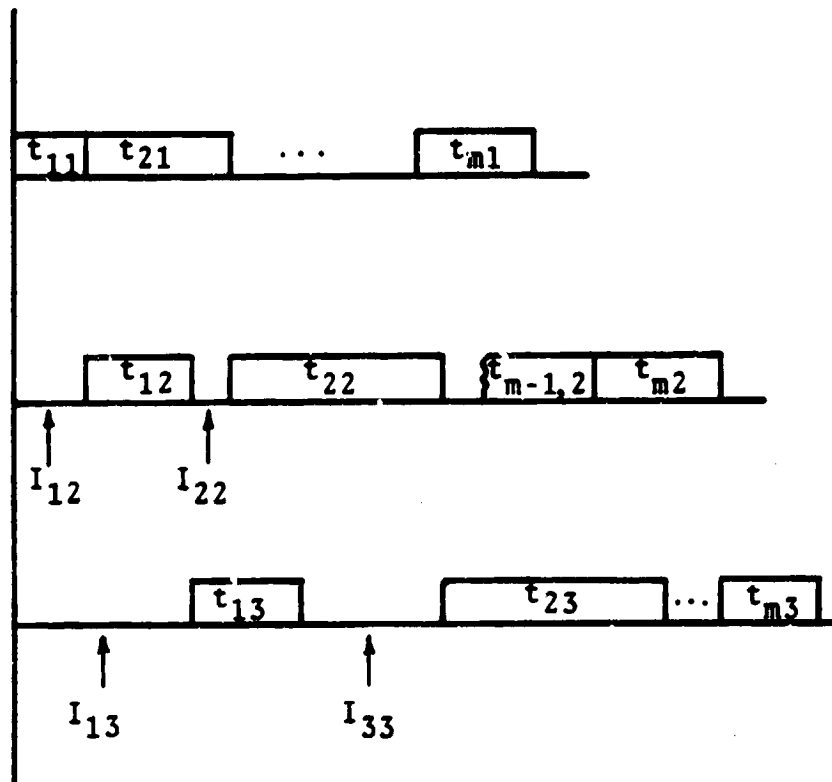


Figure 4.2. Idle Time in a Three Machine Linear Flow Shop

$$I_{m3} = \max\left\{\sum_{i=1}^m t_{i2} + \sum_{i=1}^m I_{i2} - \frac{b_3}{b_1} \sum_{i=1}^{m-1} t_{i1} - \sum_{i=1}^{m-1} I_{i3}, 0\right\} \quad (4.33)$$

The total idle time on machine 3 is:

$$\begin{aligned} \sum_{i=1}^m I_{i3} &= \max\left\{\sum_{i=1}^m t_{i2} + \sum_{i=1}^m I_{i2} - \frac{b_3}{b_1} \sum_{i=1}^{m-1} t_{i1}, \sum_{i=1}^{m-1} t_{i2} + \right. \\ &\quad \left. \sum_{i=1}^{m-1} I_{i2} - \frac{b_3}{b_1} \sum_{i=1}^{m-2} t_{i1}, \dots, \sum_{i=1}^2 t_{i2} + \sum_{i=1}^2 I_{i2} - \frac{b_3}{b_1} t_{11}, \right. \\ &\quad \left. \sum_{j=1}^2 t_{1j}\right\} \\ &= \max\left\{\frac{b_2}{b_1} \sum_{i=1}^m t_{i1} - \frac{b_3}{b_1} \sum_{i=1}^{m-1} t_{i1} + \sum_{i=1}^m I_{i2}, \frac{b_2}{b_1} \sum_{i=1}^{m-1} t_{i1} - \right. \\ &\quad \left. \frac{b_3}{b_1} \sum_{i=1}^{m-2} t_{i1} + \sum_{i=1}^{m-1} I_{i2}, \dots, \frac{b_2}{b_1} \sum_{i=1}^2 t_{i1} - \frac{b_3}{b_1} t_{11} + \right. \\ &\quad \left. \sum_{i=1}^2 I_{i2}, \sum_{j=1}^2 t_{1j}\right\} \quad (4.34) \end{aligned}$$

In general

$$\sum_{i=1}^m I_{i3} = \max_{1 \leq v \leq m} k_v \quad (4.35)$$

where

$$k_v = \frac{b_2}{b_1} \sum_{i=1}^v t_{i1} - \frac{b_3}{b_1} \sum_{i=1}^{v-1} t_{i1} + \sum_{i=1}^v I_{i2}$$

4.4.3 The General Case

The general term for the idle time I_{ij} is:

$$I_{ij} = \max\left(\frac{b_{j-1}}{b_1} \sum_{k=1}^i t_{k1} - \frac{b_j}{b_1} \sum_{k=1}^{i-1} t_{k1} + \sum_{k=1}^i I_{kj-1} - \sum_{k=1}^{i-1} I_{kj}, 0\right) \quad (4.36)$$

The total idle time on machine j is:

$$\begin{aligned} \sum_{i=1}^m I_{ij} = & \max\left(\frac{b_{j-1}}{b_1} \sum_{i=1}^m t_{i1} - \frac{b_j}{b_1} \sum_{i=1}^{m-1} t_{i1} + \sum_{i=1}^m I_{i,j-1} \frac{b_{i-1}}{b_1} \right. \\ & \left. \sum_{i=1}^{m-1} t_{i1} - \frac{b_j}{b_1} \sum_{i=1}^{m-2} t_{i1} + \sum_{i=1}^m I_{i,j-1}, \dots, \frac{b_{j-1}}{b_1} \right. \\ & \left. \sum_{i=1}^2 t_{i1} + \sum_{i=1}^2 I_{i,j-1}, \sum_{k=1}^{i-1} t_{1k}\right) \quad (4.37) \end{aligned}$$

4.5 Makespan (MS) in Linear Flow Shop

Makespan is defined as machining time of machine n (M_n) plus the idle time that is generated on machine n (I_{in}). Using equations (4.14) and (4.37) yields:

$$\begin{aligned} MS = M_n + \sum_{i=1}^m I_{in} = & \frac{b_n}{b_1} \sum_{i=1}^m t_{i1} + \max\left(\frac{b_{n-1}}{b_1} \sum_{i=1}^m t_{i1} - \frac{b_n}{b_1} \sum_{i=1}^{m-1} t_{i1} + \sum_{i=1}^m I_{i,n-1}, \frac{b_{n-1}}{b_1} \sum_{i=1}^{m-1} t_{i1} - \frac{b_n}{b_1} \sum_{i=1}^{m-2} t_{i1} + \sum_{i=1}^{m-1} I_{i,n-1}, \dots, \frac{b_{n-1}}{b_1} \sum_{i=1}^2 t_{i1} - \frac{b_n}{b_1} t_{11} + \sum_{i=1}^2 I_{i,n-1}, \sum_{k=1}^{n-1} t_{1k}\right) \quad (4.38) \end{aligned}$$

Substituting equation (4.2) into (4.38) yields:

$$\begin{aligned}
 MS = & b_n \sum_{i=1}^m n_i + \max \left\{ b_{n-1} \sum_{i=1}^m n_i - b_n \sum_{i=1}^{m-1} n_i + \sum_{i=1}^m I_{i,n-1} \cdot \right. \\
 & b_{n-1} \sum_{i=1}^{m-1} n_i - b_n \sum_{i=1}^{m-2} n_i + \sum_{i=1}^{m-1} I_{i,n-1}, \dots, \\
 & \left. b_{n-1} \sum_{i=1}^2 n_i - b_n n_1 + \sum_{i=1}^2 I_{i,n-1}, \sum_{l=1}^{n-1} t_{1l} \right\} \quad (4.39)
 \end{aligned}$$

4.6 Bounds for Performance Time in a Linear Flow Shop

4.6.1 Lower Bound (\underline{L})

The lower bound is equal to the total performance time of all the m jobs over all n machines, or equal to the total machining time of all n machines. Using equations (4.11) or (4.19) yields:

$$\underline{L} = \sum_{i=1}^m (n_i \sum_{j=1}^n b_j) = \sum_{j=1}^n (b_j \sum_{i=1}^m n_i) \quad (4.40)$$

4.6.2 Upper Bound (\bar{U})

The upper bound is the lower bound plus the idle time that is generated in the process. Using equations (4.37) and (4.40) yields:

$$\begin{aligned}
U = \underline{L} + \sum_{j=2}^n \left(\sum_{i=1}^m I_{ij} \right) &= \sum_{i=1}^m \left(n_i \sum_{j=1}^n b_j \right) + \sum_{j=2}^n \left\{ \max \left[\frac{b_{j-1}}{b_1} \right. \right. \\
&\sum_{i=1}^m t_{i1} - \frac{b_j}{b_1} \sum_{i=1}^{m-1} t_{i1} + \sum_{i=1}^m I_{i,j-1}, \frac{b_{j-1}}{b_1} \sum_{i=1}^{m-1} t_{i1} - \\
&\frac{b_j}{b_1} \sum_{i=1}^{m-2} t_{i1} + \sum_{i=1}^{m-1} I_{i,j-1}, \dots, \frac{b_{j-1}}{b_1} \sum_{i=1}^2 t_{i1} - \frac{b_j}{b_1} t_{11} + \\
&\left. \left. \sum_{i=1}^2 I_{i,j-1}, \sum_{\ell=1}^{i-1} t_{1\ell} \right] \right\} \quad (4.41)
\end{aligned}$$

CHAPTER 5
OPTIMALITY IN LINEAR FLOW SHOP SCHEDULING

This chapter describes three performance measurements of the linear flow shop scheduling: (a) Mean lateness,
(b) Mean flow time,
(c) Waiting time.

It is proved that the shortest processing time sequencing yields the optimal sequence that minimizes the given performance measurements. A numerical example is given to show how the shortest processing time sequencing yields the minimum mean lateness.

5.1 Lateness in a Linear Flow Shop

Recall that job lateness is defined as $L_i = C_i - d_i$, or the discrepancy between the due date of a job and its completion time. The objective of every production plant is to minimize the lateness in completing the orders because it is involved with penalty.

Theorem 1

The mean lateness in a two machine flow shop problem where the job times on the machines are a linear function

of the batch sizes ($t_{ij} = b_j n_i$, $b_j \geq 1$) is minimized by shortest processing time (SPT) sequencing.

Proof

The proof has two steps.

- (a) Consider a sequence S that is not a SPT sequence. That is, somewhere in S there must exist at least one pair of adjacent jobs, i and $i+1$, with $i+1$ following i , such that $t_i > t_{i+1}$. Now construct a new sequence, S' , in which jobs i and $i+1$ are interchanged in sequence and all other jobs are not changed. The situation is depicted in Figure 5.1, where T_A denotes the point in time at which job i begins in sequence S and at which job $i+1$ begins in Sequence S' . A , denotes the set of jobs that precede jobs i and $i+1$ in both schedules and B , denotes the set of jobs that follow i and $i+1$ in both schedules.

The processing times t_{i1}, t_{i2} and the due date d_i are given and completion time C_i and the lateness L_i of the job i are computed.

t_{i1} = processing time of job i on machine 1, computed
by $t_{i1} = b_1 n_i$

t_{i2} = processing time of job i on machine 2, computed
by $t_{i2} = b_2 n_i$

d_i = due date of job i

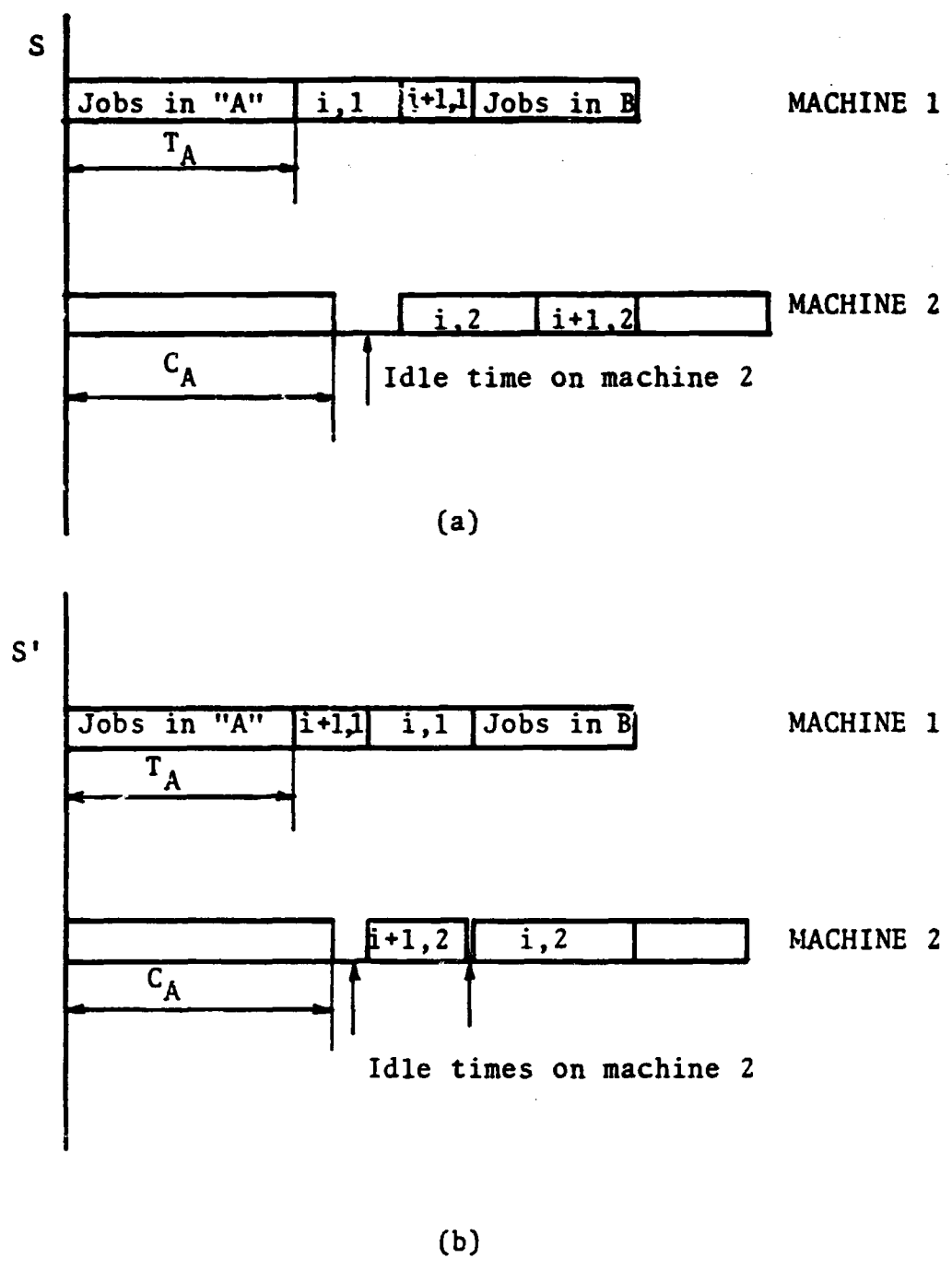


Figure 5.1 A Pairwise Interchange of Adjacent Jobs
 (a) Sequence S
 (b) Sequence S'

C_i = completion time of job i

L_i = lateness of job i , computed by $L_i = C_i - d_i$

C_A = completion time of the last job in set A .

The proof is shown for $\sum_{i=1}^m L_i$ (sum of the lateness over all the jobs), as the mean lateness $\bar{L} = \frac{1}{m} \sum_{i=1}^m L_i$.

Using equation (4.1) the processing times t_{i1} and t_{i2} are computed by:

$$t_{i1} = b_1 n_i \quad (5.1)$$

$$t_{i2} = b_2 n_i \quad (5.2)$$

From equation (5.1)

$$n_i = \frac{t_{i1}}{b_1} \quad (5.3)$$

Substituting equation (5.3) into (5.2) yields:

$$t_{i2} = \frac{b_2}{b_1} t_{i1} \quad (5.4)$$

Through all the proof steps it is considered that:

$$b = \frac{b_2}{b_1}$$

and

$$t_{i2} = b t_{i1} \quad (5.5)$$

The total lateness is:

$$\sum_{k=1}^m L_k = L_A + L_i + L_{i+1} + L_B \quad (5.6)$$

where: L_A is the total lateness of the jobs in set A.

L_i is the lateness of job i

L_{i+1} is the lateness of job $i+1$

L_B is the total lateness of the jobs in set B.

The completion times of jobs i and $i+1$ in sequence S are as follows:

$$C_i(S) = \max\{T_A + t_{i,1}, C_A\} + t_{i,2} = \max\{T_A + t_{i,1}, C_A\} + bt_{i,1} \quad (5.7)$$

$$\begin{aligned} C_{i+1}(S) &= \max\{T_A + t_{i,1} + t_{i+1}, C_i(S)\} + t_{i+1,2} \\ &= \max\{T_A + t_{i,1} + t_{i+1,1}, C_i(S)\} + bt_{i+1,1} \end{aligned} \quad (5.8)$$

Using equation (5.6):

$$\begin{aligned} \sum_{k=1}^m L_k(S) &= L_A + \max\{T_A + t_{i,1}, C_A\} + bt_{i,1} - d_i + \\ &\quad \max\{T_A + t_{i,1} + t_{i+1,1}, C_i(S)\} + bt_{i+1,1} - \\ &\quad d_{i+1} + L_B(S) \end{aligned} \quad (5.9)$$

The completion times of jobs $i+1$ and i in sequence S' are as follows:

$$C_{i+1}(S') = \max\{T_A + t_{i+1,1}, C_A\} + bt_{i+1,1} \quad (5.10)$$

$$C_i(S') = \max\{T_A + t_{i+1,1} + t_{i,1}, C_{i+1}(S')\} + bt_{i,1} \quad (5.11)$$

Using equation (5.6):

$$\begin{aligned} \sum_{k=1}^m L_k(S') &= L_A + \max\{T_A + t_{i+1,1}, C_A\} + bt_{i+1,1} - d_{i+1} + \\ &\quad \max\{T_A + t_{i+1,1} + t_{i,1}, C_{i+1}(S')\} + bt_{i,1} - \\ &\quad d_i + L_B(S') \end{aligned} \quad (5.12)$$

Let's define P and Q as:

$$P = \max\{T_A + t_{i,1}, C_A\} \quad (5.13)$$

$$Q = \max\{T_A + t_{i+1,1}, C_A\} \quad (5.14)$$

Then

$$P \geq Q \quad \text{as} \quad t_{i,1} > t_{i+1,1}$$

Substituting equation (5.13) into (5.7) yields:

$$C_i(S) = P + bt_{i,1} \quad (5.14)$$

Substituting equation (5.14) into (5.10) yields:

$$C_{i+1}(S') = Q + bt_{i+1,1} \quad (5.15)$$

Then

$$C_i(S) > C_{i+1}(S) \text{ as } P \geq Q \text{ and } bt_{i,1} > bt_{i+1,1}, b \geq 1$$

Let's define R and V as:

$$R = \max\{T_A + t_{i,1} + t_{i+1,1}, C_i(S)\} \quad (5.16)$$

$$V = \max\{T_A + t_{i+1,1} + t_{i,1}, C_{i+1}(S')\} \quad (5.17)$$

It is clear that $R \geq V$ as $C_i(S) > C_{i+1}(S')$

Substituting equations(5.13) and (5.16) into (5.9) yields:

$$\begin{aligned} \sum_{k=1}^m L_k(S) &= L_A + P + bt_{i,1} - d_i + R + bt_{i+1,1} - d_{i+1} \\ &+ L_B(S) \end{aligned} \quad (5.18)$$

Substituting equations (5.14) and (5.17) into (5.12) yields:

$$\begin{aligned} \sum_{k=1}^m L_k(S') &= L_A + Q + bt_{i+1,1} - d_{i+1} + V + bt_{i,1} - d_i \\ &+ L_B(S') \end{aligned} \quad (5.19)$$

The result of comparing equations (5.18) and (5.19) is:

$$\begin{aligned} \sum_{k=1}^m L_k(S) - \sum_{k=1}^m L_k(S') &= L_A + P + bt_{i,1} - d_i + R + bt_{i+1,1} \\ &- d_{i+1} + L_B(S) - L_A - Q - bt_{i+1,1} \\ &+ d_{i+1} - V - bt_{i,1} + d_i - L_B(S') \\ &= P + R - Q - V + L_B(S) - L_B(S') \end{aligned} \quad (5.20)$$

Equation (5.20) can be divided into two sums as follows:

$$\sum_{k=1}^m L_k(S) - \sum_{k=1}^m L_k(S') = (P + R - Q - V) + [L_B(S) - L_B(S')] \quad (5.21)$$

The first sum $(P+R-Q-V)$ represents the difference in lateness between sequence S and S' for the set of jobs which are in set A and jobs i and $i+1$. The second sum $[L_B(S) - L_B(S')]$ represents the difference in lateness between sequence S and S' for the jobs that are in set B. The first sum is independent of the second sum but the second sum is dependent on the first sum.

Observing the first sum shows that the total lateness of sequence S is greater or equal to the total lateness of sequence S' as $P \geq Q$ and $R \geq V$, it means that the first job in Set B in sequence S will start at the same time or later than the same job in sequence S' . Concludes that $L_B(S) \geq L_B(S')$ as the jobs in set B are the same for sequence S and S' but the starting point is different.

It is shown that

$$\sum_{k=1}^m L_k(S) \geq \sum_{k=1}^m L_k(S') \text{ as } P \geq Q, R \geq V, L_B(S) \geq L_B(S')$$

$$\text{and } P \geq 0, R \geq 0, Q \geq 0, V \geq 0.$$

(b) In part (a) of the proof it is shown that any interchange of a pair of adjacent jobs where a shorter job is following a longer job, can improve the total lateness or leave it unchanged.

The second part of the proof shows that in the last interchange of a pair of adjacent jobs (going from a non-SPT sequence to a SPT sequence) a strict improvement can be made in the total lateness, concluding that a SPT sequence produces the minimum total lateness. Part b of the proof has two phases. The first phase uses contradiction. Let us define a sequence S as a finite set of non-negative numbers such as

$$t_{i,1}, t_{i+1,1}, t_{i+2,1}.$$

Sequence S is a non-SPT sequence in such a way that there is somewhere in the sequence only one pair of adjacent jobs i and i+1, with i+1 following i, such that $t_{i,1} > t_{i+1,1}$. Assume that sequence S is an "optimal" sequence. Construct a new sequence S', in which jobs i and i+1 are interchanged in sequence (so that you get a SPT sequence) and show that a strict improvement can be made in this "optimal" sequence. Therefore, the conclusion is (based on part a of the proof) that it is impossible for a non-SPT sequence to be optimal.

The second phase involves a constructive proof that uses the method of mathematical induction to show that a SPT sequence minimizes the mean lateness.

In this part of the proof the jobs which precede the pair of jobs i and $i+1$ that are being interchanged, are not considered, as the total lateness of the jobs in sequence S is the same as the total lateness of the job in sequence S' . The proof is shown for the case of two jobs where only jobs i and $i+1$ are in the process, for three jobs; $i, i+1, i+2$ and for four jobs; $i, i+1, i+2, i+3$. It is assumed it is true for m jobs $i, i+1, i+2, \dots, i_{m-1}$; then it is proved it is true for $m+1$ jobs.

Two Jobs Sequence

The situation is depicted in Figure 5.2 where C_1 and C_2 are the completion time of jobs 1 and 2 respectively. The terms for C_1 and C_2 in sequence S are:

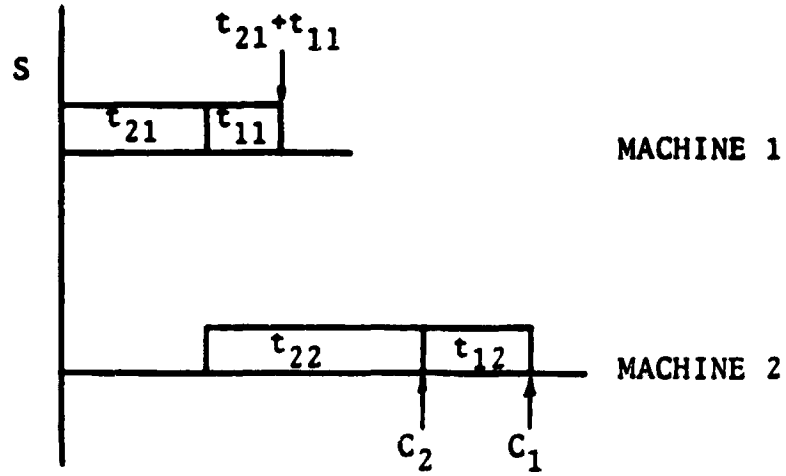
$$C_2(S) = t_{21} + t_{22} = t_{21} + bt_{21} = (1+b)t_{21} \quad (5.22)$$

$$C_1(S) = \max\{C_2, t_{21} + t_{11}\} + t_{12} = \max\{(1+b)t_{21}, t_{21} + t_{11}\} + bt_{11} \quad (5.23)$$

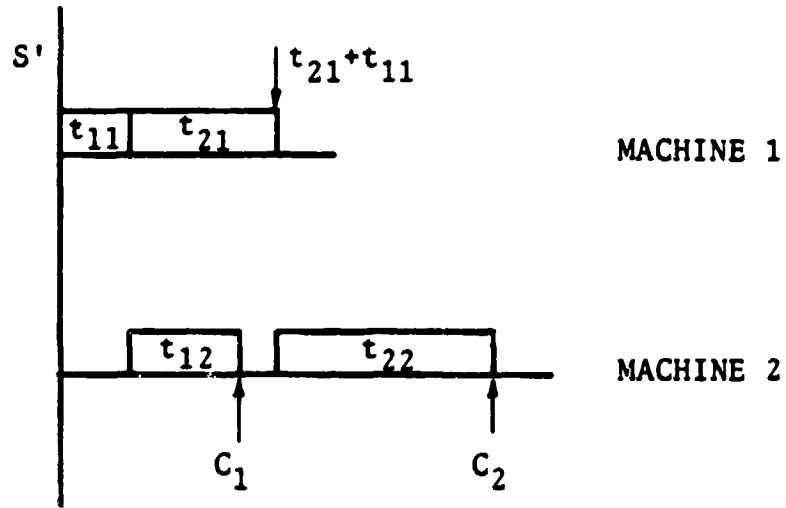
Using equations (5.22) and (5.23), the lateness is:

$$L_2(S) = (1+b)t_{21} - d_2 \quad (5.24)$$

$$L_1(S) = \max\{(1+b)t_{21}, t_{21} + t_{11}\} + bt_{11} - d_1 \quad (5.25)$$



(a)



(b)

Figure 5.2. A Two Job Linear Flow Shop
 (a) Non-SPT Sequence
 (b) SPT Sequence

Using equations (5.24) and (5.25), the total lateness of sequence S is:

$$\sum_{i=1}^2 L_i(S) = (1+b)t_{21} - d_2 + \max\{(1+b)t_{21}, t_{21} + t_{11}\} + bt_{11} - d_1 \quad (5.26)$$

The completion times in sequence S' are:

$$C_1(S') = t_{11} + t_{12} = t_{11} + bt_{11} = (1+b)t_{11} \quad (5.27)$$

$$C_2(S') = \max\{C_1, t_{11} + t_{21}\} + t_{22} = \max\{(1+b)t_{11}, t_{11} + t_{21}\} + bt_{21} \quad (5.28)$$

Using equations (5.27) and (5.28), the lateness is:

$$L_1(S') = C_1 - d_1 = (1+b)t_{11} - d_1 \quad (5.29)$$

$$L_2(S') = C_2 - d_2 = \max\{(1+b)t_{11}, t_{11} + t_{21}\} + bt_{21} - d_2 \quad (5.30)$$

Using equations (5.29) and (5.30), the total lateness of sequence S' is:

$$\sum_{i=1}^2 L_i(S') = (1+b)t_{11} - d_1 + \max\{(1+b)t_{11}, t_{11} + t_{21}\} + bt_{21} - d_2 \quad (5.31)$$

Comparing equations (5.26) and (5.31) yields:

$$\begin{aligned} \sum_{i=1}^2 L_i(S) - \sum_{i=1}^2 L_i(S') &= (1+b)t_{21} - d_2 + \max\{(1+b)t_{21}, t_{21} \\ &+ t_{11}\} + bt_{11} - d_1 - (1+b)t_{11} + d_1 - \max\{(1+b)t_{11}, \\ &t_{11} + t_{21}\} - bt_{21} + d_2 \\ &= t_{21} - t_{11} + \max\{(1+b)t_{21}, t_{21} + t_{11}\} - \max\{(1+b)t_{11}, \\ &t_{11} + t_{21}\} \end{aligned}$$

Observing the terms $\max\{(1+b)t_{21}, t_{21} + t_{11}\}$ and $\max\{(1+b)t_{11}, t_{11} + t_{21}\}$ and given that $t_{21} > t_{11}$ (by definition) it follows that $\max\{(1+b)t_{21}, t_{21} + t_{11}\} > \max\{(1+b)t_{11}, t_{11} + t_{21}\}$ as $(1+b)t_{21} > (1+b)t_{11}$ and $(1+b)t_{21} > t_{21} + t_{11}$, $b \geq 1$

it follows that $\sum_{i=1}^2 L_i(S) > \sum_{i=1}^2 L_i(S')$

It is shown that the SPT sequence gives the minimum mean lateness.

Three Jobs Sequence

The situation is depicted in Figure 5.3. The completion times of the jobs in sequence S are:

$$C_2(S) = t_{21} + t_{22} = t_{21} + bt_{21} = (1+b)t_{21} \quad (5.32)$$

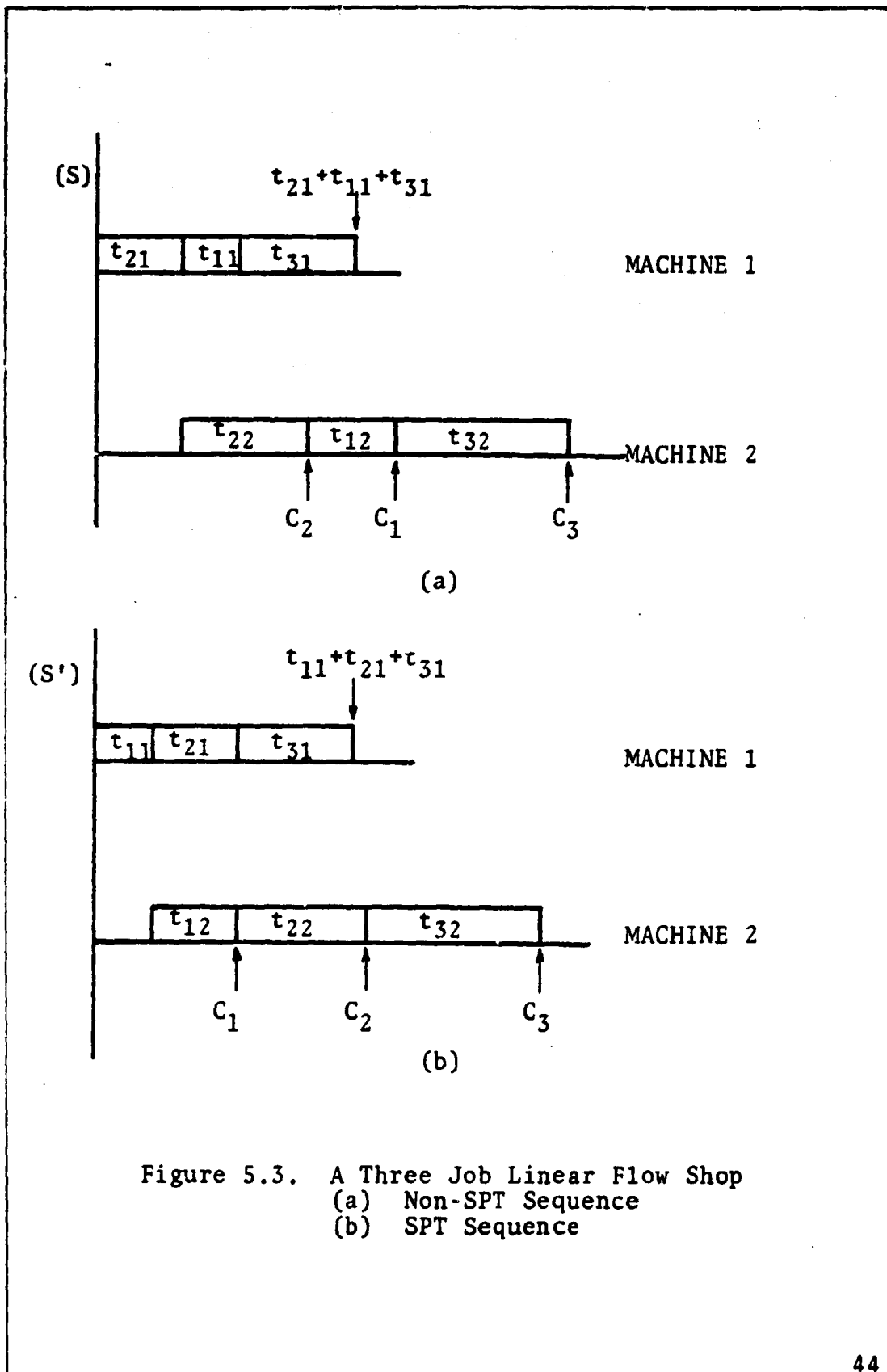


Figure 5.3. A Three Job Linear Flow Shop
 (a) Non-SPT Sequence
 (b) SPT Sequence

$$C_1(S) = \max\{C_2, t_{21} + t_{11}\} + t_{12} = \max\{(1+b)t_{21}, t_{21} + t_{11}\} + bt_{11} \quad (5.33)$$

$$C_3(S) = \max\{C_1, t_{21} + t_{11} + t_{31}\} + t_{32} = \max\{\max[(1+b)t_{21}, t_{21} + t_{11}] + bt_{11}, t_{21} + t_{11} + t_{31}\} + bt_{31} \quad (5.34)$$

Using equations (5.32), (5.33) and (5.34) the lateness is:

$$L_2(S) = C_2 - d_2 \quad (5.35)$$

$$L_1(S) = C_1 - d_1 \quad (5.36)$$

$$L_3(S) = C_3 - d_3 \quad (5.37)$$

Using equations (5.35), (5.36) and (5.37) the total lateness of sequence S is:

$$\sum_{i=1}^3 L_i(S) = (1+b)t_{21} - d_2 + \max\{(1+b)t_{21}, t_{21} + t_{11}\} + bt_{11} - d_1 + \max\{\max[(1+b)t_{21}, t_{21} + t_{11}] + bt_{11}, t_{21} + t_{11} + t_{31}\} + bt_{31} - d_3 \quad (5.38)$$

When observing equation (5.38):

$$\max\{(1+b)t_{21}, t_{21} + t_{11}\} = (1+b)t_{21} \text{ as } t_{21} > t_{11} \text{ and} \\ b \geq 1 \quad (5.39)$$

Substituting equation (5.39) into (5.38) yields:

$$\sum_{i=1}^3 L_i(S) = (1+b)t_{21} - d_2 + (1+b)t_{21} + bt_{11} - d_1 + \\ \max\{(1+b)t_{21} + bt_{11}, t_{21} + t_{11} + t_{31}\} + \\ bt_{31} - d_3 \quad (5.40)$$

The completion times of the jobs in sequence S' are:

$$C_1(S') = t_{11} + t_{12} = t_{11} + bt_{11} = (1+b)t_{11} \quad (5.41)$$

$$C_2(S') = \max\{C_1, t_{11} + t_{21}\} + t_{22} = \max\{(1+b)t_{11}, \\ t_{11} + t_{21}\} + bt_{21} \quad (5.42)$$

$$C_3(S') = \max\{C_2, t_{11} + t_{21} + t_{31}\} + t_{32} = \max\{\max\{(1+b) \\ t_{11}, t_{11} + t_{21}\} + bt_{21}, t_{11} + t_{21} + t_{31}\} \\ + bt_{31} \quad (5.43)$$

Using equations (5.41), (5.42) and (5.43) the lateness is:

$$L_1(S') = C_1 - d_1 \quad (5.44)$$

$$L_2(S') = C_2 - d_2 \quad (5.45)$$

$$L_3(S') = C_3 - d_3 \quad (5.46)$$

Using equations (5.44), (5.45) and (5.46) the total lateness of sequence S' is:

$$\begin{aligned} \sum_{i=1}^3 L_i(S') &= (1+b)t_{11} - d_1 + \max\{(1+b)t_{11}, t_{11} + t_{21}\} + \\ &\quad bt_{21} - d_2 + \max\{\max[(1+b)t_{11}, t_{11} + t_{21}] + \\ &\quad bt_{21}, t_{11} + t_{21} + t_{31}\} + bt_{31} - d_3 \end{aligned} \quad (5.47)$$

Comparing equations (5.40) and (5.47) yields:

$$\begin{aligned} \sum_{i=1}^3 L_i(S) - \sum_{i=1}^3 L_i(S') &= (1+b)t_{21} - d_2 + (1+b)t_{21} + bt_{11} - \\ &\quad d_1 + \max\{(1+b)t_{21} + bt_{11}, t_{21} + t_{11} + t_{31}\} + \\ &\quad bt_{31} - d_3 - (1+b)t_{11} + d_1 - \max\{(1+b)t_{11}, t_{11} \\ &\quad + t_{21}\} + bt_{21} + d_2 - \max\{\max[(1+b)t_{11}, t_{11} + \\ &\quad t_{21}] + bt_{21}, t_{11} + t_{21} + t_{31}\} - bt_{31} + d_3 \\ &= t_{21} + (1+b)t_{21} + \max\{(1+b)t_{21} + bt_{11}, t_{21} + \\ &\quad t_{11} + t_{31}\} - t_{11} - \max\{(1+b)t_{11}, t_{11} + t_{21}\} - \\ &\quad \max\{\max[(1+b)t_{11}, t_{11} + t_{21}] + bt_{21}, t_{11} + \\ &\quad t_{21} + t_{31}\} \end{aligned} \quad (5.48)$$

Let's define X, Y, Z as follows:

$$X = \max\{(1+b)t_{11}, t_{11} + t_{21}\} \quad (5.49)$$

$$Y = \max\{X + bt_{21}, t_{11} + t_{21} + t_{31}\} \quad (5.50)$$

$$Z = \max\{(1+b)t_{21} + bt_{11}, t_{21} + t_{11} + t_{31}\} \quad (5.51)$$

Substituting equations(5.49), (5.50) and (5.51) into (5.48) yields:

$$\sum_{i=1}^3 L_i(S) - \sum_{i=1}^3 L_i(S') = t_{21} + (1+b)t_{21} + Z - t_{11} - X - Y \quad (5.52)$$

$$\text{It is clear that } Y > X \quad (5.53)$$

The relations between Y and Z are:

$$Y = \max\{X + bt_{21}, t_{11} + t_{21} + t_{31}\}$$

$$Z = \max\{(1+b)t_{21} + bt_{11}, t_{11} + t_{21} + t_{31}\}$$

Then compare $X + bt_{21}$ and $(1+b)t_{21} + bt_{11}$ or after reducing the terms compare X and $t_{21} + bt_{11}$. Two cases have to be checked:

I. $X = (1+b)t_{11}$

It follows that $(1+b)t_{11} < t_{21} + bt_{11}$ as $t_{11} < t_{21}$

II. $X = t_{11} + t_{21}$

It follows that $t_{11} + t_{21} \leq t_{21} + bt_{11}$ as

$$t_{11} < t_{21}, b \geq 1$$

$$\text{It is found that } Z \geq Y \quad (5.54)$$

$$\sum_{i=1}^3 L_i(S) - \sum_{i=1}^3 L_i(S') = t_{21} + (1+b)t_{21} - t_{11} + Z - X - Y$$

Using equation (5.54)

Two cases have to be checked for X

I. $X = (1+b)t_{11}$

$$\sum_{i=1}^3 L_i(S) - \sum_{i=1}^3 L_i(S') = t_{21} + (1+b)t_{21} - t_{11} + Z - Y -$$

$$\text{It follows that } \sum_{i=1}^3 L_i(S) > \sum_{i=1}^3 L_i(S') \text{ as } t_{11} < t_{21},$$

$$Z \geq Y$$

II. $X = t_{11} + t_{21}$

$$\sum_{i=1}^3 L_i(S) - \sum_{i=1}^3 L_i(S') = t_{21} + (1+b)t_{21} - t_{11} + Z - Y -$$

$$\text{It follows that } \sum_{i=1}^3 L_i(S) > \sum_{i=1}^3 L_i(S') \text{ as } t_{11} < t_{21},$$

$$Z \geq Y, b \geq 1$$

It is shown that the SPT sequence gives the minimum mean lateness.

Four Job Sequence

The case is illustrated in Figure 5.4. The completion times of the jobs in sequence S are:

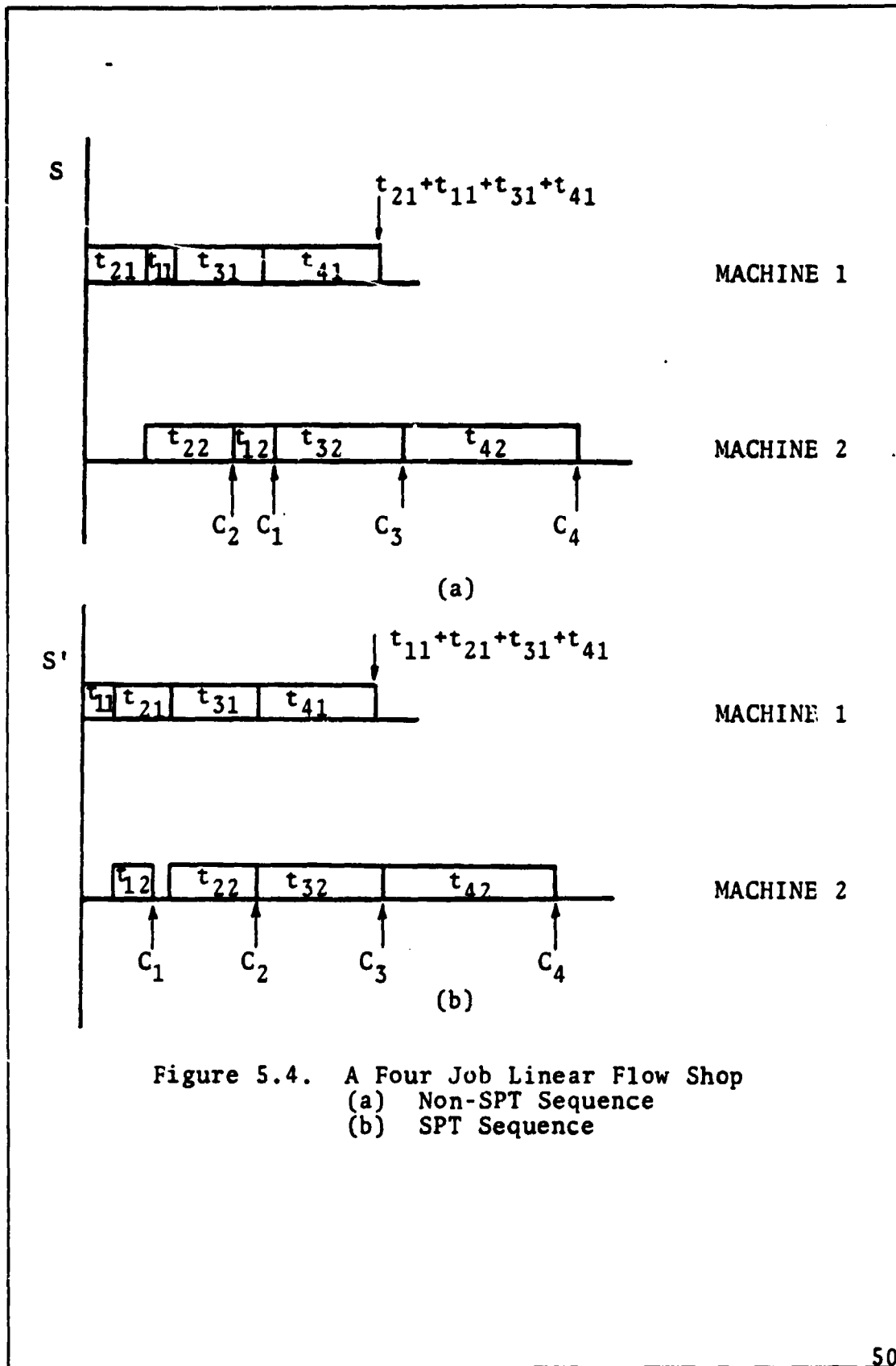


Figure 5.4. A Four Job Linear Flow Shop
 (a) Non-SPT Sequence
 (b) SPT Sequence

$$C_2(S) = t_{21} + t_{22} = t_{21}(1+b) \quad (5.55)$$

$$\begin{aligned} C_1(S) &= \max\{C_2, t_{21} + t_{11}\} + t_{12} \\ &= \max\{(1+b)t_{21}, t_{21} + t_{11}\} + bt_{11} \\ &= (1+b)t_{21} + bt_{11} \quad \text{as } t_{21} > t_{11}, b \geq 1 \quad (5.56) \end{aligned}$$

$$\begin{aligned} C_3(S) &= \max\{C_1, t_{11} + t_{21} + t_{31}\} + bt_{31} \\ &= \max\{(1+b)t_{21} + bt_{11}, t_{11} + t_{21} + t_{31}\} + bt_{31} \\ &= Z + bt_{31} \quad (5.57) \end{aligned}$$

$$\begin{aligned} C_4(S) &= \max\{C_3, t_{11} + t_{21} + t_{31} + t_{41}\} + bt_{41} \\ &= \max\{Z + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\} + bt_{41} \quad (5.58) \end{aligned}$$

Using equations(5.55) through (5.58) the lateness is:

$$L_2(S) = C_2 - d_2 \quad (5.59)$$

$$L_1(S) = C_1 - d_1 \quad (5.60)$$

$$L_3(S) = C_3 - d_3 \quad (5.61)$$

$$L_4(S) = C_4 - d_4 \quad (5.62)$$

Using equations(5.51) and (5.59) through (5.62), the total lateness of sequence S is:

$$\begin{aligned} \sum_{i=1}^4 L_i(S) &= (1+b)t_{21} + (1+b)t_{21} + bt_{11} + Z + bt_{31} + \\ &\quad \max\{Z + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\} + bt_{41} - \\ &\quad \sum_{i=1}^4 d_i \quad (5.63) \end{aligned}$$

The completion times of the jobs in sequence S' are:

$$C_1(S') = t_{11} + t_{12} = t_{11} + bt_{11} = (1+b)t_{11} \quad (5.64)$$

$$\begin{aligned} C_2(S') &= \max\{C_1, t_{11} + t_{21}\} + t_{22} \\ &= \max\{(1+b)t_{11}, t_{11} + t_{21}\} + bt_{21} \end{aligned} \quad (5.65)$$

$$\begin{aligned} C_3(S') &= \max\{\max\{C_1, t_{11} + t_{21}\} + bt_{21}, t_{11} + t_{21} + t_{31}\} \\ &\quad + bt_{31} \end{aligned} \quad (5.67)$$

$$\begin{aligned} C_4(S') &= \max\{\max\{\max\{C_1, t_{11} + t_{21}\} + bt_{21}, t_{11} + t_{21} \\ &\quad + t_{31}\} + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\} + bt_{41} \end{aligned} \quad (5.68)$$

Using equations (5.64) through (5.68) the lateness of the jobs in sequence S' is:

$$L_1(S') = C_1 - d_1 \quad (5.69)$$

$$L_2(S') = C_2 - d_2 \quad (5.70)$$

$$L_3(S') = C_3 - d_3 \quad (5.71)$$

$$L_4(S') = C_4 - d_4 \quad (5.72)$$

Using equations (5.49) (5.50) and (5.69) through (5.72), the total lateness of sequence S' is:

$$\begin{aligned} \sum_{i=1}^4 L_i(S') &= (1+b)t_{11} + X + bt_{21} + Y + bt_{31} + \max\{y+bt_{31}, \\ &\quad t_{11} + t_{21} + t_{31} + t_{41}\} + bt_{41} - \sum_{i=1}^4 d_i \end{aligned} \quad (5.73)$$

Comparing equations (5.63) and (5.73) yields:

$$\begin{aligned} \sum_{i=1}^4 L_i(S) - \sum_{i=1}^4 L_i(S') &= (1+b)t_{21} + (1+b)t_{21} + bt_{11} + Z + \\ &bt_{31} + \max\{Z + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\} + bt_{41} - \\ &\sum_{i=1}^4 d_i - (1+b)t_{11} - X - bt_{21} - Y - bt_{31} - \max\{Y + \\ &bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\} - bt_{41} + \sum_{i=1}^4 d_i \end{aligned}$$

Reducing the terms brings to:

$$\begin{aligned} \sum_{i=1}^4 L_i(S) - \sum_{i=1}^4 L_i(S') &= (2+b)t_{21} + Z + \max\{Z + bt_{31}, t_{11} + \\ &t_{21} + t_{31} + t_{41}\} - t_{11} - X - Y - \max\{Y + bt_{31}, \\ &t_{11} + t_{21} + t_{31} + t_{41}\} \end{aligned}$$

Using equation (5.54) there are two sets of terms to compare

I. $(2+b)t_{21}$ and $t_{11} + X$

Check the two cases of X

1. $X = (1+b)t_{11}$

Compare the terms $(2+b)t_{21}$ and $t_{11} + (1+b)t_{11}$

It follows that $(2+b)t_{21} > t_{11} + (1+b)t_{11}$ as

$t_{11} < t_{21}$

$$2. X = t_{11} + t_{21}$$

Compare the terms $(2+b)t_{21}$ and $t_{11} + t_{11} + t_{21}$

It follows that $(2+b)t_{21} > t_{11} + t_{11} + t_{21}$

The result is that $(2+b)t_{21} > t_{11} + X$

II. Let's define Y_1 and Z_1 as follows:

$$Z_1 = Z + \max\{Z + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\}$$

$$Y_1 = Y + \max\{Y + bt_{31}, t_{11} + t_{21} + t_{31} + t_{41}\}$$

Using equation (5.54) it is clear that:

$$Z_1 \geq Y_1.$$

It follows again that:

$$\sum_{i=1}^4 L_i(S) > \sum_{i=1}^4 L_i(S')$$

It is shown that the SPT sequence brings to minimum the mean lateness.

The Inductive Step

After proving the theorem for the cases of two jobs, three jobs and four jobs, and by using the method of the mathematical induction; assume it is true for the case of m jobs and prove that it is true for the case of $m+1$ jobs.

In general: the terms for the completion times of the jobs are as follows:

$$C_1 = (1+b)t_{11}$$

$$C_2 = \max\{C_1, t_{11} + t_{21}\} + bt_{21}$$

$$C_3 = \max\{C_1, t_{11} + t_{21} + t_{31}\} + bt_{31}$$

⋮

$$C_m = \max\{C_{m-1}, \sum_{i=1}^m t_{i1}\} + bt_{m1}$$

The total lateness is:

$$\sum_{i=1}^m L_i = \sum_{i=1}^m C_m - \sum_{i=1}^m d_i$$

Given that $t_{11} < t_{21} < \dots < t_{m-1} < t_m$

And proving for the cases $i = 2, 3, 4$ that

$$\sum_i L_i (\text{SPT}) < \sum_i L_i (\text{non-SPT})$$

Assume that

$$\sum_{i=1}^m L_i (\text{SPT}) < \sum_{i=1}^m L_i (\text{non-SPT})$$

It is necessary to check the case of $m+1$ jobs.

For the SPT sequence ($t_{11} < t_{21} < t_{31} < \dots < t_m < t_{m+1}$)

$$C_{m+1}' = \max\{C_m', \sum_{i=1}^{m+1} t_{i1}\} + bt_{m+1,1}$$

$$L_{m+1}' = L_m' + C_{m+1}' - d_{m+1}$$

for the non-SPT sequence ($t_{21} < t_{11} < t_{31} \dots < t_m < t_{m+1}$)

$$C_{m+1} = \max\{C_m, \sum_{i=1}^{m+1} t_{i1}\} + bt_{m+1,1}$$

$$L_{m+1} = L_m + C_{m+1} - d_{m+1}$$

Observing the equations for L_{m+1}' and L_{m+1}

And adding them to $\sum_{i=1}^m L_i$ (SPT) and $\sum_{i=1}^m L_i$ (non-SPT) respectively to get the general result that:

$$\sum_{i=1}^{m+1} L_i \text{ (SPT)} < \sum_{i=1}^{m+1} L_i \text{ (non-SPT)}$$

Summary

This proof has shown that in a two machine LFP problem, the optimal solution is to sequence the jobs by the shortest processing time on the first machine. The proof of optimality constructs the optimal sequence by mathematical induction. Figure 5.5 presents a block diagram that summarizes the steps of the proof.

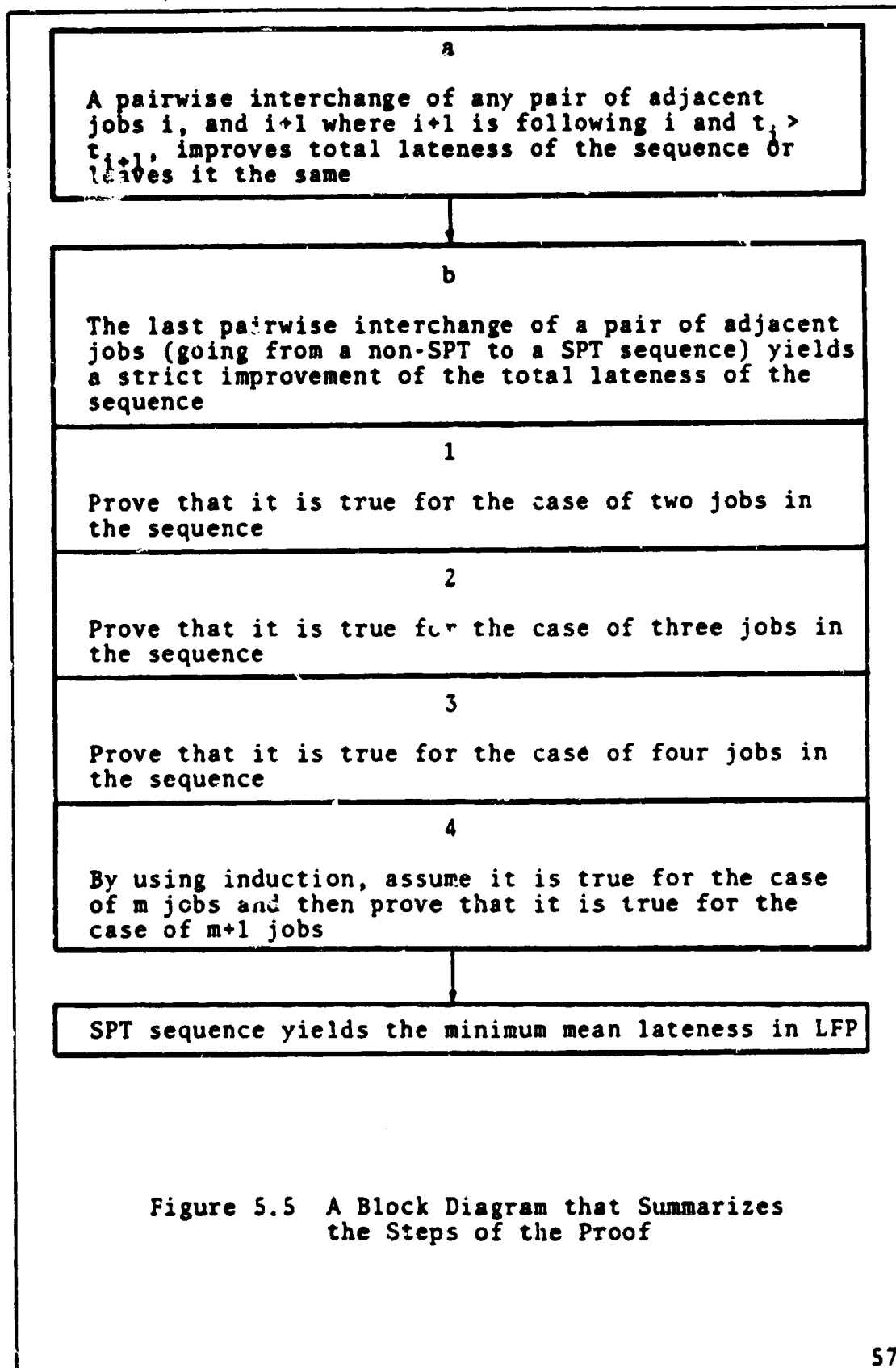


Figure 5.5 A Block Diagram that Summarizes the Steps of the Proof

5.2 Computational Results

A computer program was written to simulate numerical examples that implement the theorem. The example that is presented contains four jobs to be performed on a two machine LFP. The information includes the amount of items in each job, n_i , $i = 1, 2, 3, 4$ and the machining factors, b_j , $j = 1, 2$. The specific figures are in Table 5.1

Table 5.1
Data used in numerical example

	1	2	3	4
n	2	3	5	7
b	1	2	/	/
d	6	9	15	21

The input data contains $4! = 24$ permutations or all possible sequences, and the output is the total lateness of each sequence.

Analysis of the results show that the shortest processing time sequence, yields the minimum lateness, as proved by the theorem.

A sensitivity test is conducted for each sequence, to find out how the total lateness of the sequence is changed, when a pairwise interchange of two adjacent jobs

C

is done. The results of the sensitivity test show that each pairwise interchange of two adjacent jobs where a longer job precedes a shorter job, decreases the total lateness. Figure 5.6 describes a flow diagram that presents the algorithm that the sensitivity test follows. The sensitivity test is constructed as a network, where each block contains the sequence that is tested. The figures outside each block present the total lateness of the given sequence. There are twenty four networks. Each network shows how the lateness is decreasing after conducting a pairwise interchange of two adjacent jobs. The last block of each network presents the SPT sequence which is shown as was proven, to yield the minimum mean lateness. Each network is accompanied with a graph that shows the relations between the lateness and the number of pairwise interchanges of pairs of adjacent jobs. Each path in the network is presented as a different line in the graph. The curves show decreasing values of the total lateness for each interchange.

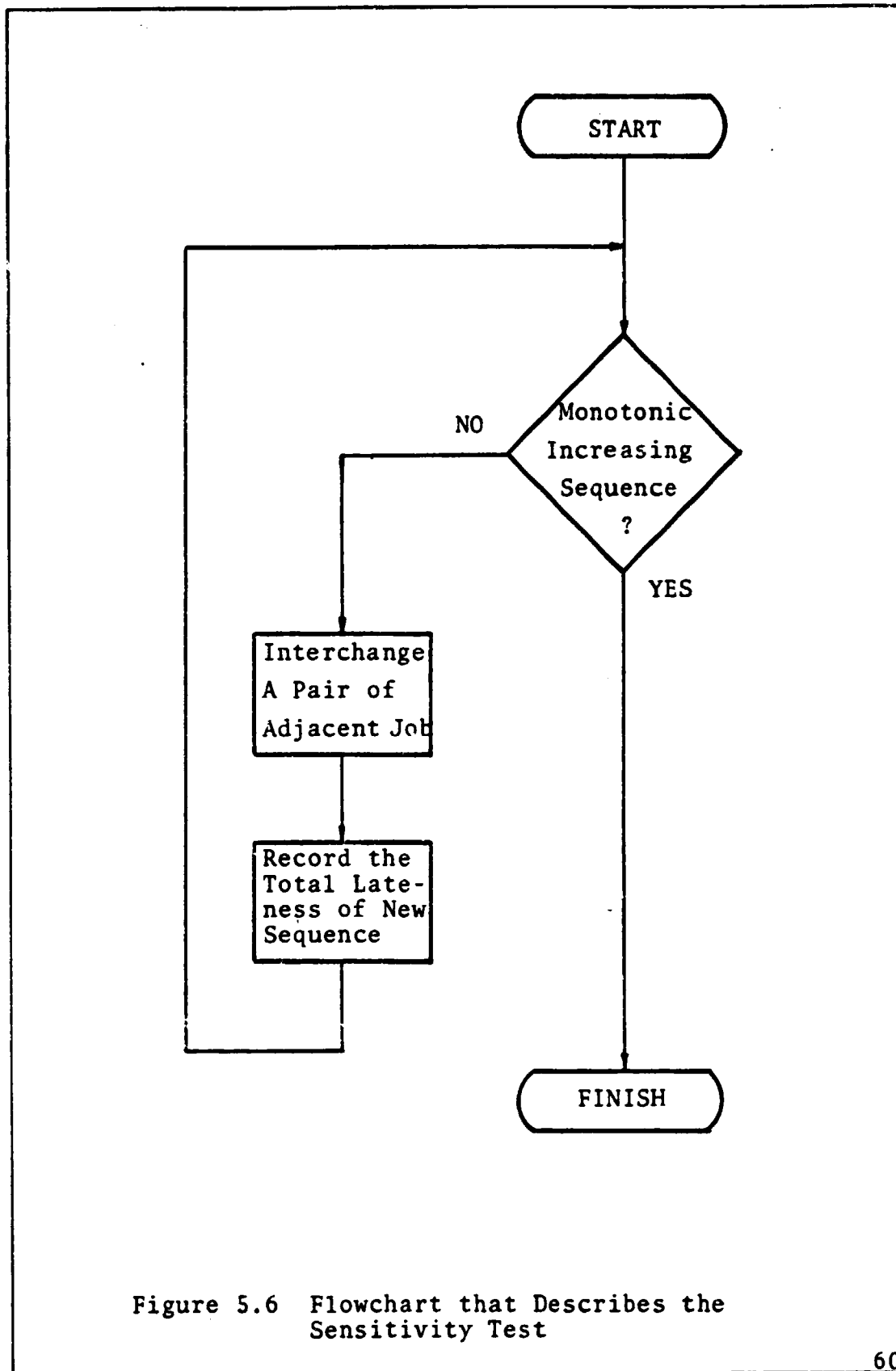
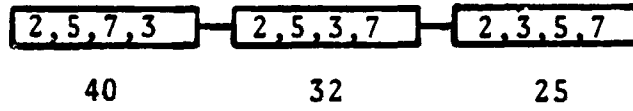
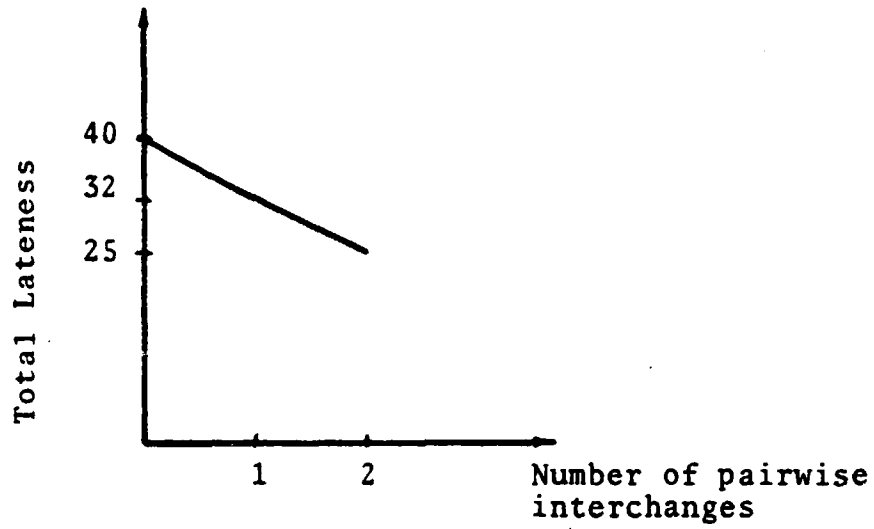


Figure 5.6 Flowchart that Describes the Sensitivity Test

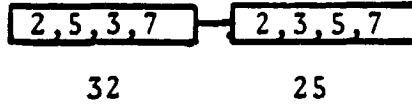


(a)

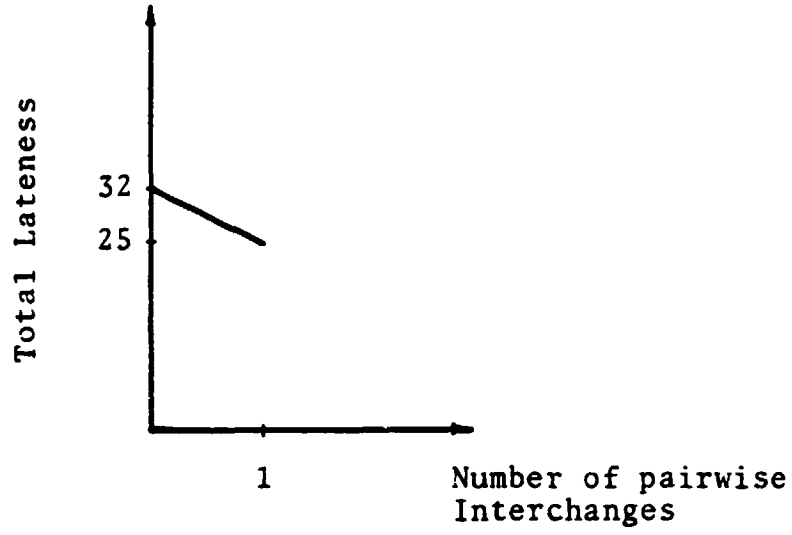


(b)

Figure 5.7 Results of Sensitivity Test for Sequence 1
 (a) Network of sequence 1
 (b) Total lateness curve for sequence 1



(a)



(b)

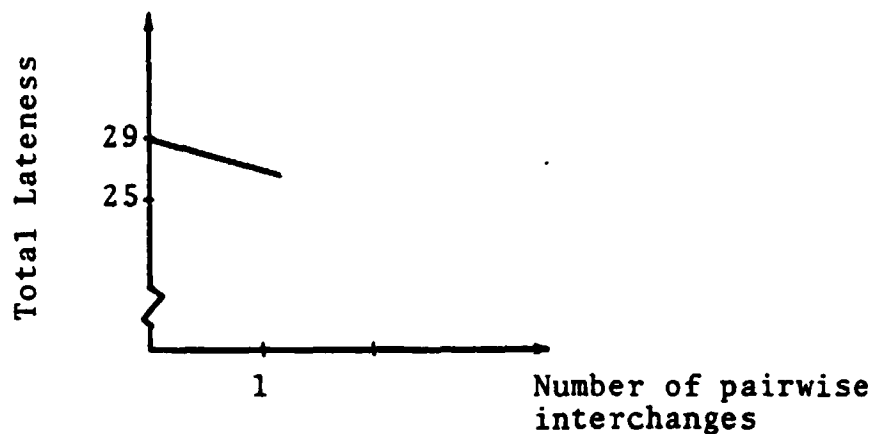
Figure 5.8 Results of Sensitivity Test for Sequence 2
 (a) Network of sequence 2
 (b) Total lateness curve for sequence 2

2,3,5,7 SPT Sequence
25

Figure 5.9 The Optimal Sequence

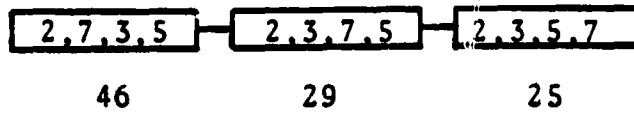
2,3,7,5 — **2,3,5,7**
29 25

(a)

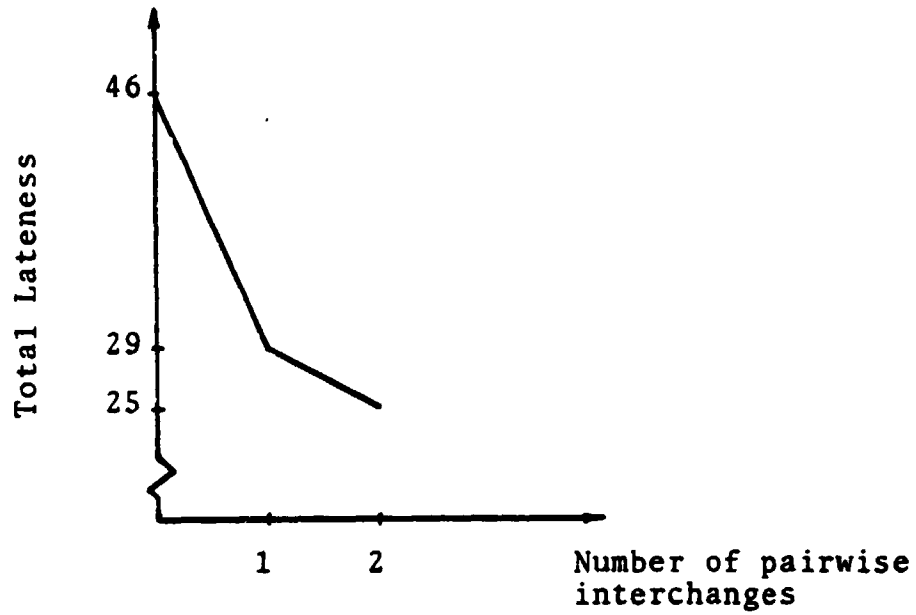


(b)

Figure 5.10 Results of Sensitivity Test for Sequence 4
(a) Network of sequence 4
(b) Total lateness curve for sequence 4

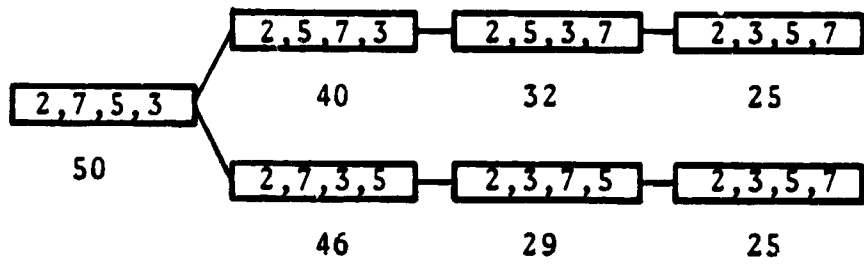


(a)

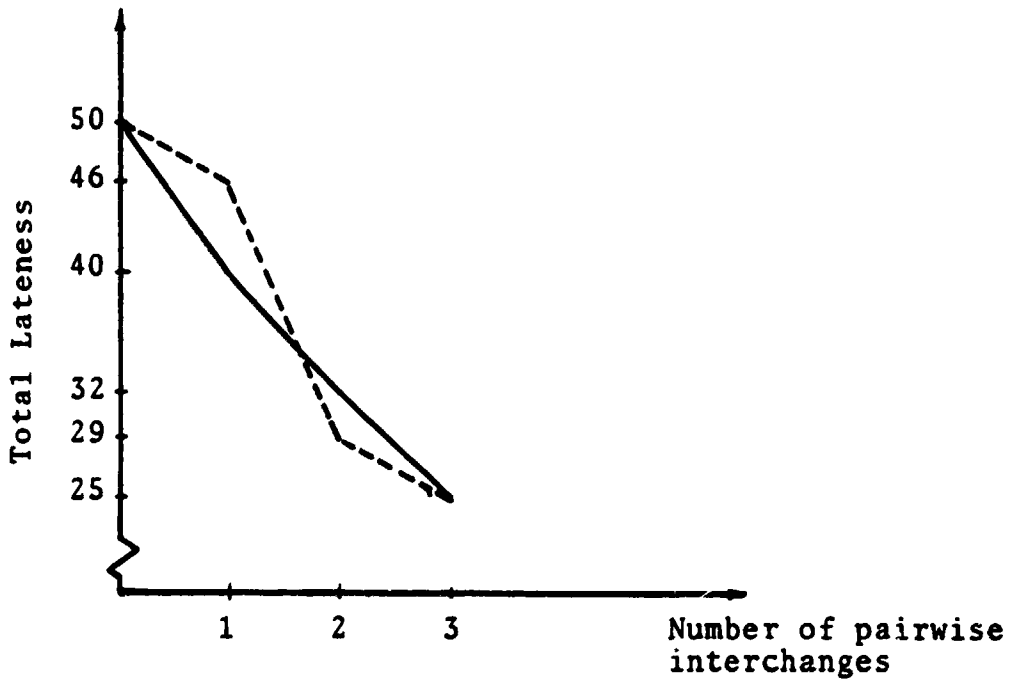


(b)

Figure 5.11 Results of Sensitivity Test for Sequence 5
 (a) Network of sequence 5
 (b) Total lateness curve for sequence 5

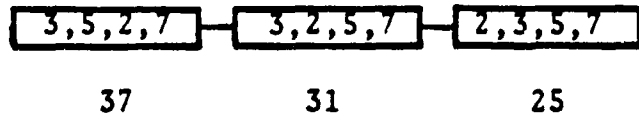


(a)

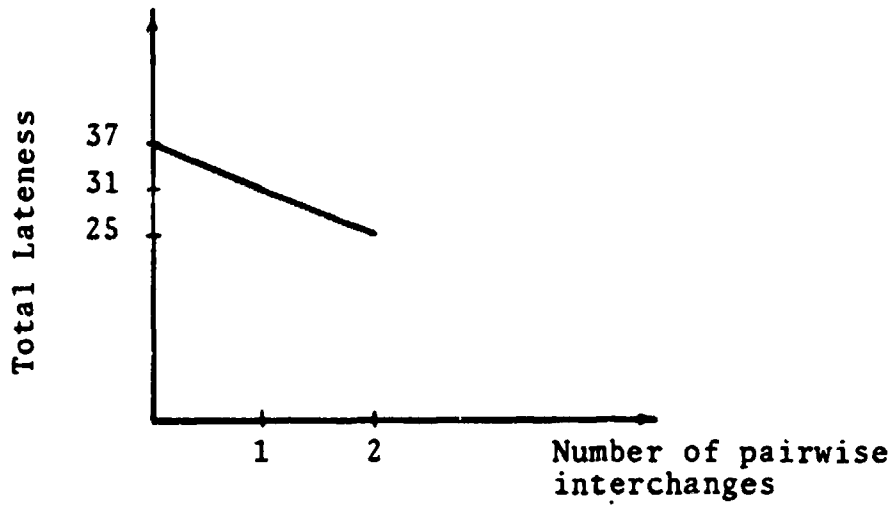


(b)

Figure 5.12 Results of Sensitivity Test for Sequence 6
 (a) Network of sequence 6
 (b) Total lateness curve for sequence 6



(a)



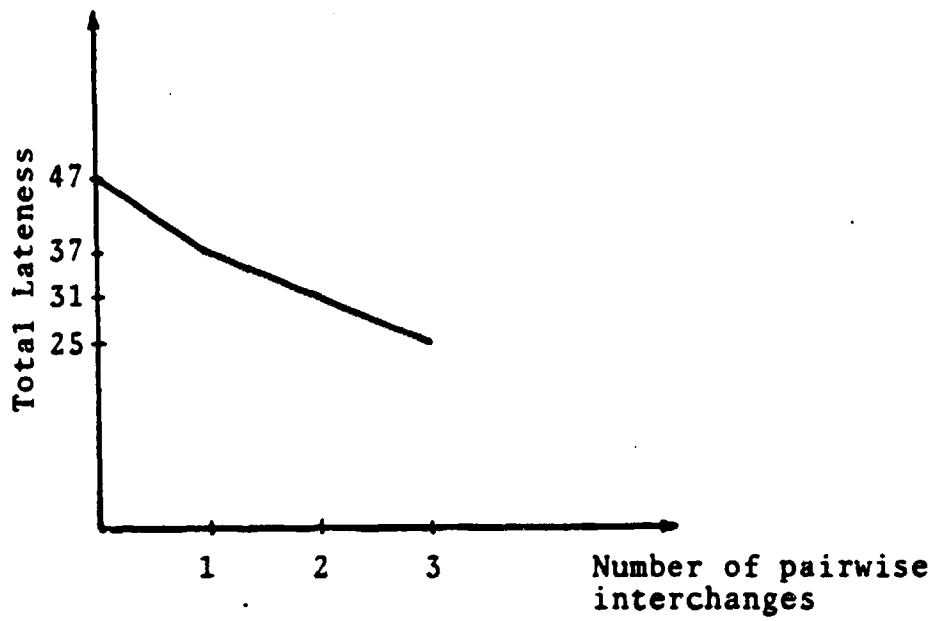
(b)

Figure 5.13 Results of Sensitivity Test for Sequence 7

- (a) Network of sequence 7
- (b) Total lateness curve for sequence 7



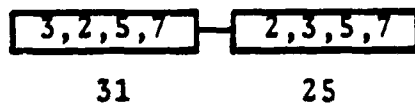
(a)



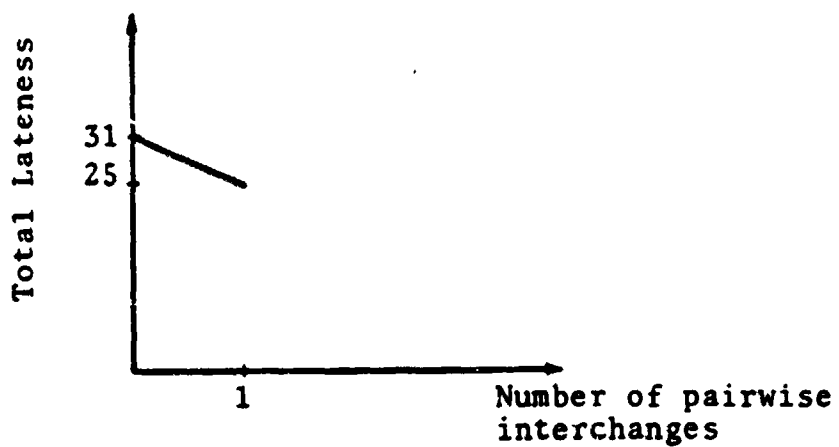
(b)

Figure 5.14 Results of Sensitivity Test for Sequence 8

- (a) Network of sequence 8
- (b) Total lateness curve for sequence 8

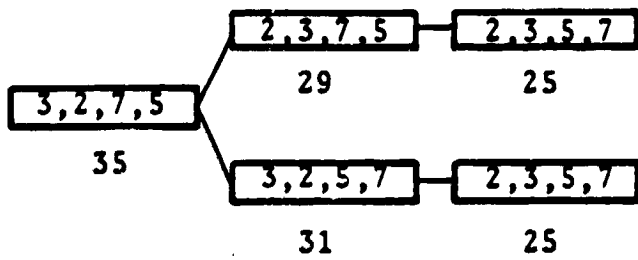


(a)

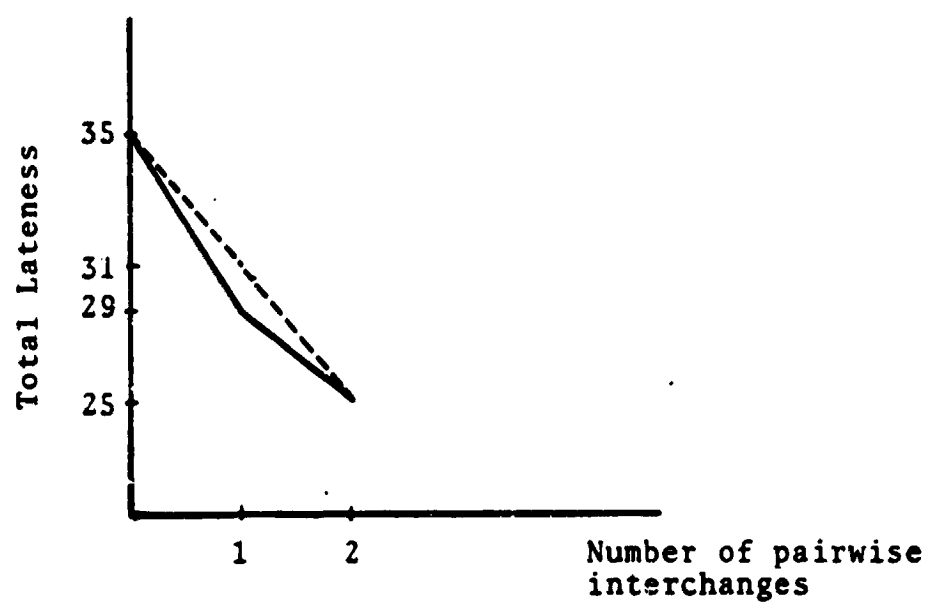


(b)

Figure 5.15 Results of Sensitivity Test for Sequence 9
 (a) Network of sequence 9
 (b) Total lateness curve for sequence 9

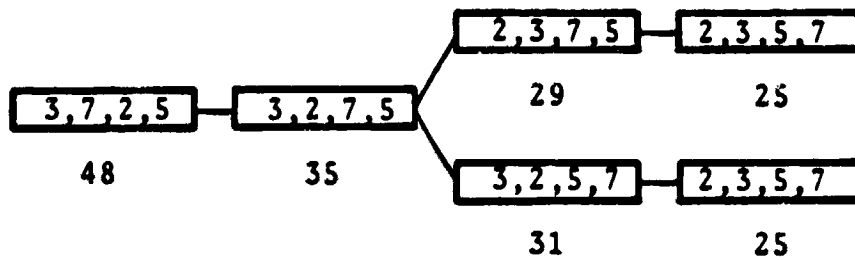


(a)

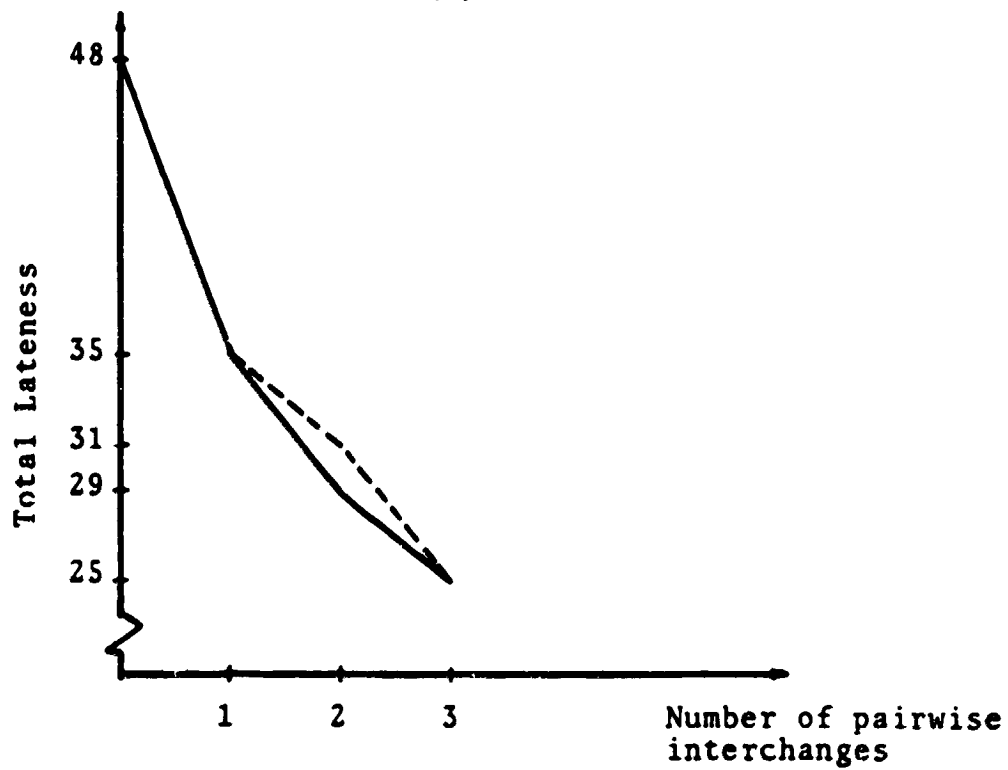


(b)

Figure 5.16 Results of Sensitivity Test for Sequence 10
 (a) Network of sequence 10
 (b) Total lateness curve for sequence 10



(a)



(b)

Figure 5.17 Results of Sensitivity Test for Sequence 11
 (a) Network of sequence 11
 (b) Total lateness curve for sequence 11

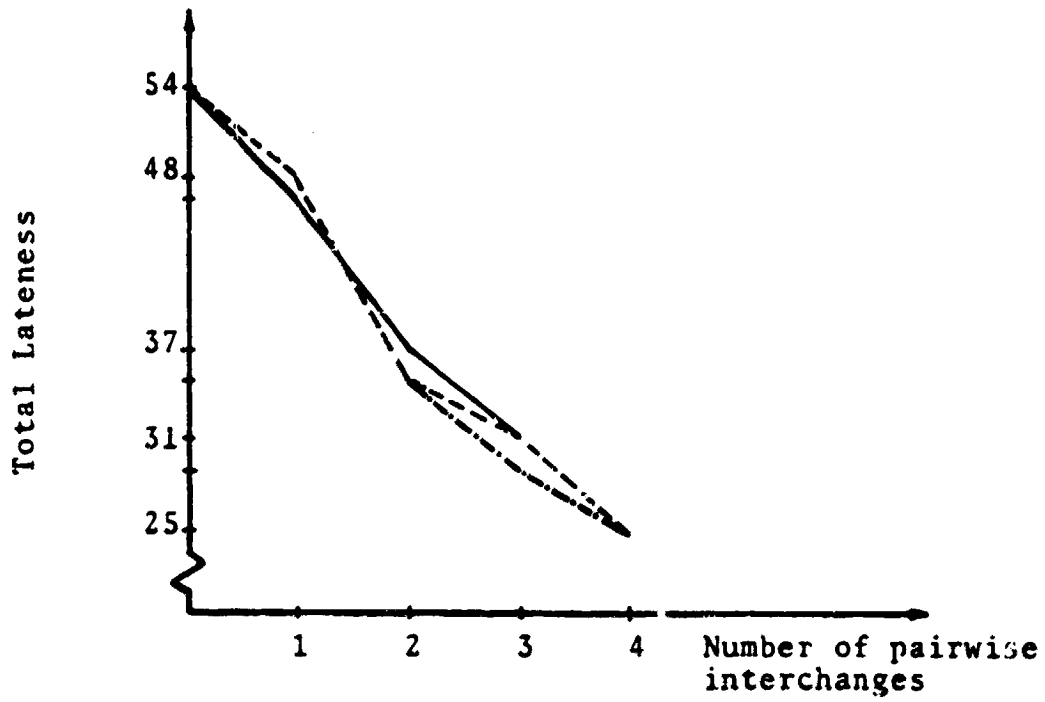
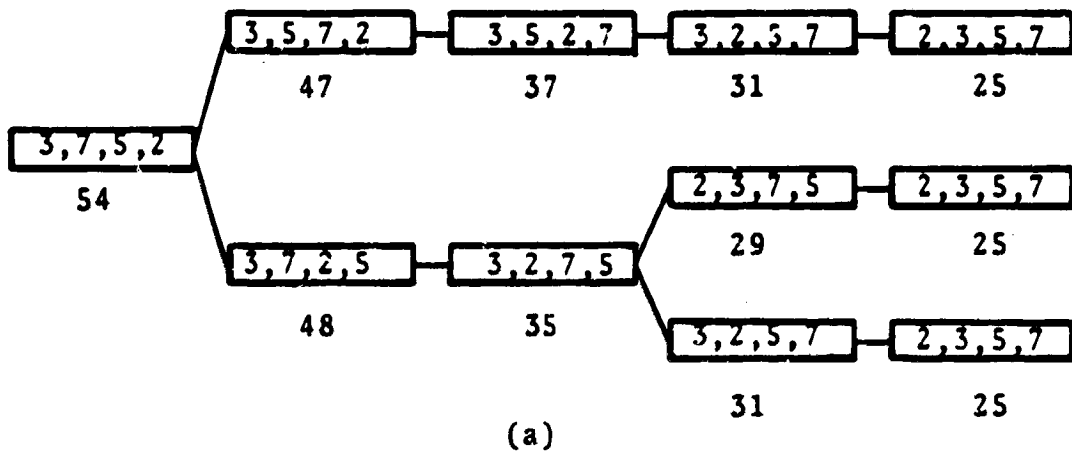
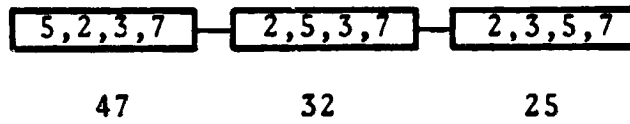
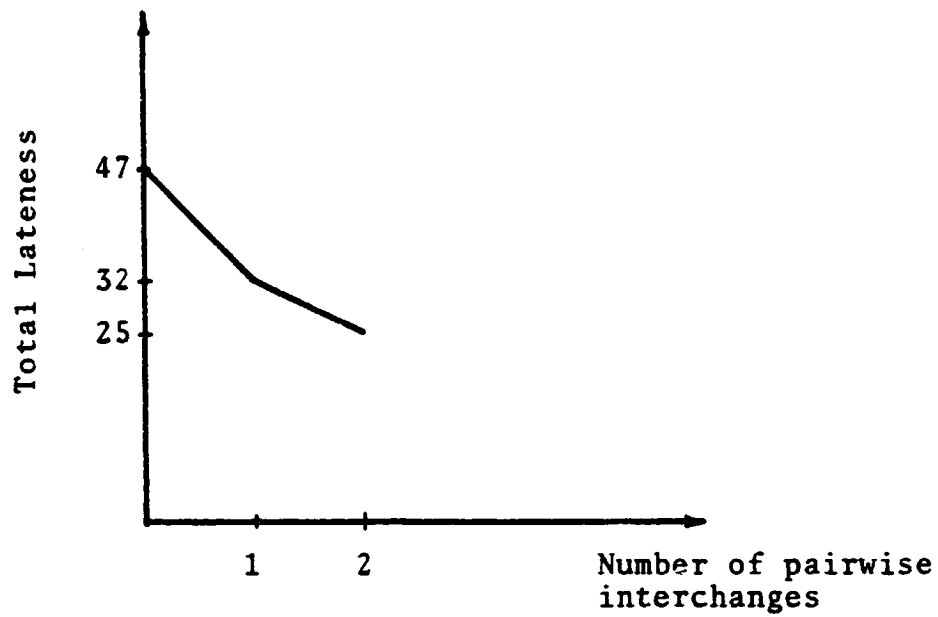


Figure 5.18 Results of Sensitivity Test for Sequence 12
 (a) Network of sequence 12
 (b) Total lateness curve for sequence 12



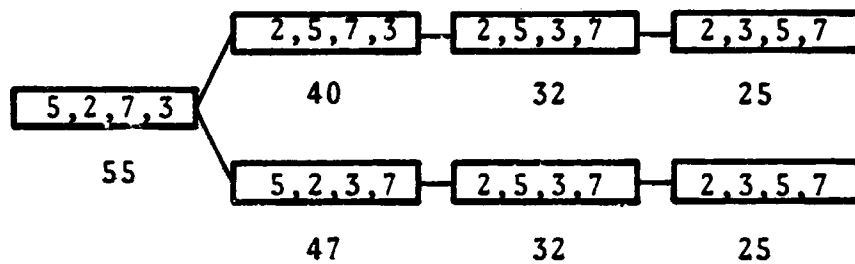
(a)



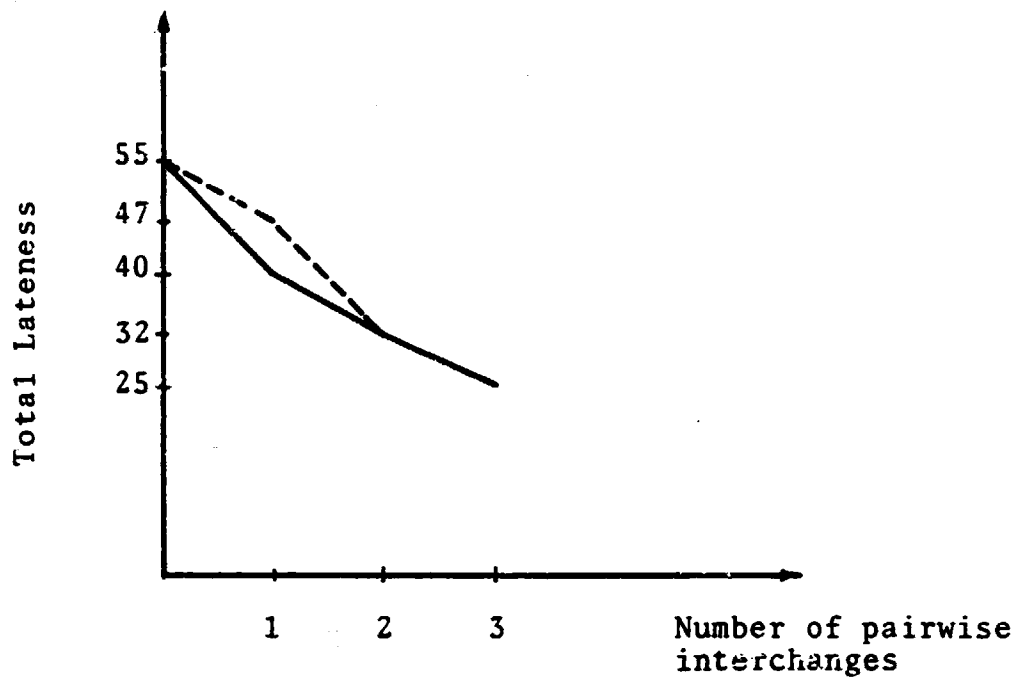
(b)

Figure 5.19 Results of Sensitivity Test for Sequence 13

- (a) Network of sequence 13
- (b) Total lateness curve for sequence 13



(a)

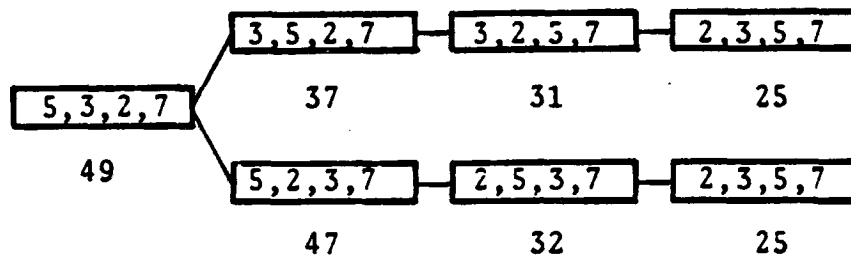


(b)

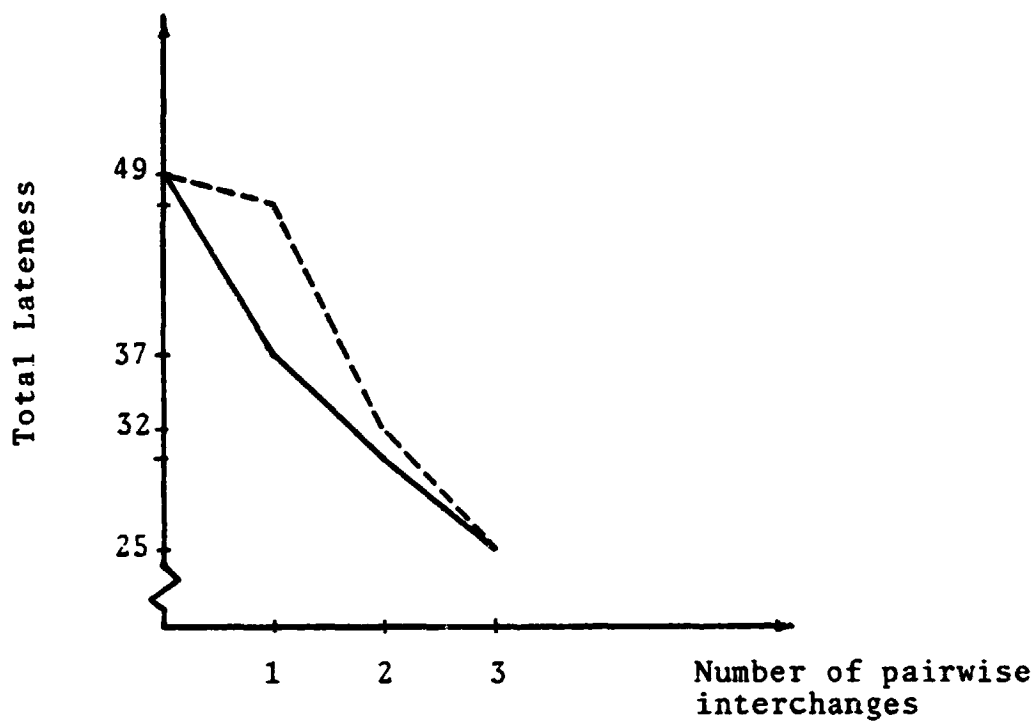
Figure 5.20 Results of Sensitivity Test for Sequence 14

(a) Network of sequence 14

(b) Total lateness curve for sequence 14



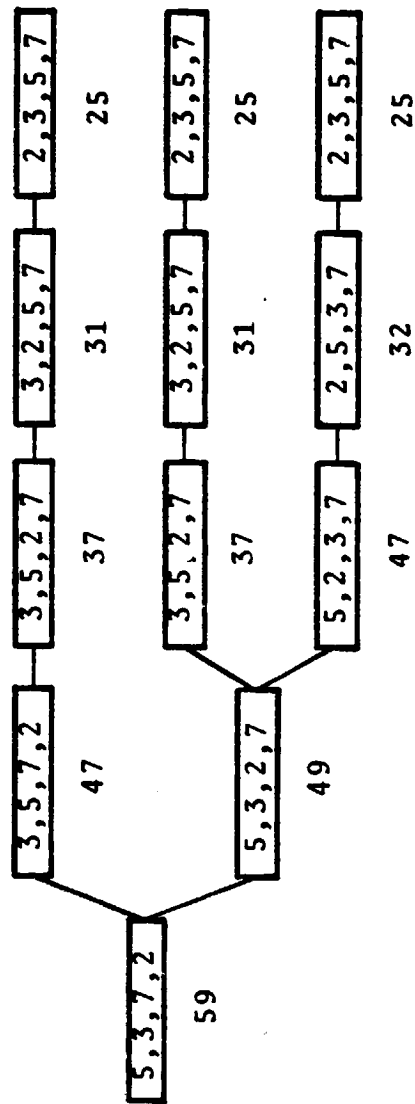
(a)



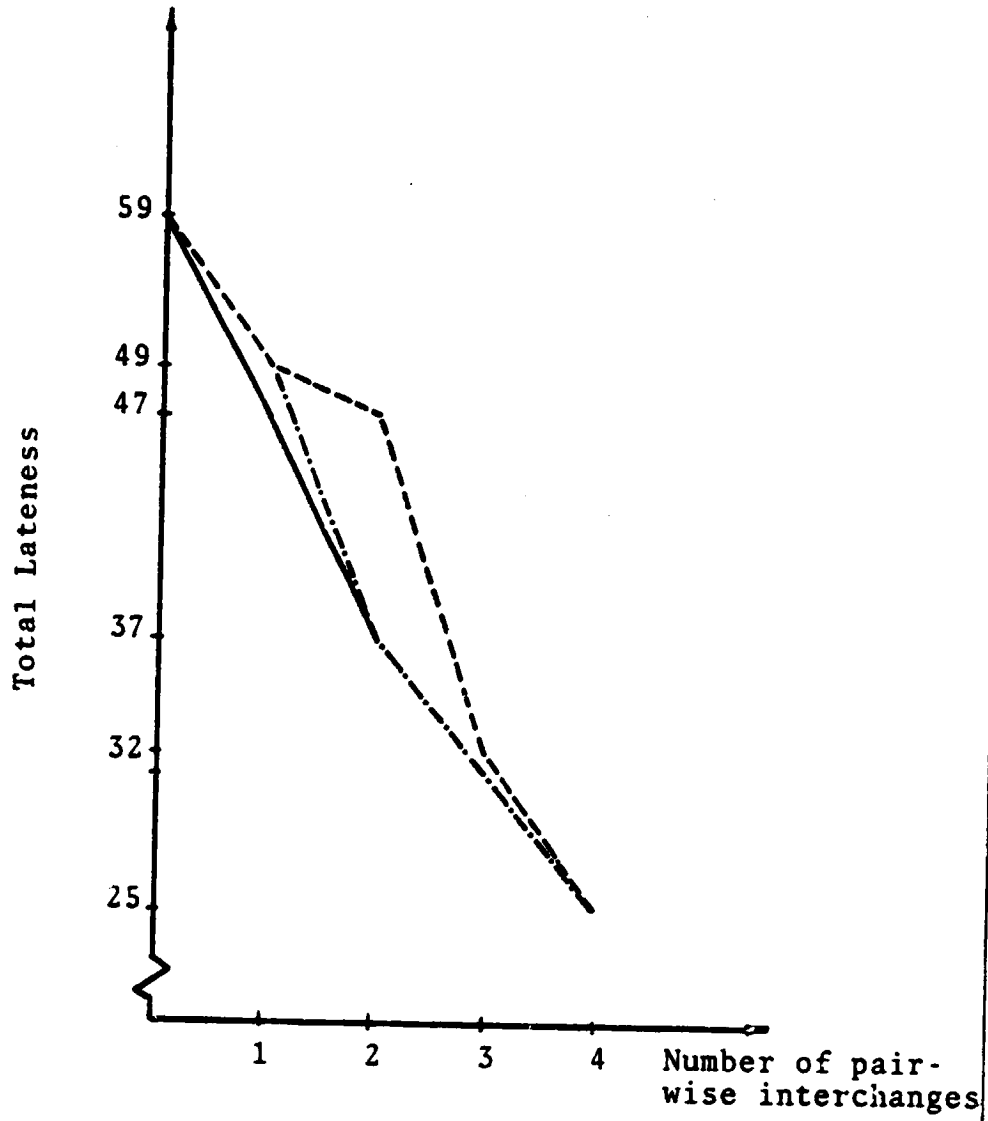
(b)

Figure 5.21 Results of Sensitivity Test for Sequence 15
 (a) Network of sequence 15
 (b) Total lateness curve for sequence 15

Figure 5.22 Results of Sensitivity Test for
Sequence 16
(a) Network of sequence 16
(b) Total lateness curve for
sequence 16

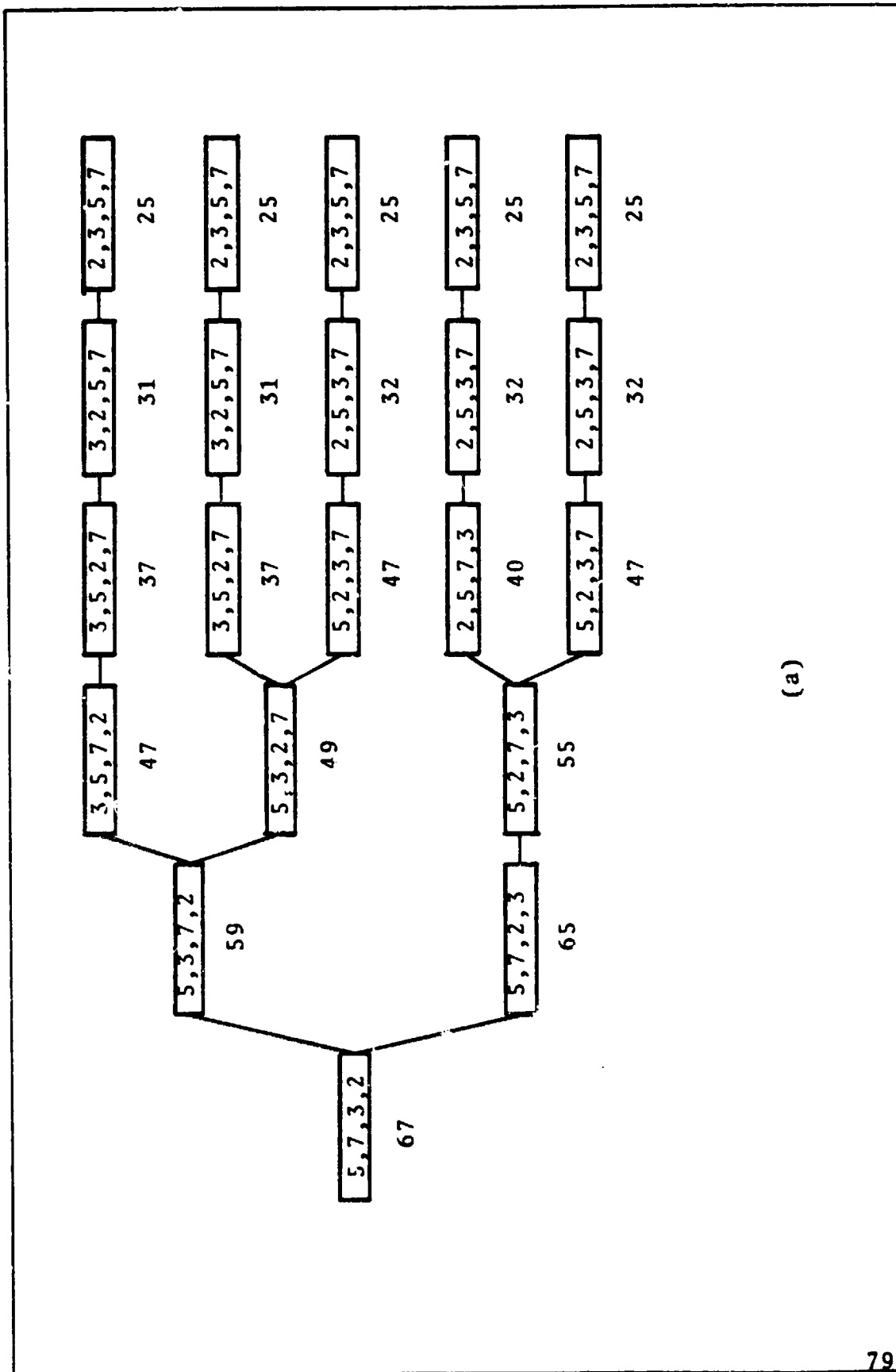


(a)

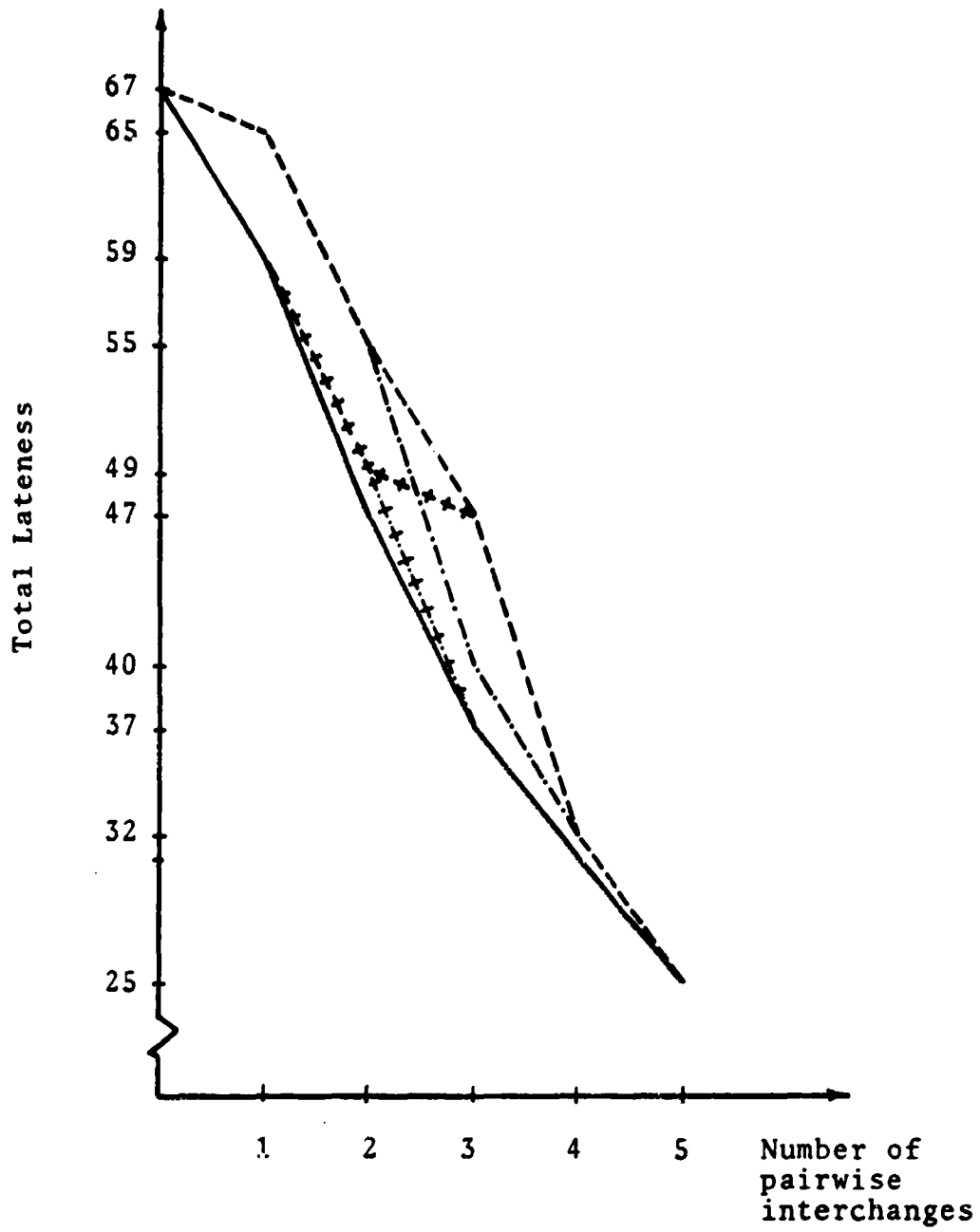


(b)

**Figure 5.23 Results of Sensitivity Test for
Sequence 17**
(a) Network of sequence 17
(b) Total lateness curve for
sequence 17



(a)



(b)

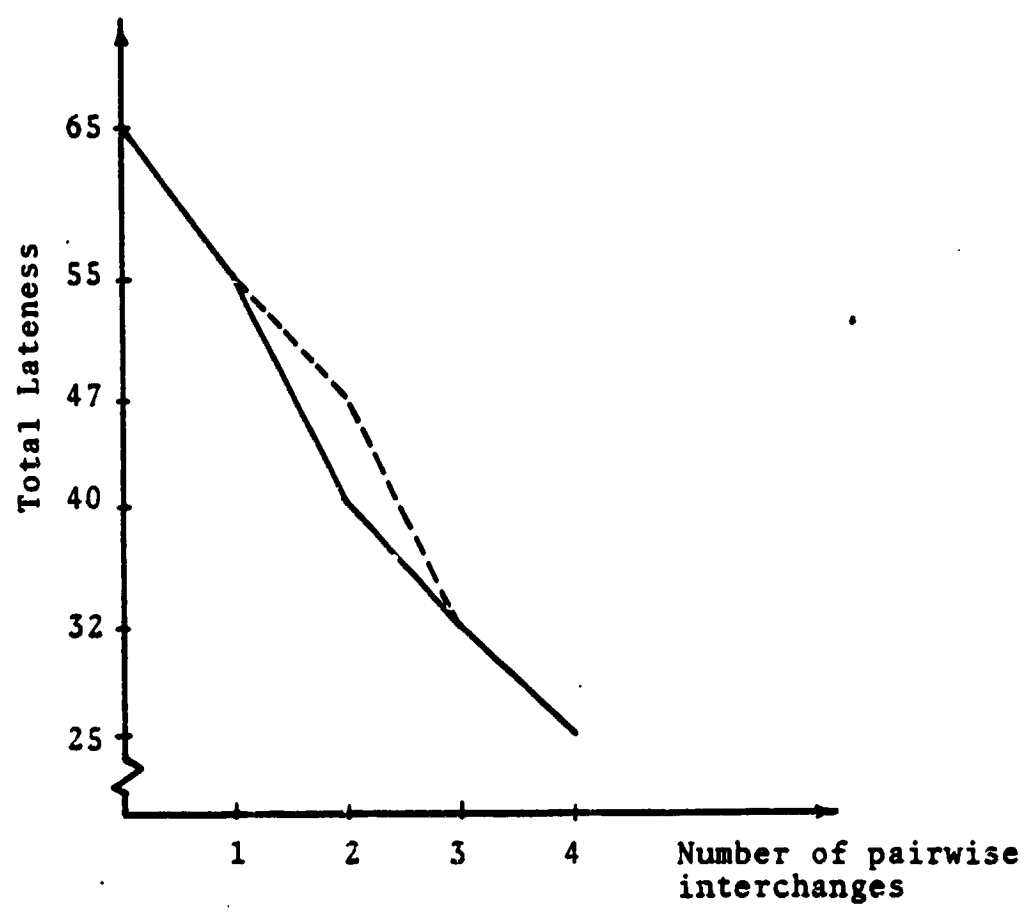
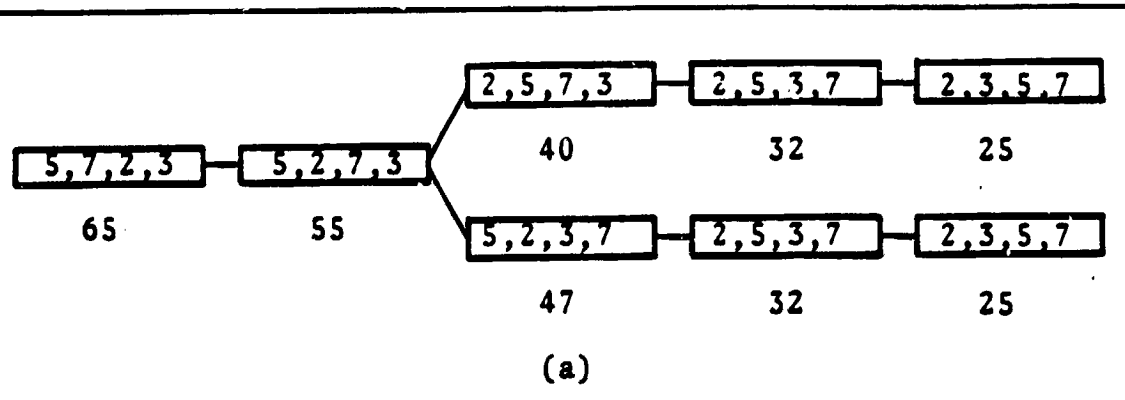
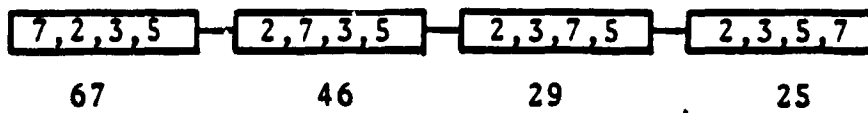
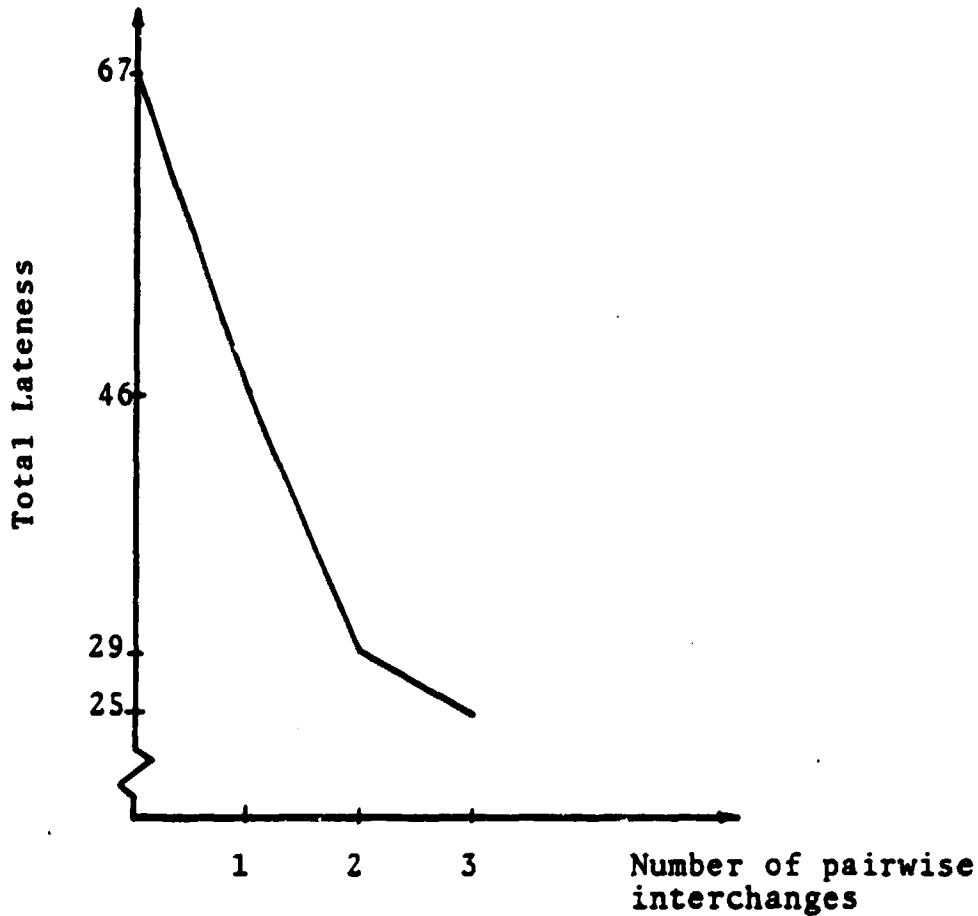


Figure 5.24 Results of Sensitivity Test for Sequence 18
 (a) Network of sequence 18
 (b) Total lateness curve for sequence 18



(a)



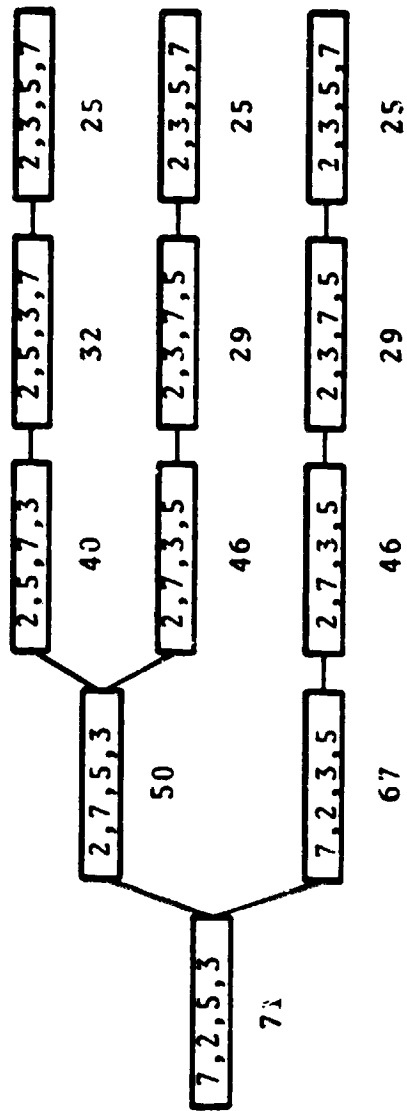
(b)

Figure 5.25 Results of Sensitivity Test for Sequence 19

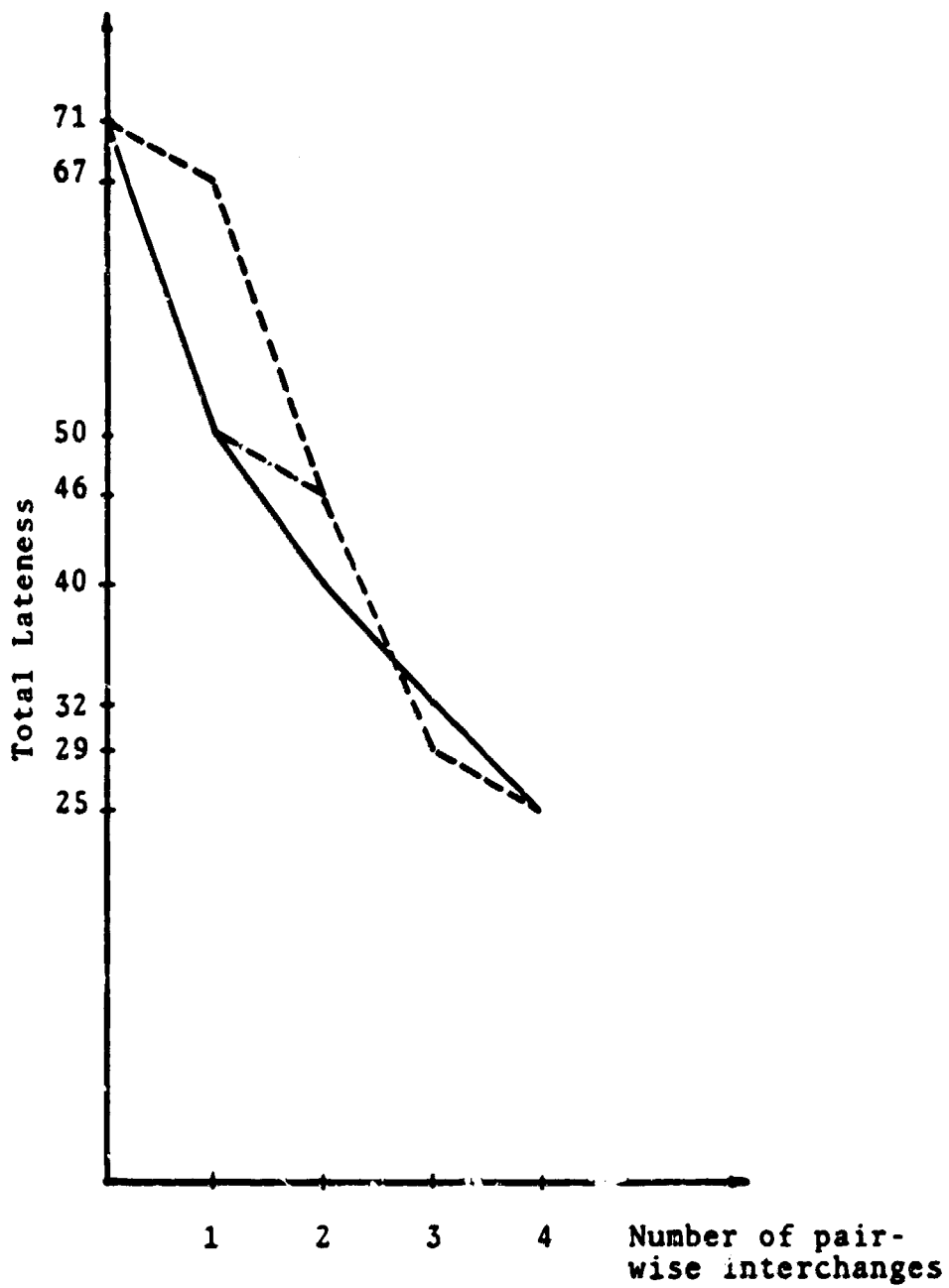
(a) Network of sequence 19

(b) Total lateness curve for sequence 19

Figure 5.26 Results of Sensitivity Test for
Sequence 20
(a) Network of sequence 20
(b) Total lateness curve for
sequence 20



(a)

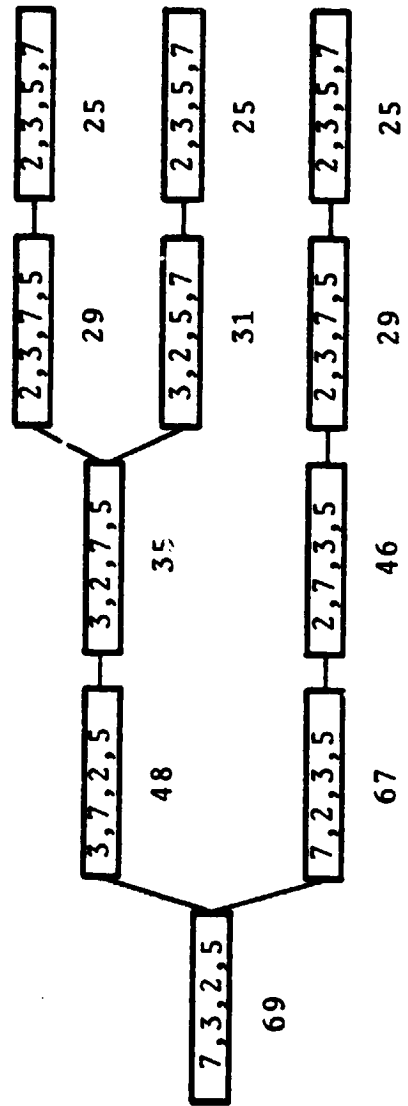


(b)

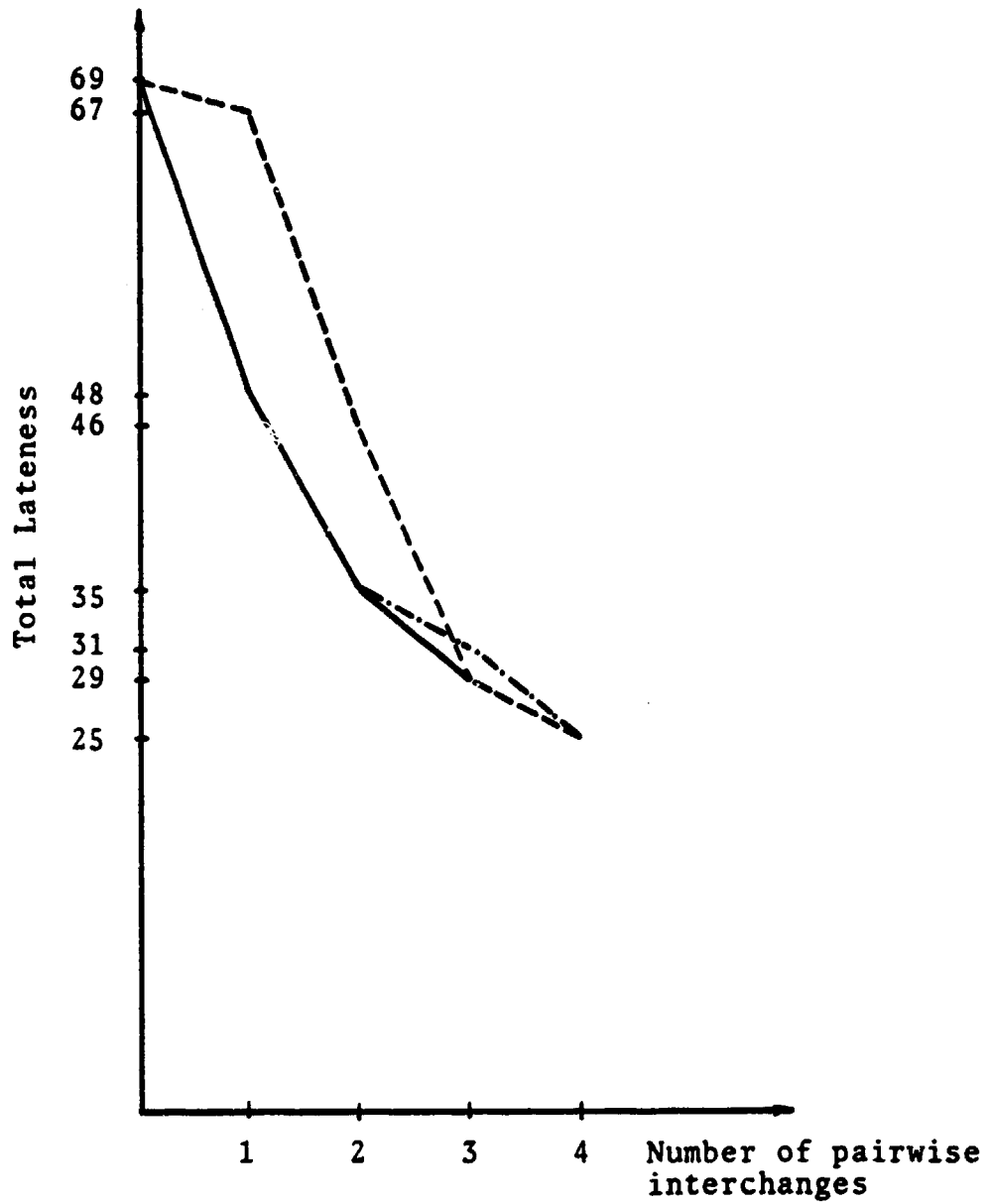
Figure 5.27 Results of Sensitivity Test for
Sequence 21

- (a) Network of sequence 21
- (b) Total lateness curve for
sequence 21

C-3

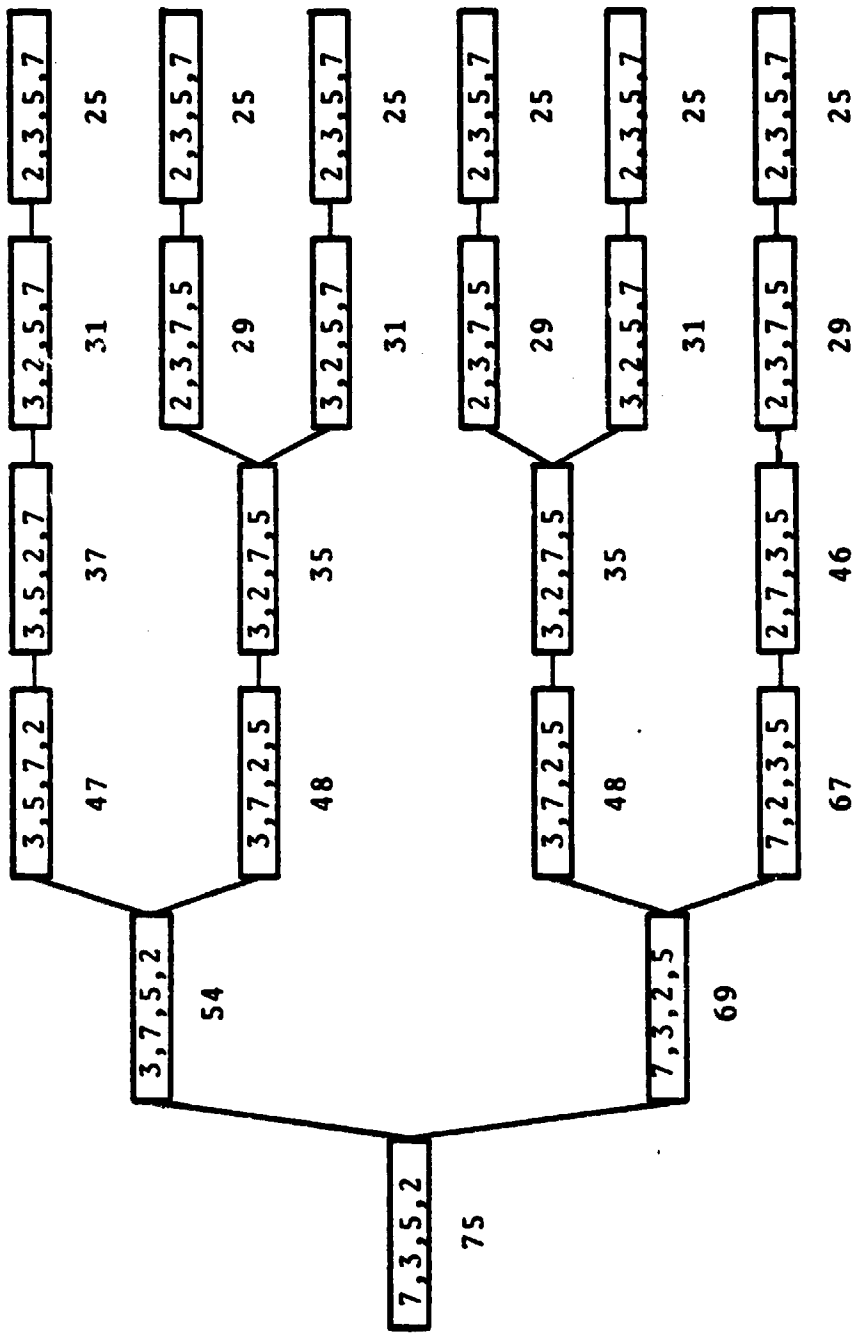


(a)

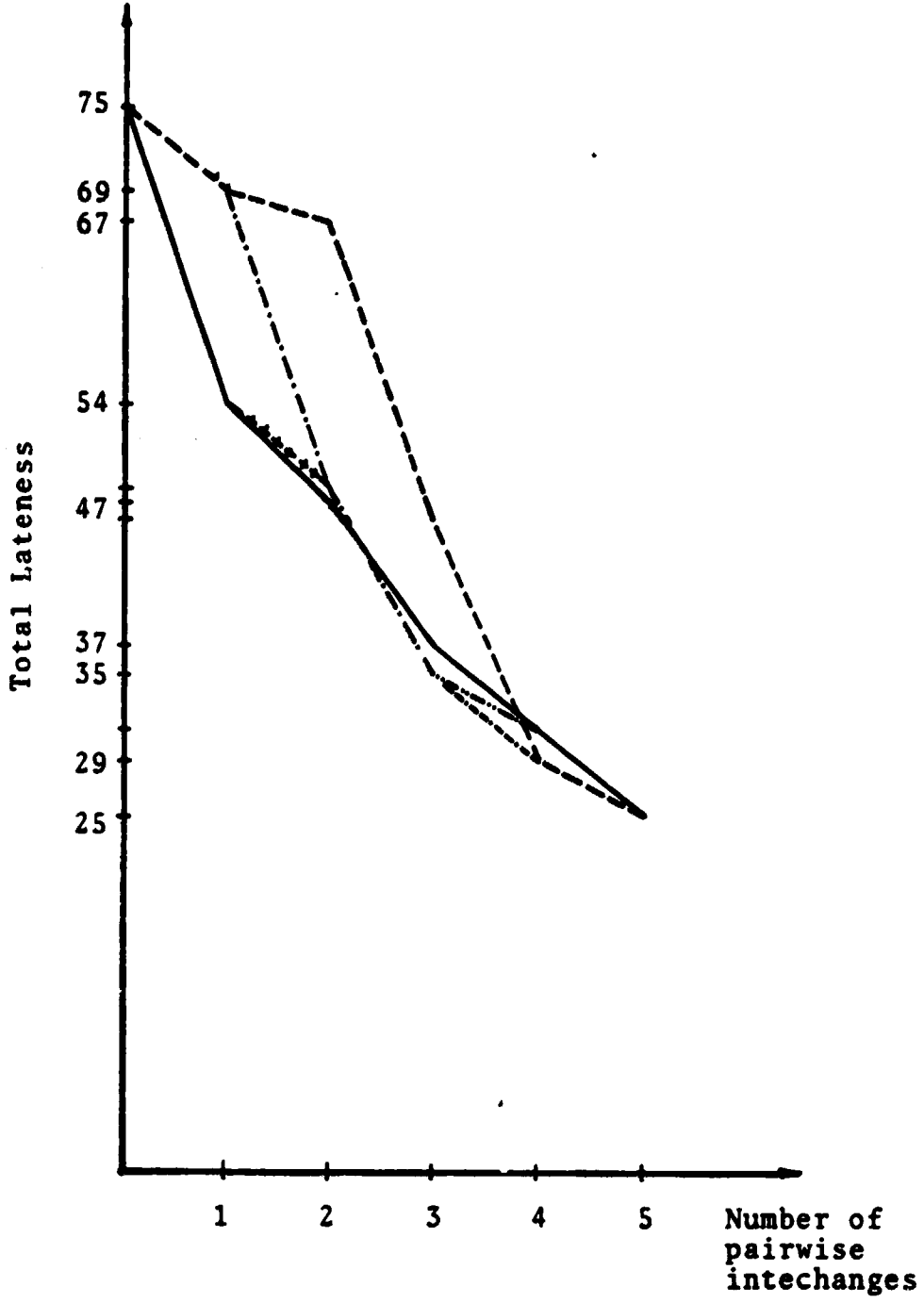


(b)

**Figure 5.28 Results of Sensitivity Test for
Sequence 22**
(a) Network of sequence 22
(b) Total lateness curve for
sequence 22

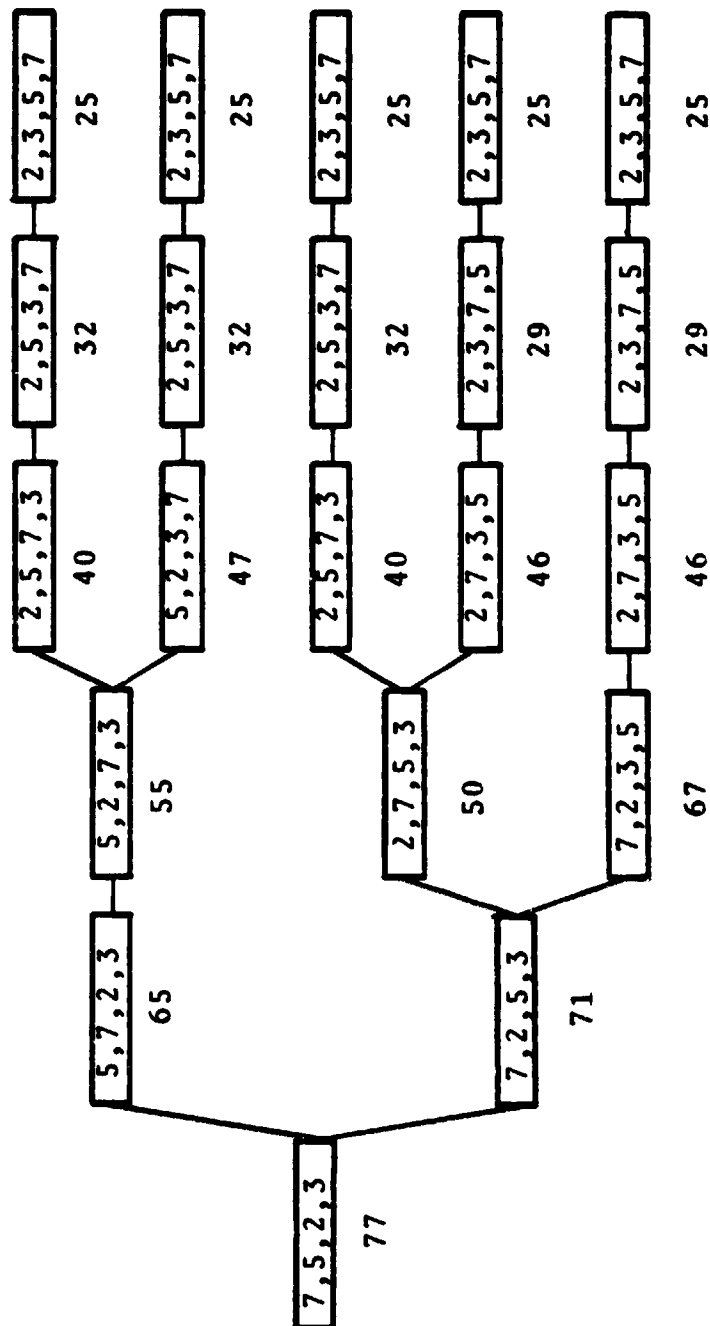


(a)

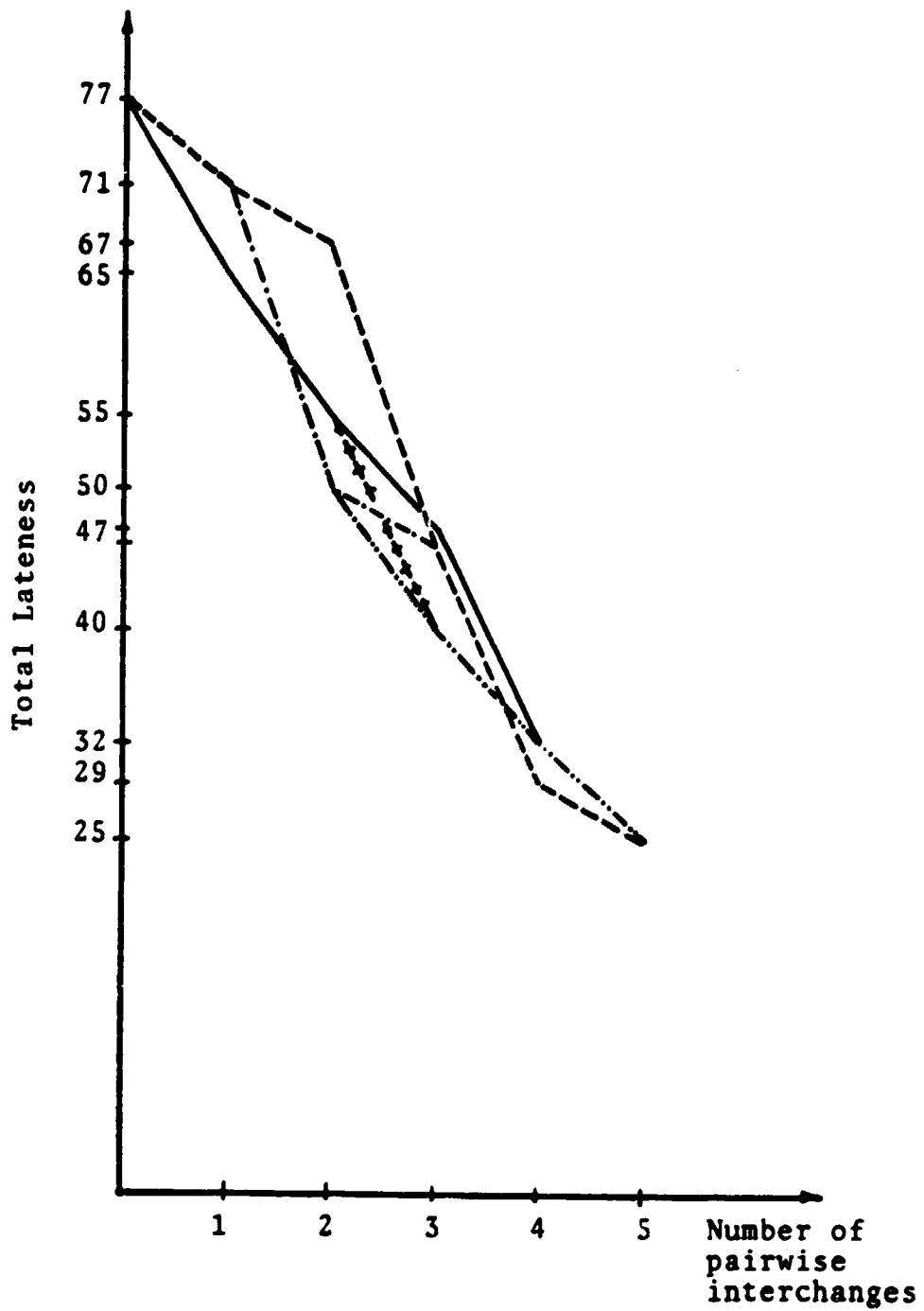


(b)

**Figure 5.29 Results of Sensitivity Test for
Sequence 23**
(a) Network of sequence 23
(b) Total lateness curve for
sequence 23



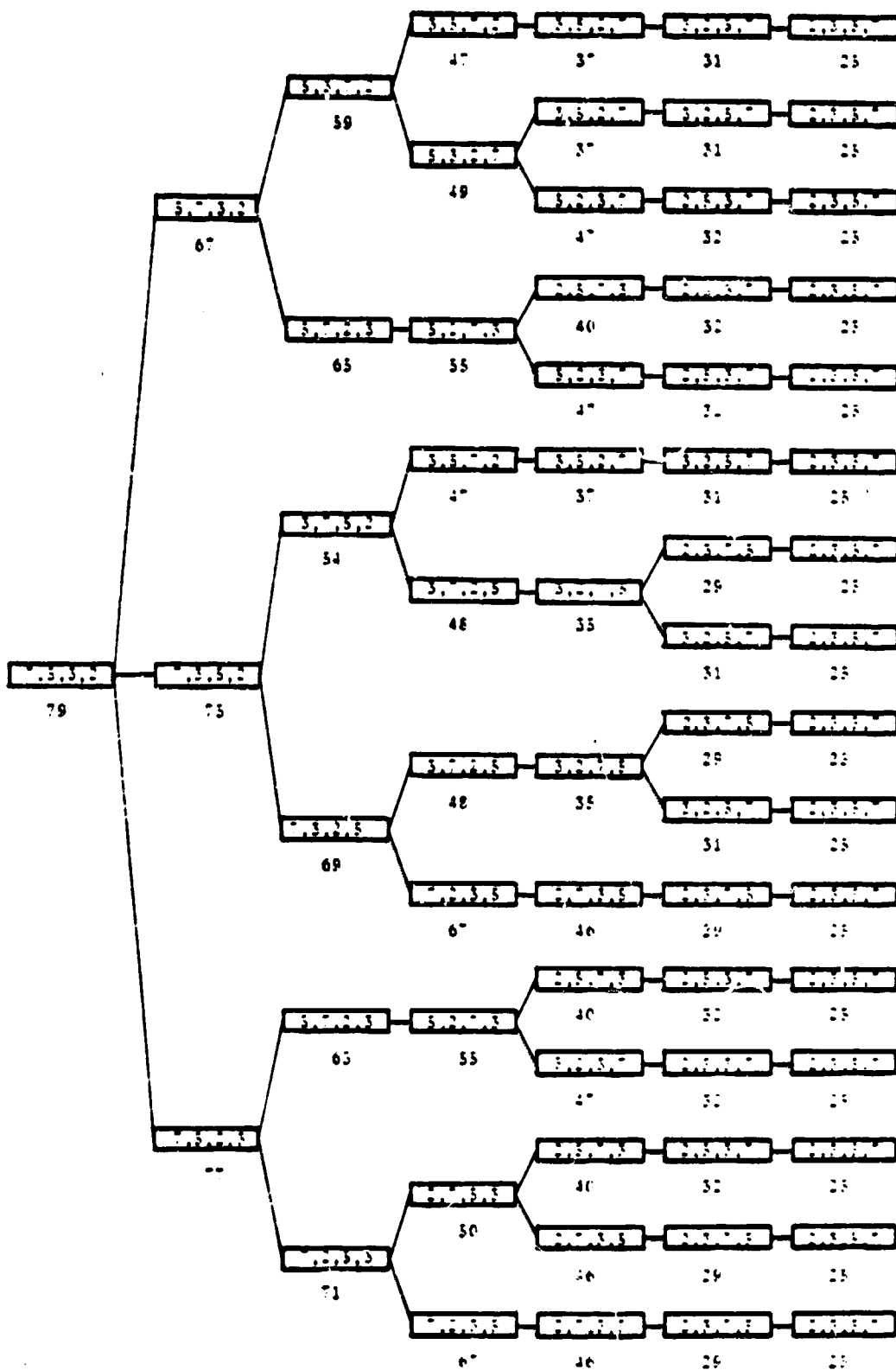
(a)



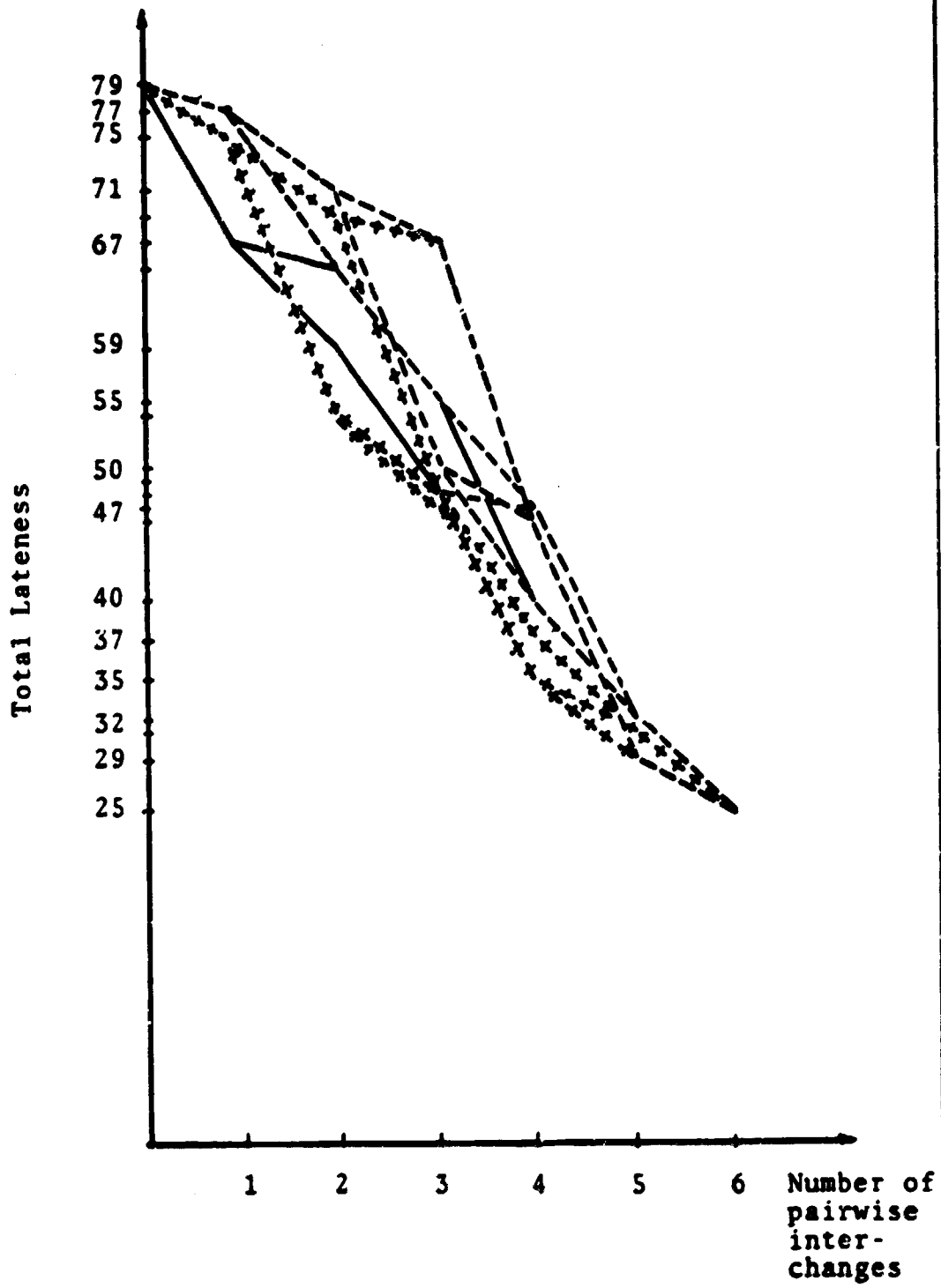
(b)

Figure 5.30 Results of Sensitivity Test for
Sequence 24
(a) Network of sequence 24
(b) Total lateness curve for
sequence 24

ORIGINAL PAGE IS
OF POOR QUALITY



(8)



(b)

5.3 Flow Time in a Linear Flow Shop

In general, flow time is defined as the amount of time that a job spends in the shop.

In the case of a two machine LFP system shown in Figure 5.31, the flow times F_k , $k \in I$ are as follows:

$$F_1 = t_{11} + t_{12} \quad (5.74)$$

$$F_2 = \max\{C_1, t_{11} + t_{21}\} + t_{22} \quad (5.75)$$

$$F_3 = \max\{C_2, t_{11} + t_{21} + t_{31}\} + t_{32} \quad (5.76)$$

Using equations (5.74) through (5.76), the flow time of job i is:

$$F_i = \max\{C_{i-1}, \sum_{k=1}^i t_{k1}\} + t_{i2} \quad (5.77)$$

where

t_{ij} is the time to perform job i on machine j

C_i is the completion time of job i

F_i is the flow time of job i .

Theorem 2

The mean flow time \bar{F} in a two machine LFP where the job times on the machines are a linear function of the batch sizes ($t_{ij} = b_j n_i$, $b_j \geq 1$), is minimized by shortest processing time (SPT) sequencing.

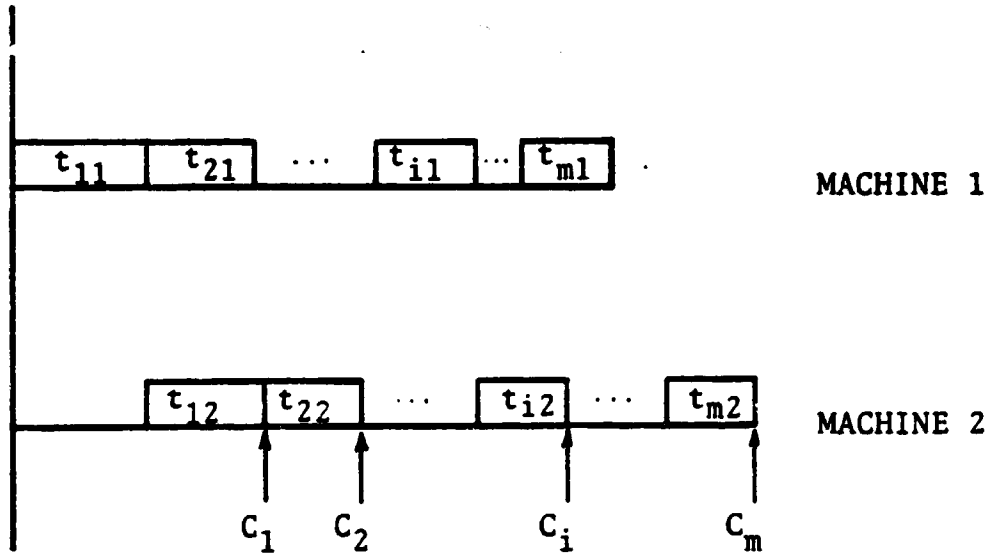


Figure 5.31 Two Machine Linear Flow Shop

Proof

The proof is a mirror image of the proof of Theorem 1, and utilizes an adjacent pairwise interchange argument.

Consider a sequence S that is not a SPT sequence. That is, somewhere in S there must exist at least one pair of adjacent jobs, i and i+1 with i+1 following i, such that $t_i > t_{i+1}$. Now construct a new sequence S', in which jobs i and i+1 are interchanged in sequence and all other jobs are not changed. This situation is depicted in Figure 5.1 where T_A denotes the point in time at which job i begins in sequence S and at which job i+1 begins in sequence S'.

"A" denotes the set of jobs that precede job i and i+1 in both schedules and "B" denotes the set of jobs that follow i and i+1 in both schedules.

The proof is shown for $\sum_{i=1}^m F_i$ (sum of the flow time over all the jobs), as the mean flow time is: $F = \frac{1}{m} \sum_{i=1}^m F_i$.

$$\sum_{k=1}^m F_k = F_A + F_i + F_{i+1} + F_B \quad (5.78)$$

where

F_A is the sum of the flow times of all the jobs in set "A"

F_i is the flow time of job i

F_j is the flow time of job j

F_B is the sum of the flow times of all the jobs in set "B".

The flow times of the jobs in sequence S are:

$$F_i(S) = \max\{C_A, T_A + t_{i,1}\} + t_{i,2} \quad (5.79)$$

$$F_{i+1}(S) = \max\{C_i(S), T_A + t_{i,1} + t_{i+1,1}\} + t_{i+1,2} \quad (5.80)$$

Using equation (5.78), the flow time of sequence S is:

$$F_k(S) = F_A + \max\{C_A, T_A + t_{i,1}\} + t_{i,2} + \max\{C_i(S), T_A + t_{i,1} + t_{i+1,1}\} + t_{i+1,2} + F_B \quad (5.81)$$

The flow times of jobs i and i+1 in sequence S' are:

$$F_{i+1}(S') = \max\{C_A, T_A + t_{i+1,1}\} + t_{i+1,2} \quad (5.82)$$

$$F_i(S') = \max\{C_{i+1}(S'), T_A + t_{i+1,1} + t_{i,1}\} + t_{i,2} \quad (5.83)$$

Using equation (5.78), the flow time of sequence S' is:

$$F_k(S') = F_A + \max\{C_A, T_A + t_{i+1,1}\} + t_{i+1,2} + \max\{C_{i+1}(S'), T_A + t_{i+1,1} + t_{i,1}\} + t_{i,2} + F_B \quad (5.84)$$

When observing equations (5.79) through (5.84), one can notice that the terms are actually equal to terms of completion times in equations (5.7) through (5.12) that are developed in Theorem 1.

The proof of the sequence that generates the minimum mean lateness (Theorem 1) is based on the completion times of the jobs, and the sum of the completion times of all the jobs in the sequence. Using the same steps of the proof can show that the mean flow time is also minimized by the shortest processing time sequencing.

It is proven that the mean flow time is minimized by the SPT sequence.

5.4 Waiting Time in a Linear Flow Shop

In a flow shop scheduling problem, a waiting time can be defined for each job in the system. It is the time that every job must wait until it can be processed. If the jobs are processed in a numerical order, the waiting time for the i^{th} object is $\sum_{k=1}^{i-1} t_{k,1}$. The idea is similar to the problem of scheduling for single stage production, since in flow shop problems there is no idle time on the first machine.

A performance measurement can be the sum of all waiting times $\sum_{i=1}^m \sum_{k=1}^{i-1} t_{k,1}$. The minimum of this expression occurs at the same time as the minimum of the expression

$\sum_{i=1}^m \sum_{k=1}^i t_{k,1}$ occurs, since the two expressions differ by the constant $\sum_{i=1}^m t_i$.

Theorem 3

The sum of waiting times of all the jobs in a flow shop problem is minimized by shortest processing time (SPT) sequencing.

Proof

In the proof, the term $\sum_{i=1}^m \sum_{k=1}^i t_{k,1}$ is used because of the reason mentioned above, as the term that represents the sum of waiting times.

$$\sum_{i=1}^m \sum_{k=1}^i t_{k,1} = \sum_{k=1}^m (n+1-k)t_{k,1}$$

According to Hardy, Littlewood and Polya (1952), this sum is least when the series are monotonic in opposite senses. Therefore the sum of all waiting times is minimized, if the jobs are sequenced according to SPT ($t_{1,1} < t_{2,1} < \dots < t_{m,1}$).

CHAPTER 6

SUMMARY AND CONCLUSIONS

This study investigates a flow shop scheduling problem which is defined as the linear flow shop problem (LFP). The purpose of the research was to find an optimal solution for real life cases that can be defined as linear flow shop problems. Three performance objectives were defined; mean lateness, mean flow time and waiting time. The results show that the shortest processing time sequencing minimizes the objectives that were defined. Implementing optimal solution for a practical problems is very easy. The jobs that have to be processed are ordered in the system by monotonic increasing performance time.

Chapter 1 of this dissertation introduces to the reader the scheduling and sequencing area. The basic concepts and basic models of scheduling are defined and the main performance measurements of a scheduling problem are formulated.

Chapter 2 defines the basic concepts of flow shop scheduling. Two variations of flow shop scheduling are discussed, the pure flow shop and the general flow shop. The only requirement on the schedule is that all movements

of jobs between machines within the shop be in uniform direction from machine j to machine $j+1$ etc.

Chapter 3 reviews past research in the flow shop area. Many studies were conducted, and many articles were written on flow shop problems. The studies that are reviewed introduce the basic ideas that were developed in the machine scheduling area and the special flow shop cases. A discussion of the differences between optimal and heuristic solutions to the problem is included. Optimal solutions are efficient in theory but difficult to implement in practical situations.

The linear flow shop problem is formulated in Chapter 4. It is a model of a practical production facility where the time to complete a batch on a specific machine and the times to complete the same batch on all other machines is a linear function of the batch size. The time to perform all jobs through all machines and the machining time required to perform the jobs is developed. The idle time involved in a linear flow shop problem is discussed in three steps. First the case of a two machine LFP, second a three machine LFP and third the consideration of the general m jobs n machines case. Upper and lower bounds for performance time are developed where the optimal solution falls in that range.

Chapter 5 describes three performance measurements of the linear flow shop scheduling, mean lateness, mean flow

time and waiting time. Three theorems regarding the defined performance measurements are stated. The optimal solution for a two machine case is proven to be a schedule that follows the shortest processing time ordering. The proofs for mean lateness and mean flow time use the method of pairwise interchange of pairs or adjacent jobs in the sequence. The method of mathematical induction is used to prove that the theorems hold for the case of m jobs in the sequence. A numerical example follows the proofs. A sensitivity test is conducted to show how the shortest processing time sequencing yields the optimal solutions.

This study presents optimal solutions for the linear flow shop problem. It was found in many studies that optimal solutions are difficult to implement in practical cases because of the complexity of the formulation and computational problems. The advantage of the results found in this dissertation is that it is very easy to construct the optimal sequence according to the shortest processing time ordering. The results can be implemented in practical production situations that fit the linear flow shop definition.

This research investigated the linear flow shop where all variables are deterministic and can be established according to data available from past years or experience of production management people. An open subject to be researched in the future is the case where the machines'

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

C

coefficients are random variables and the times to perform the jobs are the expected performance time. Further research should be conducted into production models other than the LFP. For example, a fixed setup time per machine is often necessary in parts production. Non linear production times often occur in chemical processes. Another issue in scheduling is the idle time involved in the process. One can take the advantage of idle time of machines to conduct maintenance needed in the shop. In all kinds of problems created in the scheduling area one should keep in mind the way of implementing the results of the research. People in industry want easy and understandable procedures to make their production lines run smoothly and efficiently.

BIBLIOGRAPHY

- Akers, S.B., Jr., "A Graphical Approach to Production Scheduling Problems," Operations Research, Vol. 4, No. 2 (April 1956), pp. 244-245.
- Conway, R.W., Maxwell, W.L. and Miller, L.W., Theory of Scheduling, Palo Alto: Addison-Wesley, 1967.
- Dudek, R.A., and Teuton, O.F., Jr., "Development of M-stage Decision Rule for Scheduling n Jobs through m Machines," Operations Research, Vol. 12, No. 3 (May 1964), pp. 471-497.
- Gupta, I.N.D., and Dudek, R.A., "Optimality Criteria for Flow Shop Schedules," A.I.I.E. Transactions, Vol. 3, No. 3 (September 1971), pp. 199-205.
- Hardy, G.H., Littlewood, J.E., and Polya, G., Inequalities. Cambridge: Cambridge University Press, 1952.
- Ignall, E., and Schrage, L.E., "Application of the Branch and Bound Techniques to Some Flow Shop Scheduling Problems," Operations Research, Vol. 13, No. 3 (May 1965), pp. 400-412.
- Jackson, J.R., "An Extension of Johnson's Result on Job-Lot Scheduling," Naval Research Logistics Quarterly, Vol. 3, No. 3 (September 1956), pp. 201-204.
- Johnson, S.M., "Optimal Two and Three Stage Production Schedules with Setup Times Included," Naval Research Logistics Quarterly, Vol. 1, No. 1 (March 1954).
- Palmer, D.S., "Sequencing Jobs Through a Multi-Stage Process In the Minimum Total Time - A Quick Method of Obtaining a Near Optimum," Operational Research Quarterly, Vol. 16, No. 1 (March 1965), pp. 101-108.
- Smith, W.E., "Various Optimizers for Single State Production," Naval Research Logistics Quarterly, Vol. 3, No. 1 (March 1956), pp. 59-66.

Story, A.E., and Wagner, H.M., "Computational Experience with Integer Programming for Job Shop Scheduling," Muth, J.F. and Thompson, G.L., Eds., Industrial Scheduling. Englewood Cliffs, N.J.: Prentice-Hall, 1963, Chapter 14.

PART IV
August, 1981

ROBECON
A GENERALIZED METHODOLOGY FOR ASSESSING
THE ECONOMIC CONSEQUENCES OF
ACQUIRING ROBOTS FOR REPETITIVE OPERATIONS

by
G. A. Fleischer

DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007

ROBECON
A GENERALIZED METHODOLOGY
for
ASSESSING THE ECONOMIC CONSEQUENCES
of
ACQUIRING ROBOTS FOR REPETITIVE OPERATIONS

ABSTRACT

Although problems relating to the engineering design of robots are awesome, it is the economic aspect which is fundamental to the user's decision to acquire robots for repetitive operations. Nevertheless, a review of the relevant literature suggests that very little exists in the way of providing guidance to prospective purchasers of robots as to the economic consequences of prospective acquisitions. This conclusion stems from matching critiques of more than 25 published references against a set of explicit criteria for a generalized methodology. Recognizing the need for an appropriate methodology, an exhaustive set of cost elements are identified which are to be included in a comprehensive analysis.

(This paper is intended to serve as the first in a series leading to a fully-developed model, ROBECON, which may be used for specifying the economic consequences of robot systems acquisitions. The model will be computer based and user interactive.)

ROBECON

A GENERALIZED METHODOLOGY FOR ASSESSING THE ECONOMIC CONSEQUENCES OF ACQUIRING ROBOTS FOR REPETITIVE OPERATIONS

I. INTRODUCTION

Robots Defined

The Robot Institute of America defines the robot as "a programmable, multi-function manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks." A less precise definition has been adopted by a manufacturer-users group, Computer Aided Manufacturing International (CAM-I): "a device that performs functions ordinarily ascribed to human beings, or operates with what appears to be almost human intelligence." With either definition, it is commonly understood that modern robots are programmable manipulators that can perform useful work automatically without human assistance. (The term robot comes from a Czech word for forced labor; it was invented for Karel Capek's 1921 melodrama, R.U.R.).

Robot Installations: Substantial and Growing

Beginning with the development in the mid-60's of the microprocessor, which permitted robots to be made smaller and cheaper, and spurred by endemic wage inflation, robots have been used with increasing frequency in the industrialized nations. There are differences of opinion as to the number of robots currently in place around the world. One estimate is that there are "about 7,000 working industrial robots world-wide." (Ferguson, October 12, 1980). Another estimates about 15,000 robots

C

in the Western industrialized nations, with 10,000 in Japan and 3,000 in the U.S. at the end of 1979. (TIME, Dec. 8, 1980). Still another source estimates 40,000-50,000 in worldwide use, with 30,000 of these installations in Japan (Allan, 1979).

Expert opinion appears unanimous that the forecasted growth of robot installations will be spectacular into the foreseeable future. It is estimated that installations will increase at the rate of 30%-40% over the next decade. (The first and largest of the robot manufacturers in the U.S., Unimation Inc. of Danbury, Connecticut, experienced a 30% per annum growth rate over the past seven years.) Estimates of industry sales potential range from \$2 billion to \$4 billion by 1990. (Currently, industry sales in the U.S. are about \$90 million.) A recent forecast by the Society of Manufacturing Engineers and the University of Michigan estimates that by 1987, 15% of all assembly systems in the U.S. will use robot technology.

There are several significant reasons underlying expectations for substantial growth of robot installations in the foreseeable future. First, the conditions which led users to adopt robots over the past decade will persist, principally with respect to higher wage rates. Second, unit costs can be expected to decrease because robots are becoming smaller and more flexible and new manufacturers are entering the industry. Third, applications will increase as the functional capabilities are expanded, especially with respect to the ability of robots to see properly the articles which they are manipulating.

The Problem

The engineering design aspect of robots is awesome, yet it is the economic

aspect which is fundamental to the user's decision to acquire this equipment. After all, robots generally perform no functions which cannot otherwise be performed by combinations of human workers, machines and devices. The decision to acquire robots is influenced, wholly or in part, by the economic consequences to be expected from that decision. A preliminary review of the literature suggests that this issue has received little attention relative to the design and operational characteristics of robots. Certain cost estimates are widely quoted in the literature*, but these are generally inadequate as a guide to prospective users who may be contemplating capital investments of \$5,000 to \$150,000 per installation. (Multiple installations, i.e., implementation of systems using two or more robots, are not uncommon. Capital investments in the millions of dollars may be required in these instances, of course.)

Large, relatively sophisticated firms will probably have the expertise "in-house" to conduct appropriate economic analyses. However, as robot installations become more extensive, it is likely that smaller, less sophisticated firms will be considering the acquisitions of robots, and they will need competent guidance as to the economic justification for these decisions. It is this issue which provides the justification for the research described in the following sections.

Objective

There are a variety of ways of describing the process by which prospective users arrive at the decision to acquire a specific robot or robotic system.

*For example, Unimation Inc. reports that a robot's cost is \$4.60 hourly, and this has remained relatively constant since 1961. This is a rough estimate, however; it is based on straight line depreciation rather than cost of capital recovery, and taxes are ignored.

For our purposes, here, we may focus on three principal stages. First, the appropriate decision maker(s) within the firm must focus upon a limited set of candidates from among the much larger population of robots (and related auxiliary equipment and software) currently available in the marketplace. (It is assumed, at this point, that the decision maker has already completed an analysis of the task(s) and operating environment and is reasonably convinced that a robot system may represent an optimal solution to the manufacturing* problem). At this stage it will be necessary to describe important technical requirements for the robot(s), including: capacity, drives and controls, memory, and other features such as tactile, feedback and visual sensors. These technical requirements must then be matched against availability. See Exhibit 1, for example. The central feature of this first stage is the identification of a set of candidate systems with technical characteristics suitable to the firm's operational requirements. This includes, in addition to the robots themselves, associated requirements such as changes necessary to other equipment, tooling, spare parts and test equipment for maintenance, utilities, back-up equipment to be used if and when the robot is down, safety equipment, and the like.

The second stage is an economic analysis of the consequences, or impacts, of the candidate robot systems as identified in the first stage. This is the focus of the research described here. The objective is the development of an evaluation methodology which will permit users to forecast, or assess, the economic consequences of acquiring one or more robots for repetitive operations.

*Here, "manufacturing" includes fabrication, assembly, inspection, material handling and other tasks associated with the production of manufactured goods.

ORIGINAL PAGE IS
OF POOR QUALITY

This table is taken from responses to our survey on robots and from other material released to us for publication. Although it represents a broad sample of the industry, we regret that there was neither time nor space to make a directory of all companies involved.

To get more information from any of the companies, circle the indicated number on the Reader Service Card. To inquire about your interest not covered here, write the editors.

		CIRCLE NO.		COMPANY										
		E100	E101	AMF Electrical Products Development Div	ASEA Inc									
		E102	E103	Auto-Place Inc	Binks Mfg Co									
		E104	E105	Cincinnati Milbicon	DeVilbiss Co									
		E106	E107	Industrial Automata, Inc	Modular Machine Co									
		E108	E109	Cverton Engine Co	Pickomatic Systems, Inc									
		E110	E111	Prab Conveyors, Inc	Robomation Corp									
		E112		Unimation Inc, subsidiary of Condec Corp										
CAPACITY	Horizontal Arm Movement (ft)	5	7	2	3	8	6.7	2	2	6	1.5	4.8	1	13.5
	Vertical Arm Movement (ft)	5	7	.4	6.5	13	5.4	.6	2	6	.5	3.3	1	7.5
	Rotation (degrees)	270	340	200	85	240	210	120	290	360	180	190	225	220
	Maximum Workpiece Weight (lb)	2000	132	30	30	175	25	10	50	100	30	100	25	450
DRIVES & CONTROLS	Hydraulic	•			•	•	•		•	•	•	•	•	•
	Pneumatic			•					•					
	Electric		•				•		•		•			
	Air/Oil							•						
	Mechanical (against stops)							•	•			•	•	
	Air Logic							•					•	
	NC/Computer				•	•	•							•
MEMORY	Cams										•			
	Relays, Limit Switches			•					•					
	Rotating Drum											•		
	Magnetic Tape				•	•	•							•
	RAM, ROM, PROM	•	•				•	•						•
FEATURES	Core/Plated Wire					•								•
	Tactile Sensors	•	•	•	•				•	•			•	•
	Feedback Sensors	•	•	•	•				•	•			•	•
	Self-Diagnostic	•	•	•	•				•	•			•	•
	No. of Memory Steps	3000		24		900	256	U	U		21	U	U	1024
	No. of Articulations	7	6	5	6	6	5	U	U		4	5	U	6
Cycles Per Minute	64	V	V	V	V	17	20	200	75	7	12	V		

V—Variable U—Unlimited

Exhibit 1. Characteristics of Selected Robots

Source: "Trends in Robots", article appearing in Industrial Robots, Vol. 1, William T. Tanner, Editor. Society of Manufacturing Engineers, 1979. Reprinted from Tooling and Production, August 1977.

Users are assumed to be any business firms (manufacturers, fabricators, processors, etc.) or governmental agencies who may be considering the purchase of robots as operational alternatives and for whom the economic consequences are relevant to the acquisition decision.

The third phase in this process, as illustrated in Exhibit 2, is one in which economic consequences are considered jointly with other (non-economic) consequences so as to arrive at a choice from among alternative systems. There are a variety of approaches to this "multiple criteria" problem, some of which are relatively complex. In any event, this is not an issue which we will address further at this time. The research described in this document focuses only on Stage 2, the economic analysis.

Tasks

The tasks necessary to meet the objective described above are as follows:

- (1) Specify the criteria which must, or should, be met by the economic analysis methodology, including mathematical model(s) and associated procedures.
- (2) Review the literature to determine the extent to which economic data, models, and analytical procedures are currently available to prospective users. Relevant references will be critiqued in view of the criteria identified in (1).
- (3) Identify the elements of total system costs, that is, the economic impacts which, in general, may result over the lifetime of the robotic system, from installation to ultimate disposal. In certain government applications, and in the U.S. Department of Defense in particular, these are known as life cycle costs (LCC). (See, for

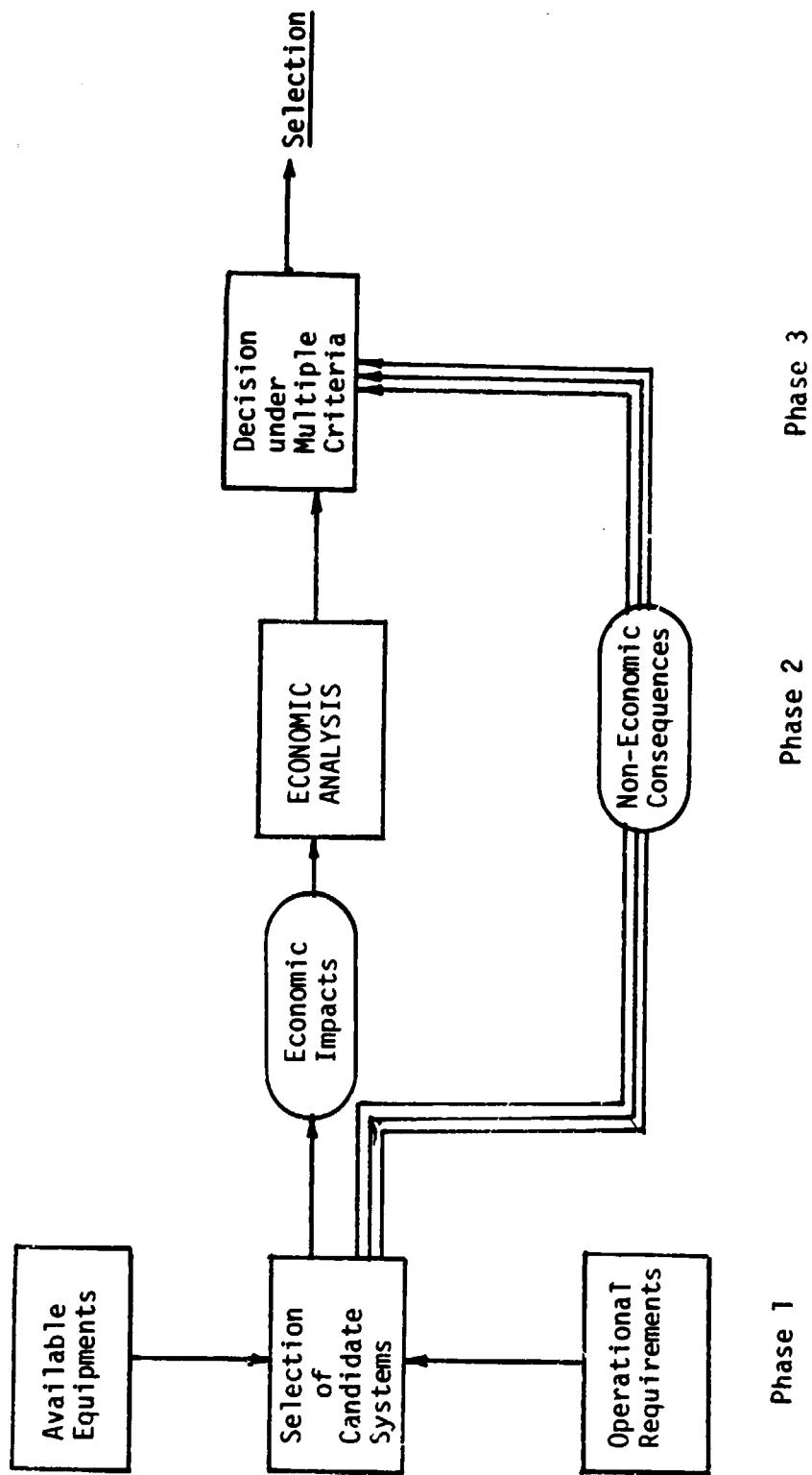


Exhibit 2. A Simplified Schematic Representation of the Decision Process for Acquiring a Robotic System

U.S. Department of Defense, 1978; U.S. Department of the Air Force, 1978; Graver and Jenkins-Stark, 1976; and Kolarik, 1980 .) Total system costs include subsequent costs as well as the initial investment.

- (4) Develop engineering cost estimates (ECEs) and/or cost estimating relationships (CERs), as appropriate. These are two principal approaches to the estimation of future economic consequences which are likely to result from a current investment decision. Generally, the CER approach relates system costs to a combination of measures of the systems (dimensions, performance characteristics, etc.)*. The cost estimating relationships are obtained through curve fitting techniques. In the ECE method, total system costs are broken down into relatively small components, or elements; the elements are related by ECEs which reflect the ways in which the system is developed, operated and maintained. It is expected that these relationships will be of sufficient generality to permit their use in a wide variety of analyses.

- (5) Develop computer-based models for generating Total System Costs. Three separate models will be developed, with increasing complexity, utilizing the ECEs and/or CERs as determined in the preceding task.

The models are characterized as follows:

- (a) In the first model we assume that all economic consequences (the amount and timing of cash flows); as well as the planning

*This method is sometimes called parametric costing.

horizon and discount rate, are deterministic. It is also assumed that the characteristics of the robotic system currently under consideration for implementation (the "challenger") are identical to those of all future challengers. That is, all future challengers are identical to the current challenger.

(b) In the second model we relax the assumption concerning the verisimilitude of current and future challengers. We now assume that future challengers are not necessarily identical to the current challenger. Indeed, it is likely that certain costs will decrease due to economies of scale and the ability of robot manufacturers to move out on the learning curve, for example; other cost elements, such as energy and labor, might be expected to increase over time.

(c) The third model differs from the second in that all economic consequences, the planning horizon and the discount rate are assumed to be stochastic. Unlike the prior models wherein all inputs were assumed to be known with certainty, we now treat these impacts as random variables.

(6) Test the implementability of each of the models through a series "of controlled" experiments. These will consist of a set of sample problems which will be presented to real-world decision makers for solution using the models. The experiences of these decision makers will be monitored and evaluated to determine the extent to which the models are useful in the capital allocation decision.

II. CRITERIA FOR A GENERALIZED METHODOLOGY

Prior to developing a generalized methodology for assessing the economic consequences of acquiring robots for repetitive operations, it is necessary to make explicit the criteria by which the efficacy of the methodology will be measured. These same criteria can also be used systematically to critique the existing relevant literature. For our purposes, then, the following criteria will be established:

- (1) Theoretically sound -- We are not interested solely in obtaining a solution. The solution must be internally consistent with the decision maker's (user's) objectives as well as the assumptions underlying the model.
- (2) Credible -- The users must have a feeling of confidence that the methodology will in fact provide solutions that are useful in the decision making process. The methodology must be believable.
- (3) Verifiable -- The user should be able to replicate, or verify, the results by tracing the chain of events from data input to ultimate solution. Verifiability is a precondition to credibility.
- (4) Comprehensive -- The economic model(s) imbedded in the methodology should include all the economic impacts which can reasonably be expected to occur as the result of the decision. (The time interval over which these impacts will occur is the planning horizon.) Thus the methodology should include the economic consequences of the total system -- equipment acquisition, operation and maintenance, taxes, and the like -- throughout the entire planning horizon. This is the Total System Costs concept.

- (5) Reasonable data requirements -- Although comprehensibility is a desirable, if not essential, element of the assessment methodology, it is unrealistic to expect that the analyst will be able to deal exhaustively with absolutely all economic impacts. To do so is neither possible nor desirable. The data requirements for the economic models should be limited to only those which are likely to have a significant affect on the user's capital allocation decision. The cost of gathering impact data and exercising the models should in no case exceed the economic advantage to be gained from the analysis.
- (6) Accuracy -- The level of accuracy should not exceed that which is necessary to identify significant differences among alternatives.
- (7) Assumptions made explicit -- The assumptions underlying the methodology and imbedded in the analytical models should be stated clearly.
- (8) Important factors stressed -- Not all elements of the analysis are of equal importance. Those which have greatest significance should be highlighted.
- (9) Uncertainty treated explicitly -- Equipment acquisition decisions are properly based upon anticipated consequences expected to result from the various alternative courses of action. These consequences lie in the future, and hence are uncertain. (Some would argue that the more distant the event, the greater is the uncertainty, but this is not necessarily so.) The extent to which this uncertainty affects the decision should be made explicit so that it may be treated by the decision maker as a separable issue.

- (10) Incorporates efficiencies over time -- The learning curve (improvement curve, progress curve, etc.) has been used for more than forty years to describe the relationship between productivity (cost/quantity) and time. During the initial stages of production, in particular, productivity is improving as the people and machines in the process "learn" to operate more effectively. Economic models should incorporate this effect.
- (11) Reflects real and relative price changes -- Economic impacts should not be expected to remain constant over time, particularly over a long planning horizon. In part these differences result from changes in the relative prices of specific goods and services, popularly known as inflation. Inasmuch as relative price changes may be of significance to the capital allocation decision, they should be incorporated into the analysis. This is especially important for those goods and services for which prices change at substantially different rates.

III. LITERATURE REVIEW

During the summer of 1981 an intensive review of the literature was conducted to identify the extent to which published material describing the economics of robotics is available to prospective users. Sources for review included newspapers and popular magazine articles, anthologies (especially W.R. Tanner's Industrial Robots), professional conference proceedings, government reports, and technical papers of professional societies (especially the Society of Manufacturing Engineers). Consultants working in this field

were also contacted for leads. More than 200 individual items were reviewed; the references appearing in the Bibliography are representative. Of these, only the dozen listed in Exhibit 3 are directly related to economic analyses of robot installations.

The Accounting Method

As indicated in Exhibit 3, these references may be characterized by one or more of several analytical procedures. The accounting method describes economic consequences (costs and benefits) in accounting terms, that is, the effect of the installation on the firm's income and expense accounts. Thus the cost of capital recovery is defined by annual depreciation expense.*

The principal objection to the accounting method is that the opportunity cost is ignored. The opportunity cost, sometimes described as the minimum attractive rate of return, is the return which would be expected from alternative investment opportunities should the specific project proposal not be funded. As described in the literature of engineering economy, the concept of capital recovery (CR) incorporates the opportunity cost as follows:

*(Allan, 1979) includes a separate item for "cost of money" in his numerical example. Thus his approach is a combination of the accounting method and discounted cash flow.

	<u>Accounting Method</u>	<u>Payback Method</u>	<u>Discounted Cash Flow</u>
1. Abraham and Beres, 1978			X
2. Ailan, 1979	(x)		(x)
3. Behuniak, 1979	X	X	X
4. Behuniak, 1980	X	X	X
5. Bublick, 1979		X	X
6. Engelberger, 1979		X	
7. Ernst, 1980 (?)	X		
8. Hanify and Belcher, 1979 (?)			X
9. Heginbotham, 1977		X	
10. Stout, 1973	X		X
11. Tanner, 1978	X	X	X
12. Meisel, 1975		X	

Exhibit 3. Publications Related to Economic Analysis of Robot Installations

$$CR = (C-L)(A/P, i, N) + Li$$

where

C = initial cost

L = net salvage (residual) value at the end of N periods

i = opportunity cost (discount rate)

N = service life of the investment

and $(A/P, i, N)$ = functional form of the algebraic expression

$$= \frac{i(1+i)^N}{(1+i)^N - 1}$$

It may be shown, in general, that capital recovery does not yield the same results as those derived from the popular depreciation methods. To illustrate, consider straight line depreciation. The annual depreciation expense (D) is given by:

$$D = (C_d - L_d)/N_d$$

where

C_d = cost basis

L_d = expected salvage value for depreciation purposes

N_d = depreciable life

To simplify our example, let us suppose that $C = C_d$, $L = L_d$ and $N = N_d$.

It may be shown that the percent error (Δ) is given by

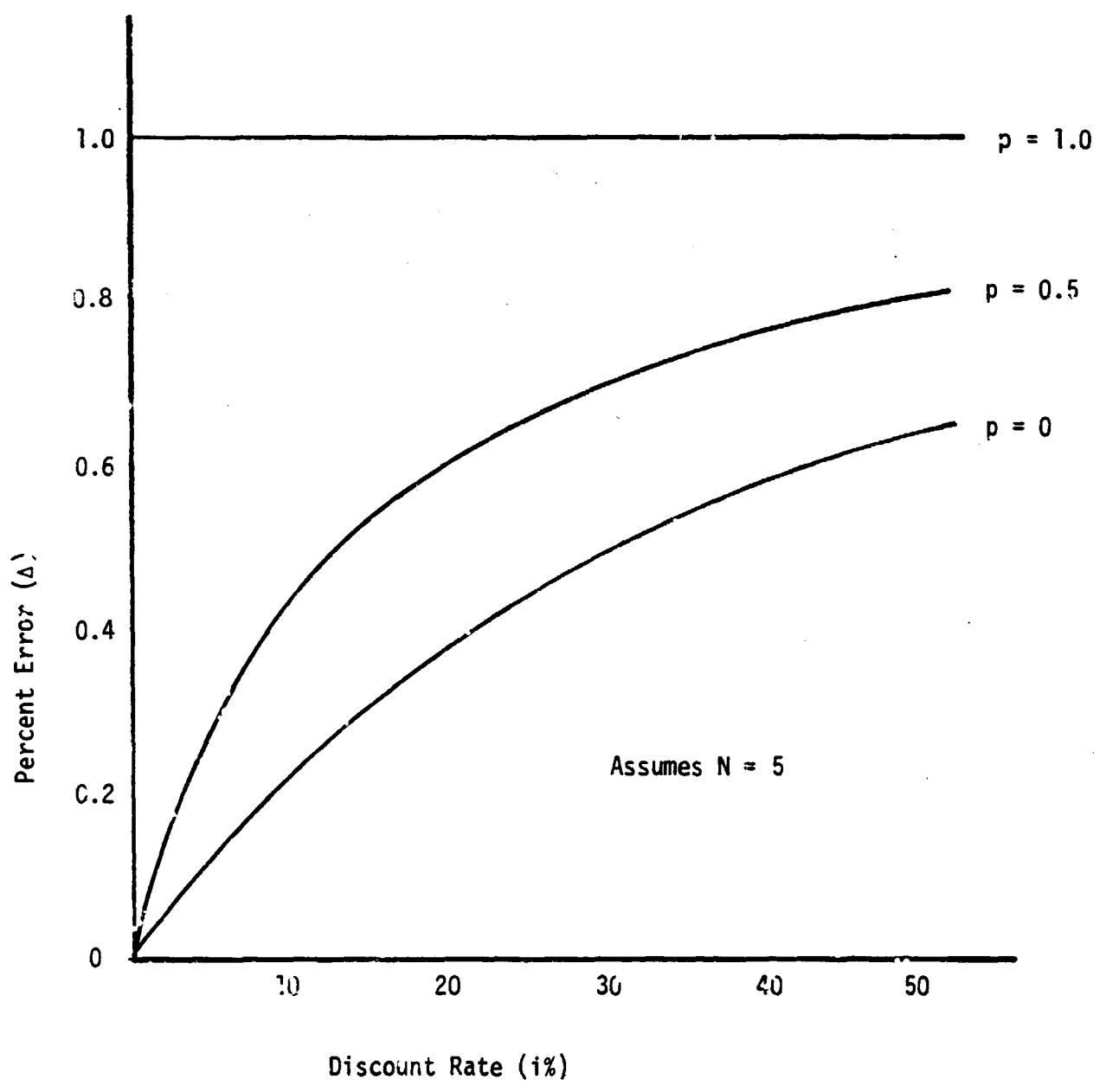
$$\Delta = 1 - \frac{(1-p)/N}{(1-p)(A/P, i, N) + pi}$$

where

$p = L/C$

The percent error (Δ) is shown graphically in Exhibit 4 for $N = 5$. The error increases with the discount rate and the ratio of salvage value to initial cost. When $i = 20\%$ and $p = 0$ (no salvage value), for example, the error is approximately 40%. When $i = 20\%$, and $p = 0.50$, the error is about 62%. (Note that $D = 0$ and $CR = 1$ for all values of N when $p = 1.00$. Thus, in this special case, $\Delta = 100\%$ for all values of N .)

Exhibit 4. Difference between Capital Recovery and Straight Line Depreciation as a Function of Discount Rate



The Payback Method

As illustrated in Exhibit 5, payback (or payout) is the number of periods required for cumulative benefits to exactly equal cumulative costs. Costs and benefits are usually expressed as cash flows, although discounted present values of cash flows may also be used. In either case, the payback method is based on the assumption that the relative merit of a proposed investment is measured by this statistic. The smaller the payback (period), the better the proposal.

Despite the apparent fact that the payback method is widely used in industry, it suffers from serious theoretical deficiencies. The most important of these is that the payback method ignores the consequences of the proposed investment after the period in which payback is completed. This may be shown with reference to Exhibit 6. Here we have two competing projects, Alternatives A and B, with payback for A less than that of B. But it is unlikely that A would be preferred to B since the latter generates far greater net cash flows over the remaining periods in the planning horizon. With very rare exception, payback should not be used as the sole criterion to measure economic efficiency.

Discounted Cash Flow (DCF)

There are a number of variations to the discounted cash flow method: present worth or present value, equivalent uniform annual cost, rate of return (or return on investment), benefit-cost ratio, and the like. They have in common recognition of the timing as well as the amounts of cash flows; money has value over time because of the existence of alternative investment opportunities. When used properly, the DCF variants lead to

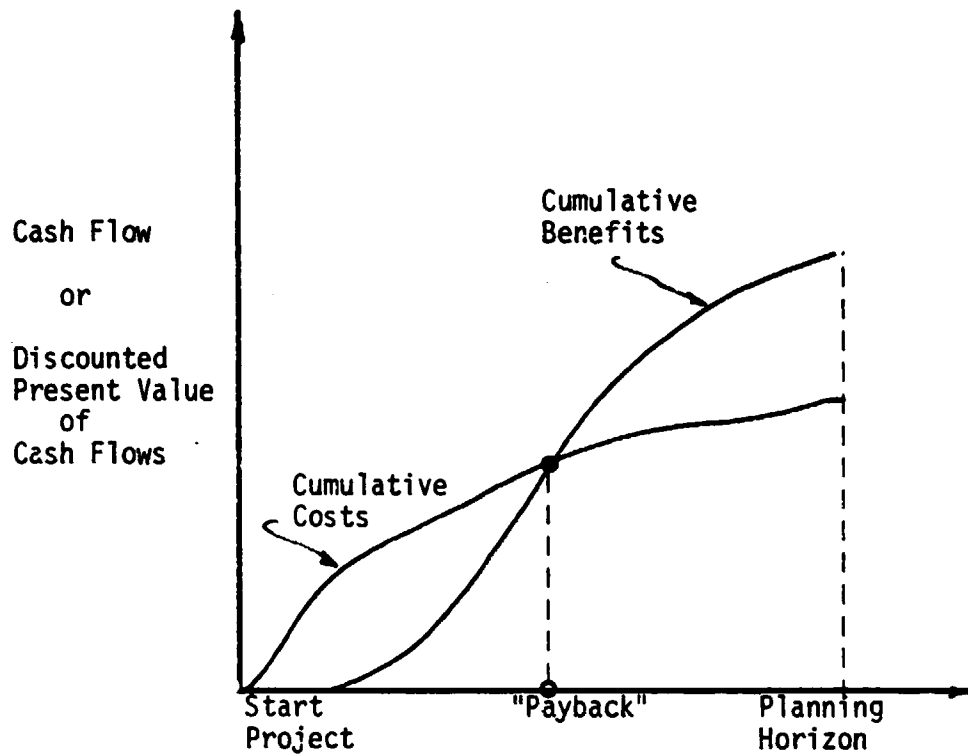


Exhibit 5. Payback (Payout) Illustrated

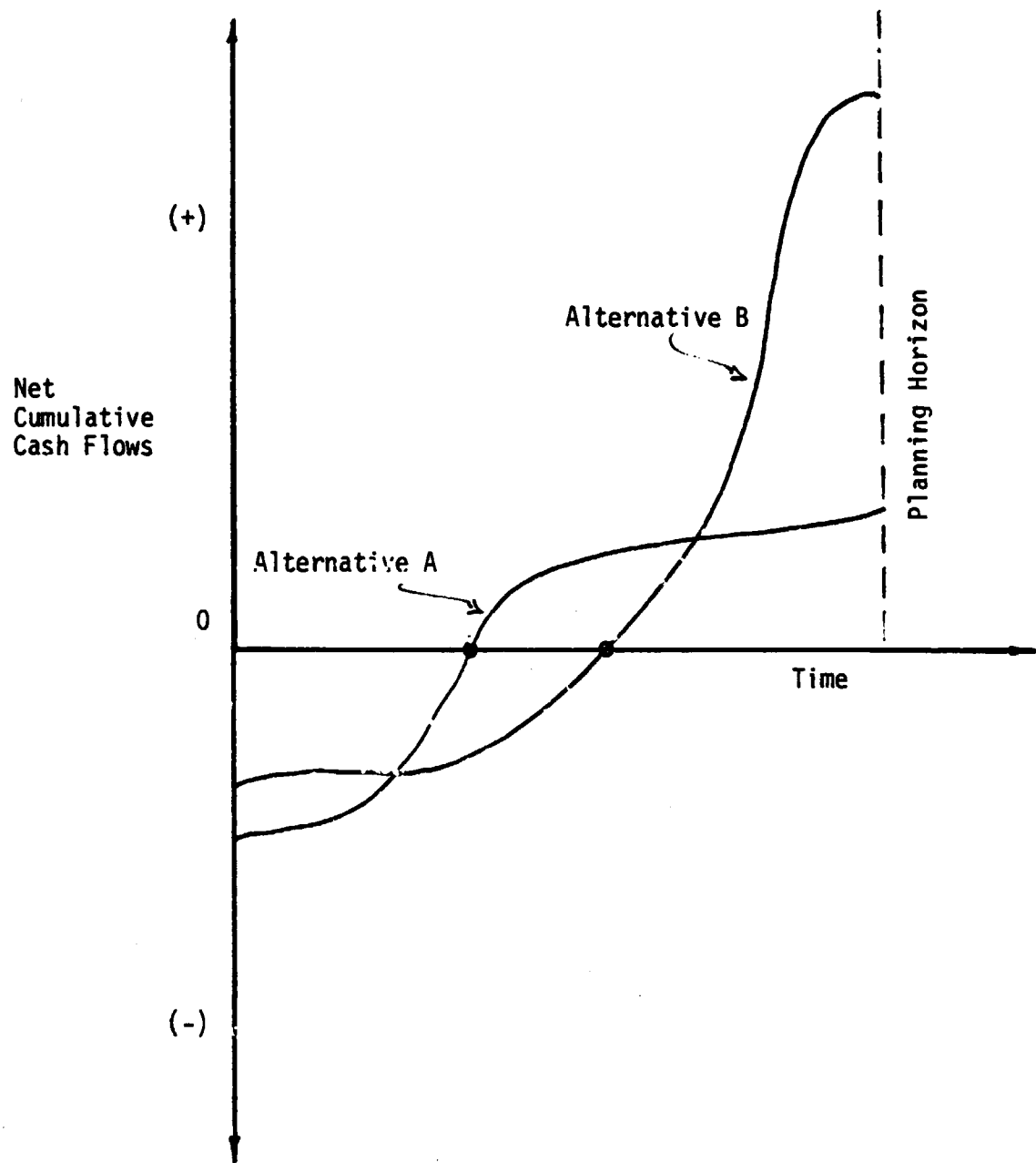


Exhibit 6. Payback for Two Competing Investment Alternatives

consistent solutions.

Our literature search revealed seven publications in which the DCF method is applied to robotics investments. These are:

Abraham and Beres, 1979. Returns on investment (ROIs) are summarized for ten candidate assembly equipments and tooling as well as fifteen separate combinations. Uncertainties are ignored. There is no description of the process by which the ROIs are computed.

Behuniak, 1979. This very brief paper describes economic analyses for three robotic applications: swaging, die casting and painting. Cash flow tables are shown for 5-year planning horizons. Three after-tax figures of merit are shown for each application: (1) payback, (2) return on investment, and (3) discounted rate of return. The formulae, or procedures, for computing ROI and DCRR are not included, and the author's results cannot be verified. Uncertainties are ignored.

Behuniak, 1980. This is similar to the author's earlier paper, except that no cash flows table is included. Results are reported for a die casting robot with initial cost of \$68,000. Payback = 3.4 years; ROI = 24%. (For the die casting robot in the author's 1979 paper, payback = 3.6 years and ROI = 29.2%.)

Bublick, 1979. This paper described an economic analysis for robots used in spray coating and finishing. Cost details are provided for both the manual and robot alternatives. The procedure for determining payback (1.17 years in this case) is detailed. A cash flow table is provided for a 7-year planning horizon, including an adjustment for inflation of 6% per annum. The author claims that ROI = 121% for this application, although the formula for determining ROI is not given. Uncertainties are ignored.

Hanify and Belcher. This is perhaps the most interesting of the papers reviewed. Dennis Hanify is affiliated with the ITT Research Institute; J.V. Belcher is with the Universal Oil Products Company. The paper describes the Industrial Robot Analysis (IRA) program initiated at the ITTRI Robot Technology Center and currently available to clients interested in the technical and economic effects of robot systems. An example problem is included -- a "real world" example -- but cost figures are fictitious since the original data are proprietary. Major costs, their timing and "probable variations" are given for this example problem. Calculations are performed using a general purpose computer program, Economic Systems Analysis (ESA), to compute the prospective rate of return, before or after income taxes. Uncertainty is addressed by evaluating the effects of using the optimistic or pessimistic cost estimates. The procedure used to determine rate of return is not detailed in the paper: no equations, no flow diagram, no computer program. It is not possible, therefore, to critique the methodology further.

Stout, 1973. Assumed costs and benefits for a "typical" project are plotted as a function of time and the ROI calculated. ROI is defined (properly).

Tanner, 1978. The author asserts that "simple 'rules of thumb' and a streamlined cost analysis method can be applied to determine the potential economic return of a contemplated robot installation." A cost analysis form is included with numerical examples illustrating the calculation of:

$$(1) \quad \text{Payback} = \frac{\text{Total Expenditures}}{\text{Total Annual Net Future Savings}}$$

and (2) ROI is that value of r such that:

$$\left\{ \begin{array}{l} \text{Current Value} \\ \text{of Savings} \end{array} \right\} = \sum_{t=1}^N \frac{\text{Total Annual Net Future Savings}}{(1+r)^t}$$

C

The calculation in (2) is described as "ROI for the Depreciated (sic) Cash Flow Method". An alternative calculation for ROI is given by

$$(3) \quad \text{ROI} = \frac{\text{Profit After Taxes}}{\text{Investment Base}}$$

The author's procedure assumes that cash flows remain constant over the planning horizon. Thus there is no cash flow table for other than the "typical" year. Uncertainties are ignored.

IV. LIFE CYCLE (TOTAL SYSTEM) COSTS

As a general principle, the economic consequences of proposed investment in a robotics system should include all significant costs that are likely to result from the investment. ("Benefits" are reductions in costs that may be obtained when comparing any pair of alternatives, and hence are implicit in this principle.) Economic consequences, or impacts, should be measured over the total life cycle of the proposed system.* Impacts should be estimated for each of the candidate systems as well as the existing process.

For our purposes, economic consequences may be grouped into three broad categories, as follows:

1. Plant and Equipment

1.1 The robot(s), including sensors and interlocks

1.1.1 Initial cost

1.1.2 Service life (not a cost)

1.1.3 Residual value (net salvage value) at the end of the service life.

*In the literature of economic analysis, life cycle is frequently referred to as the planning horizon. Strictly speaking, however, the planning horizon may be longer than the system life cycle, especially if significant economic consequences persist beyond the end of the life of the system

- 1.2 Associated tooling
- 1.3 Spare parts
- 1.4 Property taxes
- 1.5 Insurance (property only)
- 1.6 Energy requirements
- 1.7 Tax consequences
 - 1.7.1 Investment credit
 - 1.7.2 Tax savings due to depreciation
 - 1.7.3 Gain (loss) on disposal
- 1.8 Space requirements
- 1.9 Installation (including rearrangement of existing facilities)
- 1.10 Safety equipment (protective clothing, etc.)
- 1.11 Programming
- 1.12 Modification of existing equipment to ensure compatability with robot(s).

2. Operation and Maintenance

- 2.1 Operating labor
 - 2.1.1 Salaries/wages
 - 2.1.2 Fringe benefits (costs to employer)
- 2.2 Maintenance labor (for periodic maintenance)
 - 2.2.1 Salaries/wages
 - 2.2.2 Fringe benefits (costs to employer)
- 2.3 Direct cost of injuries and illness (hospitalization, medical care, etc.)
- 2.4 Absenteeism (cost of lost productivity)

- 2.4.1 illness
- 2.4.2 feigned illness
- 2.4.3 injury
- 2.5 Training
- 2.6 Supervision
- 2.7 Insurance (personnel only)
- 2.8 Overtime (not included in 2.1 and 2.2 above)
 - 2.8.1 Operating labor
 - 2.8.2 Maintenance labor
- 2.9 Labor turnover
 - 2.9.1 Termination
 - 2.9.2 Recruitment
 - 2.9.3 Training
- 2.10 Retooling and set-up costs for batch processing
- 2.11 Maintenance tools and supplies
- 2.13 Documentation (operation and maintenance)
- 2.13 Costs of interrupted production not included in 2.10, especially down time.

3. Product

- 3.1 Required changes in product design
- 3.2 Raw material requirements
- 3.3 In-process inventory
- 3.4 Effects of production rate on:
 - 3.4.1 other plant activities
 - 3.4.2 shipping schedules

3.5 Defective (sub-standard) product

3.5.1 Scrap rate (not a cost)

3.5.2 Net cost of handling and reworking defective product

3.5.3 Costs due to undetected defective product released to customer (e.g., loss of good will, responding to customers' complaints, replacing returned products)

In addition to the above, certain assumptions are required to complete discounted cash flow economic analyses. These include:

4.1 Income tax rates

4.1.1 Federal

4.1.2 State

4.1.3 Local

4.2 Engineering (and consulting) costs not included above

4.3 Cost of capital (to be used as the discount rate)

All cost estimates should be expressed in terms of probability distributions when available and where appropriate. In the absence of the full distributions, however, only the principal statistics (mean, median, range and/or standard deviation) may be estimated. Point estimates or certainty equivalents should be used only when probabilistic estimates are unavailable.

BIBLIOGRAPHY

1. Abraham, Richard G. and James F. Beres. "Cost-Effective Programmable Assembly Systems." In Industrial Robots, Vol. 2, pp. 213-235. Edited by William R. Tanner. Dearborn, MI: Society of Manufacturing Engineers. 1979.
2. Allan, Roger. "Busy Robots Spur Productivity." IEEE Spectrum, September 1979, pp. 31-36.
3. Behuniak, John A. Economic Analysis of Robot Applications. Technical Report MS 79-777. Dearborn, MI: Society of Manufacturing Engineers. (1979).
4. . Planning and Implementation of Robot Projects. Technical Paper MS 80-691. Dearborn, MI: Society of Manufacturing Engineers. (1980).
5. Bublick, Timothy. "The Justification of an Industrial Robot." In Industrial Robots, Vol. 1, pp. 39-45. Edited by William R. Tanner, Dearborn, MI: Society of Manufacturing Engineers, 1979.
6. Clapp, Neale W. Three Laws for Robotocists: An Approach to Overcoming Worker and Management Resistance to Industrial Robots. Technical Paper MS 79-775. Dearborn, MI: Society of Manufacturing Engineers. (1979).
7. . Management Resistance to Industrial Robots. Technical Paper MS 80-690. Dearborn, MI: Society of Manufacturing Engineers. (1980).
8. Engeiberger, Joseph F., "Robots Make Economic and Social Sense", Atlanta Economic Review, July-August 1977. Reprinted in Industrial Robots, Vol. 1. Edited by William R. Tanner. Dearborn, MI: Society of Manufacturing Engineers, 1979.
9. Ernst, Bruce D. "Economic Justification for Industrial Robots." A report prepared for the Corporate Users Group of the Robot Institute of America, One SME Drive, Dearborn, MI. (undated)
10. Estes, Vernon E. "Industrial Robots -- A User's Viewpoint", Proceedings of the AIIE 1979 Fall Industrial Engineering Conference, No. 1979.
11. Ferguson, Fred, "Industrial Robots Do A Man's Job -- and Don't Get Bored", Los Angeles Times, October 12, 1980.
12. Graver, C.A. and J.F. Jenkins-Stark. Life Cycle Cost Model for Comparing AGT and Conventional Transit Alternatives. UMTA-CA-06-0090-76-1, Washington, D.C. Urban Mass Transportation Administration, February 1976.
13. Groover, Mikeli P. "Industrial Robots: A Primer on the Present Technology." Industrial Engineering, Vol. 12, No. 11, November 1980, pp. 54-61.

- C
14. Hannify, Dennis W. and Jay V. Belcher. Industrial Robot Analysis - Working Place Studies. (Reference incomplete)
 15. Heginbotham, Wilfred B. Can Robots Beat Inflation. Technical Paper MS77-756. Dearborn, MI: Society of Manufacturing Engineers. (1977.)
 16. Kolarik, W.J., "Life Cycle Costing and Associated Models", Proceedings of the AIIE 1980 Spring Annual Conference, May 1980.
 17. Kuzmierski, Ted, "Robot Applications in Aerospace Batch Manufacturing", Industrial Robots, Vol. 2, pp. 169-182. Edited by William R. Tanner. Dearborn, MI: Society of Manufacturing Engineers. (1979).
 18. Maer, Gennaro C. Analysis of First UTD Installation Failures. Technical Paper MS77 - 635. Dearborn, MI: Society of Manufacturing Engineers. (1977).
 19. Marks, K.E., H.G. Massey and B.D. Bradley, An Appraisal of Models Used in Life Cycle Cost Estimation for USAF Aircraft Systems, The Rand Corporation, Santa Monica, California, October 1978, (Report R-2287-AF).
 20. Muller, R.K. Managing Costs through A Learning Curve to an Ultimate Target. Technical Paper MM78 - 308. Dearborn, MI: Society of Manufacturing Engineers. (1978).
 21. NASA Study Group on Machine Intelligence and Robotics, Machine Intelligence and Robotics: Report of the NASA Study Group, Executive Summary, Jet Propulsion Laboratory Report No. 730-51, Pasadena, California, September 1979.
 22. Nof, S.Y., J.L. Knight, Jr., and Gavriel Salvendy. "Effective Utilization of Industrial Robots - A Job and Skills Analysis Approach". AIIE Transactions, Vol. 12, No. 3, September 1980, pp. 216-225.
 23. Ottinger, Lester V. "Robots for Manufacturing and Assembly." Proceedings of the 1981 AIIE/MHI Seminar. Norcross, GA: American Institute of Industrial Engineers. (1981).
 24. "Paying the Price for Innovation." The Economist, June 7, 1980, pp. 78-79.
 25. Pollard, Brian W. RAM* for Robots (*Reliability, Availability, Maintainability). Technical Paper MS80 - 692. Dearborn, MI: Society of Manufacturing Engineers. (1980)
 26. Redman, Christopher, Otto Friedrich, Anne Hopkins, et al., "The Robot Revolution", Time, December 8, 1980, Vol. 116, No. 23.
 27. Stout, Thomas M., "Economic Justification of Computer Control Systems", Automatica, Vol. 9, 1973, pp. 9-19.
- C

28. Tanner, William R. "Selling" the Robot -- Justification for Robot Installations. Technical Paper MS78 - 702. Dearborn, MI: Society of Manufacturing Engineers. (1978).
29. "Teaching Robots How to See". The Economist. July 12, 1980, p. 83.
30. Tipnis, V.A. and Steven A. Vogel, "Economic Models for Process Planning". (Reference incomplete).
31. "Trends in Robots". In Industrial Robots, Vol. 1, pp. 81-86. Edited by William R. Tanner. Dearborn, MI: Society of Manufacturing Engineers. (1979).
32. U.S. Department of Defense. Life Cycle Costing Guide for System Acquisitions. Interim Report LCC-3. Washington, D.C., January 1973.
33. _____, Department of the Air Force. Life Cycle Cost Management Program. AF Regulation 800-11. Washington, D.C., February 22, 1978.
34. Van Cleave, David A. "One Big Step for Assembly in the Sky", Iron Age, Nov. 28, 1977. Reprinted in Industrial Robots, Vol. 2. Edited by William R. Tanner, Dearborn, MI: Society of Manufacturing Engineers. (1979).
35. Weekley, Thomas L. A View of the United Automobile, Aerospace and Agricultural Implement Workers of America (UAW) Stand on Industrial Robots. Technical Paper MS79 - 776. Dearborn, MI: Society of Manufacturing Engineers. (1979).
36. Weisel, Walter K., "Comparative Costs of Automation in Die Casting Using Robots", 8th International Die Casting Exposition and Congress, Cobo Hall, Detroit, Michigan, March 17-20, 1975. Reprinted in Industrial Robots, Vol. 2. Edited by William R. Tanner, Dearborn, MI: Society of Manufacturing Engineers. (1979).