THE OPTIMAL DESIGN

OF

DISTRIBUTED COMPUTER CONTROL SYSTEMS.

By

K S Bains.

A thesis submitted for the degree of Doctor of Philosophy in the Department of Electronic and Electrical Engineering of the University of Surrey.

Oct, 1981.
ACKNOWLEDGEMENTS

I would like to thank Dr. R E H Bywater and Mr. W Matley for the patient supervision, invaluable advice and encouragement that was received during the course of this research.

I am indebted to Mr. P Jervis (CERL) for his help, the initial ideas and continuous interest that has been shown in this research project. I would also like to thank Mr. G S Maples, Dr. D A H Jacobs and Mr. D Wells and the staff at Control and Instrumentation division (CERL) for their help. My thanks are also extended to Dr. D C Patel and Dr. N Ream for some useful discussions. My thanks are also due to Shailja, Kamal and Ron for helping me to do the diagrams.

I would like to acknowledge and thank Dr. R E H Bywater, Dr. D A H Jacobs and Mr P Jervis for their help in completing this thesis.

I am grateful to SERC / CERL for their collaborative financial support for this project.
To

Renu
And

My Family
The control of a modern power generating station is a complex task involving the acquisition and processing of a large amount of data. This involves the processing of data from transducers or other inputs which then produce the desired outputs for actuators and displays etc. The advent of small cheap digital data processing systems has made it economically desirable and indeed feasible to implement distributed computer control schemes. The overall control of the station can be achieved by an interconnected set of such computer systems, each computer being at the node of a communications network. The actual control functions are implemented as a number of co-operating modular programs resident in each of the control nodes. It is assumed that the computers will be functionally similar (hardware and software) and that the characteristics of each module (task) such as the C.P.U. loading and the inter-task communication requirements are known a priori.

This work investigates the assignment of these tasks such that the distributed computer network uses the minimum number of computers and that the overall inter-computer communication is minimised. However, this overall objective is influenced by a number of technical and operational constraints which are used to formulate a series of mathematical models that progressively include more aspects of the problem. The application of various linear and non-linear optimisation techniques to the solution of these models is investigated.
Three independant methods of optimisation are investigated to solve the computer control network problem. In each case, the aim is to construct a simple model based on certain aspects of the problem and then extend the model to include all other aspects. A complete mathematical model which applies the standard methods of optimisation is presented. It is claimed that these formulations are original.

It is shown that the complete network design problem is difficult to solve efficiently using standard methods of optimisation, because of the size and the complexity of a practical problem. Hence, the central component of this research has been the development of an algorithm to solve practical network design problems. This algorithm is claimed to be original and is computationally efficient than the standard methods of optimisation for this type of problems. The basic steps of this algorithm are to decompose the problem and then interactively solve the less complex subproblems. It is shown that this algorithm used interactively will provide a feasible solution.

This work contributes to the research into the design of distributed computing systems for process control applications, undertaken by the Central Electricity Research Laboratory (C. E. R. L., Leatherhead, Surrey). In particular it makes a major contribution to the objective of producing a design aid for such computing systems.
1. INTRODUCTION

1.1 Background

1.2 Distributed computer control networks

1.3 Network layout

1.4 Objectives of the research

1.5 Other relevant work

2. NETWORK DESIGN REQUIREMENTS

2.1 Overview

2.2 Processing requirements

2.3 Communications requirements

2.4 Communications network topology

2.5 Network design

2.5.1 General design objectives

2.5.2 Operational constraints

2.5.3 Optimality criteria

3. UNIFORM LOADING OF CONTROL CENTRES

3.1 Heuristic model

3.2 The task distribution algorithm

3.3 Comments

4. MATHEMATICAL MODELS FOR NETWORK DESIGN
4.1 Glossary

4.2 Minimising the number of control centres.

4.2.1 The problem formulation

4.2.2 Methods of solution: linear integer programming

4.2.2.1 Search methods: Branch and Bound

4.2.2.2 Cutting plane methods

4.2.3 Application of the Branch and Bound algorithm

4.2.4 Comments

4.2.5 Other relevant work

4.3 Minimisation of overall data transfer.

4.3.1 Communications Network loading.

4.3.2 Minimising the data transfer

4.3.3 Maximising the data transfer savings

4.3.4 Non-linear programming methods

4.3.4.1 Linearisation

4.3.4.2 Use of a related objective function

4.3.5 Comments.

4.4 Minimisation of the number of links in the network.

4.4.1 Complete problem formulation

4.4.2 Methods of solution
4.5 Conclusions.

5. TASK SCHEDULING USING GRAPHICAL METHODS

5.1 Glossary
5.2 Graphical models
5.3 Sequencing the tasks

5.3.1 Representation of a computational graph
5.3.2 Determination of bounds
5.3.3 Methods of solution
5.3.4 Dynamic Programming
5.3.5 Comments
5.3.6 Other relevant work

5.4 Scheduling of processors with the aid of network flows.

5.4.1 Network flows
5.4.2 Two processor scheduling
5.4.3 Two processor scheduling with constraints.
5.4.4 Comments

6. DEVELOPMENT OF THE ALGORITHM

6.1 Glossary
6.2 Reasons for the Alternative Approach.
6.3 Decomposition of the problem
6.4 Synthesis of the decomposed problem

6.4.1 Problem size reduction
6.4.2 Hierarchical structure.

6.5 Optimal sequential partitions of the reduced problem.

6.6 The algorithm

6.6.1 Summary

6.6.1.1 Stage 1 algorithm.
6.6.1.2 Stage 2 algorithm.
6.6.1.3 Stage 3 algorithm.
6.6.1.4 Mill control problem

6.6.2 Interactiveness of the algorithm

6.6.3 Efficiency of the algorithm

7. CONCLUSION

8. REFERENCES

APPENDIX 1: Uniform loading
APPENDIX 2: Branch and Bound algorithm
APPENDIX 3: Mill control problem
APPENDIX 4: The Computer program.
APPENDIX 5: Publications

Reprint of Authors paper in IEE Conf. Proc.
"Optimal assignment of control tasks in distributed computer systems", Control and its Application, IEE Conf. Publ. no.194, pp. 286-290.
1. INTRODUCTION

1.1 Background

Developments in Large Scale Integration (L.S.I.) technology have changed and are continuing to change the basic framework in which computing systems are designed and constructed. It has become feasible both economically and technically to link small computers in a network to form a powerful and reliable computer system [1, 2]. Hence, a wide range of multi-microprocessor/microcomputer systems are possible [3, 4].

These systems range from loosely coupled distributed computer systems (geographically separated) [5], to tightly coupled multiprocessor systems (functionally separated) [6, 7]. The development of these systems indicates that it would be beneficial to apply them to the control of complex and large processes [8, 9]. However, there are number of difficulties associated with the design of computer networks to control large and complex processes [10]. Hence, there is a need for an organised methodology to determine the appropriate network for a specific application. The work undertaken in this research project was to configure an optimum structure of computers to control a power station or a substation.

Until recently, the control systems employed within the C.E.G.B. power stations and substations had been based on discrete groups of equipment: that is, analogue equipment for regulating control or relay based equipment for sequencing and interlocking. The advantages of this approach have been:
* Functional segregation of the control activities
* Easy incremental commissioning and testing
* Easy scheme extension by adding new equipment.

However, such equipment does not readily lend itself to the implementation of more flexible control schemes. A number of attempts have been made to provide this flexibility by the use of a large digital computer for control applications [11]. For economic reasons, a substantial proportion of the control requirements would be committed to such a computer. This, in turn, leads to the large overheads and potential unreliability associated with the complex software systems. Therefore, a standby machine of equal size is required in case of failure, and the advantages associated with functional segregation are lost. Therefore, the alternative is to functionally sub-divide the control scheme, so that a network of computers is used to control the plant. Then, in case of failure of a single computer only a small proportion of the network (control scheme) needs to be substituted.

1.2 Distributed computer control networks

With the advent of low cost mini and microcomputers it has become feasible to consider implementing the control and instrumentation (C&I) scheme in a network of small computers. Such a scheme may be defined as one in which autonomous groups of equipment, working in parallel, are used to implement individual sub-systems, which together constitute the overall control scheme. This provides the advantages of functional segregation and introduces flexibility. Principally, the advantages of this approach are easy commissioning
maintenance, integration of modulating/sequence control schemes and flexibility for expansion or modification [12].

The envisaged network would be based upon a set of modular intelligent nodes, each node comprising a control centre with its associated plant and communications system interface. Each control centre would be able to handle a portion of the overall control scheme in a reasonably autonomous manner. However, in practice, sub-system autonomy is not absolute, since some co-ordination with other sub-systems is needed. The co-ordination or interactions between the computers could be of sequential control, alarm messages or passing of data for historical records. These interactions, to some extent, influence the choice of the network interconnections (topology) [12, 13].

1.3 Network layout

In determining the appropriate network, a number of conflicting objectives must be resolved. For example, an approach based on one control task per control centre, while admirable in terms of availability, would result in an unacceptable number of control centres and a significant loss in autonomy (a substantial transfer of data between control centres would be continually required). However, if a large number of control tasks are assigned to a single control centre then there is a possibility of overloading the control centre and, in the case of failure, it would mean the loss of control over a large plant area. Hence, the correct choice of network configuration, that is the manner in which the control tasks are
divided between a given number of control centres, is a pre-requisite to the successful implementation of a distributed control and instrumentation scheme [12, 14].

The design criteria could be based on geographical distribution (where the plant covers a large area), parallel computation to increase speed (tightly coupled network) or load sharing to increase the efficiency and the reliability of the network [10]. Generally, the design criteria comprise a number of objectives and a number of constraints to which the network must conform. Hence, it is important that an organised approach is adopted to determine the feasible network structure.

This thesis concentrates on the choice of a network, where load sharing and the reliability of the network are major design criterion. Up to now, the approach has been based on ad hoc methods which are time consuming and liable to errors [12, 15]. It is the intention of this work to formulate design criteria for the network and propose an efficient and scientific approach for determining the structure of the network. The design criteria are based on the control scheme (the software) developed at C.E.R.L for the control of a power station or sub-station.

1.4 Objectives of the research

It is the intention of this thesis to investigate the methods for achieving the following two objectives:-
(1) Determination of the optimum subdivision of the total control scheme into a number of functional groups and the assignment of these groups to appropriate control centres in the (C&I) network.

(2) Determination of the network topology i.e. the number of control centres to use and the nature of the communications system to interconnect them.

In Chapter two, the design requirements of the network are formalized. These requirements are then divided into a set of objectives (what is to be achieved) and a set of constraints (what the restrictions are). These objectives and constraints are formulated into a progressing series of mathematical models to which various optimisation techniques are applied.

In Chapter three a heuristical approach is proposed, to determine the assignment of tasks to control centres so that the control centres are uniformly loaded. Chapter four shows the mathematical programming approach to the design of a network. A mathematical formulation of the problem is proposed which aims to minimise the number of control centres in the network. This formulation is then extended to include the other aspects of the network design problem and the methods of solution of these formulations are discussed. Chapter five introduces the application of graphical methods.
It is shown that the standard methods of optimisation cannot be efficiently applied to a large and a complex problem of network design. Chapter six points out the necessity for an alternative approach. In chapter six an algorithm is developed which is efficient and can be applied to large n-processor scheduling problems of network design. It is shown that the algorithm can be used interactively and would form a very useful tool (design aid) for the design of distributed computer control networks.

1.5 Other relevant work

The implementation of a microcomputer as a single entity controller or a multi-entity controller are discussed in [1 to 13]. The various kinds of networks for distributed computer control are also discussed by Tanenbaum [10], Carson [13] and Kafura [16]. The problem of sequencing a number of tasks to a number of computers has been considered by Ibara et al [17], Rinooy kan [18], Fujii et al [19], Lawler et al [20], Jaffe [21], Eastman et al [22], Cambel et al [23] and Elmaghraby [24] and has some similarity to the problem considered in this thesis. A number of heuristic algorithms have been proposed by these authors, but they cannot be generalised to the problem considered in this thesis. The complexities associated with the general multiprocessor systems have been considered by Graham [25], Manacher [26], Bernstein [27] and Nessett et al [28]. However, the general problem of n-identical processor scheduling is much more complex and the relevant work will be considered in the appropriate sections.
2. NETWORK DESIGN REQUIREMENTS

2.1 Overview

Until recently, power stations or sub-stations control and instrumentation schemes have employed discrete groups of equipment to control and monitor separate items of plant or groups of plant items. In this sense the present C&I schemes can be considered to be 'distributed'. This description in no way reflects the physical location of the equipment, but does emphasise the separation of distinct control functions. Similarly, in considering a microprocessor control network, the emphasis is on the distribution of control processes between a number of intelligent nodes rather than on the physical location of these nodes.

In such a control system, each control process may need to communicate with other processes controlled by other nodes. It is, therefore, imperative to consider the distribution of control processes between nodes. If a large number of control tasks are resident in a single node then the loss of that node can result in having to employ manual control over a large group of plant items. However, if jobs that share a significant quantity of common data are resident in different nodes then a loss of communication between nodes could restrict the action of control jobs in those nodes.

Therefore, it is important that the network design philosophy should consider partitioning of the control scheme into a set of functional groups and the group assignment to appropriate control centres so as to avoid the above situations. The purpose of this section is to formalise a set of requirements on which the network design philosophy is to be based. This involves a basic understanding
of the software with which the scheme is to be implemented and the knowledge of the interactions between the controlled processes. This would allow us to evaluate the processing load requirements of each control task and the communications load requirements of each task.

2.2 Processing requirements

The software framework that is required to support C&I tasks in either single or multi-computer configurations naturally subdivides into a number of different activities that are required in each control centre (e.g. communications, data handling, control algorithms etc.). Each activity can be implemented as a separate software module. Each module consists, principally, of a sequence of calls to a set of standard, defined, subroutine blocks. As a result, the CPU load of each module is readily quantifiable [11].

For example, let \( N(t, b) \) be the number of times task \( t \) calls block \( b \) and let \( T(b) \) be the time to execute block \( b \), then the time to execute task \( t \) will be given by:

\[
T(t) = \sum_{b} N(t, b) \times T(b)
\]

Since each task will be regularly executed at a fixed time interval it is possible to estimate the processing load that a task will impose on a control centre. Let \( F(t) \) be the frequency at which task \( t \) is to be executed, then \( L(t) \), the processing load contributed by task \( t \), is given by:

\[
L(t) = F(t) \times T(t)
\]
Consequently $L(c)$ the processing load on control centre "c" is given by:

$$L(c) = \sum_{b} N(t,b) * T(b) * F(t)$$

Hence, we can determine the loading of each task on any computer.

2.3 Communications requirements

The processing system at each node will comprise a set of common software modules to perform executive and communications functions and frequently encountered utility functions [12], plus a particular set of programs to perform the control functions at that node. This, of course, implies that the control centres are designed to act in a reasonably autonomous manner. However, there are a number of situations in which it is necessary to transfer information between control centres. Some of the situations that call for information transfer are as follows [29, 30]:-

(1) Supervisory centres in the control network may need to provide other control centres with new set-point values (e.g. fuel/air ratio, superheater outlet temperature etc.) or may need to initiate a start-up or shut-down sequence. Conversely, a control centre would then need to inform the supervisory centre of sequence completion or failure, and perhaps of various alarm conditions.
(2) Various control centres may be required to return certain items of information to a supervisory centre for long term optimisation calculations or general data logging purposes.

(3) Programs resident in different centres may need to communicate directly for co-ordination of particular processes.

(4) Faulty transducers could prevent successful completion of sequence operations or could disrupt a modulating control loop unless some mechanism for over-riding input data is provided. A convenient means of introducing over-rides is to permit an operator to enter the over-ride via a suitable terminal at a control centre resident near the control room. The over-ride could then be transmitted to the appropriate control centre in the network.

(5) In the control and instrumentation scheme, if it is necessary for more than one job to access an item of data within a particular node, or even for a job to access an item of data in a remote node, then the concept of a distributed data base is introduced into the system. This is a part of the general data management facility [29].

At this stage, we consider that each control process may need to communicate with other control processes or with higher level co-ordinating processes resident in separate nodes. Hence, we will
need to consider the load imposed on a particular control centre in communicating with another control centre. We will assume that the communication is in the form of messages where a message comprises of destination information, the data itself and some form of self checking code [29]. Then, the simplest way to estimate the communications load is to derive the maximum number of messages that each task in each control centre can initiate on any one of its runs and then derive the total number of messages that can be initiated by a control centre in any one time interval. Hence, we can evaluate the communications load imposed when the information flow between tasks is known.

2.4 Communications network topology

The communications load imposed on the control centres is, to some extent, dependent on the topology of the communications network. For example a point to point system (fig. 2.1) implies that each node must be capable of receiving and then accepting or transmitting a message. This would impose a certain basic overhead on the memory requirements for message buffering and the control processors time for executing character reception, transmission and message routing logic. If the network took the form of a simple ring (fig.2.2), then each node would be connected to two full duplex links and the control processor would face an extra load when all four channels would be in use. On a bus communication system (fig.2.3) each message would be available simultaneously to all nodes—though the majority of the nodes may have no interest in that message.

These topologies have been investigated [29] and it was concluded that, before a sensible choice of network can be made, the
The following two factors require investigation:

(a) The overall data flows involved in the complete control network.

(b) An evaluation of the existing load on the control centre at each node.

These two factors are included in the next section in order to formalise a general network design philosophy. We will assume that all nodes are connected either directly or by other nodes.
Fig 2.1 General point to point linking

Fig 2.2 Ring structure

Fig 2.3 Nodes connected by a communication bus
2.5 **Network design**

It is intended to evolve a design philosophy for designing a distributed computer control scheme which is, as far as possible, independent of the hardware with which the scheme is to be implemented [15, 29]. The general network design objectives on which the design philosophy is to be based include the following:

2.5.1 **General design objectives**

(1) A design procedure should be followed for large systems which will minimize the number of control centres hence minimizing hardware capital and maintenance costs of the system.

(2) The overloading of any control centre or any communications link should be avoided.

(3) The design should avoid high data transfer rates between control centres.

(4) It is desirable to share the processing load evenly between control centres.

(5) To minimise the effect of control centre failure in terms of both plant safety and availability and the manual control burden imposed on the operator.
If a back-up control scheme exists for a plant area (or variable) then the control loops to implement back-up control should be in a separate control centre from the main control scheme.

Duplication of large blocks of inputs to more than one control centre should be avoided (if there are a number of tasks sharing a common data (input) it would be desirable to put these tasks in one computer to save duplication of data in other computers).

The general design objectives listed above can be divided into those that act as optimality criteria for the problem and those that act as constraints on the network.

2.5.2 Operational constraints

(1) No control task can be shared between control centres.
(2) Each task must be assigned once and only once to any one control centre.
(3) Each control centre must not be loaded above its specified capacity limit.
(4) The data link between control centres should not exceed their link capacity limit.
(5) The tasks which must be kept together should be in the same control centre.
(6) The tasks which must not be kept together should be assigned to separate control centres.
2.5.3 Optimality criteria

(1) Minimisation of the overall data transfer between the computers.

(2) Minimisation of the number of data links between computers.

(3) Minimisation of the number of computers.

(4) Uniform loading of the computers.

The objectives mentioned in the optimality criteria are in order of relative importance i.e. minimisation of overall data transfer is more important than the minimisation of data links between the computers (sub-system autonomy). The operational constraints have equal importance because they are either due to the hardware or the software with which the scheme is to be implemented.

So, the network design criteria have been formalised. Now, the aim is to optimise these objectives, simultaneously satisfying the operational constraints on these objectives. The approach adopted here is to formulate a simple model based on a single objective and its appropriate constraints and then extend the model to include all other objectives and their constraints.

As mentioned in chapter one, a slightly similar problem is that of sequencing a number of tasks on computers to optimise a single objective e.g. that of minimising the delay [17 to 28]. However, we have more than one objective and a large number of constraints. Three different types of optimisation methods are investigated, in each the formulations of the models are shown and the applicability of the methods of solution is discussed.

------------------o0o------------------
3. UNIFORM LOADING OF CONTROL CENTRES

3.1 Heuristic model

The first, and most simple, model is a heuristic approach which aims to assign the control tasks in such a way as to use the minimum number of control centres and achieve as near uniform a loading of those centres as possible. The communications aspects of the problem are not included in this model.

 Principally, the idea is to evaluate the minimum integral number of control centres by summing up the task loads and dividing the total by the capacity limit set on the control centres. It is assumed that the capacity limit on all the control centres will be the same. Then, the set of tasks is partitioned into that number of groups. The partitioning is based on the following requirements:

1. The set(s) of tasks that are to be kept separate should not be in the same group(s).
2. The set(s) of tasks that are to be kept together should be in the same group(s).
3. The total of task loads in each group should not exceed the capacity limit set on the control centres.
4. Each task must be assigned to one and only one group.

Note that in (1) the number of sets of tasks that are to be kept separate should not be greater than the total number of control centres and in (2) the sum of tasks in that set(s) should not exceed the capacity limit (valid combination) set on the control centres.

This approach can be represented by the following flowchart:
START.

\[ N_t = \text{NUMBER OF TASKS}. \]

READ \( N \) TASK LOADS.

READ LOAD LIMIT.

SEGREGATE (VALID) TASKS.

COMBINE (VALID) TASKS.

DISTRIBUTE THE TASKS.

CHANGE LOAD LIMIT.

IS THE SOLUTION OPTIMUM?

NO.

INTERCHANGE TASKS.

IS THE DISTRIBUTION EVEN?

NO.

YES.

YES.

STOP.

Fig 3.1 Algorithm flowchart
3.2 The task distribution algorithm

To accomplish this, an interactive program has been written [31]. Conceptually, in the first part of the program, control tasks are arranged in descending order of magnitude with respect to load imposition. Then, the preferred grouping (providing it is permissible) is carried out interactively with the preferred segregation. The task list is then scanned from top to bottom and allowable tasks are then grouped to maximise load up to or equal to the permitted load limit. These tasks are then eliminated from further consideration. The procedure is repeated on the reduced task list until all the tasks have been assigned. A mode of flexibility is introduced such that the load limit implied by the designer can be adjusted so that the tasks are distributed between the nearest theoretical integral value of the number of control centres.

In the second part of the program the overall load distribution of control centres is compared. It is desirable to have even loading between centres and an autonomous mechanism has been introduced to accomplish this. Principally, it is as follows:

If we define $L_i$ (for $i=1, 2, \ldots, cc$) as the difference between the load limit and load assigned to the $i$th control centre and $cc$ is the number of control centres. Then, our goal will be to have $L_i$ (for all $i$) differing as little as possible.

If we express $L = \frac{\sum_{i=1}^{cc} L_i}{cc}$

![Equation](image)
Then, ideally, we require $L_i = L$ (for all $i$). However, in practice the range of task loads is such that approximality rather than equality is sought.

3.3 Comments

The advantage of this approach is that it provides a feasible solution to the problem and it is easy to program on a computer.

The interactions between tasks and the communications link capacities between control centres have been ignored in this approach because the communications constraints are such that optimality can not be guaranteed (a large number of permutations and combinations would have been ommitted from consideration). However, the purpose of this approach is to find a feasible solution so that it can either be improved or compared with the more conventional methods of optimisation.

An example of a task list for a generating plant [32] and the corresponding task distribution obtained using this algorithm is shown in appendix 1. Heuristic methods for sequencing problems have been proposed by Elmaghraby [24] and Ibara et al [17].
4.1 Glossary

\( n \) = number of tasks

\( m \) = number of computers

\( U \) = set of tasks \((1, 2, \ldots, u, \ldots, n)\)

\( J \) = set of computers \((1, 2, \ldots, j, \ldots, m)\)

\( X_{uj} = 1 \) if \( u \)th task is on the \( j \)th computer , otherwise 0

\( Y_j = 1 \) if \( j \)th computer is active , otherwise 0

\( Z_{ij} = 1 \) if there is a link between the \( i \)th computer and the \( j \)th computer , otherwise 0

\( C_{uv} \) = The inter-task communication between task \( u \) and task \( v \).

\( P_u \) = The processing requirements of the task \( u \)

\( C_{Cj} \) = The processing capacity limit on the \( j \)th computer

\( L_{ij} \) = The link capacity on the link from the \( i \)th computer to the \( j \)th computer

\( R_{ij} \) = The value of the data transferred from the \( i \)th computer to the \( j \)th computer

\( W_j \) = The number of tasks to be assigned to \( j \)th computer

\( q_{u} \) = The lowest labelled task on a computer
In this section we consider the application of standard optimisation methods to find the optimal assignment of tasks. The network design objectives and the operational constraints are expressed as a series of dependent mathematical equations, the solution to which defines the optimal assignment of the tasks.

The initial and the simplest model is the minimisation of the number of control centres subject to the non-overloading constraints. Gradually, more aspects of the network design problem are introduced into our mathematical models. Finally, the complete model for the network design problem is proposed. Methods for the solution of each model and for the complete model are discussed.

4.2 Minimising the number of control centres.

4.2.1 The problem formulation

In this model, the objective pursued is to minimise the number of centres required to implement the overall control scheme, but the communication constraints are not taken into account. The operational constraints imposed on this objective are:-

1. Each control centre must not be loaded above its specified loading capacity limit.
2. No control task can be shared between control centres.
3. Each task must be assigned once and only once to any one control centre.
Mathematically, if we consider the assignment of a set of tasks \( U = (1, 2, 3, \ldots, u, \ldots, n) \) having known processing requirements \( P = (p_1, p_2, \ldots, p_u, \ldots, p_n) \) to a set of computers \( J = (1, \ldots, j, \ldots, m) \), each computer having a capacity limit on the maximum power available \( CC_j \). We define a binary variable \( X_{uj} \) such that

\[
X_{uj} = \begin{cases} 
1 & \text{if task } u \text{ is assigned to } j \text{th computer} \\
0 & \text{otherwise.}
\end{cases}
\]

Then we can express the operational constraints (1), (2), and (3) listed above by the equations 4.1 and 4.2 respectively.

\[
(4.1) \quad \sum_{u=1}^{n} p_u X_{uj} \leq CC_j \quad \text{for } j = 1, \ldots, m
\]

\[
(4.2) \quad \sum_{j=1}^{m} X_{uj} = 1 \quad \text{for } u = 1, \ldots, n
\]

Although some of the operational constraints have been introduced into our model, there is still no direct relationship to the objective. However, if we introduce another variable \( W \) such that no more than \( W \) tasks can be assigned to any one computer (if necessary \( W = n \)) thus generating a dummy constraint i.e.

\[
(4.3) \quad \sum_{u=1}^{n} X_{uj} \leq W Y_j \quad \text{for } j = 1, \ldots, m
\]

Where \( Y_j \) is 1 if the \( j \)th computer is active, otherwise 0.

Finally, imposing the bounds
Hence, the objective function is:

\[
\text{(4.6) } \text{Minimise } \sum_{j=1}^{m} Y_j
\]

i.e. we would like to minimise the number of active computers.

The basis of this formulation (and in fact defined by (4.4) and (4.5)) is on the assumption that no task can be shared between computers but tasks can only be added integrally. Hence, in accordance with our assumption, no individual task load can be greater than the maximum capacity of any computer. In its standard format, the complete model is thus:

\[
\text{(4.6) } \text{Minimise } \sum_{j=1}^{m} Y_j \text{ (active computers)}
\]

Subject to:

\[
\text{(4.1) computer loading } \sum_{u=1}^{n} P_u X_{uj} \leq CC_j \text{ for } j = 1, \ldots, m
\]

\[
\text{(4.2) all tasks assigned } \sum_{j=1}^{m} X_{uj} = 1 \text{ for } u = 1, \ldots, n
\]

\[
\text{(4.3) } \sum_{u=1}^{n} X_{uj} \leq \sum_{j=1}^{m} W Y_j \text{ for } j = 1, \ldots, m
\]

\[
\text{(4.4) } X_{ij} = 0 \text{ or } 1
\]

\[
\text{(4.5) } Y_j = 0 \text{ or } 1
\]
In this model, constraints (4.1) to (4.3) are all defined by linear equations and the objective function is also linear. Therefore, we can solve this problem using the methods of linear programming [33]. However, the equations (4.4) and (4.5) require the solution to be an integer. Therefore, we need the methods that would convert the continuous solution to an integer solution. These methods are referred to as integer programming methods [33, 34].

4.2.2 Methods of solution: linear integer programming

Integer programming techniques that are widely used are [34, 35]:

(1) Search methods

(2) Cutting plane methods

4.2.2.1 Search methods: Branch and Bound

This type of method is motivated by the fact that the integer solution space can be regarded as consisting of a finite number of points. In its simplest form, search methods seek to enumerate "all" such points. This would be equivalent to simple exhaustive enumeration. What makes search methods more promising than simple exhaustive enumeration, however, is that the technique can be developed to enumerate only a portion of all the candidate solutions, while discarding the remaining points as non-promising. Clearly, the efficiency of the resulting "search" algorithm depends on the power of techniques that are developed to discard the non-promising solution points.
Search methods primarily include implicit enumeration techniques and Branch and Bound techniques [35, 36]. In general, the solution space of an integer program can be assumed to possess a finite number of feasible points. A straightforward method for solving zero-one integer problems such as ours is to exhaustively (or explicitly) enumerate all such points. In this case, the optimal solution is determined by the point(s) that yield the best (maximum or minimum) value of the objective function.

The obvious drawback of the above technique is that the number of solution points may become impractically large, with the result that either the solution cannot be determined in a reasonable amount of time or the storage of data exceeds the capacity of the machine solving the problem [36]. The idea of the implicit (or partial) enumeration calls for the consideration of only a portion of all possible solution points whilst automatically discarding the remaining ones as non-promising. Implicit enumeration does not consider "complete" binary combinations. Rather, it starts with one (or more) variables being fixed at binary values and then it gradually "builds" the solution by augmenting new variables at fixed values. In the course of each augmentation, it becomes evident that (complete) solution points can be discarded without being considered explicitly. The idea is the basis of Land and Doig's Branch and Bound algorithm [37].

4.2.2.2 Cutting plane methods

The first finite cutting plane algorithm was developed by Gomory for the pure integer problem [36]. It is shown [35] how the cuts can be constructed systematically from the simplex tableau which
provides the basic continuous solution. This method was later extended to mixed integer (real and integer) problems.

Firstly, the problem is solved as a linear programming problem without paying any attention to the integer constraints. If each variable in the optimal solution is an integer then the problem is solved. If not a constraint is generated by truncating the fractional parts of the continuous solution for each integer variable and these fractional parts are employed to construct this special constraint. The addition of this constraint turns the optimal and feasible (non-integer) solution into an optimal but infeasible solution. The next step is to employ the dual simplex method [35] to arrive at a new optimal feasible solution. If this is an integer solution then the problem is solved. Otherwise, the procedure is repeated, adding these constraints, one at a time until the optimal integer solution is obtained. It has been shown that this process converges in a finite number of iterations [35].

Previously obtained results show that large problems cannot be easily solved using cutting plane techniques [36]. The convergence of these algorithms is slow and, for large problems may result in a nonfeasible solution [36]. If the structure of the problem is to be exploited, then, for our problem the enumeration techniques readily lend themselves to efficient utilisation, since it is predominantly a zero-one linear integer problem. Therefore, we will consider the application of the Branch and Bound method. The essential features of the Branch and Bound method to constrained optimisation are described by Lawler and Wood [38].
4.2.3 Application of the Branch and Bound algorithm

The algorithm used to solve some test problems, based on our model, is known as the BBMIP [39]. The code of this algorithm is running on a CDC 7600 machine at the University of Manchester. BBMIP employs a Branch and Bound algorithm implemented by Shareshian [39] and is based the Land and Doig [37] method extended to solve mixed integer programming problems.

It was noticed that this algorithm, despite finding an optimal solution quickly, failed to recognise the solution as the optimum and continued to explore a large number of other solutions. This was due to the equal importance given to all computers in our formulation. This caused the algorithm to explore a large number of solutions. This degeneracy is reduced by replacing the unity weighting factor in the objective function by an 'ordered' weighting factor, e.g.

\[
\text{Minimize } \sum_{j=1}^{m} (2)^{j} Y_j \\
\text{OR} \\
\text{Minimize } \sum_{j=1}^{m} (j)Y_j
\]

This implies that, if the computers are numbered from 1 to m, then the cost of each computer is proportional to its number. Since m is an upper bound on the number of computers that can be used the algorithm is implicitly made to choose less than or equal to m computers in an ascending order of numbers, if k computers are to be used where k<m then the computers numbered from 1 to k will be used.

It was further noticed that, if an absolute minimum bound on
the number of computers was specified, then the algorithm converged much more rapidly i.e.

\[ \sum_{j=1}^{m} Y_j \geq N \]

where \( N \) is the minimum integer number of computers needed. This bound contributed to a considerable reduction in the number of iterations executed by the algorithm, since it reduced the area to be searched for the solutions.

4.2.4 Comments

This algorithm performed satisfactorily on test problems of rather small size [40]. The execution time is determined by the proximity of the mixed integer solution to the continuous solution values. It was noticed that imposing an upper bound on binary variables increased the execution time while a good upper bound on the objective function reduced it (since it discarded non-acceptable integer solutions).

The workspace memory requirements in (words) for this algorithm is:

(a) Central memory for continuous solution

\[ = 4(N) + (M+1)(N+1) + M + 1 \]

(b) Plus central memory for Branch and Bound \( = 8 \times \text{INTVAR} \).

Where

\[ M = \text{Number of constraints} \]
\[ N = \text{Number of variables} \]
\[ \text{INTVAR} = \text{Number of integer variables} \]
The largest problem that we were able to solve on the CDC machine, using this algorithm, was for \( n = 20 \) and \( m = 5 \). The explanation of the algorithm, taken directly from Shareshian [39] is given in appendix 2.

4.2.5 Other relevant work

V. Balachandaran [41], Srinivasan and Thompson [42] have proposed similar algorithms, based on branch and bound techniques to solve problems such as ours. Gertsbakh and Stern [43] have proposed an enumerative method to determine the assignment of jobs to computers. However, all these methods treat the problem as of sequencing a number of jobs to a given set of computers. Their objectives are slightly different and there are no capacity constraints such as ours. Therefore the use of their methods is restricted to trivially small problems. The efficient construction of a branch and bound type of algorithm to various other problems is shown by Klee[44], Balas[45] and by Gen et al [46].
4.3 Minimisation of overall data transfer.

4.3.1 Communications Network loading.

In this section we consider the interactions between tasks and then introduce the concept of communications load on the computers in handling these interactions. The objective is to find an assignment of the tasks so that data transferred between computers is minimal. The operational constraints imposed are the same as in our previous model.

In this formulation we again consider a set of tasks \( U = (1, \ldots, u, \ldots, n) \) having known processing power requirements \( P = (p_1, p_2, \ldots, p_u, \ldots, p_n) \). Furthermore, we consider the interactions between the tasks i.e. the communications between the tasks. The communications, at this stage, can be thought of as a package of data that a task sends to another task for processing, updating or storing. We assume a computer has the ability to transmit, receive and pass on a message. Then, our aim is to distribute these tasks to a limited number of computers so that the minimum of data is transferred between computers. We will assume that, if two tasks are co-resident in one computer, then the communication load between those tasks is zero.

If we consider the communication between the tasks to be in the form of messages, then we can say the interaction between the tasks is as shown in figure 5.1.
Fig 4.1 Inter-task communication

$C_{uv}$ is the number of messages that task $Tu$ transmits to task $Tv$. We will assume that the communications load is directly proportional to the number of messages being transmitted. This can now be represented in a $n$ by $n$ matrix $C$: $C_{uv}$ for $u=v =1, \ldots, n$.

$$C = \begin{pmatrix}
C_{11} & C_{12} & \cdots & C_{1n} \\
C_{21} & C_{22} & \cdots & C_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
C_{n1} & C_{n2} & \cdots & C_{nn}
\end{pmatrix}$$

The terms, Communications Load and Data Transfer will be used interchangably in this context but they basically refer to the task interactions. Since we are interested in the total data transfer
between two tasks if they reside in separate computers, we can reduce
the number of elements of this matrix that we need to consider. If we
assign \( C_{uv} := C_{uv} + C_{vu} \) for all \( u < v \), for \( v=1, 2, \ldots, (n-1) \), and
assign \( C_{vu} = 0 \) for all \( v < u \), for \( u = 2, 3, \ldots, n \). Then we only need
to consider the upper triangular half of this matrix since the lower
triangular half elements are all zero. Furthermore, in accordance
with our assumption, the elements on the main diagonal of this matrix
are zero.

4.3.2 Minimising the data transfer

Mathematically, the objective can be expressed as follows:
given an \( n \) dimensional square matrix \( C \), find an \( n \) by \( m \) solution matrix
\( X = X_{uij} \) such that \( Z \) in (4.7) is minimum.

\[
Z = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} C_{uv} X_{ui} X_{vj}
\]

(intertask data transfer)

Here, the first two summation signs basically mean "sum over all the
interactions between all the computers but not within the same
computer". The variables \( X_{uij} \) have the same meaning as in the last
model (section 4.1) and the operational constraints (1), (2) and (3)
are similarly defined i.e.
This model represents the necessary conditions but not the sufficient conditions, because it does not consider the link capacity, the message receiving and passing-on-loads between the computers. But, at this stage the emphasis is on building a simple model and satisfying the fundamental requirements.

4.3.3 **Maximising the data transfer savings**

In the above formulation, we have considered the minimisation of data transfer between the computers. Let us consider the converse of this. If tasks Tu and Tv reside in the same computer, then the data transfer is saved by a factor of \(( C_{uv} + C_{vu} )\). Hence, we can think in terms of maximising our "savings".
Proposition:

\[ \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} C_{uv} X_{ui} X_{vj} + \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} \sum_{j \neq i}^{m} C_{uv} X_{ui} X_{vj} = \text{constant}. \]

Proof:

Let

\[ \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} C_{uv} X_{ui} X_{vj} + \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} \sum_{j=i}^{m} C_{uv} X_{ui} X_{vj} = g(z) \]

Then

\[ \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} C_{uv} \left[ \sum_{i=1}^{m} \sum_{j=1}^{m} X_{ui} X_{vj} + \sum_{i=1}^{m} \sum_{j=i}^{m} X_{ui} X_{vj} \right] = g(z) \]

But this is

\[ \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} C_{uv} X_{ui} X_{vj} = g(z) \]

The L.H.S. represents the sum of all the elements of \( C \), which is the total data transferred, if each task were assigned to a separate computer.

Hence, the alternative formulation is :-
Maximise $Z = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} C_{uv} X_{ui} X_{vi}$

Subject to:

(4.1) **Computer loading** \( \sum_{u=1}^{n} P_u X_{uj} \leq C_{Cj} \) for \( j = 1, \ldots, m \)

(4.2) **All tasks assigned** \( \sum_{j=1}^{m} X_{uj} = 1 \) for \( u = 1, \ldots, n \)

(4.4) \( X_{uj} = 1 \) or 0

We note that the number of combinations considered in (4.8) is much less than in (4.7), although the constraints are the same.

Both of these models represent non-linear objective functions with linear constraints. Furthermore, the set of dependent variables impose integrality constraints, i.e. they are discrete variables. The objective function is classified as a quadratic function [33]. Hence, the models are constructed with a view towards implementation by quadratic programming techniques [47, 48].
4.3.4 Non-linear programming methods

These models are very similar to the generalised quadratic assignment problem [48, 49]. This parallelism of similar objective functions naturally assumes that the methods of quadratic programming, such as, Wolfe's and Beale's [48], can still be applied. However, the underlying concept of definiteness and convexity behind these methods causes difficulty. These methods require the matrix C to be positive semidefinite [39, 48].

For definiteness, an n by n real symmetric matrix A is positive definite if $X'AX$ is positive for an all real non-zero n column vector X. Similarly A is negative definite if $X'AX$ is negative for an all real non-zero n column vector X. If A is a square matrix with at least one diagonal entry, $A_{ii} = 0$ such that $A_{ij} + A_{ji} > 0$ for some j, then A is indefinite. The matrix C in our formulation is square, non-zero and has $C_{ii} = 0$, for all i. Thus it is indefinite.

If A is positive definite then the quadratic form $X'AX$ is convex. Similarly, if A is negative definite, then the quadratic form $X'AX$ is concave. But, in our case, the matrix C is indefinite therefore Z is non-convex [48].

Because of the inapplicability of these methods, we consider reformulating our problem in an equivalent form so that efficient methods can be applied. For non-linear, non-convex problems with discrete variables the usual approach is described in references [49, 50].

(1) Linearisation
These approaches can be applied to our problem as follows:

4.3.4.1 Linearisation

In the previous formulation we had:

\[
\text{(4.8) Maximise } z = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{j=1}^{m} C_{uv} X_{uj} X_{vj} \\
\text{Subject to:}
\]

\[
\text{(4.1) } \sum_{u=1}^{n} P_{u} X_{uj} \leq C_{Cj} \quad \text{for } j = 1, \ldots, m
\]

\[
\text{(4.2) } \sum_{j=1}^{m} X_{uj} = 1 \quad \text{for } u = 1, \ldots, n
\]

\[
\text{(4.4) } X_{uj} = 1 \text{ or } 0
\]

This quadratic assignment problem can be converted to a linear
problem by making the change of variables

\begin{equation}
Yuv = \sum_{j=1}^{n} Xuj Xvj
\end{equation}

To obtain the equivalent problem. Hence, find \( Yuv = 0 \) or \( 1 \) that maximizes \( Z \), where \( Z \) is

\begin{equation}
Z = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} Cuv Yuv
\end{equation}

The next step is to express the constraints in this equivalent form. If we consider that the tasks \( u, v \) and tasks \( v, r \) are in the same computer then it necessarily follows that tasks \( u, r \) must be in the same computer and each task must be assigned. Hence, we can replace (4.2) with

\begin{equation}
\begin{aligned}
Yuv + Yur - Yvr &\leq 1 \\
Yuv - Yur + Yvr &\leq 1 \\
-Yuv + Yur + Yvr &\leq 1
\end{aligned} \quad \text{for all } u<v<r.
\end{equation}

Now, if another variable \( q = 0 \) or \( 1 \) is introduced such that, if the \( u \)th labelled task is the lowest numbered task assigned to a computer then \( qu = 1 \), otherwise \( qu = 0 \). Then we can introduce two new constraints of the form:

\begin{align*}
\end{align*}
(4.12) \[ \sum_{u=1}^{v} Y_{uv} + q_{v} \geq 1 \quad \text{for } v = 2, \ldots, n \]

(Note that \( u = v \) is a trivial case since \( Y_{uu} = 0 \). Hence for \( v = 1 \) we will just have \( q_{1} \geq 1 \) since at least one computer must be active. Thus we replace the inequality with the equality.)

(4.13) \[ \sum_{u=1}^{v} Y_{uv} + (v-1) q_{v} \geq (v-1) \quad \text{for } v = 3, 4, \ldots, n \]

(Again we note that for \( v = 1, 2 \) this constraint will be represented by (4.12))

The constraint in (4.12) implies that if \( k \) computers are active then all computers labelled from 1 to \( k \) are active. The constraint (4.13) is very similar to (4.3) but, in this case, it implies that \( q \) takes the values of 0 or 1. The capacity constraint (4.1) is replaced by its equivalent form:

(4.14) \[ \sum_{v=1}^{n} \sum_{v \neq u} P_{v} Y_{uv} + P_{u} q_{v} \leq C_{C_{u}} \quad \text{for } u = 1, \ldots, n \]

Finally, we can impose an upper bound on the number of computers to use as follows

(4.15) \[ \sum_{u=1}^{n} q_{u} \leq \text{(some upper bound)} \]

Hence, the equivalent form of the problem is:
Maximize $z = \sum_{u=1}^{n-1} \sum_{v=1}^{n} C_{uv} Y_{uv}$

Subject to:

\[
\begin{align*}
Y_{uv} + Y_{ur} - Y_{vr} &\leq 1 \quad (4.11) \\
Y_{uv} - Y_{ur} + Y_{vr} &\leq 1 \\
-Y_{uv} + Y_{ur} + Y_{vr} &\leq 1
\end{align*}
\]

\[\text{for all } u < v < r\]

\[
\sum_{u=1}^{v} Y_{uv} + q_{v} \geq 1 \quad \text{for } v = 2, 3, \ldots, n \quad (4.12)
\]

\[
\sum_{u=1}^{v} Y_{uv} + (v-1) q_{v} \geq (v-1) \quad \text{for } v = 3, \ldots, n \quad (4.13)
\]

\[
\sum_{v=1}^{n} \sum_{u=1}^{v} P_{u} Y_{uv} + P_{u} q_{u} \leq C_{Cu} \quad \text{for } u = 1, \ldots, n \quad (4.14)
\]

\[
\sum_{u=1}^{n} q_{u} \leq \text{(some upper bound)} \quad (4.15)
\]

\[
Y_{uv} = 0 \text{ or } 1 \quad (4.16)
\]

\[
q_{u} = 0 \text{ or } 1 \quad (4.17)
\]
When the optimal Yuv are found, the Xuj are obtained by inspection. For example, if Yuv = 1 then Xuj = Xvj = 1 for some arbitrary j.

4.3.4.2 Use of a related objective function

Lawler [49] has suggested that an indefinite matrix can be changed to a positive semidefinite matrix by adding a positive constant k to the main diagonal in the C matrix. However, this means that the objective function can increase by a factor of n*k hence altering the true objective therefore the solution obtained may not be the global optimum. Similarly Carlson and Namhauser [51] have proposed a heuristic approach to solve this type of problem. However their approach is specific to assignment problems. They have proposed [51] that if a feasible assignment is first found, then the possibility of obtaining a better solution is explored by heuristically interchanging the assignments, initially derived until a near optimum solution is obtained. The idea is that, if the initial assignment provides the upper bound on the objective, and interchanging the assignment reduces it, then that is a better solution. Their approach cannot be directly applied because our problem is more rigidly constrained than a pure assignment problem. Very limited computational experience is reported [49] for methods of this type and it is not sufficient to base any general conclusions concerning the general applicability. A problem of backboard wiring proposed by Steinberg [52], which has similar objective function as that of ours, illustrates the difficulties of obtaining a global optimum solution to these problems.
4.3.5 Comments.

The linearised equivalent form of the problem was solved using the Branch and Bound method described earlier. For \( n<7 \) the method was quite efficient and the convergence of the algorithm was within satisfactory limits. However, for \( 7<n<12 \) the algorithm failed to converge and for \( n>12 \) the problem became too large to handle. This can be expected because the number of constraints generated in the equivalent form increases cubically i.e.

\[
\text{Number of constraints} = \frac{(n!)/(n-3)!}{2} + n + (n-2) + n + 1 \\
\approx \frac{1}{2} \cdot (n-1)^{3} + n
\]
4.4 Minimisation of the number of links in the network.

In this section, we extend our mathematical model to include the inter-linking of computers. In our last model, we introduced the concept of communication between the tasks if two tasks resided in different computers and we assumed that a link existed between each and every pair of computers in the network. Now, as discussed in chapter 2, our objective is to minimise the number of links within the network as well as to minimise the overall communications. This minimisation of the number of links is restricted in that the remaining links required in the network should not be overloaded.

This objective, and the constraint imposed upon it can easily be introduced into our model developed in the last section. Since this objective relates the other objectives (e.g. minimising the number of computers and minimising the data transfer) considered earlier, then clearly, it is the basis of the complete network design problem. In proposing this complete mathematical formulation of the complete network design problem, we have considered the objectives listed in chapter 2 to have a relative importance. The relative importance given to these objectives in this formulation is as in the following order ((a) being most important).

4.4.1 Complete problem formulation

(a) Minimise the overall data transfer.
(b) Minimise the number of links.
(c) Minimise the number of computers.
The constraints imposed upon these objectives are:
(d) Computer links should not be overloaded.
(e) Computers should not be overloaded.
(f) Certain tasks should be kept together.

(g) Certain tasks should not be kept separate.

(h) All tasks must be assigned.

In this formulation, we have assigned an arbitrary weighting factor to the objectives e.g. objective (b) is twice as important as (c) and (a) is twice as important as (b). The constraints (d), (e), (f), (g) and (h) have equal importance.

We assume that, each link between the computers is a bidirectional and the total data transferred from one computer to the other is the sum of that in each direction as explained in section (4.3). Furthermore, there is no indirect means of transferring of data i.e. data meant for computer k from computer i does not go via j. Hence, the complete formulation of the problem can be stated as;

Let

\[ X_{uj} = 1 \text{ if } u \text{th task is on the } j \text{th computer} , \text{ otherwise } 0 \]

\[ Y_j = 1 \text{ if } j \text{th computer is active} , \text{ otherwise } 0 \]

\[ Z_{ij} = 1 \text{ if there is a link between the } i \text{th computer and the } j \text{th computer} , \text{ otherwise } 0 \]

\[ C_{uv} \text{ = The inter-task communication between task } u \text{ and task } v. \]

\[ P_u = \text{ The processing requirements of the task } u \]

\[ C_{Cj} = \text{ The processing capacity limit on the } j \text{th computer} \]

\[ L_{ij} = \text{ The link capacity on the link from the } i \text{th computer to the } j \text{th computer} \]

\[ R_{ij} = \text{ The value of the data transferred from the } i \text{th computer to the } j \text{th computer} \]

\[ W_j = \text{ The number of tasks to be assigned to } j \text{th computer} \]
Minimise \[ \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} \sum_{j=1}^{m} \left( 4C_{uv}X_{ui}X_{vj} + \frac{(2Z_{ij})}{2} + \right) \sum_{i=1}^{m} Y \]

\text{Subject to :}

(4.19) \text{link limit} \quad \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} C_{uv}X_{ui}X_{vj} \leq L_{ij} \quad \text{for } i=1,\ldots,m

and \( j \neq i, j=1,\ldots,m \)

(4.20) \text{link loading} \quad \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} C_{uv}X_{ui}X_{vj} = R_{ij} \quad \text{for } i=1,\ldots,m

and \( j \neq i, j=1,\ldots,m \)

(4.21) \text{computer loading} \quad \sum_{u=1}^{n} Pu_{u}X_{uj} \leq C_{Cj} \quad \text{for } j=1,\ldots,m

(4.22) \text{all tasks assigned} \quad \sum_{j=1}^{m} X_{uj} = 1 \quad \text{for } u=1,\ldots,n

(4.23) \text{no. of tasks} \quad \sum_{u=1}^{n} X_{uj} \leq W_{Yj} \quad \text{for } j=1,\ldots,m

(4.24) \quad R_{ij} = Z_{ij}L_{ij} \quad \text{for } i=1,\ldots,m

and \( j \neq i, j=1,\ldots,m \)

(4.25) \quad X_{uj} = 1 \text{ OR } 0

(4.26) \quad Y_{j} = 1 \text{ OR } 0

(4.27) \quad Z_{ij} = 1 \text{ OR } 0
In this model, the equation (4.18) expresses the total minimisation of three objectives with a given weighting factor which denotes the relative importance. The constraint (d) is expressed by (4.19) and the constraint (e) is expressed by (4.21). The equations (4.20), (4.23) and (4.24) are to hold the values of Y's and Z's to 1 or 0. The equation (4.22) satisfies the constraint (h). The equations (4.25) and (4.26) and (4.27) impose the integrality constraints.

The constraints (f) and (g) are made implicit in the construction of communications matrix C. If two tasks u and v are to be kept together, then the element Cuv is made very large and if u and v are to be kept separate then Cuv is set to 0.

4.4.2 Methods of solution

This formulation shows that the problem has a non-linear objective function and a mixture of linear and non-linear integer constraints. We have already seen in section (4.3) that even a simpler model of this problem did not readily lend itself to be solved by the conventional calculus based methods of optimisation. Sahni et al. [53] and Garey et al. [54] have shown that this type of problem falls into the group (non-deterministic time of solution) for which there is no efficient method of solution. Methods based on heuristics have been proposed by Geoffrion [55], Paviani et al [56] and Celebiler [57] for assignment type problems but they do not guarantee a global solution.
However, if an attempt is made to linearise this formulation as in section (4.2), then there is a dramatic increase in the number of variables and in the number of constraints. Thus, the linearised form would only be useful for trivially small problems. Therefore, a practical problem of large size and such complexity cannot easily be solved by the standard methods of optimisation.

4.5 Conclusions.

It has been shown that mathematical programming techniques can be applied to the problem of optimal network design. We began by constructing a simple model to minimise the number of computers in the network. Next, the inter-task communication was introduced and another model was presented, which attempted to minimise the overall communication within the network. Finally, the inter-linkage between the computers was introduced and a complete formulation of the network design problem was presented. The methods of solution for each model were investigated.

It is concluded that the complete network problem is so complex that the standard methods of optimisation would only be useful for small problems.

The apparent inapplicability of these methods to a complete problem warrant the need for an efficient method to decompose the problem. The aim of this method would be to decompose the problem either in terms of reduction in size or reduction in complexity.

---------------oOo-----------------
5. Task Scheduling Using Graphical Methods

5.1 Glossary

**GRAPHS**

Index terms = Incidence matrix, Reduced graph, Dependancy graph, Dynamic Programming

\[ T = \text{The incidence matrix of a graph} \]
\[ E = \text{The earliest precedence} \]
\[ L = \text{The latest precedence} \]
\[ J = \text{A completed stage in the graph} \]
\[ P = \text{The current stage in the graph} \]
\[ D = \text{The cardinality of the set} \]
\[ y = \text{The number of processors} \]
\[ x = \text{The number of active processors} \]

**NETWORK FLOWS**

\[ N = \text{Number of nodes in the network} \]
\[ A_{ij} = \text{The weight on the arc connecting node } i \text{ to node } j \]
\[ s = \text{The source node} \]
\[ t = \text{The sink node} \]
\[ C(i,j) = \text{The value of the cut separating node } i \text{ and node } j \]
Definitions:

The **Weight** or **Capacity** of an arc represents the maximum amount of flow that can go through the arc.

A **Feasible Flow** is a set of arc flows assigned to the arcs such that:

1. The capacity of the arcs is not exceeded.
2. The flow is conserved at every node except source and sink.

A **Value** of a flow is the net flow out of source node.

A **Maximal Flow** is a feasible flow whose value is the maximum among all other feasible flows.

A **Cut** of a network is a minimal set of arcs, the removal of which separates the source from the sink.

A **Minimal Cut** of a network is a cut which separates the source from the sink and no other cut is a subset of this cut.

The **Weight** or **Capacity** of a cut is the sum of the capacities of the arcs in the cut.

**Minimum cost assignment** is one which minimises the objective.

**Minimum Feasible cost assignment** is an assignment which minimises the objective and satisfies the constraints.
5.2 **Graphical models**

In this model, the nodes of the graph represent tasks and the node weights represent the processing requirements of the tasks. The arcs connecting these nodes indicate the interactiveness between the tasks. The arcs can be directed or undirected depending on our objective. We pursue the following two objectives:

(5.1) **Sequencing the tasks to minimise the idle time of the control centres.**

(5.2) **Assignment of the tasks with regard to the link capacities connecting the control centres.**

The model based on our first objective requires the arcs between nodes to be directed and is known as the graph model [58]. The model based on our second objective is known as the network flow model [59]. We consider the construction of these models and look at the methods of solution that are available to solve them.

5.3 **Sequencing the tasks**

In this model, we consider the processing times required by the tasks rather than their load imposition. It is shown in section 2 that the load imposed by a task is determined by the product of the
processing time required by that task and the frequency of its execution. If we assume that each task is to be executed once per second (which will be the case for our problem [32]), then our aim is to sequence the tasks so that each control centre is able to repeat the cycle after one second. This is because we assume that the interactive nature of the tasks is such that it warrants the need for a sequential operation then data will be immediately available where required. Therefore, the basis of this model is that no task has either to wait or work on invalid data. Hence, the added objective is to sequence the tasks so that the idle time of each control centre is minimised. This is achieved by ordering the tasks in terms of precedence. This implies that task i precedes task j (or i<j ) if there is a directed arc from node i to node j. Hence, if task j is processed after task i then the data required by task j is immediately available. This ordering can be associated with a dependency graph. The dependency graph is defined as the graph with no local or global loops, a unidirectional system [58]. Hence, our objective is to develop a schedule using the dependency graph so that it gives a description of the tasks to be done by each control centre as a function of the processing times. The schedule must not violate any of the precedence relationships or the operational constraints of chapter 2.

5.3.1 Representation of a computational graph

In this section we consider the construction of the dependency graph and its representation as a parallel computational graph [58]. First determine the matrix T, from the task interconnection graph (reduced graph) [60]. The elements of T are defined to be either 1 or 0. An element Tij is a "1" if and only if
the jth node has one of its inputs as the output of node i; otherwise $T_{ij}$ is a "0".

To determine the parallel task graph the reduced graph is partitioned into $E$ and $L$ partitions [58]. The task set $T$ can be partitioned into $s$ subsets ($E_1, E_2, \ldots, E_s$) called the earliest (or $E$ or column) precedence partitions such that

$$E = \bigcup_{j=1}^{s} E_j = \bigcup_{k=1}^{n} T_k$$

Where $T = [T_1, T_2, \ldots, T_n]$ is the set of tasks and $E_j \cap E_k = 0$ when $j \neq k$.

The meaning of the $E$ precedence is as follows: $E_1$ is the subset of tasks that can be initiated and executed in parallel at the very start. $E_2$ is the subset of tasks that can be initiated and executed in parallel after the tasks in $E_1$ are done and so on. Thus, the elements of $E_j$ represent those tasks that can be processed at the earliest time and corresponding to level $j$ even though some of them can be postponed.

The $E$ precedence partitions can be determined very simply from the connectivity matrix $T$. From $T$ a column (or columns) containing only zeroes is located. The tasks corresponding to these all zero column(s) form partition $E_1$. Next, both the column(s) and row(s) corresponding to the tasks in $E_1$ are deleted from $T$. Using the remaining portion of $T$, tasks that correspond to the columns
containing only zeroes is then located. The second E-precedence is E2. This procedure is repeated to obtain other E precedence partitions until no more columns remain in the T matrix. It has been shown by Ramamoorthy [58] that this procedure is valid for all reduced graphs.

The tasks can also be partitioned into L subsets called the latest (or L or row) precedence partitions (L1, L2, .., Ls) such that

\[
\text{s} \quad \text{n}
\]
\[
L = \bigcup_{i=1}^{s} L_i = \bigcup_{k=1}^{n} T_k
\]

Where \( T = (T_1, T_2, .., T_n) \) and \( L_i \cap L_k = 0 \) when \( i \neq k \).

\( L_i \) represents the subset of the tasks that must be executed at least by the end of level \( i \). The L partitions are obtained by performing the precedence partitioning on the transpose of the matrix T. This process is also called the row partitioning.

We will only consider the task graphs in which the process starts out as a single stream, spawns a number of parallel tasks and finally terminates as a single completed task in the end. Any other variations can be taken accounted for without loss of generality by a single entry single exit graph. If there are multiple input nodes (multiple entry points), add a new node to the graph, allowing branches from it to reach the entry points. Such a node will correspond to a computation requiring zero time. Similarly where there is more than one exit node, a new node is added such that branches converge into it from every exit node.
5.3.2 **Determination of bounds**

The parallel task graph is used to determine the bounds associated with our problem.

1. The minimum number of computers required to process the task graph in the least time is a low bound on the minimum number of computers required to accomplish overall control.

2. The minimum time required to process the task graph is a low bound on the minimum time required to accomplish overall control.

The minimum time to process the graph is determined from the path with the greatest length from the entry node to the exit node of the task graph. The length of the path $T(1), T(2), \ldots, T(i), \ldots, T(m)$ is defined as $t_1 + t_2 + \ldots + t_i + \ldots + t_m$, where $t_i$ is the time required to process task $i$.

The minimum number of computers required to compute the task graph in the least possible time is bounded below by

$$\max \forall j \quad \{ \mid L_j \cap E_j \mid \}$$

and above by

$$\min \{ \max \forall j \mid L_j \mid \text{ OR } \max \forall j \mid E_j \mid \}$$
Proof [58].

The tasks can be completed in the minimum number of levels with at most

$$\text{Max } \forall i \{ |L_i| \} \quad \text{OR} \quad \text{Max } \forall i \{ |E_i| \}$$

Computers, which ever is smaller. Since $L_i \cap E_i$ is the smallest number of essential tasks at level $i$.

$$\text{Max } \forall i \{ |L_i \cap E_i| \}$$

specifies the absolute minimum number of computers.

The technique used to schedule the tasks within these bounds is called Dynamic Programming.

5.3.3 Methods of solution

Methods of dynamic programming have been used to solve a number of scheduling problems [61]. Basically, these methods interpret the problem as a multistage decision problem. A stage corresponds to a specific milestone on the solution route [62]. Bellman's principle [62] of optimality provides the basis for linking an $n$-stage process so that the optimal decision is reached in a stepwise manner, proceeding from one step to another.
The implication of this optimality principle is that, starting at a given stage, the optimal policy for the remaining stages depends only on the state* at that starting stage and not on how the system arrived at that state. This optimality principle leads us to the conclusion that, if we can build the optimality of each stage on the previous one then ultimately, we can find the optimality of any state as a function of the initial state of the system.

Conceptually, dynamic programming treats the problem in the following manner [63]:

1. Start with a small segment of the problem
2. Find an optimal solution to the segment
3. Enlarge the problem to include the next stage. Based on the optimality of (2), determine the optimal policy for these two stages.
4. Enlarge the problem to include three stages. Determine the optimal policy as in (3) and continue to progress to the final stage.

* At each stage a system is found in one of several possible conditions or states, described by the state variables [63]. The decision as to how to move from one state to another is, usually, a decision of how to transform the system from one state to another. Each stage is associated with either finite or infinite states.
We note that this partitioning of the problem is implicit in our model i.e. the E and L partitions of the parallel task graph. Hence, in our case we determine the optimal schedule at each level with respect to the last level and proceed to the next level until all the tasks have been assigned. This approach is based on the approach adopted by Ramamoorthy [58] to a problem similar to ours.

The states of the dynamic program are classified as the intervals 1, 2, 3.. where the i th interval corresponds to the i th unit of time. Zero time is the instant at which the graph starts getting processed. If there are n tasks in the graph, indexed 1, 2, .., n, and if ti is the processing time required for the i th task, then a state in the dynamic program will be described by the sets (J, P)

Where

\[ J = j_1, \ldots, j_q; \quad j_v \in 1, 2, \ldots, n \quad \text{for all } v \] and

\[ P = p_1(r_1), p_2(r_2), \ldots, p_u(r_u); \quad p_v \in 1, 2, \ldots, n, \quad \text{for all } v \]

rv are assumed to be integer where 0<rv<tv

State (J, P), at the i th interval represents a set of tasks that are completely or partially processed in the first i units of time. If J =j_1, j_2, \ldots, j_q and P =p_1(r_1), p_2(r_2), \ldots, p_u(r_u) then tasks j_1, j_2,
.., jq have been completed at the end of the i th unit of time; tasks pl, p2, ..pu will have started being processed but will not be completely processed at the end of the i th unit of time and task pv requires rv additional units of time to be completely processed, v =1, 2, .., u. In general, there is one state, at the i th interval, for each distinct (J, P). It is shown [58] that several of these states can be ignored by the principle of dominance.

The dominance of task i over task j at level k is defined as:-
if we define the set of predecessor P(i) tasks of i as P(i)=v \ v<i, in other words, task i can only be started if and only if all tasks in P(i) completed. The set of successor tasks of task i is written as S(i) and defined as S(i) = v \ i<v. Now, we can say that task i dominates task j if and only if

\[ P(i) \subseteq P(j) \]

\[ S(i) \supseteq s(J) \]

It can be proved [58] that there exists an optimal schedule of computers represented by a sequence of feasible states (J1, P1), (J2, P2), .., (Jf, Pf) where no state (Ji, Pi) for i=1, 2, .., f, is dominated.

5.3.4 Dynamic Programming

The algorithm developed to schedule the tasks is similar to
the one developed by Ramamoorthy et al. [58], but extended to consider the capacity constraint on the computers and the interactions between the computers. The sequential decision process involved is based on the following heuristics:

Heuristic A: Let there be \( y \) tasks \( S_v \in S \) where \( r_v > 1 \). Then during all successor states of \((P, S)\) which are in the \((i+1)\)th interval, \( y \) computers will be occupied processing these tasks. However, it may be preferable to keep some of the \((k-y)\) computers idle through the \((i+1)\)th unit interval. Let \( D \) be the set of tasks (other than in \( S \)) that are for processing in the next interval. Then in the \((i+1)\)th interval we may choose to process any subset of \( D \) with cardinality (the number of elements in \( D \)) not exceeding \((k-y)\) (the number of free computers). Let the cardinality of \( D \) be \( q \).

If there are \( x \) computers working on tasks in \( D \) in the \((i+1)\)th interval \((0 \leq x \leq (k-y))\) and \((k-(y+x))\) computers idle, then there are \( \binom{q}{x} \) different alternatives for choosing the tasks in \( D \). Hence, there are a total of

\[
\sum_{x=0}^{x=x'} \binom{q}{x} \quad \text{where} \quad x' = \min\left\{ q, (k-y) \right\}
\]

Heuristic B: If there is more than one task competing for the same computer, choose the task which receives the greater amount of data from that computer.
5.3.5 Comments

A small (n=10) problem was formulated using this approach and it was solved using dynamic Programming. The problem and its solution are presented in appendix 3. Although this approach is very easy to mechanise on digital computers its generalisation to our problem cannot be made at this stage. This is because a related objective function has to be introduced and the optimal solution based on that objective function is not necessarily the overall optimum solution. This is the case with the problem that was solved using this approach and three feasible solutions were obtained.

5.3.6 Other relevant work

The basic applications of Dynamic Programming to scheduling problems have been proposed by Bellman [61]. More specific to our work, Sahni [64] has developed algorithms for scheduling of independant tasks on computers to minimise the total execution time. Muntz and Coffman [65] have considered pre-emptive scheduling (task shared by more than one computer) on two computer systems. Segal [66] has successfully applied Dynamic Programming to a problem of dynamic file assignment in a network of computers to solve for some optimal strategies. Although their work considers simpler single objective problems, there are some similarities with our objectives.
5.4 Scheduling of processors with the aid of network flows.

In this model we assume that the arcs connecting the nodes can be directed or undirected and each arc is weighted i.e has a number associated with it. This number represents the number of messages that need to be passed between the nodes. This model is then equivalent to a network in which a certain commodity is flowing. Hence, we consider that we wish to partition this network into a number of groups so that flow from one group to another is restricted by the capacity of the link connecting these groups. If each group forms the set of tasks that are assigned to one computer then the aim is to partition the network, satisfying the operational constraints of section 2. The algorithms that are available are known as the Network Commodity Flow Algorithms [69].

Network commodity flows is a relatively new approach credited much to the work of Ford and Fulkerson [59]. This approach has been fairly successful in its varied applications [67, 69]. The usefulness of this approach to scheduling problem was first highlighted by H.Stone [67]. It was first implemented by H.Stone [67] to solve a two processor module assignment problem. He considered this problem in two separate formulations. In his first paper [67] he considered minimising the interprocessor communications and in his second [68] paper the critical loading of the processors was also taken into account.

In this section, we first, outline the theory of network flows and define the necessary terminology. Then the application of network flows is shown (as implemented by Stone[67]). Although the problem considered by Stone is much simpler than ours, it is nevertheless very useful in understanding application of network flow methods. We also
review the work of Rao et al [70] who have considered the same problem as Stone, but with a constraint. One of the processors is constrained to have limited memory. From these applications of network flows to a two processor network design problem, we will investigate, possible methods for use in a general n processor network design problem.

5.4.1 Network flows

A Graph consists of a set of nodes (vertices, points) and a set of arcs (edges, links, branches, lines) connecting these nodes. The arcs can be directed or undirected.

![Network diagram](image)

Fig 5.1 Network of nodes.

Commonly, Ni is used to indicate the node i and Aij is used to indicate the arc leading from Ni to Nj. The term Network is used when every arc Aij has associated with it a positive integer bij called its capacity[69]. There are two special nodes, predefined in a network, one is called a Source denoted by Ns and the other is called the Sink denoted by Nt. However, in some cases, each node in the network can act as source or a sink, in which case the network is known as a Multiterminal Network [69].
The network can be considered as a pipeline system with arcs representing the pipelines, the source being the inlet of water, the sink being the outlet of water and all other nodes being the junctions between the pipelines. The capacity of each arc can be considered to represent the cross sectional area of the pipeline. In such a system we are interested in the maximum flow that can be put through from the source to the sink. In a network a source can generate an infinite amount of flow while the sink can absorb an infinite flow.

This analogy is of course, not quite accurate for our case, but is used merely to present a picture of the network flows. In our case the numbers on the arcs represent the inter-task communication and we will assume that these numbers represent the capacities of the arcs. What we would like to determine is the maximal flow from source to sink for a nominated source and sink. This maximal flow will be shown to be the minimal communication between those two modules, if these modules are assigned to different computers.

The necessary definitions are as follows:
The **Weight** or **Capacity** of an arc represents the maximum amount of flow that can go through the arc.

A **Feasible Flow** is a set of arc flows assigned to the arcs such that:

1. The capacity of the arcs is not exceeded.
2. The flow is conserved at every node except source and sink.

A **Value** of a flow is the net flow out of source node.

A **Maximal Flow** is a feasible flow whose value is the maximum among all other feasible flows.

A **Cut** of a network is a minimal set of arcs, the removal of which separates the source from the sink.

A **Minimal Cut** of a network is a cut which separates the source from the sink and no other cut is a subset of this cut.

The **Weight** or **Capacity** of a cut is the sum of the capacities of the arcs in the cut.
Max-Flow Min-Cut theorem: [Ford and Fulkerson [59]]

For any network, the maximal flow value from the source to the sink is equal to the capacity of a minimal cut separating the source and the sink.

This theorem is of fundamental importance in determining the optimal partitioning of a network (minimal cut) into two groups. One group includes the source and the other includes the sink. Ford and Fulkerson [59] has developed an algorithm based on this theorem for evaluating the minimal cut in the network. The algorithm basically consists of \( O(|V|^2) \) steps. First, a path is located from the source to the sink and then the maximal flow value is determined. This equals the smallest capacity of an arc in the path. Second, this maximal flow is assigned to all the arcs in the path. Third, the arcs on which the flow equals the capacities are cut by the minimal cut.

This algorithm is referred to as the Max-Flow Min-Cut Algorithm [59]. A fuller exposition of this algorithm is also given by Hu [69] and Hadley [71]. This algorithm is computationally very efficient as shown by Dinic [73], Karzanov [74] and Edmonds and Karp [75]. We will consider the application of this algorithm to problems slightly similar to our own, by reviewing the work of Stone [67] and Rao [70].
5.4.2 Two processor scheduling

Stone [67] considered the following problem:

In a distributed computing system a modular program must have its modules assigned among two processors so as to avoid excessive interprocessor communication while taking advantage of specific efficiencies of each processor in executing some program modules.

In the assignment of modules to processors, there are two types of costs: The cost of execution of a module on a processor and the cost of interprocessor communications.

The network model in fig 5.2 represents modules as nodes of the network, with arcs between the nodes if and only if the modules communicate with each other.

![Diagram of module assignment network]

Fig 5.2 Module assignment network
The number against the arcs represent the cost of communication between modules, when the modules are not co-resident on the same processor. Intermodule communication costs between, a specified pair of modules are assumed to be zero when the modules are co-resident. The table below shows the execution costs of the modules when run on processor P1 or processor P2. An infinite (inf) cost indicates that a module cannot be assigned to that processor.

<table>
<thead>
<tr>
<th>Module</th>
<th>P1-time</th>
<th>P2-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>inf</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>d</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>f</td>
<td>inf</td>
<td>4</td>
</tr>
</tbody>
</table>

Given this module interconnection network, Stone [67] has shown how to obtain the minimum cost assignment for two processors P1 and P2. Firstly, using the following procedure a Modified Network is constructed as shown in fig 5.3.
Fig 5.3 Modified network and the minimal cost cut.
(1) Add two nodes $s$ and $t$ representing the processors $P_1$ and $P_2$, respectively.

(2) For every node other than $s$ or $t$, add an arc from that node to each of $s$ and $t$ as shown in fig 5.3. The weight of the arc to $s$ is the cost of executing the module on $P_2$ and the weight of an arc to $t$ is the cost of executing the module on $P_1$.

(3) Using the Max-flow Min-cut algorithm we find the minimal cut i.e. the partition of nodes representing the modules in two groups, one associated with each processor. This partition is shown by the heavy line in fig 5.3.

A cutset is defined as a collection of nodes such that:

(1) When removed from the graph, node $s$ is disconnected from node $t$.

(2) No proper subset of a cutset is also a cutset.
The dark line in fig 5.3 indicates a cutset. It is noted that each cutset corresponds to a module assignment, and that every assignment corresponds to a cutset.

The weight of the cutset is the sum of the weights of the arcs in a cutset. In the network fig 5.3, the weight of the cutset is equal to the corresponding module assignment, since the weight of the cutset is the sum of the execution and communication costs for the assignment. For example, if module 1 is assigned to P1 then the arc to P2 is a cut but this arc carries the cost of execution on P1. Similarly, other arcs cut between module 1 and other nodes of the graph represent actual communication costs incurred by this assignment for communication between module 1 and other corresponding nodes.

The main point of this discussion is: Each cutset in fig 5.3 corresponds in a one-to-one fashion with module assignment. The optimal assignment corresponds to a minimum weight cutset. The optimal assignment can then be solved by the use of network flow algorithms, which are among the class of algorithms with low computational complexity [59, 69].

Stone has shown a rather modest application of the Max-Flow Min-Cut algorithm to a two processor scheduling problem. This problem is slightly similar to our problem because of the objectives that it pursues, but is not so rigidly constrained. The constraint that certain modules should be assigned to particular processor is easily satisfied by an arc of infinite capacity from the module node to the processor node. This problem differs from ours in that the processors are identical in our case, hence, there are other objectives and a large number of other constraints.
Rao et al [70] have considered the same problem as Stone but with an extra constraint, that is one of the processors is constrained to have limited memory. We will consider the approach of Rao et al [70] and investigate whether it can be applied to our problem.

5.4.3 Two processor scheduling with constraints.

Rao et al [70] considered the problem of assignment of modules in a distributed processor system with limited memory. This introduces a constraint which makes the problem much more difficult to solve. Let us assume that every node has a weight that is the memory requirement of the node. Hence, the sum of the weights assigned to one processor should not exceed the memory of that processor.

The approach adopted by Rao et al [70] is as follows: first modify the network as shown by Stone [67], then determine the minimal cost assignment. Let t denote the processor with limited memory and s to be the other processor. Then the feasible assignment is an assignment that does not exceed the memory of t. If it is exceeded then it is necessary that some of the modules assigned to t should be taken out and assigned to s. In order to do that, Rao et al [70] proposed an interesting theorem to show that Minimum cost feasible assignment does not reassign to t any of the modules assigned to s by the minimum cost assignment.

Theorem 5.2 [Rao et al [70]].

The modules assigned to t by the Minimum feasible cost assignment form a subset of the modules assigned to t by the Minimum cost assignment.
Proof [70]:

Let I denote the cut producing a minimum cost assignment and II the cut producing a minimum cost feasible assignment as shown in fig 5.4.

First, observe that cut II cannot be as shown by dotted line, because this cut assigns modules to t in addition to the modules assigned by I. If cut II is the dashed line, then the theorem is trivially proved. Thus, consider the case where cut II is as shown by the solid line. Since cut I and cut II cross each other, the nodes are partitioned into four regions marked A, B, P and Q.

Let $C(u, v)$ be the sum of weights of the arcs between regions $u$ and $v$.

Since I is the minimum cut separating s and t its capacity is no bigger than any other cut separating s and t, and in particular, the cut produced by re-routing I to include region A with Q. Thus
(5.1) \[ C(A,Q) + C(P,Q) + C(A,B) + C(B,P) \leq C(A,P) + C(P,Q) + C(B,P) \]

OR

(5.2) \[ C(A,Q) + C(A,B) \leq C(A,P) \]

OR

(5.3) \[ C(A,Q) \leq C(A,P) + C(A,B) \quad (\text{since } C(A,B) \geq 0) \]

adding \( C(P,Q) + C(B,Q) \) to both sides of (5.3):

(5.4) \[ C(A,Q) + C(P,Q) + C(B,Q) \leq C(A,P) + C(A,B) + C(P,Q) + C(B,Q) \]

i.e. the cut \((A \cup B \cup P, Q)\) has less or the same capacity as \(II\).

Since, the cut \((A \cup B \cup P, Q)\) also separates \(s\) and \(t\) then it highlights the partition of nodes which demands less memory of \(t\), because \(Q\) is the subset of \(Q \cup B\) which was the partition implied by \(I\).

QED.

Thus in searching for a minimum cost feasible assignment, we need only to produce the minimum cut and then re-assign some modules from \(t\) to \(s\).

\[ \text{Let } \text{min}(t) \text{ be the set of nodes assigned to } t \text{ by the minimum cost assignment. Rao et al [70] propose that, in order to find the minimal} \]

PAGE 80
cost feasible assignment, condense all nodes in \( s \) into a single node then introduce an arc of an infinite capacity from the condensed node to \( N \epsilon \min(t) \). Then find another minimal cost assignment. This procedure is repeated for all \( N \epsilon \min(t) \) and the least cost assignment which satisfy the memory constraint on \( t \) is selected. This is the minimal cost feasible assignment.

5.4.4 Comments

Stone [67] and Rao et al [70] have shown a successful application of Network Flow Methods to solve a two processor scheduling problem. However, in our problem, the number of processors is very much greater than two and all processors are identical. As a result there are a large number of constraints.

In the next chapter, we show that Stone's approach cannot be applied directly to a problem as large and as complex as ours. We also show that Rao et al's approach would also be computationally inefficient for a general n processor scheduling problem.

However, these examples will be used to illustrate the development of an algorithm based on network flows which can be generalised to a general n identical processor task assignment.
6. DEVELOPMENT OF THE ALGORITHM

6.1 Glossary

\( N \) = number of nodes (tasks)
\( s \) = The source node
\( t \) = The sink node
\( C(i,j) \) = The value of the cut separating node \( i \) and node \( j \)
\( x \) = subset of the network containing source node
\( x' \) = subset of the network containing sink node
\( V_{ab} \) = The link value joining subsets \( a \) and \( b \)
\( CC_i \) = The capacity of subset \( i \)
\( g_i \) = The limit on the value of subset \( i \)
\( C_{ij} \) = The arc weight connecting node \( i \) to node \( j \)
\( T \) = The vector containing the costs of the assignments
\( L \) = The decision vector for choosing the feasible cost
\( dij \) = The sum of node weights inclusive from \( i \) to \( j \)

Index terms = Cutsets, Cut-Tree, Hierarchical Structure, Partial Order
Graph, Sequential Partitioning, Dynamic Programming
6.2 Reasons for the Alternative Approach.

So far, three independent methods of optimisation have been applied to solve a computer control network design problem. In each case, the aim has been to construct a simple model, based on certain aspects of the problem and then to extend that model to include all other aspects. However, in each case, although the simple models could be solved efficiently, the method of solution could not be generalised to the complete problem. This is due to the size and the complexity of multi-objectives with large number of constraints in the problem.

For instance, the heuristic method of solution to obtain a uniform loading on the computers could not be generalised to include the inter-computer communication objectives and their constraints. Although a complete mathematical model was constructed in chapter 4, but it could not be solved efficiently, hence, it was only useful for trivially small problems. In chapter 5, a successful application of network flow methods for a two processor scheduling problem was shown, but it has not been generalised to n identical processor scheduling problems.

There is also an added complexity, in that the optimal solution to a single unified model may not be very useful to the designer. Because of some desired objectives (as discussed in chapter 2), the designer must look for a number of feasible solutions and then select the one which satisfies the design criteria.

Clearly, the alternative approach is to decompose the problem into simpler sub-problems and then interactively solve these simpler
problems. This would mean that the decomposition procedure has to be optimal so that the combined optimalities of decomposition and subproblem solutions provide a feasible solution to the general problem.

In this chapter such a procedure is developed. This procedure is based on network flow theory and dynamic programming. First, we show that the approach adopted by Stone [67] and Rao et al [70] would fail for identical processors, as is the case in our problem. A different and more efficient approach is proposed for the case of identical processors. We show that the problem can be decomposed into a subproblem which is less complex and the solution to the sub-problem is a solution to the original problem. There are three stages in this procedure and it is shown that each stage is optimal. Because of the stagewise structure of this procedure we show that it can be used to solve either a complete problem or certain parts of the problem. Hence, it can be used interactively by the designer to explore number of solutions at each stage.

In this chapter, we first show why the approach adopted by Stone [67] fails for the case of identical processors. Instead a better approach is proposed, which is based on multi-terminal network flows and is used to decompose the problem. Then, we show that the implicit structure of the decomposed problem is such that it allows us to compute the interprocessor communication objectives and interactively satisfy the constraints on these objectives. Finally, the reduced problem is modelled as a partially ordered graph. The k partitions of this graph represent the minimum number of processors. Also, the interprocessor communication is minimised.

In section 6.6, a summary of the proposed algorithm and its
application to solve a problem is presented. Finally, the
interactiveness and efficiency of this algorithm is shown.
6.3 Decomposition of the problem

In this section, we first consider Stone's [67] approach to find the assignment of tasks to two processors but with the exception that the processors are identical. Because the processors are identical then the arc weight $C(s, i) = C(t, i)$ (for $i=a, b, c, d, e, f$), in other words the run times of tasks on both processors, would be the same.

![Modified graph for identical processors.]

**Fig 6.1** Modified graph for identical processors.

**LEMMA 6.1:** If the arcs from $s$ to all other nodes except $t$ and arcs from $t$ to all other nodes except $s$ have the same weights then the minimum cut $C(x, x')$ assigns $s$ to set $x$ and all other nodes to set $x'$. 
Fig 6.2 Minimal cuts for identical processors.

Proof: We note that cut I represents the minimum cut-set and is outside the task interconnection graph of N nodes. It therefore would only cut N arcs joining s to all other nodes except t.

Now, cut II is inside the task interconnection graph, therefore would cut the arcs of interconnections and the N arcs joining the nodes from the task interconnection graph to s or t.

Therefore:
Number of arcs cut by cut I = N
Number of arcs cut by cut II >= N.
Clearly, if cut II cuts more than $N$ arcs and since the arc capacities are non-negative then cut $I \leq$ cut II.

Since no subset of a cut-set can be a cutset, then cut I is the minimal cut-set.

QED

Therefore, if we want to construct a modified graph for two identical processors then an alternative approach is necessary. Let us assume that a node from the task interconnection graph is joined by an arc of infinite capacity to $s$. Similarly, we nominate another node to be joined by an arc of infinite capacity to $t$, as shown below.

![Fig 6.3 New modified graph for identical processors.](image-url)
The maximal flow minimal cut for this modified graph is as shown above by the heavy line. We note that by assigning an arc of infinite capacity from node b to node s, we have implied that task b should be assigned to processor P1 and similarly, task f should be assigned to processor P2.

Infact, we have indirectly defined node b to be the source and node f to be the sink in the new modified graph. This raises a further point: are the two additional nodes s and t necessary for identical processors network? The following lemma shows that for identical processors, we do not need s and t as additional nodes—the task which must reside in s, can take the role of s and that in t, the role of t.

**LEMMA 6.2**: The minimal cut assignment for two identical processors, obtained from the new modified graph would be equivalent to the minimal cut assignment obtained from the task interconnection graph.

**Proof:**

![Module interconnection graph](image)

Fig 6.4 Task interconnection graph with two additional nodes.
Because of the presence of infinite capacity arcs within the new modified graph, the minimum cut must lie within the module interconnection graph. Let us consider the two cuts in that region: cut I and cut II as shown in fig 6.4. We note that any cut in that region must cut N arcs joining s or t plus, the arcs within the module interconnection graph.

Let the weight (cost) of cutting N arcs = w  
Let the weight of cut I be $C'(s, t) = w+y$  
Let the weight of cut II be $C(s, t) = w+z$

If cut I $\leq$ cut II then $w+y \leq w+z$  
or $y \leq z$

But this is the minimal cut in the original task interconnection graph.

QED

This is a useful result because it shows that, for two identical processors, the minimal cost assignment can be found from the original task interconnection graph and therefore it is unnecessary to modify the graph as indicated by Stone [67]. Hence, we propose that for identical processors, the original interconnection graph is used to find the minimal cost assignment.

In terms of our problem, if we assume that an arc $A_{ij}$ from node $i$ to node $j$ denotes the amount of communication between these two tasks. Then lemma 6.2 implies the following: if we were to assign task $i$ and task $j$ to separate processors, then the absolute minimum
communication between these two tasks is denoted by the weight of the minimum cutset $C(x, x')$, where $x$ is the subset of tasks including task $i$ and $x'$ is the subset of tasks including task $j$. Having found the minimal communication between these two tasks then subsets $x$ and $x'$ represents the assignment of the tasks to two identical processors.

In our problem, the subsets $x$ and $x'$ are bounded i.e. there are constraints imposed on these subsets. For instance, we have a size constraint such that the total processing power required by the tasks, in any one subset, should not exceed a certain capacity limit. We also, have design constraints such that certain tasks should not be assigned to the same processor or certain tasks should be assigned to the same processor. Therefore, in such cases it may be necessary to further subdivide one subset or both subsets to satisfy these constraints.

We have already discussed Rao's [70] approach for this further subdivision of a subset for a two processor network to satisfy the memory size constraint on one of the processors. Rao et al [70] proposed a procedure such that, assume subset $x'$ is bounded i.e. has a memory size constraint. Then, choose a node from subset $x'$ and combine it with subset $x$ to compute another minimal cut $C(y, y')$. This operation is repeated for all nodes in subset $x'$, except $t$, and then the minimal cut $C'(y, y')$ is selected so that $y'$ (containing $t$) satisfy the memory size constraint.

In our problem, we have a large number of tasks and large number of constraints so that the number of processors required will be very much greater than two. It is conceivable that Rao's [70] approach of finding a feasible subset from a given set can be carried on until the set is divided into number of feasible subsets, where each subset represents the group of tasks assigned to a processor. This approach would have two disadvantages. First, each feasible subset will be
derived from a reduced set thereby representing only a local optimality. Second, since the number of processors required for our problem is likely to be very much greater than two, then this approach would require a considerable amount of computation.

Instead we propose an alternative approach which is computationally very efficient and can easily be generalised to \( n \) processor assignment satisfying large number of constraints. The proposed approach is: First, find the minimum communication between each and every task, if they are assigned to separate computers. This procedure constructs a tree network, where each arc in the tree network represents the minimal amount of communication. Then based on these evaluations, certain subsets of tasks are condensed into single task subsets so that some constraints (e.g. the upper bound on the link capacities between computers) are satisfied. We show that the tree network is highly informative and the condensation procedure is developed from that information. We also, show that the resulting condensed network implies a partial order of the tasks. This partial order of the tasks is then used to find a feasible partition of the network into \( n \) disjoint subsets so that the communication between subsets is minimised. Each of these subsets specifies the feasible assignment of the tasks to \( n \) processors.

Having outlined the approach, we will now provide the more formal definition of this method. First of all, we consider the evaluation of the minimum communication between each and every task. This is based on the recent work of Gomory and Hu [76], who have put forward a very efficient technique.

If we are interested in finding maximal flow values (minimum amount of communication) between \( p \) nodes where \( 2 \leq p \leq n \) then instead of doing \( p(p-1)/2 \) maximal flow computations we need do only \( (p-1) \) computations. Furthermore, each of the flow computations is done on a
To begin the discussion of Gomory and Hu's [69] technique, let us assume that a maximal flow problem has been solved with some node $s$ as the source and another node $t$ as the sink, thereby locating a minimal cut $(x, x')$ with $s$ in $x$, $t$ in $x'$ as shown below in fig 6.5.

Suppose that next we wish to find $C(p, q)$ where both $p$ and $q$ are on the same side of the minimal cut $(x, x')$, say both $p$ and $q$ are in $x$. Gomory and Hu [69] have shown that, for this purpose, all the nodes in $x$ can be "condensed" into a single node to which all the arcs of a minimal cut are attached. Several arcs, joining the pair of nodes, can be replaced by a single arc, as shown in fig 6.6. The resulting network is known as the "condensed network" [69].
LEMMA 6.3. The minimum amount of communication $C'(p, q)$, between nodes $p$ and $q$ of the condensed network, is equal to the minimal amount of communication $C(p, q)$ in the original network.

Proof:--
Let $(y, y')$ be a minimal cut, separating $p$ and $q$ in the original network, as shown in fig 6.7. We define the following sets:

Fig 6.7 Crossing cuts.
Let \( A = x \cap y \quad A^\prime = x \cap y^\prime \quad B = x^\prime \cap y \quad B^\prime = x^\prime \cap y^\prime \)

Here \( A^\prime \) is the complement of \( A \) in \( x \), \( B^\prime \) is the complement of \( B \) in \( x^\prime \).

We assume that \( p \in A \), \( q \in A^\prime \) and \( s \in A \).

Case 1: Let \( t \in B \).

What we would like to show is that the cut \((x, x^\prime)\) and cut \((y, y^\prime)\) do not cross each other. In other words, we would like to show that in this case set \( B^\prime = x^\prime \cap y = 0 \) i.e. \( B^\prime \) is an empty set.

Now

\[
C(x, x^\prime) = C(A, B) + C(A^\prime, B) + C(A, B^\prime) + C(A^\prime + B^\prime)
\]

\[
C(y, y^\prime) = C(A, A^\prime) + C(A, B^\prime) + C(B, A^\prime) + C(B, B^\prime)
\]

Since \( C(y, y^\prime) \) is the minimal cut separating \( p \) and \( q \) and since \((A \cup B \cup B^\prime, A^\prime)\) also separates \( p \) and \( q \) we have

\[(6.1) \quad C(y, y^\prime) - C(A \cup B \cup B^\prime, A^\prime) \leq 0\]

or

\[
C(A, A^\prime) + C(A, B^\prime) + C(B, A^\prime) + C(B, B^\prime) - \left\{C(A, A^\prime) + C(B, A^\prime) + C(B^\prime, A^\prime)\right\} \leq 0
\]

\[
= C(A, B^\prime) + C(B, B^\prime) - C(A^\prime, B^\prime) \leq 0
\]

Since \((x, x^\prime)\) is a minimal cut separating \( s \) and \( t \) and since \((A \cup A^\prime \cup B^\prime, B)\) separates \( s \) and \( t \), we have

\[(6.2) \quad C(x, x^\prime) - C(A \cup A^\prime \cup B^\prime, B) \leq 0\]

or
\[ C(A, B) + C(A^-, B) + C(A, B^-) + C(A^-, B^-) - \left\{ C(A, B) + C(A^-, B) + C(B^-, B) \right\} \leq 0 \]

\[ = C(A, B^-) + C(A^-, B^-) - C(B^-, B) \leq 0 \]

Adding (6.1) and (6.2) shows that \( C(A, B^-) \leq 0 \) and, hence \( C(A, B^-) = 0 \). It then follows from (6.1) and (6.2) that \( C(B, B^-) - C(A^-, B^-) = 0 \). Hence \((A \cup B \cup B^-, A^-) = (A \cup x^-, A^-)\) is also a minimal cut separating \( p \) and \( q \).

Case 2: Let \( t \in B^- \)

A similar proof shows that \((A, A^- \cup x^-)\) is a minimal cut separating \( p \) and \( q \) in this case and set \( B \) is an empty set.

Hence, in both cases we have shown that the cut \((x, x^-)\) and cut \((y, y^-)\) do not cross each other.

QED

We have shown that there is always a minimal cut separating \( p \) and \( q \) such that the set of nodes in \( x^- \) are on one side of this cut. Consequently, condensing \( x^- \) to a single node does not affect the value of the minimal cut \((p, q)\). This lemma plays a fundamental role in finding the minimum amount of communication between each and every task. We would only have to do \((p-1)\) flow computations for a network with \( p \) nodes (tasks), furthermore, each flow computation would be done on a smaller network. We can now describe the formal procedure for such evaluations as proposed by Gomory and Hu [76].
Select two nodes arbitrarily from the original network and solve a maximal flow problem. This locates a minimal cut \((x, x')\), which is represented symbolically by two nodes connected by an arc of capacity \(v_1 = C(x, x')\) as in the fig 6.8.

![Fig 6.8 Arbitrary minimal cut.](image)

In one node the individual nodes of \(x\) are listed; in the other, those of \(x'\). Next, choose two nodes in \(x\) and solve the resulting maximal flow problem in the \(x'\) condensed network. The resulting minimal cut has capacity \(v_2\) and is represented by an arc of this capacity connecting the two parts into which \(x\) is divided by the cut, say \(x_1\) and \(x_2\). The node \(x'\) is attached to \(x_1\) if it is in the same part of the cut as \(x_1\); to \(x_2\) otherwise. Let us assume \(x_2\) is connected to \(x'\) as shown in fig 6.9.
This process is continued until each set contains only one node. To illustrate the general step, suppose we have arrived at the stage indicated by fig 6.10, with $y$ to be split.
Removal of the arcs attached to $y$ leaves the disconnected components $y; x_1; x_2, x_3; x_4, x_5, x_6$. Then in the original network, the nodes of $x_1$ are condensed as are of $x_2 \cup x_3$ and $x_4 \cup x_5 \cup x_6$. Solving a maximal flow problem between two nodes of $y$ in the condensed network might then lead to the new stage as shown in fig 6.11.

If the original network has $n$ nodes then the algorithm terminates after $(n-1)$ maximal flow computations. We note that the final diagram is a tree network with $n$ nodes and $(n-1)$ arcs joining these nodes.

The method of analysis described above constructs a flow equivalent network which is a tree. There can be more than one flow equivalent network but the flow equivalent tree constructed by the above described procedure has another property, namely, each link of the tree represents a minimal cut of the original network. Therefore it is called a cut-tree Gomory and Hu [69].

![Fig 6.11 General stage of cut-tree.](image)
6.4 Synthesis of the decomposed problem

In the previous section we have just shown how to find the minimum communication between two tasks if they were assigned to separate computers. We note that, if two nodes are neighbours in the cut-tree then the value of the arc joining those nodes represents the absolute minimum communication between those two tasks. We can go further and generalise this by saying that the tree network represents the minimal communication between each and every task.

LEMMA 6.4: The minimum amount of communication between any two tasks (nodes) $Ni$ and $Nj$ of the original network is equal to

$$\min (v_{i\alpha}, v_{\alpha\beta}, \ldots, v_{dj})$$

where $v_{i\alpha}, v_{\alpha\beta}, \ldots, v_{dj}$ are values associated with links of the cut-tree which forms a unique path from $Ni$ to $Nj$.

Proof: [Gomory and Hu [69]].

Consider a link in the cut-tree connecting two circles with one circle containing node Na and the other circle containing Nb: Let Na $\in x$ and Nb $\in x'$. 

Then the value associated with this link is $C(x, x')$, which will be shown later to be equal to the minimum communication between $a$ and $b$, $(C_{ab})$. Let us assume that the minimum cut $C(x, x')$ is for $Ni \in x$ and $Nj \in x'$. The minimum communication value $C_{ab}$ is equal to the value of the cut $C(z, z')$ in the original network, say $Na \in z$ and $Nb \in z'$.

Thus, $(z, z')$ is a minimum cut separating $Na$ and $Nb$. Let $z \subset x$ and $z' \supset x'$ (the case for $z \subset x'$ and $z' \supset x$ will be similar) as in Fig 6.12.

Case 1: Let $Ni \notin z$ i.e. as in Fig 6.12.

Then $(z, z')$ separates $Ni$ and $Nj$, $C(z, z') \geq C_{ij} = C(x, x')$; otherwise it would contradict the fact that $(x, x')$ is a minimum cut.
Case 2: Let \( N_i \not\in z \cap x \) i.e. as \( L \) in fig 6.12.

Let \( C_{ia} = C(y, y') \)

From the previous lemma, we know that \( (y, y') \) does not cross \( (x, x') \) or \( (z, z') \), furthermore, \( N_a, N_b \) and \( N_j \) belong to \( y' \). Then \( (y, y') \) is a cut separating \( N_i \) and \( N_j \).

\[
C(y, y') \geq C_{ij} = C(x, x') \quad (6.4)
\]

Also as \( (z, z') \) is a cut separating \( N_a \) and \( N_i \), we have

\[
C(z, z') \geq C_{ia} = C(y, y') \geq C(x, x') \quad (6.5)
\]

Therefore, in both cases, we have

\[
C(z, z') \geq C(x, x') \quad (6.6)
\]

Since \( (x, x') \) is a cut separating \( N_a \) and \( N_b \)

\[
C(x, x') \geq C_{ab} = C(z, z') \quad (6.7)
\]

(6.6) and (6.7) imply that \( C(z, z') = C(x, x') = C_{ab} \)

Having established that the neighbouring nodes in the cut tree have a minimum communication value of the connecting link, we can say that for any two nodes (tasks) connected by a series of links in the tree,

\[
C_{ij} \geq \text{Min} \ (C_{ia}, C_{ab}, \ldots, C_{ej}) = \text{Min} \ (v_{ia}, v_{ab}, \ldots, v_{ej})
\]

where \( N_i \) and \( N_j \) are not neighbouring nodes in the cut tree.
Hence, given a Gomory–Hu cut tree we can find the minimum value of the communication between each and every task.

QED

Consider the network in fig 6.13. First, let us arbitrarily select N1 and N3 as the source and sink respectively. We obtain a cut (N1, N2, N6 | N3, N4, N5) with value 13, this is indicated symbolically by fig 6.14.

Fig 6.13 Network example
Next, we find the minimal communication between N3 and N4, from the condensed network as shown in Fig 6.15. The corresponding cut is \((N1, N2, N6, N3, N5 \mid N4)\) with value 14 as shown in Fig 6.16.
Next, we find the minimal communication between N3 and N5, which again should be done on the network shown in Fig 6.15. The corresponding cut obtained is \((N_1, N_2, N_6, N_3, N_4 \mid N_5)\) with value of 15. The cut-tree so far obtained is as shown in Fig 6.17.
Since each circle in one subset has only one node and the minimum communication values are $C_{13} = C_{14} = C_{15} = 13$, $C_{34} = C_{45} = 14$ and $C_{35} = 15$, this network then has the same values $C_{13}, C_{14}, C_{15}, C_{34}, C_{35}$ and $C_{45}$ as the original network in fig 6.13.

Next, we move to the other subset ($N_1$ and $N_6$) as the source and sink respectively and find the minimal communication using the network as shown in fig 6.18. The result is minimal cut ($N_1, N_2 \mid N_6, N_3, N_4, N_5$) with value 17. The resulting tree is as shown in fig 6.19. The remaining computation would then be carried out on the network shown in fig 6.20.

![Fig 6.19 Fourth minimal cut](image1)

![Fig 6.20 Condensed network](image2)

At this stage, $N_1$ and $N_2$ are chosen as the source and sink. The result is minimum cut ($N_1 \mid N_2, N_3, N_4, N_5, N_6$) with value 18. The
The final cut tree is as shown in fig 6.21.

![Diagram of the final Gomory-Hu cut tree](image)

Fig 6.21 The final Gomory - Hu cut tree.

From the final cut-tree, we can determine the minimal communication between all tasks. In this case it is shown in table 6.2.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00</td>
<td>18</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>00</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>13</td>
<td>00</td>
<td>14</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>13</td>
<td>14</td>
<td>00</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td>13</td>
<td>15</td>
<td>14</td>
<td>00</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>17</td>
<td>17</td>
<td>13</td>
<td>13</td>
<td>13</td>
<td>00</td>
</tr>
</tbody>
</table>

Table 6.2 Minimum communication matrix.
A cut-tree in Fig 6.21 shows (n-1) minimal cuts of the original network which do not cross each other. The (n-1) cuts are as shown in fig 6.22.

Fig 6.22 The (n-1) minimal cuts.

6.4.1 Problem size reduction

A cut \((x, x')\) in the original network is said to separate a link \(L_{pq}\) in the cut-tree if \(N_p \notin x\) and \(N_q \notin x'\). In other words, the two terminal nodes of the cut-tree are on different sides of the cut. Since the existence of the link implies that \(L_{pq}\) equals the minimum
communication between Np and Nq, any cut \((y, y')\) separating Np and Nq must have value \(C(y, y') \geq L_{pq}\).

We propose that if any link \(L_{pq}\) in the cut-tree has value exceeding the link capacity between any two computers then the tasks \(p\) and \(q\) should be condensed to a single node for the remainder of the computation. This method of node condensation may reduce the size of the problem and in any case would insure the feasibility (links not overloaded) of the final solution.

Hu and Ruskey [77] have proposed other procedures for condensing nodes for single objective unconstrained problems. In such cases, large problems are reduced to trivially small problems. However, since our problem is multi-objective and has a large number of constraints, it is felt that the use of such procedures at this stage would be premature and prejudicial to the final solution of the problem.

6.4.2 Hierarchical structure.

In this section, we show that the Gomory - Hu cut-tree exhibits a hierarchical structure between the subsets of tasks. This is evident because of the way the cut-tree is constructed. In order to illustrate this point, consider the cut-tree shown in fig 6.23.
Fig 6.23. Illustration of hierarchical structure.

let us assume that we wish to assign tasks d and e to separate computers. Then the minimum value of the communication would be Cde, since, Cde is the minimum cut in the original network. We note that the minimum cut \((x, x')\) implies that tasks e, f, g and h should be assigned to one computer and the tasks a, b, c and d to the other.

Let us assume a further subdivision of subset \(x'\) into \((y, y')\) as shown in fig 6.23 or equivalently, we could represent this as follows
Let us assume that $x$, $y$ and $y'$ are assigned to three computers $cc1$, $cc2$ and $cc3$ respectively. Then, we note that the communication value between $cc1$ and $cc2$ would be $L_{de}$ and communication between $cc2$ and $cc3$ would be $L_{fg}$. We know that the $L_{de}$ and $L_{fg}$ satisfy the computer link capacities constraints (otherwise the tasks would have been condensed). Furthermore, $L_{de}$ and $L_{fg}$ represent the absolute minimum communication value. We also note that any messages from $x$ to $y'$ will pass through $y$ without violating any constraints. Hence, fig 6.24 represents a hierarchical structure.

Since $y \in x'$ and $y' \in x'$.

Therefore $y \cup y' = x'$

If $y < y'$ i.e. $y$ precedes $y'$ and if $x < x'$
Then it follows that $x < y < y'$.

In general, we can say that, if there is a link in the Gomory – Hu cut-tree from task $i$ to task $j$, then task $i$ precedes task $j$. This means that for any assignment of tasks cutting $L_{ij}$ on the Gomory – Hu cut-tree, the task $i$ and its predecessors must be assigned to one group and the task $j$ and its successors should be assigned to the other group. The value $L_{ij}$, of the link $ij$, represents the minimum communication value. Furthermore, each group can be further subdivided into sub-groups without any change in $L_{ij}$.

Now, this is a very important result, because it implies that to find a minimum communication feasible assignment in the original network we have only to use the information from the Gomory – Hu cut-tree. Because, for minimum communication, each link in the cut-tree implies a partial order of the tasks, then clearly, to find a minimum cost feasible assignment, we only need to consider order preserving cuts specified by the partial order of the tasks. In order to clarify this point, consider the following partially ordered graph representing the partial order of the tasks.

![Partially ordered graph](image)

Fig 6.25 Partially ordered graph.
The cut \((x, x')\) implies that if task \(i\) were to be separated from task \(j\) then this cut would dominate all other cuts separating tasks \(i\) and \(j\) in the original network. This inherent property of the Gomory–Hu cut-tree not only reduces the size and complexity of the problem but also is a very useful source of information to the designer. In next section, we show how some of the other constraints (e.g. assignment of back-up tasks, separation of tasks, combination of tasks) can be implied in the construction of the partial order graph.

In order to illustrate the construction of the partial order graph consider the cut-tree obtained in fig 6.21.

![Fig 6.26 Gomory-Hu cut-tree.](image)

The partial order implied by this cut-tree is \(1<2<6<5<4\) or \(3\). In order to determine the exact partial order, construct the network as in fig 6. 27.

![Fig 6. 27 condensed network.](image)
The minimal cut on this network is as shown and the assignment is (1, 2, 6, 5, 3 | 4). Therefore, the exact partial order is.

Fig 6. 28 The exact partial order

This partially ordered graph is flow equivalent to the original network[69].

In this section we proceed to find the solution to the remainder of our problem. Having constructed a partially ordered graph that minimises the communication the next problem is: Given a partial order of the tasks for minimum communication, find the assignment of these tasks to the minimum number of computers so that the size constraints are satisfied. The partitioning of graphs into number of disjoint subsets has been considered by Christofedos et al [78], Perl et al [79], Glover [80] and Kernighan [81]. Kernighan [81] has proposed an efficient Dynamic Programming method, which is applicable to partially ordered graphs.

6.5 **Optimal sequential partitions of the reduced problem.**

Let \( G \) be a graph of \( n \) nodes \{1, 2, \ldots, n\}. The nodes of \( G \) have weights \{\( w_1, w_2, \ldots, w_n \)\} such that \( 0 \leq w_i \leq C_{Ci} \), where \( C_{Ci} \) is a positive integer denoting the capacity limit of computer \( i \), where
A feasible partition of \( G \) is a division of the nodes of \( G \) into \( k \) disjoint subsets \( g_1, g_2, \ldots, g_k \), such that for \( i = 1, \ldots, k \),

1. \( |g_i| \leq C_{Ci} \)
2. The nodes in any \( g_i \) have contiguous numbers that is, \( g_i \) contains nodes \( j, j+1, \ldots, m-1, m \).

Each arc \( (i, j) \) of \( G \) has a positive number associated with it, \( C_{ij} \), which represents the communication between task \( i \) and task \( j \). Then the cost of partition is \( \sum C_{ij} \), with \( i \) and \( j \) in different computers. A feasible optimal partition is an admissible partition of communication.

The algorithm starts by augmenting \( G \) by two nodes, 0 and \((n+1)\), with communication costs \( C_0, i = C_i, n+1 = 0 \), for \( i = 1, \ldots, n \).

The distance \( d_{ij} \) from node \( i \) to node \( j \) (\( i < j \)) in a graph \( G \) is

\[
d_{ij} = w_i + w_{(i+1)} + \ldots + w_j;
\]

the distance is thus the sum of weights of all nodes from \( i \) to \( j \) inclusive.

Let \( C(x) = \sum_{i < x \leq j} C_{ij} \). (\( x = 1, 2, \ldots, n+1 \)).

\( C(x) \) is the sum of costs on all edges cut by a break point at \( x \). Let \( T(x) \) be a minimum partial cost (as far as node \( x \)). Thus \( T(1) = 0 \), \( T(x) \) will be evaluated iteratively.

Intuitively, the algorithm operates roughly as follows: For \( x = 1, 2, \ldots, n+1 \), set \( T(x) = C(x) + T(y) \), where \( y \) is such that \( d_{xy} \leq CC \)
and where $T(y)$ is minimal over all such $y$.

This stage terminates after $T(n+1)$ values has been computed; $T(n+1)$ is the cost of optimal segmentation. The particular $y$ value which was used to compute $T(n+1)$ defines the last break point $b_k$.

Now $T(b_k)$ was computed as $C(b_k) + T(b_{k-1})$ where $d(b_{k-1}, b_{k-1}) \leq CC$. So $b_{k-1}$ defines the next break point. This process continues until $b_1 = 1$ is reached. At this time a set of break points $\{b_1, \ldots, b_k\}$ has been found and the total cost evaluated.

The algorithm is essentially a form of Dynamic Programming and has execution time linearly dependent on the number of nodes. Kernighan [81] has shown the necessary proofs to the following theorems.

Theorem 6.1: This procedure finds an optimal admissible partition of a graph.

Theorem 6.2: The running time of this procedure is linearly proportional to the number of nodes in the graph.

Thus, in conclusion, the necessary algorithm has been developed. We have seen that this algorithm has three stages, each stage individually providing optimal solution. The move from one stage to the other is dependent on the previous stage. Because this move is made interactively, the algorithm insures the feasibility of the final solution.

In the next section, we summarise the algorithm and present it in a stepwise manner. A numerical problem is solved. The interactiveness of the algorithm and the efficiency of the algorithm are fully discussed in the next section.
6.6 The algorithm

6.6.1 Summary

Given a graph $G = (N, W, A)$ where $N$ represents the set of nodes (tasks) and $A = a_{ij}$ represents non-negative numbers (communication between tasks) joining two nodes, $W_i$ represents the weight (the processing power requirements) on node $i$. Using the maximal flow algorithm we construct a tree network of $N$ nodes with $(N-1)$ links. Each link in this tree represents the minimal cost (minimum data transfer) partition of the original network.

In the second stage of the algorithm, certain subsets of nodes are condensed into single node subsets, as indicated by the tree network. If the link weight in the tree exceeds a certain value, then the two neighbouring nodes of the link are condensed into a single node. The tree obtained from this procedure implies a hierarchical order of the subsets of the nodes. This inherent property of the tree is used to construct a partial order of the nodes (tasks). This shows that, for minimum data transfer, the neighbouring node in the tree takes precedence on all other nodes. It is intended that this partial order of the nodes will be constructed interactively with the designer's requirements.

In the final stage of the algorithm, we use a Dynamic Programming type of algorithm on a much reduced and less complex sub-problem. This part of the algorithm finds a feasible $k$ partitions of an $N$ node network with a given partial order, such that the cost of partitions (the data transfer) is minimised and the size constraints (link capacity, processing power capacity requirements) are satisfied.
6.6.1.1 *Stage 1 algorithm.*

Comment: this part of the algorithm constructs the tree.

TAG is used to label the \((n-1)\) links of the tree.

(1) Set up linear arrays: \( \text{FROM}[i] \) to \( \text{TO}[i] \) the value of the data transfer \( \text{RCAP}[k] \), for \( i=1, \ldots, N \), for \( k=1, \ldots, V \), where \( V=N^2N/2 \).

(2) Define \( \text{SOURCE} \) and \( \text{SINK} \); set \( \text{TAG}=1 \).

(3) Evaluate maximal flow (minimum communication).

(4) Determine minimal cut. Let that be \( C(x, x^-) \).

(5) Assign \( \text{LABEL}[i]=\text{TAG} \) for all nodes in \( x \) otherwise \( \text{LABEL}[i]=\text{TAG}+1 \).

(6) If there is only one element in \( x \) then GOTO step 9.

(7) Condense subset \( x^- \).

(8) Define \( \text{SOURCE} \) and \( \text{SINK} \) and GOTO step 3.

(9) IF there is more than one element in \( x^- \), then condense \( x \) and GOTO 8; otherwise GOTO 10.

(10) Increment \( \text{TAG}=\text{TAG}+1 \).

(11) If \( \text{tag} < N \) then GOTO 6.

(12) \((N-1)\) links have been found and the cut-tree is constructed.
6.6.1.2 Stage 2 algorithm.

Comment: This stage constructs the partial order of the graph.

(13) Determine, from the cut-tree, a node which has no predecessors (This node would be represented by a minus sign)

(14) Label this node S.

(15) Proceed to the successor node to S. If S is only connected to one node, then GOTO 16; ELSE DO

(a) Condense the predecessors to S, and condense S into a single node.
(b) Condense the successors of S on each link leading from S, into single node subsets.
(c) Compute the maximal flow on the condensed network, where S is the source and the node of the lightest link is the sink. Determine the minimal cut.
(d) The node which is in the same group as S is the successor (s) to S.

(16) If there are successors to this node, then GOTO 14; ELSE re-number the nodes in an ascending order. This specifies the partial order of the tasks.
6.6.1.3 Stage 3 algorithm.

Comment: The following procedure will find a feasible partition of the network of $N$ nodes in a given partial order.

Define $T$ as the vector of running total costs.
$L$ as the vector of pointers.

At any node, a chain of entries of $L$, starting with $L(x)$ is precisely the sequence of nodes which is used in the computation of $T(x)$.

Let $C(x, y) = \sum_{y \leq i < x, j \geq x} C_{ij}$.

where $C(x, y)$ is the incremental cost of the breakpoint at $x$, given that the previous breakpoint is at $y$.

(17) Set $T(1) = 0$.

(18) For $x$ from 2 to $(N+1)$ in steps of 1

Set $T(x) = \min_y [ T(y) + C(x, y) ]$

where the minimisation is done over all $y$ such that $d_{y,(x-1)} \leq CC$ (node capacity).

Set $L(x) = y$.

(19) Set total cost = $T(N+1)$.

(20) Set $Z(1) = N+1$.

(21) While $Z(k) > 1$; DO $Z(k+1) = L(Z_k)$; $k := k+1$.

Breakpoints are $Z_1, Z_2, \ldots, Z_k$ in descending order.
6.6.1.4 Mill control problem

The following problem reflects an aspect of the control scheme for a power station.

![Control Diagram]

**Fig 6.29 Functional diagram of a mill**

The communications and processing requirements of the modulating control loops are as in the following table:

<table>
<thead>
<tr>
<th>TASK NO.</th>
<th>NAME</th>
<th>LOAD(%)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GOVERNER</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FURNACE</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FAN A</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FAN B</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TEMP. A</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TEMP. B</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>FEED. A</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>FEED. B</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>LOGGER</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>EFFICIENCY</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Given that information the problem can be stated as:

Derive the optimum number of control centres and the distribution of the tasks such that each control centre is evenly loaded. No control centre must be loaded above 50%. No communication link must carry more than 10 messages per second in each direction. It is desirable to have FAN A, TEMP A, and FEED A to be together and FAN B, TEMP B, and FEED B to be kept together.

SOLUTION:

The network of tasks and the interactions is as follows.

Fig 6.30 The network for mill control
Using stage 1 algorithm, the following cut-tree is obtained.

Fig 6.31 The cut-tree for mill control problem

We note that in fig 6.31, node 2 is connected to nodes 5, 6 and 9. But the link between nodes 2 and 9 exceeds the minimum inter-computer link capacity. Therefore, nodes 2 and 9 can be condensed to a single node. Using algorithm 2, the following partial order of the tasks is obtained. (Note that in the partial order, the tasks that are to be kept together are kept as close as possible.)

Fig 6.32 The partial order of the tasks.
From the partial order of the tasks, following network is then constructed:

Fig 6.33 The partially ordered network.
Using the stage 3 algorithm, the following partition of the network is obtained. Each partition represents the computer assignment i.e. three computers are needed.

Fig 6.34 The feasible partition of the network.

Hence, the final computer assignment of the tasks is as shown in fig 6.35

Fig 6.35 The computer assignment of the tasks.

We note that the computer capacity constraint and the computer link constraints are also satisfied.
6.6.2 Interactiveness of the algorithm

We have already seen how the partial order graph is obtained from the original network of tasks. However, this partial order is influenced, to some extent, by the Gomory-Hu cut-tree. The information obtained from the cut-tree can be used interactively to construct the partial order so that some other objectives are also satisfied. The following is an indication of the interactiveness of the algorithm.

(1) Combination of tasks: If certain tasks are to be combined together then those tasks should be combined in the original network, thereby reducing the size of the problem.

(2) Separation of tasks: If it is desirable that certain tasks should be not be assigned to the same computer, either for autonomy or for safety reasons, then those tasks should be kept apart in the partial order graph. This can be achieved by having an arc of zero weight between those nodes in the original network.

(3) Assignment of back-up tasks: It is proposed that the back-up tasks of the major tasks should be assigned after the assignment of major tasks has been found. Generally, it would be feasible to assign back-up tasks to either the \((j+1)\)th or \((j-1)\)th computer if the major task is in the \(j\)th computer.
6.6.3 Efficiency of the algorithm

Edmonds and Karp [75] have shown that the maximal flow algorithm has an execution time of the order of $N^3$, where $N$ is the number of nodes (tasks). Since we would require $(N-1)$ flow computations, then the execution time for constructing the Gomory-Hu cut-tree would be approximately $N^4$. Although this algorithm can be used for real numbers, it was found to converge rapidly with integer data. Hu [69] has shown that the MAX FLOW- MIN CUT algorithms do converge in a finite number of steps for integer data.

The storage space required is also less than the standard methods of optimisation such as Branch and Bound [39]. This is because the algorithm only requires the non-zero entries from the network. Since the task interactions are sparse (i.e. each task only connected to three or four tasks), larger problems can easily be solved.

A test problem of 32 nodes has been solved on the HP 21 computer. The problem and its solution is shown in appendix 4. The computer code of this algorithm has been written in CORAL 66 for the HP 21 computer. It was found that the memory requirement of the program and its workspace is approximately 24 K (words) for a 40 node problem. The execution time of the 32 node problem (appendix 4) was 15 seconds.
7. CONCLUSION

There is a strong case for computer based assistance in the design of complex and large distributed computer control networks. The purpose of this thesis was to highlight the problems encountered in optimising the network configuration and assigning jobs to computers.

The objectives and the requirements of this problem were formulated as realistic mathematical models. Three independent methods of optimisation have been applied to solve this computer control network problem. Although in each case, the simple model could be solved efficiently, the method of solution could not be generalised to the complete problem. This is due to the size and the complexity of multi-objectives with a large number of constraints in the problem.

An alternative approach has been proposed. An algorithm has been developed which is used to decompose the problem and interactively solve the less complex subproblems. It has been shown that the algorithm is computationally efficient and is capable of solving large, practical size, problems. This algorithm can be used interactively to explore the feasible solutions to network design problems, hence it forms the basis of a very usefull tool for the design aid for distributed computer control networks.

An inherent property of the algorithm is that it allows fault conditions to be simulated and their effects on the system to be evaluated. For each fault or combination of faults eg a processor or communications path failure, it is possible to calculate the new loading levels and thereby determine any overloading. By so doing, design weaknesses can be identified.


[8] G. HOPE. "Distributed microcomputer system for a large


PAGE 130


[52] L. STEINBERG. "The backboard wiring problem: a placement


[61] R.BELLMAN. "Mathematical aspects of scheduling theory". J.


The following table shows the list of control tasks developed at C.E.R.L for Drax completion [13]. In this appendix, we consider the assignment of these tasks to computers such that all the computers are uniformly loaded. This illustrates the application of the heuristic algorithm developed in chapter 3.


<table>
<thead>
<tr>
<th>TASK NO</th>
<th>NAME</th>
<th>LOAD(%)</th>
<th>EXECUTION TIME (mS)</th>
<th>TIME STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unit step</td>
<td>23.1</td>
<td>230.74</td>
<td>1 s</td>
</tr>
<tr>
<td>2</td>
<td>Duct pressure</td>
<td>8.2</td>
<td>82.27</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Back-up mill temp.</td>
<td>5.2</td>
<td>52.55</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Main feedvalve (A)</td>
<td>23.4</td>
<td>233.98</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Start-up feedvalve (A)</td>
<td>7.9</td>
<td>78.99</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Steam air ratio</td>
<td>7.2</td>
<td>71.88</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Furnace pressure (A)</td>
<td>14.7</td>
<td>147.83</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Feeder (A)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Feeder (B)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Feeder (C)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Feeder (D)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Feeder (E)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Feeder (F)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>TASK NO.</td>
<td>NAME</td>
<td>LOAD (%)</td>
<td>EXECUTION TIME (mS)</td>
<td>TIME STEP</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td>----------</td>
<td>---------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>32</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td>1 s</td>
</tr>
<tr>
<td>33</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Feeder (G)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Feeder (H)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Feeder (J)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Primary air</td>
<td>6.3</td>
<td>63.19</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Secondary air</td>
<td>10.3</td>
<td>103.16</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Feeder (K)</td>
<td>5.4</td>
<td>54.11</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Mill outlet temp.</td>
<td>8.2</td>
<td>82.40</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Main feedvalve (B)</td>
<td>23.4</td>
<td>233.98</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Furnace pressure (B)</td>
<td>14.7</td>
<td>147.83</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Start-up feedvalve (B)</td>
<td>7.9</td>
<td>78.99</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Electric feedvalve</td>
<td>11.1</td>
<td>111.21</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>2nd stage S/H temp. (L1)</td>
<td>10.9</td>
<td>109.26</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>2nd stage S/H temp. (L2)</td>
<td>10.9</td>
<td>109.26</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>2nd stage S/H temp. (L3)</td>
<td>10.9</td>
<td>109.26</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>2nd stage S/H temp. (L4)</td>
<td>10.9</td>
<td>109.26</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>1st stage S/H temp. (L1)</td>
<td>13.1</td>
<td>130.91</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>1st stage S/H temp. (L2)</td>
<td>13.1</td>
<td>130.91</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>1st stage S/H temp. (L3)</td>
<td>13.1</td>
<td>130.91</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>1st stage S/H temp. (L4)</td>
<td>13.1</td>
<td>130.91</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>Reheat feedpump</td>
<td>7.3</td>
<td>72.76</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>Main feedpump</td>
<td>20.9</td>
<td>209.26</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>Emergency reheate</td>
<td>9.2</td>
<td>91.76</td>
<td></td>
</tr>
</tbody>
</table>
The following charts show the distribution of these tasks using the heuristic algorithm. The numbers in charts correspond to the task load imposition on the control centre CC. The initial capacity limit set is 85% loading of the computers.

<table>
<thead>
<tr>
<th>Load</th>
<th>CC1</th>
<th>CC2</th>
<th>CC3</th>
<th>CC4</th>
<th>CC5</th>
<th>CC6</th>
<th>CC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>83.9%</td>
<td>83.9%</td>
<td>83.8%</td>
<td>83.8%</td>
<td>83.6%</td>
<td>82.2%</td>
<td>81.0%</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td></td>
<td>8.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>5.2</td>
<td></td>
<td></td>
<td>8.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.9</td>
<td>5.4</td>
<td></td>
<td></td>
<td>8.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>7.2</td>
<td>8.2</td>
<td>5.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>4.2</td>
<td>9.2</td>
<td>7.3</td>
<td>8.2</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>10.3</td>
<td>10.3</td>
<td>10.9</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>10.3</td>
<td>13.1</td>
<td>10.9</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>14.7</td>
<td>13.1</td>
<td>11.1</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.9</td>
<td>20.9</td>
<td>14.7</td>
<td>13.1</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>7.9</td>
<td>10.9</td>
<td>23.4*</td>
<td>23.4*</td>
<td>23.1*</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
</tbody>
</table>

Fig AP 1.1 The initial distribution

* tasks to be kept separate.
<table>
<thead>
<tr>
<th>Load</th>
<th>83.0%</th>
<th>83.3%</th>
<th>83.8%</th>
<th>83.8%</th>
<th>83.0%</th>
<th>83.4%</th>
<th>81.9%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit 85%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.4 +</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>8.2</td>
<td>5.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>5.2</td>
<td>8.2</td>
<td>5.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.9</td>
<td>5.4</td>
<td>8.2</td>
<td>5.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>7.2</td>
<td>8.2</td>
<td>6.3+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>4.2</td>
<td>9.2</td>
<td>7.3</td>
<td>8.2</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>10.3</td>
<td>10.3</td>
<td>10.3+</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>10.3</td>
<td>13.1</td>
<td>10.9</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>14.7</td>
<td>13.1</td>
<td>11.1</td>
<td>10.3</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>8.2</td>
<td>10.3</td>
<td>20.9</td>
<td>14.7</td>
<td>13.1</td>
<td>10.9+</td>
<td>6.3</td>
<td></td>
</tr>
<tr>
<td>7.9</td>
<td>10.9</td>
<td>23.4*</td>
<td>23.4*</td>
<td>23.1*</td>
<td>10.9+</td>
<td>6.3</td>
<td></td>
</tr>
</tbody>
</table>

**Fig AP 2.1 The initial distribution**

* Tasks to be kept separate.

+ Tasks that have been re-distributed
Branch and Bound algorithm

This algorithm is based upon Land and Doig's method (as extended by Sharesian) to solve mixed integer programming problems [20]. The linear (minimisation) problem is first solved without regard to the integral constraints. From this point on, the program proceeds as if to enumerate the set of all possible mixed integer solutions by constraining each integer variable singly, in turn, to an integer value within its range. A dual simplex is used as a bound establishing mechanism, immediately after each integer variable is constrained.

Conceivably, one could enumerate all possible solutions to the mixed integer problem. Simultaneously constraining each integer variable to an integer value within the range would be the candidate for the optimal solution. But such an approach would be prohibitive since there is no procedure which would discard the non-promising solutions. Therefore the alternative is to consider each integer variable one by one, rather than simultaneously.

Constrain the first variable, say \(x_i\) to an integer value within its range. This yields a linear program which we proceed to solve using the simplex method. Assuming a feasible solution exists, the first variable is held at its designated value and the second variable is constrained in like manner. We proceed in this fashion, alternately constraining another variable and solving the resulting program, until either we arrive at a feasible solution having constrained all the \(x_i\) otherwise the integer choices for the variables
do not admit a feasible solution. In the first case, we have a candidate for the optimal solution. In the second case it makes no sense to proceed, since a linear program obtained by adding a constraint to a non-feasible linear program must also be non-feasible. In either case, we make a new choice for the integer value of the latest variable and proceed as before. If we have exhausted the range of the first variable the procedure terminates and the solution with the lowest objective function is the optimum solution.

It is possible to eliminate additional resultant problems by making use of information available from the solution of the feasible linear program. We notice that as we proceed in the forward direction, the objective function cannot decrease. In effect, we have established a lower bound on the optimal solution to the (partially) constrained problem immediately following each decision point. Once a feasible solution is obtained, it represents an upper bound on the optimal solution to the original problem. Therefore, at any stage in the procedure if the objective function for the partially constrained problem equals or exceeds that for the current "best" feasible solution, it is unnecessary to continue examination. To this effect dual simplex L.P. algorithm is employed to test for "dominance" after each pivot.

In order to clarify this, consider the case where we wish to constrain $X_k$ to an integer value, if in the previous L.P. solution it had a value $X'k$. We note that the integer value for $X_k$ which results in the smallest increase in the objective function in the succeeding L.P. is either $[X'k]^*$ (the truncated part of $[X'k]$) or $[X'k+1]$ and the farther we proceed from $X'k$ the 'worse' is the objective function. Therefore, if we establish that constraining $X_k$ to $X'k+i$ ( $i$ +ve
integer) produces an increase in the objective function, it need not be considered further. This discussion also applies to the values less than \([X^\prime k]\).

Computationally, the following steps are taken:

1. All\(\geq\) inequalities are complemented to \(=\langle\)

2. Each row is assigned a 'slack/artificial' variable \(W_i\) with \(W_i\) to satisfy
   \[ W_i = \langle \text{if the corresponding constraint is an inequality} \]
   \[ W_i = 0 \text{ if the corresponding constraint is an equality} \]

3. Algorithm is a dual feasible algorithm. This means that the algorithm requires a non-negative initial cost row. If any variable \(X_k\) has a corresponding \(C_k < 0\) then \(X_k\) is replaced by its complement \(X^\prime k\) where \(X^\prime k = (U_k - X_k)\) and \(U_k\) is the upper bound on the variable. This reverses the sign of \(C_k\).

The problem is solved when all \(W_i = \langle 0\).

------------------------------oOo-------------------------------
APPENDIX 3

Mill control problem

The following problem reflects an aspect of the control scheme for a power station.

Consider the following graph.

![Graph Model](image)

Fig AP 3.1 The graph model

The communications and processing requirements of the modulating control loops are as in the following table:

<table>
<thead>
<tr>
<th>TASK NO.</th>
<th>NAME</th>
<th>LOAD(%)</th>
<th>NO. OF MESSAGES/SECOND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GOVERNOR</td>
<td>10</td>
<td>1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>2</td>
<td>FURNACE</td>
<td>10</td>
<td>1 1 2 2 4</td>
</tr>
<tr>
<td>3</td>
<td>FAN A</td>
<td>15</td>
<td>1 0.5</td>
</tr>
<tr>
<td>4</td>
<td>FAN B</td>
<td>15</td>
<td>0.5 0.5</td>
</tr>
<tr>
<td>5</td>
<td>TEMP. A</td>
<td>10</td>
<td>0.5</td>
</tr>
<tr>
<td>6</td>
<td>TEMP. B</td>
<td>10</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>FEED. A</td>
<td>20</td>
<td>0.5</td>
</tr>
<tr>
<td>8</td>
<td>FEED. B</td>
<td>20</td>
<td>0.5</td>
</tr>
<tr>
<td>9</td>
<td>LOGGER</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>EFFICIENCY</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Given the following conditions

(a) If a control centre sends $x$ messages per second, this imposes an extra $2x$ % load on that particular centre.

(b) If a control centre receives $x$ messages per second, an extra $x$ % load is caused on that centre.

(c) If a control centre passes on $x$ messages per second, an extra $x$ % load is caused on that centre.

Given that information the problem can be stated as:

Derive the optimum number of control centres and the distribution of the tasks such that each control centre is evenly loaded. No control centre must be loaded above 50 %. No communication link must carry more than 5 messages per second in each direction. Solve for the following two cases.

Case 1 : There is no constraint on the way in which tasks can be distributed.

Case 2 : Fan A, Temp. A and Feed A must be kept together
          Fan B, Temp. B and Feed B must be kept together

SOLUTION:
First, we evaluate the E and L precedence graphs from the connectivity matrix.

<table>
<thead>
<tr>
<th>TASK NO.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig AP 3.2 The connectivity matrix

From the definition of the reduced graph we observe that the connectivity matrix holds the condition i.e. $T_{ij} = 0$ for all $i < j$. 
The E precedence graph

The L precedence graph
Having constructed \( E \) and \( L \) partitions, the next step is to schedule the tasks starting from the left hand side of the \( E \) precedence graph. The decision process at each level is based on heuristics \( A \) and \( B \). The algorithm is as follows:-

**INITIALISATION**:
Construct one state at the first level. This is state 1 consisting of just task 1. (Since we have assumed the graph such that there is no task graph such that \( (P, S) < 1 \))

**STEP 1:**
\[ i := t_i \]

**STEP 2:**
If all states at the \((i-1)\)th interval of time have been completed, go to step (3), otherwise go to (4).

**STEP 3:**
Pick the terminal state with minimum cost. This is the optimal schedule. Stop.

**STEP 4:**
If all states in the \((i-1)\)th interval of time have been either completed or are being processed go to \((i+1)\)th interval otherwise proceed, to (5).

**STEP 5:**
Pick any state \( S \) at the \((i-1)\)th interval that has not been completed or is being processed. Determine the remaining task graph \( R \in (P, S) \) of state \( (P, S) \) is the subgraph of the task graph obtained by deleting all nodes in \( (P \cup S) \).

**STEP 6:**
Extend \( S \) to the \( i \)th interval. Let \( D(S) = j_1, \ldots, j_q \) be the candidates for processing in the next time interval. Construct a successor state to \( S \) for each distinct subset \( i_1, \ldots, i_k \) of \( D(S) \). Using the dominance criterion together with heuristics \( A \) and \( B \) eliminate all dominated tasks from further consideration. These are therefore \(<(\frac{q}{k})\>
Such successor states (we define \((\frac{q}{k}) = 1 \) if \( q < k \) where \( k \) is the number of active processors). Go to step 2.

Three schedules are developed using this algorithm. The schedules are represented by the Gantt charts [§8] overleaf.

The two computer schedule shows that minimum time to obtain overall control is 85 units. The link capacity and the load limit is exceeded in this case.

The three computer schedule shows that the minimum time to obtain overall control is 75 units. Although the link capacity constraint is satisfied, the load limit is exceeded.

The four processor schedule shows that the minimum time to obtain overall control is still 75 units but all the conditions are satisfied.
THE TWO COMPUTER SCHEDULE

The Gantt chart below illustrates this schedule:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

The communications and overall loading is as follows:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL PROCESSING LOAD (%)</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>TOTAL TRANSMISSION LOAD (%)</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL RECEIVING LOAD (%)</td>
<td>0</td>
<td>9.5</td>
</tr>
<tr>
<td>TOTAL LOAD ON PROCESSOR</td>
<td>84</td>
<td>74.5</td>
</tr>
</tbody>
</table>
THE THREE COMPUTER SCHEDULE

The Gantt chart below, illustrates this schedule:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3</td>
<td>IDLE</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>P2</td>
<td>IDLE</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>P3</td>
<td>10</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

---

The communications and overall loading is as follows:

<table>
<thead>
<tr>
<th>PROCESSING LOAD (%)</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSMISSION</td>
<td>45</td>
<td>35</td>
<td>50</td>
</tr>
<tr>
<td>RECEIVING</td>
<td>20</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>PASSING ON</td>
<td>0</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>TOTAL LOAD (%)</td>
<td>65</td>
<td>42</td>
<td>60</td>
</tr>
</tbody>
</table>

---

PROCESSING TIME

---
THE FOUR COMPUTER SCHEDULE
The gantt chart below, illustrates this schedule:

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>10</td>
<td>15</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>IDLE</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PROCESSING TIME

The communications and overall loading is as follows:

<table>
<thead>
<tr>
<th>PROCESSING LOAD (%)</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSMISSION</td>
<td>19</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>RECEIVING</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>9.5</td>
</tr>
<tr>
<td>PASSING ON</td>
<td>0</td>
<td>3.5</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>TOTAL LOAD</td>
<td>49</td>
<td>41.5</td>
<td>42</td>
<td>39.5</td>
</tr>
</tbody>
</table>
The following program is written in "CORAL, 66", for the HP 21 computer. The program requires the INPUT DATA to be in the INTEGER form. An example illustrating the format of the INPUT and OUTPUT of data for this program is provided later in this appendix.

The program is in three parts. The first part, GHCUT program is used to construct the Gomory-Hu cut-tree. The INPUT DATA for this program consists of the following parameters.

- \( N \) = The number of nodes (tasks) in the network. 
  \( N \) should be less than 40 for HP 21 computer.

- \( RCAP \) = A vector \( (N \times N) \) containing the interactions between the tasks. Only the upper triangular half of this matrix is needed (since \( RCAP(i,j) = RCAP(j,i) \) and \( RCAP(i,i) = 0 \)).

  \[ i.e \text{ the communication FROM}[i] \text{ to TO}[j] \text{ is } RCAP[(i-1)N+j]. \]

This data is stored in a file called IDATA. The number of tasks \( N \) being the first value. Then the upper triangular half of the task interactions matrix is written in order of \( RCAP(1, 2), RCAP(1, 3) \ldots \) \( RCAP(N-1, N) \). Then the nominal value of the SOURCE and the SINK are asked for in the program. These values should be in the range of 1 to \( N \) and SOURCE not equal to SINK.

The output from this program is stored in a file called TE and this consists of the following data.

- \( GHLINKS \) = A \((N \times N)\) matrix showing the links in the cut-tree.

- \( MAFLOW \) = An \((N \times 1)\) vector showing the weights (minimum communication) on the corresponding links in the \( GHLINKS \).

- \( LABELS \) = A \((N \times 1)\) vector showing the grouping of the tasks.

The second part of the program is called the PORDER. The information from the GOMORY -HU cut tree provides the PARTIAL ORDER of the tasks. The task numbers and the processing requirements of each task are then written in a file called PODATA as shown in Table AP 4.4. The last entry in this file should be the LOADLIMIT on the computers.

The output of this program is stored in a file called SEQDATA. The information stored in SEQDATA is:

- \( N \) = The number of nodes in the PARTIAL ORDER GRAPH,

- \( LOADLIMIT \) = The capacity limit on computers.
PO[N] = The PARTIAL ORDER of the tasks.

RCAP[NxN] = The communication between the tasks and their processing requirements.

The third part of the program is SEQPART which is used to partition the PARTIALLY ORDERED GRAPH constructed by PORDER. The INPUT DATA for this program is from SEQDATA as constructed by PORDER (above).

The output of this program is the BREAKPOINTS in the PARTIALLY ORDERED GRAPH. These BREAKPOINTS are listed in a descending order and each partition is specified. The output is stored in a file called TE2. This file contains the information of the number of partitions in the PARTIAL ORDER GRAPH and each partition (group) indicates the assignment of tasks (the numbers of the tasks correspond to the original numbers in IDATA). Finally, the inter-computer communication is represented in a matrix called LINKAGE, where the element (i,j) of this matrix represents the data transfer from computer i to computer j. The element (i,i) indicates the processing load on the ith computer. The first group of tasks refers to the first computer and so on.
PROGRAM GHCUT
LIBRARY ("&SIOLB::LB");
SEGMENT ONE
BEGIN

INTEGER I, J, K, V, N, NODE, NN, P, TT, TAG, SOURCE, SINK, SWITCH, MFLOW, FILL;
INTEGER X1, X2, X5, X7, IN, OSTREAM;
INTEGER "ARRAY" LABJ, SET, LASTLABEL, P1, MAFLOW, LABEL[1:40];
INTEGER "ARRAY" CAP, RCAP, FROM, TO, GH, GHLINK, FLOW[1:1600];
PROCEDURE MAXFLOW("INTEGER" "ARRAY" FROM, TO, LABJ, CAP, FLOW;
VALUE "INTEGER" V, N, SOURCE, SINK;
LOCATION "INTEGER" MFLOW);

COMMENT This procedure determines the minimum communication
between the nominated SOURCE and the nominated SINK.
The value of the minimum communication returned to
the main subroutine is MFLOW.

BEGIN
INTEGER L, J, K, R, Q, LK, EK, U, S, D, INF, EPS, GJK;
INTEGER "ARRAY" LOW, LABF, UP, KLIST[1:40], IND[1:1600];
L:=1;
INF:=32000;
EPS:=1;
FOR J:=1STEP 1 UNTIL N DO
BEGIN
LOW[J]:=L;
FOR R:=1STEP 1 UNTIL V DO
BEGIN
IF FROM[R]=J THEN
BEGIN
IND[L]:=R;
FLOW[L]:=CAP[L];L:=L+1;
END;
END;
UP[J]:=L-1;
END;
MFLOW:=0;
LAB:
FOR J:=1STEP 1 UNTIL N DO
BEGIN
KLIST[J]:=0;
LABF[J]:=0;
LABJ[J]:=0;
END;
LABF[SOURCE]:=INF;

COMMENT labeling.

; J:=SOURCE; LK:=0;
EK:=0;
S:=0;
PATH:
U:=UP[J];
FOR S:=LOW[J]STEP 1 UNTIL U DO
BEGIN
L:=IND[S];

PAGE 155
K:=TO[L]; GJK:=FLOW[L];
'IF' GJK<0 'THEN' Q:=-GJK 'ELSE' Q:=GJK;
'IF' LABJ[K]='NE''O 'OR' Q<EPS 'THEN' 'GOTO' NED;
LABJ[K]:=J;
LABF[K]:='IF' GJK<LABF[J] 'THEN' GJK 'ELSE' LABF[J];
'IF' K=SINK 'THEN' 'GOTO' REACHED;
LK:=LK+1; KLIST[LK]:=K;
NED:
'END'';
EK:=EK+1; J:=KLIST[EK];
'IF' J='NE''O 'THEN' 'GOTO' PATH 'ELSE' 'GOTO' MAX;

'COMMENT' SINK is labeled, find path and FLOW.

REACHED:
J:=SINK; D:=LABF[J]; MFLOW:=MFLOW+D;
LOOK:
K:=LABJ[J];
U:=UP[K];
'FOR' S:=LOW[K] 'STEP' 1 'UNTIL' U 'DO'
'BEGIN'
L:=IND[S];
'IF' TO[L]=J 'THEN' FLOW[L]:=FLOW[L]-D
'END'';
U:=UP[J];
'FOR' S:=LOW[J] 'STEP' 1 'UNTIL' U 'DO'
'BEGIN'
L:=IND[S];
'IF' TO[L]=K 'THEN' FLOW[L]:=FLOW[L]+D
'END'';
J:=K; 'IF' J='NE''SOURCE 'THEN' 'GOTO' LOOK;
'GOTO' LAB;
MAX:;

'COMMENT' Minimum communication value found.

; ;

'FOR' L:=1 'STEP' 1 'UNTIL' V 'DO'
FLOW[L]:=CAP[L]-FLOW[L];
'END' OF PROCEDURE MAXFLOW;
PROCEDURE SORT(VALUE INTEGER N;
LOCATION INTEGER V;
INTEGER ARRAY FROM,TO,RCAP,CAP);

COMMENT This procedure is used to select the non-zero values of the interactions between the tasks. The non-zero entries i.e the task interactions FROM [i] to TO[j] is RCAP[v], where v=(i-1)*N+j. This value is stored in CAP[v]. The vectors CAP[v], FROM[i], TO[j] are then updated for the procedure MAXFLOW.

; BEGIN
  V:=0;
  FOR I:=1 STEP 1 UNTIL N DO
  BEGIN
    FOR J:=1 STEP 1 UNTIL N DO
    BEGIN
      IF RCAP[(I-1)*N+J]>0 THEN
      BEGIN
        V:=V+1;
        FROM[V]:=I;
        TO[V]:=J;
        CAP[V]:=RCAP[(I-1)*N+J];
      END;
    END;
  END;
END OF PROCEDURE SORT;

PROCEDURE CHECK(VALUE INTEGER NODE,TAG;
LOCATION INTEGER TT,P;
INTEGER ARRAY LABEL);

COMMENT This procedure is used to determine the number of tasks in a particular group i.e the group containing SOURCE or the group containing SINK. The number of tasks in that group is P. TT is the highest numbered task belonging to the other group.

; BEGIN
  INTEGER X4;
  P:=0;
  TT:=0;
  FOR X4:=1 STEP 1 UNTIL NODE DO
  BEGIN
    IF LABEL[X4]=TAG THEN P:=P+1;
    IF LABEL[X4]=TAG THEN TT:=X4;
  END;
END OF PROCEDURE CHECK;
PROCEDURE FINDSOURCE(VALUE INTEGER NODE, K, SWITCH, TAG;
LOCATION INTEGER SOURCE, SINK;
INTEGER ARRAY LABEL, PI);

COMMENT This procedure is used to find the SOURCE node and the SINK node from the condensed network. The nodes that are condensed are represented by the vector PI[i]. If PI[i] has a positive value then node i has been condensed.

BEGIN
INTEGER X3, X9;
X3 := 1;
FINISH: IF LABEL[X3] NE TAG THEN
BEGIN
X3 := X3 + 1;
GOTO FINISH;
END;
SINK := 0; SOURCE := 1;
FOR I := 1 STEP 1 UNTIL NODE DO
BEGIN
PI[I] := 0;
IF LABEL[I] = TAG THEN
BEGIN
SINK := SINK + 1;
X9 := I;
END;
END;
K := 0;
MAP:
K := K + 1;
IF K = TAG THEN GOTO MAP;
IF K > SWITCH THEN GOTO RASS;
CHECK(NODE, K, TT, P, LABEL);
IF P < 1 THEN GOTO MAP;
PI[TT] := TT;
IF X3 > TT THEN
BEGIN
SOURCE := SOURCE + 1;
END;
IF X9 > TT THEN SINK := SINK + 1;
GOTO MAP;
RASS:
X2 := X3; X5 := X9;

COMMENT X3 is the new SOURCE and X9 is the new SINK.

; ENDOF PROCEDURE FINDSOURCE;
PROCEDURE PARTITION(VALUE INTEGER N, SOURCE, SINK, TAG;
  INTEGER ARRAY SET, LABJ);

COMMENT This procedure determines the minimal cut. The group which contains the SOURCE node is labeled TAG and the group which contains the SINK node is labeled TAG+1.

BEGIN
  FOR I:=1 STEP 1 UNTIL N DO
  BEGIN
    IF LABJ[I]>0 THEN SET[I]:=TAG ELSE SET[I]:=TAG+1;
  END;
  SET[SOURCE]:=TAG; SET[SINK]:=TAG+1;
END OF PROCEDURE PARTITION;

PROCEDURE GHSTAT(VALUE INTEGER NODE, SOURCE, TAG, MFLOW;
  INTEGER ARRAY GH, LABEL, LASTLABEL, LABJ, MAFLOW);

COMMENT start constructing the GH matrix which would show the minimum communication between each and every task.

BEGIN
  FOR K:=1 STEP 1 UNTIL NODE DO
  BEGIN
    TAG:=LASTLABEL[K]; MFLOW:=MAFLOW[K];
    FOR I:=1 STEP 1 UNTIL NODE DO
    BEGIN
      IF LABJ[I]=TAG+1 THEN
      BEGIN
        FOR J:=1 STEP 1 UNTIL NODE DO
        BEGIN
          IF LABJ[J]=TAG THEN
          BEGIN
            IF GH[(J-1)*NODE+I]>MFLOW THEN
            GH[(J-1)*NODE+I]:=MFLOW;
            GH[(I-1)*NODE+J]:=MFLOW;
            END;
          END;
        END;
      END;
    END;
  END;
END OF PROCEDURE GHSTAT;
This procedure determines the links in the GOMORY-HU cut-tree. First, all the nodes are considered to be linked to the SOURCE node if they are in different group then, gradually they are set to zero if they do not remain in the same group as the SOURCE node. The SOURCE node is represented by a positive value and the SINK is represented by the -ve value.

\[
\text{BEGIN} \ \ \ \ \\
\text{FOR} \ I:=1 \ \text{STEP} \ 1 \ \text{UNTIL} \ \text{NODE} \ \text{DO} \ \\
\text{BEGIN} \ \\
\quad X7:=X7+1; \\
\quad \text{IF} \ \text{LASTLABEL}[I]=\text{TAG} \ \text{THEN} \ \\
\quad \quad \text{BEGIN} \\
\quad \quad \quad \text{IF} \ \text{LABJ}[I]=\text{TAG} \ \text{THEN} \ \text{GHLINK}[X7]:=I; \\
\quad \quad \quad \text{IF} \ \text{LABJ}[I]=\text{TAG}+1 \ \text{THEN} \ \text{GHLINK}[X7]:=-I; \\
\quad \quad \quad \text{END} \\
\quad \quad \text{ELSE} \ \text{GHLINK}[X7]:=0; \\
\quad \quad \text{IF} \ \text{P1}[I]=I \ \text{OR} \ \text{GHLINK}[X7]=0 \ \text{THEN} \\
\quad \quad \text{BEGIN} \\
\quad \quad \quad \text{FOR} \ K:=1 \ \text{STEP} \ 1 \ \text{UNTIL} \ (\text{FILL}-1) \ \text{DO} \\
\quad \quad \quad \quad \text{BEGIN} \\
\quad \quad \quad \quad \quad \text{IF} \ \text{GHLINK}[(K-1)*\text{NODE}+I] \neq 0 \ \text{THEN} \\
\quad \quad \quad \quad \quad \quad \text{BEGIN} \\
\quad \quad \quad \quad \quad \quad \quad \text{FOR} \ J:=1 \ \text{STEP} \ 1 \ \text{UNTIL} \ \text{NODE} \ \text{DO} \\
\quad \quad \quad \quad \quad \quad \quad \quad \text{IF} \ \text{LABJ}[J] \neq \text{LABJ}[I] \ \text{THEN} \ \text{GHLINK}[(K-1)*\text{NODE}+J]:=0; \\
\quad \quad \quad \quad \quad \quad \quad \quad \text{END}; \\
\quad \quad \quad \quad \quad \quad \quad \text{END}; \\
\quad \quad \quad \quad \quad \quad \text{END}; \\
\quad \quad \quad \quad \quad \text{END}; \\
\quad \quad \text{END}; \\
\quad \text{END}; \\
\text{END} \ \text{OF} \ \text{PROCEDURE} \ \text{GHLINKS};
\]
"PROCEDURE" ASSIGN("VALUE" "INTEGER" NODE,TAG;
 "LOCATION" "INTEGER" SWITCH;
 "INTEGER" "ARRAY" LABEL,SET,P1,LABJ);

"COMMENT" Determine the nodes that have been partitioned and label them same if they are in the same group. The nodes that have not yet been partitioned are labeled greater than the present TAG value.

; BEGIN "INTEGER" SJ;
 SJ:=1;
 SWITCH:=SWITCH+1;
 FOR I:=1 "STEP" 1 "UNTIL" NODE "DO"
 BEGIN IF LABEL[I]=TAG "THEN"
 BEGIN
 LAB J [I] :=SET[SJ];
 LABEL[I]:=SET[SJ];
 SJ:=SJ+1;
 END
 ELSE "BEGIN"
 IF LABEL[I]>TAG "THEN" LABEL[I]:=LABEL[I]+1;
 IF P1[I]=1 "THEN"
 BEGIN
 FOR J:=1 "STEP" 1 "UNTIL" NODE "DO"
 IF LABEL[J]=LABEL[I] "THEN" LAB J [J] :=SET[SJ];
 SJ:=SJ+1;
 END;
 END;
 END; END OF PROCEDURE ASSIGN;
PROCEDURE CONDENSE("VALUE" INTEGER NODE;
    "LOCATION" INTEGER V,SINK,SOURCE,N;
    INTEGER ARRAY CAP,GH,FROM,TO,LABEL,PI);

COMMENT This procedure is used to condense the nodes in the network. The nodes that are to be condensed are labeled greater than TAG value or less than the TAG value and this is also indicated by the non-zero value in the P1 vector. This procedure also calls the FINDSOURCE procedure in order to determine the new SOURCE and the new SINK from the condensed network.

BEGIN
INTEGER TEMP,VIV,DIV;
N:=0;V:=0;VIV:=0;DIV:=0;K:=0;
FINDSOURCE(NODE,K,SWITCH,TAG,SOURCE,SINK,LABEL,P1);
FOR I:=1 STEP 1 UNTIL NODE DO
BEGIN
  IF LABEL[I] NE TAG AND I NE P1[I] THEN
    BEGIN
      K:=LABEL[I];
      CHECK(NODE,K,TT,P,LABEL);
      IF TT NE 0 THEN
        BEGIN
          FOR J:=1 STEP 1 UNTIL NODE DO
          BEGIN
            IF LABEL[J]=TAG OR J NE TT THEN
              BEGIN
                TEMP:=(TT-1)*NODE+J;
                GH[TEMP]:=GH[TEMP]+GH[(I-1)*NODE+J];
                TEMP:=(J-1)*NODE+TT;
                GH[TEMP]:=GH[TEMP]+GH[(J-1)*NODE+I];
              END;
          END;
        END;
      END;
    END;
  END;
END;
FOR I:=1 STEP 1 UNTIL NODE DO
BEGIN
  IF LABEL[I]=TAG OR I=P1[I] THEN
    BEGIN
      VIV:=VIV+1;
      DIV:=0;
      FOR J:=1 STEP 1 UNTIL NODE DO
      BEGIN
        IF LABEL[J]=TAG OR J=P1[J] THEN
          BEGIN
            V:=V+1;
            DIV:=DIV+1;
            FROM[V]:=VIV;
            TO[V]:=DIV;
            CAP[V]:=GH[(I-1)*NODE+J];
          END;
      END;
    END;
  END;
END;
N:=VIV;
TT:=0;
END OF PROCEDURE CONDENSE;
This is the main procedure which reads in the data and prepares the data for the minimal cut procedure and then builds the GOMORY-HU cut-tree.

OPEN STREAM(CONSOLE,OSTREAM);
OUT STREAM(OSTREAM);
OPEN STREAM("IDATA::KB",IN);
IN STREAM (IN);
IN INT (N,"I");
"FOR" I:=1 "STEP" 1 "UNTIL" (N) "DO"
"BEGIN"
    P1[I]:=0;
    "FOR" J:=(I+1) "STEP" 1 "UNTIL" N "DO"
    "BEGIN"
        IN INT (RCAP[(1-1)*N+J],"I");
        RCAP[(J-l)*N+I]:=RCAP[(I-1)*N+J];
        CAP[(1-1)*N+J]:=RCAP[(I-1)*N+J];
        CAP[(J-l)*N+I]:=RCAP[(I-1)*N+J];
    "END";
    RCAP[(I-1)*N+I]:=0;
    CAP[(I-1)*N+I]:=0;
"END"
CLOSE STREAM(IN);

"COMMENT" The number of nodes and their interactions have been read in the RCAP vector and duplicated in the CAP vector.

OPEN STREAM(CONSOLE,IN);
REDO:
    OUT CHAR(NEWLINE);
    OUT TEXT("SOURCE = ");
    OUT CHAR(NEWLINE);
    IN STREAM(IN);
    IN INT(SOURCE,"I");
    OUT CHAR(NEWLINE);
    OUT TEXT("SINK = ");
    OUT CHAR(NEWLINE);
    IN INT(SINK,"I");
    OUT CHAR(NEWLINE);
    "IF" SOURCE "GT" N "OR" SINK "GT" N "THEN" "GOTO" REDO;
    "IF" SOURCE "LT" 1 "OR" SINK "LT" 1 "THEN" "GOTO" REDO;
    "IF" SOURCE = SINK "THEN" "GOTO" REDO;

"COMMENT" make certain that the nominated SOURCE and the SINK are within the specified number of nodes.

CLOSE STREAM (OSTREAM);
OPEN STREAM("TE::KB",OSTREAM);
"COMMENT" open the file TE to store the results.

;

OUT STREAM(OSTREAM);
"FOR" I:=1 "STEP" 1 "UNTIL" N "DO"
LABEL[I]:=1;
NODE:=N;
NN:=N;
SWITCH:=1;
TT:=0;
XI:=1;
TAG:=1;
X2:=SOURCE;X5:=SINK;
FILL:=0;
X7:=0;
SORT(N,V,FROM,TO,RCAP,CAP);
"COMMENT" the non-zero values in the CAP vector have been
eliminated and the positive entries are stored
in FROM[nodei] to TO[nodej] the flow is CAP[v].

RPT: .  '  '
MAXFLOW(FROM,TO,LABJ,CAP,FLOW,V,N,SOURCE,SINK, MFLOW);
FILL:=FILL+1;
MAFLOW[FILL]:=MFLOW;
"FOR" I:=1 "STEP" 1 "UNTIL" NODE "DO"
LASTLABEL[I]:=LABEL[I];
"COMMENT" Determine the minimal cut.

;

PARTITION(N,SOURCE,SINK,TAG,SET,LABJ);
ASSIGN(NODE,TAG,SWITCH,LABEL,SET,PI,LABJ);
GHLINKS(NODE,FILL,TAG,X7,LASTLABEL,LABJ,GHLINK,PI);
CHECK(NODE,TAG,TT,P,LABEL);
HOME: "IF" (NN-P)=1 "AND" P>1 "THEN"
"BEGIN"
"COMMENT" If SINK is the only node in the group then there is
no need for further condensation of the network ,
simply use the present network.

;

FINDSOURCE(NODE,K,SWITCH,TAG, SOURCE,SINK,LABEL,PI);
NN:=NN-1;
"IF" SOURCE='NE' SINK "THEN" 'GOTO' RPT;
"END";
"IF" P>1 "THEN"
"BEGIN"
"COMMENT" There are more than one node in the SOURCE
group therefore donot change the tag value
but condense the network and find the new
SINK value.

;

NN:=P;
FOR I:=1 STEP 1 UNTIL NODE*NODE DO
GH[I]:=RCAP[I];
CONDENSE(NODE,V,SINK,SOURCE,N,CAP,GH,FROM,TO,LABEL,P1);
SORT(N,V,FROM,TO,CAP,CAP);
GOTO RPT;
END;
LUVE: TAG:=TAG+1;

COMMENT The SOURCE node is the single node in the group
therefore move to the next higher labeled node
which has not been labeled yet.

;
IF TAG>NODE THEN GOTO GHLINKER;
CHECK(NODE,TAG,TT,P,LABEL);
IF P<=1 THEN GOTO LUVE;
FOR I:=1 STEP 1 UNTIL NODE DO
IF LABEL[I]<TAG AND LABEL[I]>=X1 THEN LABEL[I]:=X1;
COMMENT The nodes that were in the same group at
the previous tag value at the first
partition are labeled with that tag value
to form a single subset for condensation.
;
X1:=TAG;
GOTO HOME;
GHLINKER:

COMMENT (N-l) computations have been carried out.
Hence print the GOMORY-HU cut-tree links and the
weights (representing minimum communication) of
the links.
;
OUT TEXT(" ");
OUT TEXT("GHLINKS");
OUT TEXT(" ");
OUT CHAR(NEWLINE);
FOR I:=1 STEP 1 UNTIL (NODE-1) DO
BEGIN
FOR J:=1 STEP 1 UNTIL NODE DO
BEGIN
X7:=(I-1)*NODE+J;
OUT INT (GHLINK[X7],"14");
END;
END;
OUT CHAR (NEWLINE);
END;
OUT CHAR(NEWLINE);
OUT CHAR(NEWLINE);
OUT TEXT("THE MINIMUM COMMUNICATION VALUES ARE :");
OUT CHAR(NEWLINE);
FOR I:=1 STEP 1 UNTIL (NODE-1) DO
OUT INT(MAFLOW[I],"14");
OUT CHAR(NEWLINE);
OUT TEXT("THE FINAL LABELS ARE");
OUT CHAR(NEWLINE);
"FOR" I:=1 "STEP" 1 "UNTIL" NODE "DO"
"BEGIN"
    OUT INT (LABEL[I],"I4");
"END";
OUT CHAR(NEWLINE);
CLOSE ALL STREAMS;
"END"
"FINISH"
This is the second part of the program, which constructs the PARTIAL ORDER GRAPH. The input to this procedure is \( N \) (the number of nodes) and vector \( PO[N] \) which indicates the partial order of the nodes (tasks).

LIBRARY ("&SIOLB::LB");
SEGMENT ONE
BEGIN
INTEGER I, K, L, NODE, IN, OSTREAM, LOADLIMIT;
INTEGER ARRAY PO[1:40];
INTEGER ARRAY CAP, RCAP[1:1600];
OPEN STREAM("SEQDATA::KB", OSTREAM);
OUT STREAM(OSTREAM);
OPEN STREAM("PODATA::KB", IN);
IN STREAM (IN);
IN INT (NODE, "I");
FOR I := 1 "STEP" 1 "UNTIL" NODE "DO"
BEGIN
IN INT(PO[I], "I");
K := PO[I];
IN INT(CAP[(K-1)*NODE+K], "I");
END;
IN INT (LOADLIMIT, "I");
CLOSE STREAM(IN);
OPEN STREAM("IDATA::KB", IN);
IN STREAM (IN);
IN INT (NODE, "I");
FOR I := 1 "STEP" 1 "UNTIL" (NODE-1) "DO"
BEGIN
FOR J := (I+1) "STEP" 1 "UNTIL" (NODE) "DO"
BEGIN
IN INT (CAP[(I-1)*NODE+J], "I");
CAP[(J-1)*NODE+I] := CAP[(I-1)*NODE+J];
END;
END;
END;
CLOSE STREAM(IN);
FOR I := 1 "STEP" 1 "UNTIL" (NODE) "DO"
BEGIN
K := PO[I];
FOR J := I "STEP" 1 "UNTIL" NODE "DO"
BEGIN
L := PO[J];
RCAP[(I-1)*NODE+J] := CAP[(K-1)*NODE+L];
END;
END;
END;
OUT INT(NODE+1, "I4");
OUT CHAR(NEWLINE);
OUT INT(LOADLIMIT, "I4");
OUT CHAR(NEWLINE);
FOR I := 1 "STEP" 1 "UNTIL" NODE "DO"
BEGIN
OUT INT (PO[I], "I3");
OUT CHAR(NEWLINE);
END;
FOR J := I "STEP" 1 "UNTIL" NODE "DO"
BEGIN
END;
`BEGIN`
  OUT INT (RCAP[(I-1)*NODE+J],"I4");
`END``
  OUT TEXT("  0");
  OUT CHAR(NEWLINE);
`END``
  OUT TEXT("  0");
  OUT CHAR(NEWLINE);
CLOSE ALL STREAMS;
`END`
`FINISH`
This is the third part of the program, which determines the feasible partitions of the PARTIALLY ORDERED GRAPH. The data is read in as in the GH CUT but this time the element RCAP(i,i) which represents the processing requirements of task i is also required. The task grouping is specified by the BREAKPOINTS vector. The matrix LINKAGE is used to represent inter-computer data transfer.

`LIBRARY` ("&SIOLB::LB");
`SEGMENT` ONE
`BEGIN`

`INTEGER` I,J,K,LOADLIMIT,SETWEIGHT,C,Y,X,NODE,IN,OSTREAM,P,Q,R,TDATA;
`INTEGER` L,M,PQ;
`INTEGER` `ARRAY` TASKSET,BREAKPOINT,PO[1:40];

`COMMENT` LOADLIMIT i the processing capacity limit on the computer.

`INTEGER` `ARRAY` RCAP,LINKAGE[1:1600];
OPEN STREAM("TE2::KB",OSTREAM);
OUT STREAM(OSTREAM);
OPEN STREAM("SEQDATA::KB",IN);
IN STREAM (IN);
IN INT (NODE,"I");
IN INT (LOADLIMIT,"I");
TDATA:=0;
`FOR` I:=l `STEP` 1 `UNTIL` (NODE-1) `DO`
`IN` INT(P0[I],"I");
`FOR` I:=1 `STEP` 1 `UNTIL` (NODE) `DO`
`BEGIN`
`FOR` J:=(I) `STEP` 1 `UNTIL` (NODE) `DO`
`BEGIN`
IN INT ( RCAP[(I-1)*NODE+J],"I");
RCAP[(J-1)*NODE+I]:=RCAP[(I-1)*NODE+J];
`IF` J `NE` I `THEN` TDATA:=TDATA+RCAP[(I-1)*NODE+J];
`END``END``FOR` X:=2 `STEP` 1 `UNTIL` NODE `DO`
`BEGIN`
REP:
SETWEIGHT:=0;

PAGE 169
FOR $I := Y \text{ STEP } 1 \text{ UNTIL } (X-1) \text{ DO}$
BEGIN
   SETWEIGHT := SETWEIGHT + RCAP[(I-1)*NODE+J];
END;
IF SETWEIGHT $GT$ LOADLIMIT THEN
BEGIN
   Y := Y + 1;
   IF Y $LT$ X THEN GOTO REP;
END;
TASKSET[X] := 32000;
FOR $I := Y \text{ STEP } 1 \text{ UNTIL } (X-1) \text{ DO}$
BEGIN
   C := 0;
   FOR $K := I \text{ STEP } 1 \text{ UNTIL } (X-1) \text{ DO}$
   BEGIN
      FOR $J := X \text{ STEP } 1 \text{ UNTIL } NODE \text{ DO}$
      BEGIN
         C := C + RCAP[(K-1)*NODE+J];
      END;
   END;
   IF TASKSET[X] $GT$ TASKSET[I] + C THEN
   BEGIN
      TASKSET[X] := TASKSET[I] + C;
      BREAKPOINT[X] := I;
   END;
END;
END;
BREAKS:
COMMENT The feasible partitions have been found.
The partitions are listed in the decending order.

; $K := \text{BREAKPOINT}[Y]$;
$X := X + 1$;  COMMENT count the number of computers needed;
TASKSET[X] := Y;
OUT CHAR(NEWLINE);
OUT TEXT("THE TASKS IN THIS GROUP ARE");
OUT CHAR(NEWLINE);
FOR $I := K \text{ STEP } 1 \text{ UNTIL } (Y-1) \text{ DO}$
OUT INT(PO[I],"I4");
Y := K;
IF Y $GT$ 1 THEN GOTO BREAKS;
OUT CHAR(NEWLINE);
OUT CHAR(NEWLINE);
OUT CHAR(NEWLINE);
OUT TEXT ("THE INTER-COMPUTER COMMUNICATION IS ");
"FOR" I := 1 "STEP" 1 "UNTIL" (X-1) "DO"
"BEGIN"
Y:=TASKSET[I];
K:=BREAKPOINT[Y];
LINKAGE[(I-1)*X+I]:=0;
"FOR" J:=(I+1) "STEP" 1 "UNTIL" X "DO"
"BEGIN"
R:=(I-1)*X+J;
LINKAGE[R]:=0;
L:=TASKSET[J];M:=BREAKPOINT[L];
"FOR" P:=K "STEP" 1 "UNTIL" (Y-1) "DO"
"BEGIN"
"FOR" Q:=M "STEP" 1 "UNTIL" (L-1) "DO"
"BEGIN"
PQ:=(P-1)*NODE+Q;
LINKAGE[R]:=LINKAGE[R]+RCAP[PQ];
"END";
"END";
"END";
"END";
OUT CHAR(NEWLINE);
"FOR" I := 1 "STEP" 1 "UNTIL" X "DO"
"BEGIN"
Y:=TASKSET[I];
K:=BREAKPOINT[Y];
LINKAGE[(I-1)*X+I]:=0;
"FOR" P:=K "STEP" 1 "UNTIL" (Y-1) "DO"
"BEGIN"
LINKAGE[(I-1)*X+I]:=LINKAGE[(I-1)*X+I]+RCAP[(P-1)*NODE+P];
"END";
"END";
"FOR" I := 1 "STEP" 1 "UNTIL" X "DO"
"BEGIN"
"FOR" J:= I "STEP" 1 "UNTIL" X "DO"
"BEGIN"
OUT INT(LINKAGE[(I-1)*X+J],"I5");
"END";
OUT CHAR (NEWLINE);
"END";
OUT CHAR(NEWLINE);
OUT CHAR(NEWLINE);
OUT TEXT("TOTAL DATA TRANSFERRED = ");
OUT INT(TASKSET[NODE],"I5");
OUT CHAR(NEWLINE);
OUT TEXT("TOTAL DATA SAVED = ");
OUT INT (TDATA-TASKSET[NODE],"I5");
OUT CHAR(NEWLINE);
CLOSE ALL STREAMS;
"END"
"FINISH"
An Example : Application of the Program

The problem considered here is the same as in appendix 1 but we consider the complete solution of the problem. Given a set of tasks, find the assignment of these tasks subject to the design criteria mentioned in chapter two. The processing requirements and the communications requirements of these tasks is as follows:

(Note that the tasks 8, 9, 10 and 11 are condensed into a single task (no.8) as shown below and the task loads are converted to integers.)

NB. The letters A, B, C, D, E, F, G, H, J, K, denote the mills.

Table AP 4.1 The task loads and their interactions.
The data required in the first part of the program (GHCUT) is typed by the user. The data is in a file called IDATA and the format of the input is as follows. (Note the task loads are omitted at this stage).

Table AP 4.2 The data input file.
The output from GHCUT is written by the computer in a file called TE, which specifies the GOMORY-HU cut-tree links and the weight (minimum communication) on them. The weight on the link in the jth row is represented by the jth element (from the left) in the minimum communication vector. The table AP4.3 shows the output from GHCUT. Fig AP4.1 overleaf shows the corresponding GOMORY-HU cut-tree.

Table AP 4.3 The cut-tree links.
Fig. A4.1 The illustration of minimum communications cut-tree.
The labels in Table AP 4.3 show the grouping of the tasks. This information and the cut-tree is used to decide the PARTIAL ORDER of the tasks. This PARTIAL ORDER and the processing requirements of the tasks are written by the user in a file called PODATA as shown below in Table AP 4.4. The first value in each row is the task number and the second is its processing requirements. The last entry is the capacity limit set on the computers.

<table>
<thead>
<tr>
<th>Task</th>
<th>Processing Requirements</th>
<th>Capacity Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>3 52</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>92</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>131</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>382</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>234</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>954</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table AP 4.4 Partial order of the tasks.
The output from PORDER is as shown in Table AP 4.5. This is stored in a file called SEQDATA and this file is the input file for the final part of the program. The first value in Table AP4.5 corresponds to the number of nodes in the PARTIAL ORDER GRAPH. The second value is the capacity limit set on the computers. Then the ORDER of the tasks is shown i.e. task 3 becomes task 1 in the PARTIAL ORDER GRAPH. The task interactions are arranged accordingly by PORDER as shown below.

Table AP 4.5 The data input for feasible partitions.
The output from SEQPART is stored in a file called TE2. The computer assignment of the tasks is represented by the groups and the task numbers in each group correspond to the original task numbers in Table AP 4.1. Finally, the inter-computer communication is shown in the LINKAGE matrix in Table AP 4.6. The element (i, i) of this matrix shows the task loads on the ith computer and the element (i, j) shows the data transfer from the ith computer to the jth computer. First group corresponds to the first row of the matrix and the second corresponds to the second and so on.

```
1 1 1 1 1 1 1 9 0 8 9 8 8 9 12 15 16 17 18 19 28 21 22 23 24 24 26 28 29
0 3 30 32 64 48 68 56 57 59 66 92 92 92 92 92 92 92 92 92 92 109 115 112 110 117 114 113 111

THE TASKS IN THIS GROUP ARE
31 32 20 39
THE TASKS IN THIS GROUP ARE
18 4 21
THE TASKS IN THIS GROUP ARE
9 8
THE TASKS IN THIS GROUP ARE
11 18
THE TASKS IN THIS GROUP ARE
13 12
THE TASKS IN THIS GROUP ARE
15 14
THE TASKS IN THIS GROUP ARE
7 6 17 16
THE TASKS IN THIS GROUP ARE
25 24 23 22
THE TASKS IN THIS GROUP ARE
3 2 19 29 28 27 26

THE INTER-COMPUTER COMMUNICATION IS
532 10 0 0 0 0 3 0 0
915 0 0 0 0 5 0 0 0
604 0 0 6 0 6 0
604 0 6 0 6 0 6
604 6 0 6 0 6
604 6 0 6 0 6
604 6 0 6 0 6
604 6 0 6 0 6
623 4 15
436 12
945

TOTAL DATA TRANSFERRED = 110
TOTAL DATA SAVED = 64
```

Table AP 4.6 Computer assignment and their communications.
INTRODUCTION

Background

Developments in L.S.I. technology have changed the basic framework in which computing systems are designed. It has become feasible economically and technically to link small computers ranging from loosely coupled distributed system (1) to tightly coupled multiprocessor systems (2). The work undertaken in this paper is to configure an optimum network structure for computers to control a power station.

Until recently, the control systems employed within C.E.G.B. power stations have been based on groups of equipment: that is, analogue equipment for regulating control or relay based equipment for sequencing and interlocking. The advantages of this approach have been:

- Functional segregation of the control activities
- Easy incremental commissioning and testing
- Easy scheme extension by adding new equipment.

However, such equipment does not readily lend itself to the implementation of more flexible control schemes. A number of attempts have been made to provide this flexibility by the use of a large digital computer for control applications (3). For economic reasons a substantial proportion of the control requirements would be committed to such a computer. This in turn leads to the large overheads and unreliability associated with the complex software systems. Therefore, a standby machine of equal size would be required in case of failure, and the advantages associated with functional segregation are lost.

Distributed computer control network

With the advent of low cost mini and microcomputers it has become feasible to consider a distributed control and instrumentation (C&I) scheme by a network of small computers. Such a scheme may be defined as one in which autonomous groups of equipment working in parallel are used to implement individual sub-systems which together constitute the overall control scheme. The advantages associated with this approach are easy maintenance, integration of modulating/sequence control schemes and flexibility for expansion or modification (4).

The envisaged network would be based upon a set of modularly designed intelligent nodes, each node comprising a control centre with its associated plant and communications system interface. Each control centre will be able to handle a portion of the overall control scheme in a reasonable autonomous manner. However, in practice sub-system autonomy is not absolute, since some co-ordination with other sub-systems is needed.

Network layout

In determining the appropriate network a number of conflicting objectives must be resolved. For example, an approach based on one control task per control centre, while admirable in terms of availability would result in an unacceptable number of control centres and a significant loss in autonomy (since a substantial transfer of data between control centres would be continually required). Hence, the correct choice of network configuration, that is the manner in which the control tasks are divided between a given number of control centres, is a pre-requisite to the successful implementation of a distributed control and instrumentation scheme (4).

Aim of the paper

It is the intention of this paper to propose mathematical models to achieve the following two objectives:

1. Determination of the optimum subdivision of the total control scheme into a number of functional groups, and the assignment of these groups to appropriate control centres in the (C&I) network.

2. Determination of the network topology i.e. the number of control centres to use and the nature of the communications system to interconnect them.

In this paper the design requirements of the network are formalised. These requirements are then divided into a set of objectives and a set of constraints. These objectives and constraints are expressed as a progressing series of mathematical models to which various optimisation techniques are applied.

NETWORK DESIGN REQUIREMENTS

In considering a distributed computer control network, the emphasis is on the distribution of control processes between a number of intelligent nodes rather than on physical location of these nodes. In such a control system each control process may need to communicate with other processes controlled by other nodes. If a large number of control tasks are resident in a single control node then the loss of that node can result in having to employ manual control over a large group of plant items. However, if jobs that share a significant quantity of common data are resident in different nodes than a loss of communication between nodes could restrict the action of control jobs in both nodes. The purpose of this section is to formalise a set of requirements on which the network
Design philosophy is to be based. This involves a basic understanding of the software with which the scheme is to be implemented and the knowledge of interactions between the controlled processes.

Processing requirements

The software framework that is required to support C&I tasks in either single or multi-computer configurations naturally subdivides into a number of different activities that are required in each control centre (e.g. communications, data handling, control algorithms etc.). Each activity can be implemented as a separate software module. Each module consists, principally of a sequence of calls to a set of standard defined subroutine blocks. As a result, the cpu load of each module is readily quantifiable (4). (See Appendix).

Communication requirements

The processing system at each node will comprise a set of common software modules to perform executive and communications functions and frequently encountered utility functions (5), plus a particular set of programs to perform the control functions at that node. Although this implies that the control centres are designed to act in a reasonably autonomous manner, there are nevertheless a number of situations in which it is necessary to transfer information between control centres (e.g. new set point values, data logging, alarms, overrides etc.).

The communication load of a task can be determined by assuming that the communication is in the form of messages, where a message is comprised of destination information, the data itself and some form of self checking code (5). Then, the simplest way to estimate the maximum communications load is to derive the number of messages that each task in each control centre can initiate on any one of its runs and then derive the total number of messages that can be initiated by a control centre in any one time interval.

Network design

The communication load imposed on the control centres is to some extent dependent on the topology of the communications network. However, before a sensible choice of network can be made the following factors require investigation (4):

1. The overall data flows involved in the complete control network.
2. An evaluation of the existing load on the control centre at each node.
3. General design objectives.
   a. A design procedure should be followed for large systems which will minimize the number of control centres hence minimizing capital and maintenance costs of the system.
   b. The overloading of any control centre or any communications link should be avoided.
   c. The design should avoid high data transfer rates between control centres.
   d. It is desirable to share the processing load evenly between control centres.
   e. To minimize the effect of control centre failure in terms of both plant safety and availability, and the manual control burden imposed on the operator.
   f. If a back-up control scheme exists for a plant area (or variable) then the control loops to implement back-up control should be in a separate control centre from the main control scheme.
   g. Duplication of large blocks of inputs to more than one control centre should be avoided.

The general design objectives listed above can be divided into those that act as optimality criteria for the problem and those that act as constraints on the network.

Operational constraints

a. No control task can be shared between control centres.
b. Each task must be assigned once and only once to any one control centre.
c. Each control centre must not be loaded above its specified capacity limit.
d. The data link between control centres should not exceed their link capacity limit.
e. The tasks which must be kept together should be in the same control centre.
f. The tasks which must not be kept together should be assigned to separate control centres.

Optimality criteria

a. Uniform loading of the control centres.
b. Minimize the number of control centres.
c. Minimize the data transfer between control centres.

MATHEMATICAL MODELLING

As part of the general problem formulation, consider that we have a set of tasks I = (1, 2, 3, ..., n) with known processing power requirements R = (p1, p2, ..., pn) and a set of computers J = (1, 2, 3, ..., m), where the jth computer has a capacity limit on the maximum power available Lj. With this much information available simple mathematical programming models can be constructed to pursue the above mentioned objectives.

MINIMIZING THE NUMBER OF CONTROL CENTRES

We define variables Xij and Yj such that

\[ X_{ij} = \begin{cases} 1 & \text{if the task } i \text{ is assigned to } j \text{ th computer} \\ 0 & \text{otherwise} \end{cases} \]

\[ Y_j = \begin{cases} 1 & \text{if } j \text{ th computer } j \text{ active} \\ 0 & \text{otherwise} \end{cases} \]

Then the objective function i.e. to minimize the number of active computers can be expressed as follows

\[ \text{Minimize} = \sum_{j=1}^{m} Y_j \]

The operational constraints c, a and b are then expressed by the equations (2) and (3).

\[ \sum_{i=1}^{n} X_{ij} = L_j \quad \text{for } j = 1, \ldots, m \quad \text{(2)} \]

\[ \sum_{j=1}^{m} X_{ij} = 1 \quad \text{for } i = 1, \ldots, n \quad \text{(3)} \]
The objective function is related to the constraints by the introduction of variable \( W_j \) such that no more than \( W \) tasks can be assigned to the \( j \) th computer.

\[
\sum_{i=1}^{n} X_{ij} = W_j Y_j \quad \text{for} \quad j = 1, \ldots, m \quad (4)
\]

In the standard format the model is

\[
\text{Minimize } \sum_j Y_j \quad (1)
\]

Subject to constraints (2), (3), (4) and

\[
X_{ij} = 0 \text{ or } 1 \quad \ldots \quad (5)
\]

\[
Y_j = 0 \text{ or } 1 \quad \ldots \quad (6)
\]

This is a simple model, the solution to which is obtained by methods of integer programming (6). In this case a Branch and Bound technique is used in preference to the cutting plane technique because of the dominance of zero-one variables.

Application of Branch and Bound algorithm

The algorithm used to solve some test problems is known as the BBMIP (8). The code of this algorithm is running on a CDC 7600 machine at the University of Manchester.

It was noticed that a rather large number of solutions were explored by the algorithm, despite finding the optimal solution quickly. This was due to the equal importance given to all computers which introduced degeneracy into the model. This degeneracy is reduced by replacing the unity weighting factor in the objective function by an 'ordered' weighting factor, e.g.

\[
\text{Minimize } \sum_j (2)^{j-1} Y_j \quad \text{or} \quad \sum_j (j) Y_j
\]

Also if an absolute minimum bound on the number \( N \) of computers is specified then the algorithm converges much more rapidly i.e.

\[
\frac{1}{J} \sum_j Y_j = N \quad \ldots \quad (7)
\]

This algorithm performed satisfactorily on test problems of rather small size. The largest problem that was solved on the CDC machine using this algorithm was for \( n = 20 \) and \( m = 5 \).

MINIMIZATION OF DATA TRANSFER

The communication between the tasks can be thought of as a package of data that a task sends to another task for processing, updating or storing. We assume a computer has the ability to transmit, receive and pass on a message. Then, our aim is to distribute these tasks to a limited number of computers so that the minimum of data is transferred between computers. We will assume that if two tasks are co-resident in one computer then the communication load between those tasks is zero.

The communication can be represented by an \( n \times n \) matrix \( C \), where the element \( C_{uv} \) represent the number of messages that a task \( u \) sends to task \( v \). Since we are only interested in the total data transfer between two tasks if they reside in different computers, we only need to consider the upper triangular half of the matrix i.e. if we assign \( C_{uv} := (C_{uv} + C_{vu}) \) for all \( u < v \), for \( v = 1, \ldots, \) \( n \) and assign \( C_{vu} = 0 \) for \( v < u \) = 2, 3, \ldots, \( n \). Then the elements of the lower triangular half are all zero. Furthermore, in accordance with our assumption the elements on the main diagonal are all zero.

The problem is then expressed as follows: given an \( n \) dimensional square matrix \( c \), find an \( m \) by \( n \) solution matrix \( X = X_{ij} \) such that \( Z \) in (8) is minimum.

\[
Z = \sum_{\substack{u=1 \\& v=u+1 \\& j=1 \\& j+1}}^{n-1} \sum_{\substack{u=1 \\& v=u+1 \\& i=1 \\& j=1}}^{n-1} C_{uv} X_{ui} X_{vj} \quad (8)
\]

Subject to constraints (2), (3) and (5).

Here, the first two summation signs basically mean to sum over all the interactions between all the computers but not within the same computer. The variables \( X_{ij} \) have the same meaning as in the last model and the operational constraints (c), (a) and (b) are similarly defined.

In this formulation we have considered the minimization of data transfer between the computers. Let us consider the converse of this, if task \( Tu \) and task \( Tv \) reside in the same computer, then the data transfer is saved by a factor of \( (C_{uv} + C_{vu}) \). Hence, we can think in terms of maximizing our savings'.

Proposition:

Let

\[
\sum_{\substack{u=1 \\& v=u+1 \\& i=1 \\& j=1}}^{n-1} \sum_{\substack{u=1 \\& v=u+1 \\& i=1 \\& j=1}}^{n-1} C_{uv} X_{ui} X_{vj} + \text{constant}
\]

Then

\[
\sum_{\substack{u=1 \\& v=u+1 \\& i=1 \\& j=1}}^{n-1} \sum_{\substack{u=1 \\& v=u+1 \\& i=1 \\& j=1}}^{n-1} C_{uv} X_{ui} X_{vj} = g(z)
\]

But this is

\[
\sum_{\substack{u=1 \\& v=u+1 \\& i=1 \\& j=1}}^{n-1} \sum_{\substack{u=1 \\& v=u+1 \\& i=1 \\& j=1}}^{n-1} C_{uv} X_{ui} X_{vj} = g(z)
\]

Hence, the alternative formulation is
Maximize $Z = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} \sum_{i=1}^{m} C_{uv} X_{ui} X_{vi} \ldots (9)$ 

Subject to the constraints (2), (3) and (5).

We note that the number of combinations considered in (9) is much less than in (8), although the constraints are the same.

Methods of solution

These models are very similar to the generalised quadratic assignment problem (8,9). This parallelism naturally assumes that the methods of quadratic programming such as Wolfe’s and Beale’s (8) can still be applied. However, the underlying concept of definiteness and convexity behind these methods causes difficulty. These methods require the matrix C to be definite (9,10). The matrix C in our formulation is square, non-zero and has $C_{ii}=0$, for all i. Thus it is indefinite.

Because these methods cannot be applied directly we consider expressing our problem in an equivalent form so that these efficient methods can be applied. For non-convex, non-linear problems with discrete variables the usual approach is (9,10).

1. Use of a related convex objective function

2. Linearisation

Use of a related convex function. Lawler (8) has suggested that an indefinite matrix can be changed to a positive semidefinite matrix by adding a positive constant $k$ to the main diagonal in the $C$ matrix. However, this means that the objective function can increase by a factor of $n^2$, hence altering the true objective therefore the solution obtained may not be the global optimum. Similarly Carlson and Nahmuser (10) have proposed a heuristic approach to solve assignment problems. Their approach cannot be directly applied because our problem is more rigidly constrained than a purely assignment problem. Very limited computational experience is reported for methods of this type and it is not sufficient to base any general conclusions concerning the applicability of these methods.

Linearisation. Lawler (8) and Greenberg (9) have shown that quadratic assignment problems by making the change of variables

$$Y_{uv} = \sum_{j=1}^{m} X_{uj} X_{vj} \ldots \ldots \ldots (10)$$

Where $Y_{uv}$ is 1 if task $u$ and task $v$ are in the same computer. Then the equivalent problem for the latter model is: find $Y_{uv} = 0$ or 1 that maximizes $Z$, where $Z$ is

$$Z = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} C_{uv} Y_{uv} \ldots \ldots \ldots (11)$$

The next step is to express the constraints in this equivalent form. If we consider that the tasks $u, v$ and tasks $v, r$ are in the same processor then it necessarily follows that $u, r$ must be in the same computer and each task must be assigned. Hence, we can replace (3) with

$$Y_{uv} + Y_{ur} - Y_{vr} \leq 1$$

$$Y_{uv} - Y_{ur} + Y_{vr} \leq 1$$

Now, if another variable $q = 0$ or 1 is introduced such that, if $i$ th task is the lowest numbered task in a computer then $q_i = 1$ otherwise $q_i = 0$. This is ensured by the following two constraints:

$$\sum_{u=1}^{v-1} Y_{uv} + q_v \geq 1 \quad \text{for } v=1, \ldots, n \ldots (13)$$

$$\sum_{v=1}^{u} Y_{uv} + (v-1) q_v \leq (v-1) \quad \text{for } v=1, \ldots, n \ldots (14)$$

The capacity constraint (2) is then replaced by its equivalent form

$$\sum_{v=1}^{P} Y_{uv} + q_u \leq L \quad \text{for } u=1, \ldots, n \ldots (15)$$

(Note that all computers are assumed to be identical and the capacity limits are also the same).

Finally, we can impose an upper bound on the number of computers as follows

$$\sum_{u=1}^{n} q_u \leq \text{(some upper bound)} \ldots \ldots \ldots (16)$$

Hence, the equivalent for the problem is:

Maximize $Z = \sum_{u=1}^{n-1} \sum_{v=u+1}^{n} C_{uv} Y_{uv} \ldots \ldots \ldots (11)$

Subject to constraints (12), (13), (14), (15) and

$$Y_{uv} = 0 \text{ or } 1 \ldots \ldots \ldots \ldots (17)$$

$$q_u = 0 \text{ or } 1 \ldots \ldots \ldots \ldots (18)$$

When the optimal $Y_{uv}$ are found, the $X_{uj}$ are obtained by inspection. For example, if $Y_{uv} = 1$ then $X_{uj} = X_{vj} = 1$ for some arbitrary $j$.

The linearised equivalent form of the problem was solved using the Branch and Bound method described earlier. For $n<7$ the method was quite efficient and the convergence of the algorithm was within satisfactory limits. However, for $7<n<12$ the algorithm failed to converge and for $n>12$ the problem became too large to handle. This can be expected because the number of constraints generated in the equivalent form increases cubically i.e.

Number of constraints =

$$\frac{(n!)^2}{(n-3)!} \times 1/2 + n^2(n-2) + n$$

= $$(1/2) (n-1)^2 + n$$

CONCLUSION

It has been shown that mathematical programming techniques can be used to configure an optimum structure for the
design of complex and large distributed computer control networks. Realistic mathematical models of the design objectives and operational constraints are formulated. The available standard mathematical programming algorithms have been used to solve medium size problems. To solve larger problems it would be necessary to use special algorithms exploiting problem structure.

ACKNOWLEDGEMENT

We wish to acknowledge the support that was received for the preparation of this paper from P. Jervis and D. W. Wells, C.E.R.L., Leatherhead.

REFERENCES


APPENDIX

For example, let N(t, b) be the number of times task 't' calls block 'b' and let T(b) be the time to execute block 'b', then the time to execute task 't' will be given by:-

$$T(t) = \sum_{b} N(t, b) \times T(b)$$

Since each task will be regularly executed at a fixed time interval it is possible to estimate the processing load that a task will impose on a control centre. Let F(t) be the frequency at which task 't' is to be executed, then L(t), the processing load contributed by task 't' is given by:-

$$L(t) = \sum_{b} N(t, b) \times T(b) \times F(t)$$

Hence the loading of each task can be evaluated and consequently L(c) the processing load on control centre 'c' is given by:-

$$L(c) = \sum_{t} L(t)$$

Hence, the processing load of each task can be determined.