A SYSTEM FOR THE SIMULATION OF HARDWARE TO SOFTWARE ALLOCATION AND PERFORMANCE EVALUATION

JOHN WIELGOSZ

ABSTRACT

In this thesis we develope a representation of both hardware and software based on general directed graphs. In a hardware graph arcs represent processors and nodes represent memory stores. In a software graph the arcs are process descriptions and the nodes are groups of data. Program execution is modelled as the binding together of elements of these two graphs, the set of bound elements characterizing the program state at a given time. Binding is regarded as a resource allocation process, and the method of selecting one from the set of binding alternatives as the allocation strategy.

This modelling system was implemented as a program whose input consists of the two graph descriptions in sequential form. The program reconstructs the topology of the graphs within the computer memory using pointers, and proceeds to bind the two graphs until a terminal state is reached. During binding data is gathered using a set of performance measures. On completion statistics are calculated and a summary of the observations is produced. A log of the binding activity is also available.

The latter part of the thesis is concerned with the application of the modelling system to computer networks. The program was validated by modelling a simple store and forward network, and the results proved satisfactory at the ninety-five per cent confidence level. The system was then applied to a proposed linkage between computers in the United Kingdom and the Advanced Research Projects Agency computer network in the United States. The results of this application are described in the penultimate chapter. Finally conclusions drawn from the work are presented and possible extensions discussed.

ACKNOWLEDGMENTS

I wish to express my thanks to Professor Peter T. Kirstein for his guidance, patience, and friendship during the course of this research. My gratitude also goes to all the members of staff, both academic and secretarial, at the Institute of Computer Science who have been most generous with their time and help. I have been greatly aided by the service and facilities made available to me at the University of London Computing Centre, and by the cooperation of Control Data Corporation. Finally I would like to thank my wife for her unfailing encouragement and help while this work was carried out.

CONTENTS

		Page
Figures		6
Notation		10
Chapter I	Introduction	
1.1	Background	13
1.2	Computing power	15
1.3	Research aims and methods	17
1.4	Organization of subjects	19
1.5	Summary of results	20
Chapter II	Review	
2.1	Graph models	22
2.2	Models of resource allocation and utilization in computing systems	29
2.3	Computer networks	36
Chapter III	Theory	
3.1	Graphical representation of hardware and software	49
3.2	Recursive structure of SIGMA and PIgraphs	60
3.3	Execution of a process by a processor	67
3.4	Allocation of a processor to a process	76
3.5	The hardware allocation problem in team execution	85
3.6	Properties of nodes in SIGMA and PIgraphs	96
3.7	Data dependence and reentrance	109
Chapter IV	Implementation	
4.1	General criteria	129
4.2	Graph input	134

Page

		Page
4.3	The allocator	151
4.4	Ties and IFloops	176
4.5	Hardware measurement	187
4.6	Software measurement	201
Chapter V	Validation	
5.1	The choice of validation	208
5.2	Store and forward networks	211
5.3	The validation model	216
5.4	The statistical test	231
5.5	Validation results	235
Chapter VI	Application	
6.1	A UK link to the ARPA network	243
6.2	Analysis of the link	248
6.3	The link model	263
6.4	Results	280
Chapter VII	Conclusion	
7.1	Summary of research aims achieved	300
7.2	Suggestions for further research	304
Appendix I	Bibliography	307
Appendix II	SIMULA 67	327
Appendix III	SHAPE limitations	338
Appendix IV	SHAPE user information	345

FIGURES

		Page
2-1	The ARPA computer network as of May, 1972	40
3-1	Relationship of graph and subgraph	52
3-2	Contact between arcs s and P	57
3-3	Correspondence between software and hardware representations	58
3-4	Structure of a graph at more than one level	61
3-5	Planar representation of fine structure	62
3-6	List structure representation showing subgraphs	64
3-7	Recursive call of SIGMA by itself	65
3-8	Execution of an arc S by Processor P_0	69
3-9	Range of a processor P_i at s_0 on S	72
3-10	Division of S when P is allocated	77
3-11	Microprogramming under program control	79
3-12	Necessary correspondence between data for P and S	81
3-13	Measures characterizing function execution	82
3-14	Software graph showing cut zone	86
3-15	Memory matrix for a node k	90
3-16	Use of dummy arcs	92
3-17	Reallocation on more than one level	94
3-18	Repartition Matrix R of a node	98
3-19	Binding of an arc S and its terminal node N'	100
3-20	Branching arcs	103
3-21	Loop representation	104
3-22	R-matrix for process one of Dijkstra's interlock algorithm	108

		Page
4-1	Node and arc class definition	137
4-2	Node and arc linkage	138
4-3	Subgraph linkage	139
4-4	Node class definitions	142
4-5	Arc class definitions	143
4-6	Index usage for graph input	146
4-7	Outline of class allocator	153
4-8	Allocator parameters	156
4-9	Multiple INarcs	161
4-10	Transaction entry to REP matrix	164
4-11	Four state representation of a processor	178
4-12	DOloop examples	182
4-13	Summary of IFcode actions	186
4-14	Processor utilization and efficiency	189
4-15	Node statistics in arrays QD, QT, QS	202
4-16	Cut statistics	204
4-17	Arc statistics in array STARC	206
5-1	Validation network	217
5-2	Proportional traffic matrix	218
5-3	Message routing	219
5-4	Link traffic, delay and capacity	220
5-5	Message delay matrix Z	221
5-6	Directed semi-network	223
5-7	SIgraph and PIgraph topologies	224
5-8	Nodes 1 and 2 of model SIgraph	225
5-9	Nodes 3, 4, and 5 of the model SIgraph	225
5-10	Example arc data of the SIgraph	227
5-11	Example node and arc data of the PIgraph	228

Page

5-12 Values of t corresponding to given probabilities 233

5-13 Values for t-test and $\bar{x}_k$ 236

5-14 Mean message delay by message type for eight runs 237

5-15 Confidence limits for $\bar{x}_k \times 10^{-3}$ 238

5-16 Mean message delay 241

6-1 Tentative ARPA network, logical map, May 1973 244

6-2 UK-ARPANET linkage 249

6-3 Traffic on each channel of the subnet 252

6-4 Values of T_i and P_i for L = 1/30 254

6-5 Values of T_{600} for various combinations of C_1 and C_3 255

6-6 Graph of mean message length against mean packet length in subnet 257

6-7 Values of T_a and C for various T_b 259

6-8 Values of C_i corresponding to T_b shown in Fig. 6-7 260

6-9 Graph of T_a and T_b against C 261

6-10 Hardware graph of subnet 264

6-11 Typical data for hardware graph 265

6-12 Stream zero 267

6-13 Stream five: nodes 15, 25, 35 268

6-14 Stream five: nodes 45,55,95 269

6-15 Stream three 270

6-16 Stream seven 271

6-17 Initialization node one 272

6-18 Means of truncated negative exponential distributions 275

6-19 Means of truncated negative exponential distributions (continued) 276

6-20 Subnet response when C = 4.8 281

6-21 Subnet response when C = 9.6 282

6-22 Subnet response when C = 50 283

6-23 Non-cyclic generation of messages 286

6-24 Variation of response with packet length 288

6-25 Graphs of response against think period for C = 4.8, 9.6, 50 Kb 289

6-26 Variation of response with block length 290

6-27 Background traffic generation 292

6-28 Stream zero with background 293

6-29 Initialization node one including background 294

6-30 Subnet response for various combinations of C_1 and C_3 when P = 30 296

6-31 T_a for various combinations of C_1 and C_3 when P = 30 297

6-32 Variation of T_a with C_3 for various C_1 298

NOTATION

Σ, SIGMA	Software graph
Π, PI	Hardware graph
Φ	Null processor
σ	Software subgraph
S	Arc of software graph, process
P	Arc of hardware graph, processor
P_I	Identity processor
P_O	Ideal processor
s	Quasi-distance on arc S
δw	Quantity of computation
j(s)	Computation density
T_O	Time for P_O to execute S
w	Total computation of S
u	Scaling factor for processor power
T	Time for P to execute S
r(p,s)	Range of P at s on S
$r_O(s)$	Quantal range at s on S
$\eta_O(u)$	Time for P_O to execute $r_O(s)$
L	Loss
$\alpha_i(s)$	Redundancy of P_i at s
$n(\sigma)$	Number of software functions for subgraph
ϕ_i	Number of times i th function is executed
K	Allocation procedure
A	Graph analysis procedure
REP	Repartition matrix
REP[i,j], μ_{ij}^k	Element of repartition matrix of node K
SIgraph	Software graph

SIarc	Arc of software graph
SInode	Node of software graph
PIgraph	Hardware graph
PIarc	Arc of hardware graph
PInode	Node of hardware graph
INarc	Arc entering a node
OUTarc	Arc leaving a node
λ, LAMBDA	Size of initial arc data
β, BETA	Cut generation time
ACT	Activity matrix of a node
v	Portion of tie duration due to store characteristic
ut	Utilization
ef	Efficiency
f_i	Function i of SIarc
t_i	Time for processor to execute f_i
c_j	Cost of processor component j
t_{ij}	Time component j in use during j_i
ut_i	Utilization of processor during function i
ef_i	Efficiency of processor during function i

The notation above is that used in describing the SHAPE system. We have not included variable names from the SHAPE program, and these are defined when used in Chapter IV. Variables used to describe the systems modelled in validating and applying the SHAPE system only appear, and are defined, in Chapter V and VI respectively.

CHAPTER I

INTRODUCTION

1.1 Background.

One of the goals of computer designers and users in creating new equipment is increased computing power. Such a goal is not difficult to justify. If achieved it reduces the cost of current computing activity, or allows expansion at a lower price; a previously uneconomic solution to a problem may now seem more attractive; perhaps less frequently, a solution is made feasible on an acceptable time-scale. Intuitively computing power is not a difficult idea to grasp, but interpretations vary and are seldom precise.

Computing power is usually described in relative terms. For example, twice the work done per day implies twice the computing power; alternatively, the same work done in half the time. In practice these need not be the same thing. Such relative comparisons tend to beg the question of what we really mean by computing power or computing work. It is worth emphasizing that computing power (in its normal intuitive sense) is very dependent on the task to be performed. In this sense it is dynamic, and not a function of hardware alone.

Expansion of a computing facility by adding more equipment of the type already in use may be called lateral expansion. Replacement by differently designed, faster, or more appropriate equipment may be termed vertical expansion. It is said, and may generally be the case, that there is more computing power per unit cost in a large system than in a small one. Consequently, simply spending more for a larger system may well do as a first step to increased computing power per unit cost. Having reached some financial limit, a differently designed, or in some sense intrinsically more powerful machine, for the same

price is needed.

Usually a mixture of these approaches is adopted. Another possibility is to design and build a new machine of the required power, though this is beyond the scope of most users. It is, however, part of a manufacturer's motivation.
A counter-productive side-effect in increasingly powerful systems is the difficulty of using that power efficiently. Significant numbers of comparatively trivial tasks under-utilize hardware, and difficulty in organizing work flow leads to high system overhead and idle time. At best only partial solutions to these problems have been found.

1.2 Computing power.

Theoretical limits of computing power undoubtedly exist given the current state of physical science. Laws such as the uncertainty principle will limit switching speeds of stores, transmission speed of data, and packing density of information. If we consider the hypothetical situation arising when computer technology reaches these limits, then only one strategy for achieving increased computer power remains. This is the organization of laterally expanding systems to process work in a parallel fashion.

Such an approach makes the implicit assumption that a significant amount of computational work is amenable to parallel processing. Though we are very far from the absolute limits mentioned above, the situation has a practical analog in the problem of a real-time system which is already using the fastest appropriate computer available, and is still unable to meet the completion constraints for some task or set of subtasks. The only way to meet the constraints is to reorganize the task so that it is amenable to parallel processing and then execute it on a laterally expanded system.

A visible trend in recent computer design is functional dispersal. This is based on the view that if too many functions are combined in one module, then much of it is idle, much of the time. Consequently, greater efficiency is obtained by having the funcrions in separate modules of appropriate cost and computing power. These are used when needed by the task and free for other work the rest of the time. The gain in efficiency presupposes enough different tasks in the system to ensure that individual module utilization is high. Attempts to meet this need have

recently led to pipeline design in some large computers.

Elementary function dispersal is present in computing equipment at the time of writing. Separation and concurrency of computation, I/O, telecommunications control, display regeneration, and so on, is evident in most third generation machines. Such function dispersal places increasing emphasis on the net-like aspects of computer facilities. Net representation of a computing facility can be applied at any level of detail, from computer networks, where the complete computer is the quantal object, to a single processor, where each logic subassembly is considered separately.

Clearly a program organized for parallel processing can take advantage of function dispersal to minimize its total execution time, and to select the functional hardware best suited to its individual processes. This is potentially a means of improving performance over a sequential version of the same task. Indeed, a sequential program may be regarded as one member of the class of parallel programs which achieve the same result.

1.3 Research aims and methods.

We have mentioned above the problems of using powerful computer systems effectively. In this thesis we attempt to provide a framework for the solution of such problems. Some elements common to a wide range of computing processes are isolated and identified. From these a technique for modelling computational activity in complex computer systems is developed. It is hoped that the technique will prove useful both as an aid to problem definition, and as a practical tool in the solution of a problem once it has been defined.

We have tried to introduce measures for aspects of the computational process which will be relevant in most circumstances and useful in evaluating the performance of systems under investigation.

We view computational activity as a hardware to software allocation process. That is to say that a task is realized, or results produced, by the allocation of a task processor to a task description. The basic operation in this process is chosen to be the production of one dataset from another through the action of a processor. A complete task is then regarded as a number of such steps occurring sequentially or in parallel.

The modelling technique uses directed graphs to represent software description and a hardware configuration. Execution of the former by the latter can then be modelled as a dynamic connection, or binding, of the two graphs. The system has been implemented in a high-level language (SIMULA 67) whose syntax provides features which correspond closely to the needs of such a modelling system.

Our goal has been to provide a system which can be used to evaluate and compare various combinations of hardware and software which perform a given task, and so provide a means of optimizing task performance both in existing and proposed computer systems.

1.4 Organization of subjects.

The material which follows is organized into six chapters each dealing with one phase of the research that was carried out. In Chapter II there is a brief discussion of related work in the fields of modelling, allocation problems, computing and transmission networks, and performance measurement.

In Chapter III we develope the concepts and theoretical considerations on which the modelling technique is based. We then describe the technique itself and show how it can be applied to computational processes. Chapter IV gives an account of the implementation of the technique on a CDC 6600 computer using the SIMULA 67 programming language.

Chapter V contains the results of a validation of the system, using a small store and forward network as a test situation. In Chapter VI, we apply the system to a proposed network linkage between the U.K. and the ARPA (Advanced Research Projects Agency) computer network in the United States. We describe the way in which the modelling technique was used to investigate the performance of the linkage under various conditions and present the results obtained.

Chapter VII discusses the conclusions which can be drawn from the research undertaken, and makes suggestions for further study. We have added four appendices for reference purposes. These are some remarks on SIMULA 67 and the CDC 6600, information required to use the implementation of the modelling system, a description of the limitations of the implementation and a bibliography.

1.5 Summary of results.

We have designed a system, based on graphical representation, which is sufficiently general to model a large class of computational processes. This has required the identification of a set of basic functions which are necessary for such modelling, and a program incorporating them has been written. To create such a system we have had to isolate and define the operation of these functions in some depth, and as a result we believe the modelling system corresponds well with the underlying structure of computational activity.

As necessary adjuncts we have produced computer input procedures which convert a sequential graph description to a topological replica within the computer, as well as a set of performance measures by which different model executions may be compared.

The implementation has been validated using a model of a store and forward network, and the modelling technique was applied to a computer network link between Britain and the U.S. Results predicting the performance of the link under various conditions have been obtained, and hardware parameters for link operation estimated.

CHAPTER II

REVIEW

2.1 Graph models of computation.

In this section we give some of the history and bibliography of models of computational processes which use graph representations. In the following sections we deal with resource allocation in computer systems and the design of computer networks. The references quoted are to be found in Appendix I, which also contains a separate comprehensive bibliography of material related to computer networks.

A number of researchers have produced graph models of computation. The use of graph representations is widespread in the literature of the theory of computation, and has also extended to modelling or describing processes which involve existing hardware or software systems. The utility of such descriptions can be seen, for example, in the short paper by K. A. Bartlett, R. A. Scantlebury and P. T. Wilkinson which gives an algorithm for the detection of errors during data transmission [BART 69]. Here the finite automata state diagram is used in the solution of a highly practical problem in computer communications.

One of the earliest widely quoted models of computational activity is the one put forward by R. M. Karp and R. E. Miller in 1966 [KARP 66]. Their model is called a computation graph. This is a directed graph in which nodes denote operations and arcs denote storage elements where results are placed in first-in-first-out queues. Associated with each arc are four non-negative integers A_p, U_p, W_p and T_p where $T_p > W_p$. For an arc directed from node i to node j these parameters are interpreted as follows: A_p is the number of data words initially in the queues; U_p is the number of words added to the queue upon

completion of the operation associated with node k; and

T_p is a threshhold giving the minimum queue length of the arc before the operation of node j is initiated. Karp and Miller show that computations represented by these graphs are deterministic. They also give a test to determine whether a computation terminates, and study properties of the data queues associated with the arcs, deriving conditions for the queue lengths to remain bounded.

Another type of model, similar to those above but probably more oriented to hardware representation, is one in which a set of operations are connected to a memory as in Karp and Miller's model but the control is entirely local and is incorporated into the values stored in the memory. Each operation monitors the values in its domain locations and can apply whenever the values belong to a specified set. When an operation applies it replaces the values in its range locations as determined by the current domain values. Models of this type have been investigated by Luconi [LUCO 68] and Petri [PETR 62]. Luconi considers schemata in which only a subset of the memory cells need contain unique sequences of values. Such schemata are called output functional and are realized by allowing more than one determinate computation to nondeterminately "share" operations. Sufficient conditions for a schema to be determinate are given and synthesis procedures for output functional schemata are provided.

E. Van Horn [VANH 66] has proposed an abstract model called machines for coordinated multiprocessing or MCMs. An MCM consists of a set of cells, a count matrix, and a scheduler to control operations. Each cell may behave either as a memory

(value) cell or as a computing (clerk) cell. In the latter case, a table of transactions is associated with the cell where each transaction may read and write cells or modify the count matrix. On the basis of the values in the cells and in the control matrix the scheduler determines which cells are enabled, i.e. can perform one transaction. The scheduler selects a subset of the enabled cells and directs them to perform their transaction. Van Horn has demonstrated that the action of the scheduler insures that the behaviour of any MCM is asynchronously reproducible.

G. Estrin and R. Turn [ESTR 63B] and D. Martin [MART 66] have introduced a directed graph model for computer programs in which the vertices represent computational tasks and the arcs represent data dependency between nodes. In this model, the conditions for the initiation of the computation denoted by a vertex is expressed by writing a boolean expression in terms of boolean variables associated with the arcs incident into the node. A boolean variable associated with an arc is true when the data in that arc becomes available. A computation may be initiated when the boolean expression of the corresponding node, called the vertex input control, is true. There are three types of vertex input control: conjunctive, disjunctive and compound. Vertices with conjunctive input control may be initiated only when all input data are available. Vertices with disjunctive input control may be initiated only when precisely one set of input data (i.e. one arc) becomes available. The compound input control is a combination of the other two. Vertices also have output control which is used to specify the program flow from a

vertex to a subset of its immediate successors. A vertex with conjunctive output control simultaneously makes data available at all of the arcs incident out of the vertex. A vertex with disjunctive output control makes data available at precisely one of its output arcs. Thus vertices with disjunctive output control effectively perform data dependent decisions to control the program flow. The model can properly represent only cycle free graphs. It has been used primarily as a tool for the *a priori* assignment and sequencing of computation in parallel processor systems. This model, described below, has been developed in a sequence of research reports by Turn, Martin, J. L. Baer, D. P. Bovet, E. C. Russell, S. A. Volansky, and V. G. Cerf, working with Professor G. E. Estrin at the School of Engineering and Applied Science at U.C.L.A.

Cyclic to acyclic graph transformations are the subject of [MART 67B] by Martin and Estrin, and other properties of the model are derived in [ESTR 63A, MART 67A, 67C, 69] by the same authors. Baer [BAER 68] has investigated the assignment of computations to processors by various scheduling techniques. Bovet [BOVE 68, 70A, 70B] has analyzed the model to determine profiles for memory allocation and Russell [RUSS 69] has used the model as a basis for the limited detection of parallelism and developed a system for the automatic generation of graph model descriptions, including attribute sets, from FORTRAN programs. Baer and Bovet have presented a method to test the legality of the initiation/termination conditions described by the graph model [BAER 70].

Volansky [VOLA 70] has further extended use of the model with an investigation of the detection and implementation of

parallelism in a multi-processor environment. Cerf [CERF 72] has considered the flow of program control which can be represented in the model, and determined condition for the proper termination of programs so modelled.

The U.C.L.A. model has been further developed by J. Rodriguez [RODR 69] to study the determinacy of the execution of a program where the parallelism is shown. Further control is introduced on the arcs of the graph. These can be idle, enabled, disabled, and blocked, while nodes are classified by their computational functions (control, data modification, loop junction) and logic (AND, EOR, and OR).

Other work on graph models of computation is that of H. Eisner [EISN 62], in which he has generalized the PERT network technique to take into account alternatives in performing project phases. This was achieved by assigning probabilities to different arcs out of decision nodes.

D. R. Slutz [SLUT 68] has extended the work of Karp and Miller. His models are called Flow Graph Schemata, and contain two structures. The first, called a data flow graph, indicates the paths of data flow and includes both operations to perform data transformations and memory cells to store intermediate results. The second is called a control graph and represents a mechanism to effect sequencing of operation activations. Using these structures Slutz has investigated the problems of determinacy and equivalence.

Three papers of interest in the use of graphs for modelling systems of processes are those of D. L. Parnas [PARN 69A], and of S. Crespi-Reghizzi and R. Morpurgo [CRES 70], and of Pfaltz [PFAL 72].

Parnas deals in some depth with the simulation of simultaneous events and gives an algorithm for the derivation of an efficient sequential process equivalent to a given network of parallel processes, where the network has unconditional rules of immediate dependency, and no delayless loops. Crespi-Reghizzi and Morpurgo present a language for representing graphs. The language uses linked lists to provide facilities such as addition and deletion of nodes and arcs, traversal of graphs, union, intersection, and so on. Pfaltz describes graph structures which allow the introduction of extra subsequences of arcs at nodes and other similar substitutions.

The works referenced above are mostly attempts to model the behaviour of parallel computations. To insure determinate behaviour it is necessary to provide some mechanism that would disallow more than one operation to change the contents of a shared memory cell at one time. Such mechanisms are also present in current proposals for practical parallel and multi-programmed computer systems.

Dijkstra [DIJK 66] considers a method by which asynchronous sequential processes may communicate 'harmoniously'. The processes are provided access to common integer variables called semaphores. The semaphores can be manipulated by means of two synchronizing primitives, the 'P' and 'V' operations which decrement and increment, respectively, the value of a semaphore by one. The P operation can be executed only when the current value of a semaphore is greater than zero. Thus the facility is available for one process to block another from entering a 'critical section' such as data accessible to

both. A number of interesting examples using semaphores are given. Dijkstra [DIJK 68] has incorporated semaphores into the design of a multiprogramming system and A. Habermann [HABE 69] has provided a theoretical justification of the logical structure. Holt [HOLT 71] has discussed Habermann's work and shown that artificial deadlocks can occur when Habermann's methods are used, and that they do not necessarily eliminate cases of permanent blocking. Holt gives a solution for these situations. Hebalkar [HEBA 71] has extended Habermann's analysis with a graph model of process resource requirements and defined algorithmic tests relevant to resource allocation with the intention of precluding deadlocks.

The interested reader is referred to various other papers on aspects of graph models of computation: [BERN 66, ABLO 68, BRUN 71, CORN 70, IRAN 71, EARN 72, SHOS 69, LOWE 70, GILB 72, TESL 68, CONS 68, COHE 68, GONZ 69, DENN 68, KOTO 68].

2.2 Models of Resource Allocation and Utilization in Computing Systems.

Many of the models mentioned in the previous section have been used to investigate resource allocation strategies. In particular Bovet [BOVE 68] has examined memory allocation profiles using the U.C.L.A. model. P. J. Denning [DENN 68] has also used graph models when investigating multiprocessor assignment.

The literature of resource allocation and utilization in computational systems is extensive. Much of it uses queueing theory to provide mean values for quantities of interest such as service times, waiting times, throughput rates and idle times. However, there is also a wide range of non-stochastic analyses.

One of the earliest papers in this field is that of J. Heller [HELL 61] which deals with the scheduling of the tasks of a computational job among the processing units which can carry them out. Solutions are obtained for completion times of the tasks, and idle times of the processing units, and these are then extended to the concurrent execution of more than one job.

G. K. Manacher [MANA 67] has provided a more extensive treatment of problems similar to that investigated by Heller. In this paper the assignment of tasks to processors is controlled by a task list, which orders all tasks according to servicing priority. A free processor is assigned to the highest priority task available. Two types of constraint are used, start-times and completion times. Tasks with start-times may not commence before those times, and tasks with completion times must terminate before them. Algorithms are developed to give

schedules which guarantee the execution of tasks within their deadlines, and allow the inclusion of non time-critical tasks in these schedules.

T. C. Hu [HU 61] uses a graphical model to derive an algorithm for the optimum sequencing of the tasks of a job in two cases. The first case is to provide a schedule which satisfies a completion constraint on the whole job with a minimum of processors, and the second is to provide the schedule with the earliest completion time when the number of processors are fixed.

The models described above have been greatly extended by the work of R. R. Muntz and E. G. Coffman [MUNT 69A, 69B, 70]. The authors have used acyclic, directed graphs not unlike the U.C.L.A. description to model computational activity, and have allowed preemption in task scheduling. Two important results are derived in [MUNT 70]. The first is the equivalence of Preemptive Scheduling and General Scheduling. Preemptive Scheduling is a scheduling discipline where a processor, instead of working continuously on a task once assigned to it, can be interrupted and assigned to another task. General Scheduling is a discipline where a fraction of a processor can be assigned to a task, and this fraction varied. The equivalence of these two disciplines is used in the implementation of the modelling system put forward in this thesis.

The second result is the statement and proof of an algorithm for the optimal scheduling of free-structured computations.

Another paper concerned with scheduling in multiprocessor systems is that of J. L. Rosenfeld [ROSE 69]. In this paper, execution of a certain type of program by N identical processors is simulated, and it is shown that with proper programming the solution time approaches 1/N of the single processor solution time.

Further results in this area of research can be found in: [BOWD 69, RAMA 72, SCHW 61, REIT 68, AOKI 63, KATZ 66, GOSD 66, GRAH 66].

The work described above is concerned mostly with scheduling to meet timing constraints. Another body of work deals with scheduling resources in a statistical demand environment, where it is the average behaviour of the system which is of interest. Typically this research has often centred on the response of time-sharing systems, and makes use of queuing theory in many of the results. A well known study of this type, augmented by simulation is that of A. L. Scherr [SCHE 67].

Detailed research has also been undertaken on the behaviour of specific devices. For example, the behaviour of the IBM 2314 disc is the subject of a paper by Abate, Dubner and Weinburg [ABAT 68], and drum scheduling has been investigated by Fuller [FULL 72]. Frank [FRAN 69] has also performed a more general study of disc usage in time sharing systems.

Markovian models have been used to study computational systems and resource usage within them. An example is the paper by J. D. Foley [FOLE 67] on the University of Michigan executive system. The executive is considered to have nine states and transition probabilities between them are provided

from experimental observation of the Michigan system. Results are obtained for the fraction of time spent by the executive in any state, and the effect of changes to the system.

Simulation has been a widely used tool in examining computer behaviour. In particular it is often used to see how well theoretical models predict the behaviour of real systems, and so determine their validity. In most cases the models have been of unique systems, for example [NIEL 66], and consequently the results have not been easily applicable to other situations.

An example which suffers less than most from this disadvantage is B. Randall's paper [RAND 69] on storage fragmentation. Here external fragmentation is defined as the loss in storage utilization caused by the inability to make use of all available storage after it has been fragmented into a large number of separate blocks, and internal fragmentation is the loss of utilization caused by rounding up a request for storage rather than allocating only the exact number of words required. A number of simulation experiments are used to show that rounding up requests for storage, to reduce the number of different sizes of blocks co-existing in the storage, causes more loss of storage by increased internal fragmentation than is saved by decreased external fragmentation. A method of segment allocation and an accompanying technique for segment addressing which take advantage of this result are then derived.

Space does not permit us to list the numerous papers which describe specific simulations, but more general discussions can

be found in: [ZEIG 72, HUTC 65, WEBE 64, NIEL 67, PARN 69B].

Some important results which are applicable to models of computation have been derived by G. F. Newell and W. J. Gordon in the area of queueing theory [NEWE 67A, 67B]. In the first of these papers closed queueing systems are considered. These are characterized by having N customers and M stages each with r_i parallel exponential servers of the same mean service rate. Such closed systems are shown to be stochastically equivalent to open systems in which the number of customers cannot exceed N, and equilibrium equations for the joint probability distribution of customers are derived. In the second paper closed cyclic queueing systems with restricted queue lengths are shown to be equivalent to open systems in which the number of customers is a random variable. The differential-difference equations for the time-dependent stochastic structure of the system are derived, and solutions given for a number of special cases.

Queueing theory has been applied to time-sharing systems and related computing situations by L. Kleinrock in a number of papers: [KLEI 66, 67, 68, 70B, 71, 72]. In the first of these papers [KLEI 66] a group of processors is considered to act in sequence on subsets of data belonging to a problem. Such a chain of sequential processing machines (SPM) has been described in [AOKI 63]. Kleinrock shows that the system may be viewed as a cyclic queue, and gives results for the case of two sequential processing stages, where their intermediate buffer is of arbitrary size. Assuming exponentially distributed service times for timeslices of subset processing, the ratio of

expected time to process n subsets by the SPM system and a single processor is derived. An approximation is then derived for an SPM system with 2^p processors by applying the previous result to pairs of processors, each of which represents a pair of processors, p times.

In [KLEI 67] time-shared computer systems are treated as queueing systems, where the time sharing effect is obtained by giving each request a timeslice Q of processor time and then requeueing it. Results are given for the expected time a request spends in the system by applying queueing theory to the case for which Q --> 0. These are extended to include systems in which requests belong to priority groups which determine the size of their timeslice.

In [KLEI 68] time-shared systems with M consoles are analysed and results given for the behaviour of the normalized average response time. Consoles are again serviced in a time-slicing fashion and after completion of a request, delay for an exponentially distributed think time before requesting service again. A definition of system saturation is given, and the original system is considered as a special case of the class of systems in which the Nth class consists of N processors with capacity 1/N of the original processor and serving M/N consoles each.

Scheduling algorithms for time-shared systems are the subject of [KLEI 70B], and further results for response time are given in [KLEI 71]. In [KLEI 72] the application of queueing theory as Q --> 0 is again used to provide results for the class of algorithms where the scheduling discipline may change as a function of the accumulated service. In

particular solutions are given for the average response time as a function of the service required by a request.

Further results on aspects of time sharing are given in the following papers: [FIFE 66, LASS 69, LEWI 71, NAKA 71, NIEL 67, RAMA 72, RASC 70, SHEM 67, SMIT 66, STIM 69] which are only a selection of the large body of research in this field.

2.3 Computer Networks.

Perhaps the earliest attempt to interconnect a large number of computers was the SAGE (Semi-Automatic Ground Environment) air defence system[EVER 57, MART 69]. This system, developed by the military to collect, analyze and display radar data from sensors scattered over the continent, became operational in 1958 and has subsequently been improved. At about the same time the American Airlines SABRE Reservation System [PLUG 61, EVAN 67] was being developed on a commercial basis. Due to the success of this system, similar systems are now in use by other airlines, hotels, etc. The Ticketron real-time reservation system [DUBN 70] is one such example.

The need by the military for improved data communications led to the development of the AUTODIN (Automatic Digital Network) Communications System in 1963 [HAMS 68, MILL 68]. This system utilized both line switching and message switching facilities and its design was influenced heavily by network survivability and vulnerability considerations. In contrast to military requirements for ultra-reliability, many commercial and experimental networks have relied upon simple interconnections or dial-up telephone lines for communications. Examples of such systems are the Chrysler Message Switching system [ISSA 68], the Rio Grande Railroad Message Switching Transportation System [DAY 68], the Control Data Corporation Cybernet and Kronos Systems [GAIN 71], and the DATRAN (Data Transmission Company) common-carrier network [BINA 71, FISH 71, GAIN 71].

Several networks have been designed using a central store-and-forward message switch which reduces the network cost.

The network topology for this type of design takes the form of the classic Star network. Examples of such networks are the COINS (Community On-Line Intelligence Network System) and the Lawrence Radiation Laboratory OCTOPUS System.

The Lawrence Radiation Laboratory network was called OCTOPUS due to its star-like topology. The central computer is a PDP-6 which serves as a store-and-forward switch between the large processors such as CDC 6600, 7600, and STAR, as well as the IBM Stretch and 360/91 computers. The central switch also provides access to the huge photo-store mass memory by any of the other machines,and allows an evolutionary growth of the multi-computer complex since new computers can be connected to the system resources and can gradually be brought up to operational status.

The third star network is the IBM computer network, NETWORK/440, which has several unusual features [MCKA 71A]. The central node was initially to be a medium size 360/50 computer, but was later changed to be a partition in the large 360/91, which serves not only as a store-and-forward switch, but also as a master operating system. The network consists of several IBM 360 computers and a Control Data 6600 computer, the latter being connected via a small Honeywell DDP-516 preprocessor. The non-IBM machine introduces a degree of generality into the network due to the considerable difference in the CDC and IBM architecture and data structures.

In 1964 the Rand Corporation completed a comprehensive study, "On Distributed Communications" [BARA 64A, BOEH 64, SMIT 64], and a proposal for a distributed store-and-forward

message switched digital network. Although Rand's system was never implemented, their approach has influenced the design philosophy of some military networks and the ARPA Computer Network. During the study, Baran was responsible for the definition of a "packet" and for the "hot potato routing algorithm."

In 1966 Lichtenberger [LICH 66] proposed a network of identical computers; however, this network was only partially implemented. Also in 1966, an experiment was conducted by interconnecting the TX-2 computer at the Lincoln Laboratory and the Q-32 computer at System Development Corporation to test the basic philosophy of a network connection. This experiment showed that resource sharing was possible between two computer systems.

In 1967 the National Physical Laboratory (NPL) in England made a comprehensive proposal [DAVI 67] for a general purpose store-and-forward network. The NPL network was to be a store-and-forward network using interface computers and 1.5 Mb/sec. transmission lines for the message switching net, with an expected network response time (the time from the receipt of a packet to the beginning of the output at the destination) of less than 100msec. Packets were defined as any multiple of 128 bit segments up to a maximum of 1024 bits. Details of the proposed network operation appeared a year later [BART 68, DAVI 68, SCAN 68, WILK 68]. To date, only one node has been implemented and can be described as a multiaccess computer system controlled by a time-sharing computer [BARB 69, SCAN 69, WILK 69]. The authors have so far concentrated on the local rather than trunk level.

A small experimental computer network is being developed at Carnegie Mellon University, consisting of two DEC PDP-10 computers, a pair of PDP-8 minicomputers, and a hybrid computer. All five computers are located together and since the communications costs are insignificant, experiments with completely connected nets as well as with more typical network interconnection topologies have been planned.

In 1968 the Advanced Research Projects Agency released a Request for Quotation to construct a store-and-forward computer The contract was awarded to Bolt, Beranek and Newman, Inc. located in Cambridge, Massachusetts. The basic ARPA Network community consists of about 26 ARPA-sponsored research sites. Some of these sites have areas of specialization such as the graphics work at the University of Utah, picture processing at the University of Southern California, the man-machine interactive work at System Development Corporation, the text editing and information retrieval work at Stanford Research Institute and the network measurement and modelling work at UCLA. Other sites have specialized hardware capability such as the ILLIAC IV computer and the trillion bit laser memory.

Figure 2-1 shows the configuration of the ARPA Computer Network. The various sites (HOSTS) are interconnected via a distributed message switching communication net consisting of IMPs (Interface Message Processors) and dedicated 50 kbit/sec. full duplex communication lines. Each site typically consists of one or more computers, called HOSTs, operating in a time-shared environment, but

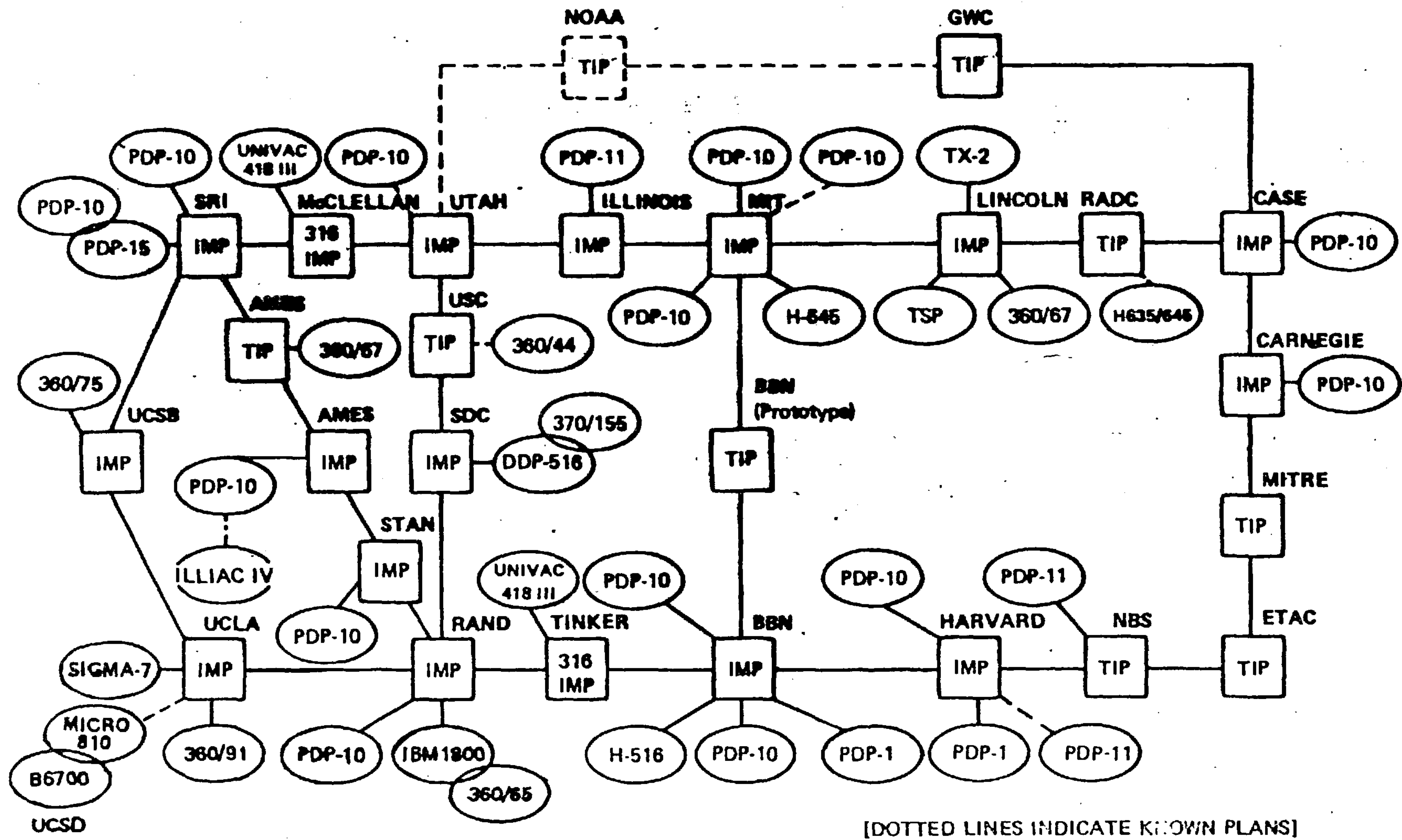


Figure 2-1 The ARPA Computer Network as of May 1972.

range in size from a TIP (a terminal IMP) to the ILLIAC IV. Two series of papers presented at Spring Joint Computer Conferences [CARR 70, CROC 72, FRAN 70, FRAN 72, HEAR 70, KLEI 70A, ORNS 72, ROBE 70, ROBE 72, THOM 72] discuss the design, performance and operational aspects of the network.

The problems of message delay, nodal storage requirements and the network routing strategy are some of the more interesting aspects of such a network from a modelling, analysis and operational viewpoint. Once the node locations are given, the network topology is influenced by the required system reliability, by projected user traffic requirements, nodal processing speeds, and total dollar cost allowed for the construction of the network. Then a protocol for passing messages between the nodes must be chosen and the nodal processing programs designed.

Another problem is the specification of an operating system communication protocol which allows for the establishment of a connection between HOST computers [CARR 70]. This task is handled by the NCP (Network Control Process) which is generally a part of a HOST's executive program. Progresses located within a HOST communicate with the network through the NCP.

Finally, a higher level of protocol is needed when a high degree of interaction is required between a user and a particular subsystem in a foreign HOST. This requires the development of interprocess communication techniques; for example, file transfer techniques, communication between dissimilar graphics stations, remote job entry, and interactive terminals.

In the assessment of performance of a general store-and-forward computer-communication network, it is necessary to examine the assignment of channel capacities, the effect of queue discipline, choice of the message routing procedures, nodal processing delays, nodal storage requirements, and the design of the network topology. A network performance measure is required to determine how various choices of the above parameters affect performance.

There are basically two classes of performance measures. The first class does not relate in any simple way to individual messages in the network, but rather to the performance of particular components that compose the network. Examples of such performance measures are: average channel utilization, nodal storage utilization, and channel error rates. Many of these performance measures can be computed analytically. The second class of performance measures relates more directly to individual messages. An example of such a performance measure is the average message delay. This provides a measure of system response which may be directly observed and which can be estimated. L. Kleinrock has investigated the minimization of this measure under various constraints in [KLEI 64]. Amongst other results the use of an "independence assumption" was shown to allow analytic solution for the optimal channel capacities in store-and-forward communication nets. A further description of this work is given in Chapter V.

One of the problems of current design is the application of general theories to the analysis and design of store-and-forward computer-communication networks.

Four main problems in these networks are construction of models to predict message delay, message routing strategies, channel capacity assignments, and topological design of networks.

All these are dealt with to greater or lesser degree in [KLEI 64]. This work is further developed in [KLEI 69A and 69B] in which exact and approximate analysis, simulation, and measurement are compared to obtain results for networks of the ARPA type. The discussion is carried further in [KLEI 70A] and [FULT 71, 72].

Routing procedures have been investigated from various approaches. Prosser [PROS 62A], Kleinrock [KLEI 64], Shapiro [SHAP 66] and Benes [BENE 66] have examined the effect of random routing procedures on message delay. Their conclusions were that random routing techniques are highly inefficient in terms of message delay, but are relatively unaffected by small perturbations in traffic intensity or network structure. Boehm and Baran [BOEH 64] and Smith [SMIT 64], Boehm and Mobley [BOEH 66], Kahn and Teitelman [TEIT 69] and Kleinrock [KLEI 70A] have examined some stochastic computing techniques. Deterministic routing procedures have been investigated by Prosser [PROS 62B], Boehm and Mobley [BOEH 66], and Kleinrock [KLEI 69A]. Their approaches have been slightly different. Prosser gave an approximate analysis of directory procedures which showed an increase in efficiency and amount of data transfer as compared to random routing, but at the expense of maintaining the directory. Kleinrock has computed average message delay as a function of traffic intensity for a fixed network topology and fixed routing procedures. Boehm and Mobley considered the problem of

computing a fixed routing procedure from estimates of network delay.

The topological design of ARPA-like computer-communication networks has been attacked by the Network Analysis Corporation [FRAN 70, NAC 70A-B, NAC 71A-B]. Their procedure is derived from a natural gas pipeline study [FRAN 69]. In their procedure, both the network topology and channel capacity assignments are varied during the optimization, while the routing procedure is essentially held fixed (it is deterministic for a given network topology). Since this problem defies a precise solution, their results must be viewed as giving good, but not necessarily optimal, network realizations.

Implicit in the optimal design of a network is the network performance function. For most network design problems average message delay has been selected because it is mathematically tractable, because it represents the global performance of such networks, and because it can be measured. Meister, Mueller and Rudin [MEIS 72] considered a slightly different performance measure: a weighted sum of powers of the average message delay in each channel. From this performance measure, they are able to obtain a channel capacity assignment for fixed routing which reduces the variation in delay from channel to channel at the expense of only a moderate increase in average message delay. This technique reduces the delay markedly on lightly utilized channels where, as the authors state, the user would be very much aware of this decrease when using the network.

Measurement of the behaviour of the ARPA network is the subject of [COLE 71]. A measurement collection system is

described and implemented, and data accumulated by observing normal and artificially generated traffic is analysed.
J. F. Zeigler [ZEIG 71] has investigated nodal blocking in ARPA-like networks with the aid of a two-state Markov process model. Results for the fraction of blocked nodes in a network are given, and developed for "clumps" of adjacent blocked nodes.

A further effort in computer networks based on the ALOHA system [ABRA 70] is the current examination of satellite communications as a means of extending the ARPA network. Their use, particularly in broadcast mode, is the subject of a series of ARPANET Satellite System Notes. In Note 12 [ASSN 72] L. Kleinrock and S. S. Llam derive expressions for channel efficiency and expected number of retransmissions. In the system analysed simultaneous, or overlapping, broadcasting is regarded as failure of transmission for both messages, which are retransmitted after a stochastic delay.

A study is currently taking place of methods of providing a computer network for a number of Canadian Universities. A first stage in the study is described in [DEME 72A and 72B] by J. DeMercado. These reports deal with the synthesis of minimum cost networks in which either simultaneous or time-shared transmission occurs.

Some interesting papers on computer networks are to be found in the proceedings of the ACM/IEEE Second Symposium on Problems in the Optimization of Data Communication Systems, October, 1971. The ARPA network is the subject of two papers. The first is by G. D. Cole, which is materially similar to [COLE 71], and the second is by R. E. Kahn and W. R. Crowther

on flow control [KAHN 71A]. In this paper the authors describe the various types of storage deadlock which can occur in the ARPA network and present the precautions which were taken against such occurences.

There are also two papers on the NPL network in the proceedings. The first also deals with congestion and proposes an "isorithmic" solution [OAVI 71]. That is to say that there should be a fixed number of packets in the network at all times, whether or not they carry data. The second paper describes various levels of protocols to be used in the NPL network for computer-to-computer communication, [SCAN 71].

A description of Tymshare Inc.'s TYMNET system and its history is given in [BEER 71], while reliability in centralized networks is the subject of [HANS 71]. Two papers in the proceedings deal with distinct loop-type networks. In [HAYE 71] results are given for mean message delay and other characteristics, and confirmed by simulation. In [SPRA 71] loops consisting of a central processor and a number of terminals are analysed and parameters obtained for the variation in terminal message delay with terminal loop position. Error control is the subject of [TRAF 71], which deals with computer-to-computer links involving transmission via satellite.

Current developments in the design and operation of computer networks are described in a number of papers presented at the First International Conference on Computer Communications, 1972. In [ANSL 72] methods of data transmission used by the British Overseas Airways Corporation are surveyed, and in [BARB 72] an outline is given of a project for a European

network initially linking research establishments in France, Italy, Switzerland and the United Kingdom. Methods of operation and maintenance in the ARPA network are the subject of [MCKE 72]. In this paper the detection and diagnosis of network faults by the HOST computer at the Network Control Centre are described. The Centre has the function of receiving IMP situation reports, determining the actual state of the network, and initiating repair activity when appropriate.

In [WHIT 72] V. Kevin Moore Whitney has compared various algorithms which have been used to obtain (heuristically) least cost network topologies. The same networks are submitted for solution by each algorithm and resultant topologies compared. The comparisons are shown to be remarkably consistent, and demonstrate some advantages of the Steepest Ascent Hill Climbing (SAHC) algorithm.

The operation of a network under conditions of saturation is discussed in [DESP 72], and network characteristics for such operations are presented. The performance of satellites for network data transmission is described in [HUST 72] and figures for both performance objectives and measured performance are given. Data management in networks is the subject of [FARB 72 and BOOT 72] in which the problems of safeguarding, accessing and updating dispersed data by equally dispersed users are discussed. Finally a survey of European network development is given in [KIRS 72] which describes current ventures by universities, research establishments, post offices, together with those of some industrial and commercial concerns. In view of the extensive material available, we have added a section on computer network design to the bibliography in Appendix I.

CHAPTER III

THEORY

3.1 Graphical representation of hardware and software.

For the sake of descriptive convenience in the material below we define the terms team and net as follows: a team is defined as a set of interdependent cooperating programs executing concurrently in real time to perform some well-defined function. A net is any collection of hardware modules, i.e. processors, memories, peripherals, I/O controllers, message switchers, connected by data channels. A net can of course be one computer or many, and generally exhibits the properties of hardware-sharing, function dispersal, and concnrrency of operation.

A team can be represented as a directed graph, Σ, whose arcs represent the execution of individual sub-programs, and whose nodes represent events where the subprograms interact. Such interaction may be simultaneous completion or initiation of subprograms, or communication of information between two or more subprograms. Processing within an arc is considered logically independent of that within other arcs. That is to say that all interaction between subprograms which is implicit in the intrinsic logic of the overall task occurs only at the nodes. This does not imply that the arcs themselves are purely sequential programs; further, there may be interaction between them because of hardware allocation constraints in the net.

The word processor will be used in the following to denote any hardware module which performs a transformation and/or movement of data. This includes devices such as I/O controllers, multiplexors, regenerators, and so on. In this sense a processor need not possess the full set of functions of a general purpose

computer. Consequently not all processors will be able to execute all programs. A processor P can be regarded as a hardware operator on data. Each arc of a team Σ is a subprogram S executed by some hardware module of the net on which Σ executes. We define <u>regular</u> execution of a team to be execution where hardware allocation only changes at the nodes of Σ. Running programs to completion is regular execution, hardware sharing is not. Transmission of data, without any transformation, may be regarded as processing by an identity processor P_I. Storage of data for a period of time can be regarded as processing by the null processor Φ.

We now consider an aspect of modelling which might be termed focusing. In constructing any model, a decision must be made as to what level of detail the model will reach. The situation is analogous to choosing the degree of magnification appropriate when using a microscope. Too small a magnification may not show the process of interest, too large a magnification may make it impossible to view the entire process or obscure it with irrelevant detail. For convenience the level of detail a model reaches will be called its <u>depth</u>. When the depth of a model is chosen, this is in effect a decision to treat all objects below that level as black box or quantal ones (if not, then there would be a further level of detail below the chosen depth, which is a contradiction in terms). However this choice is imposed by the model builder; objects at the model depth are of course structured in reality. Consequently the choice of model depth is in effect a decision to ignore (or a cut-off point for) the appropriate fine structure.

In terms of the foregoing, we suggest that graphical representation of a team can be used for modelling computer activity at any depth from the execution of a single machine instruction (which can be regarded as a team of microprograms) to considering entire computers as quantal objects. If we have a graph Σ representing some task performed by a team, we are implicitly deciding to treat the members of the team (arcs of Σ) as black box processes, since we stipulate that logical interaction between the members occurs only at nodes of Σ. That is to say, we are interested in the change in system state caused by the execution of an arc, but not concerned with the interactions occurring within the execution of an arc.

We can of course include this level of interaction if desired, by replacing each arc S of Σ by a subgraph σ of processes, at the next (convenient) level of detail down, which perform the function previously represented by the single arc S. We use the word subgraph here to mean a graph representing the structure of a single arc of another graph (at a higher level) rather than in the normal graph theoretic meaning of a subset of graph elements.

It may be that certain arcs of Σ are of critical interest. In this case a more detailed picture may be obtained by replacing only the arcs concerned by subgraphs, while leaving the rest of Σ as before. Thus the graphical representation is recursive in the sense that any arc may be replaced by a subgraph. If the graph Σ and its attributes are considered as a named data structure, then the name of an arc of Σ may be an element, or the name of a further data structure, i.e. a subgraph. If we envisage a procedure A performing analysis,

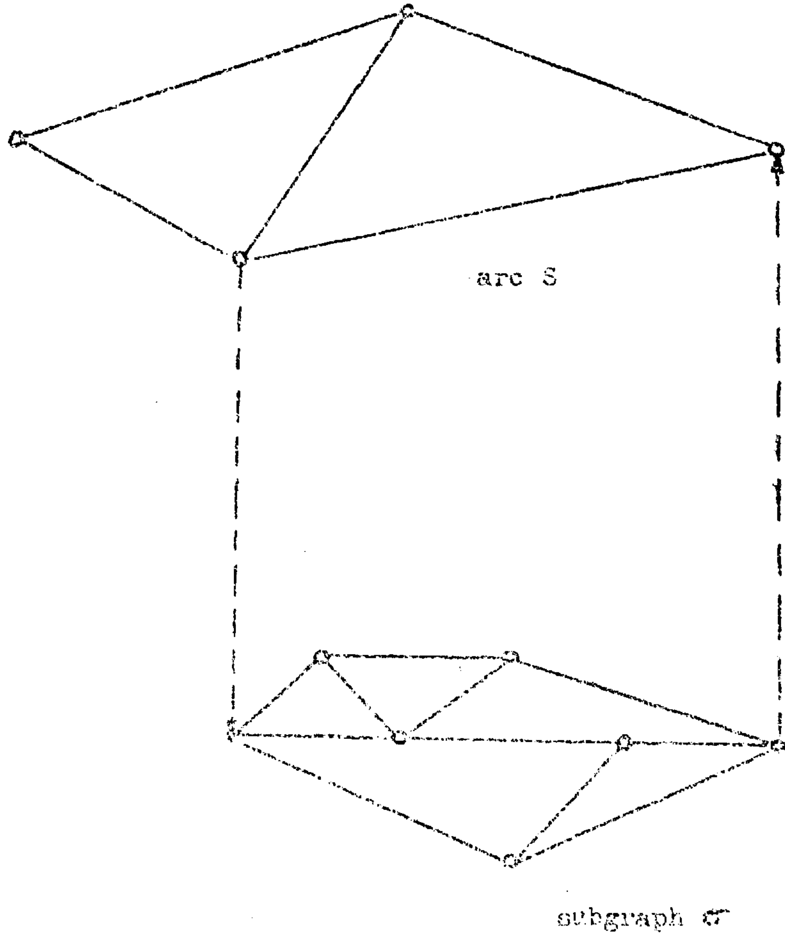


Fig. 3-1 Relationship of graph and subgraph.

or other processing, of Σ , then an individual arc of Σ may undergo the same analysis by recursive call to A, if the structure of the appropriate subgraph is available.

The progress of a team towards completion can be regarded as the execution of arcs of Σ by processors allotted to them by a control algorithm. The time taken to execute an arc will depend on the computing power of the allotted hardware relative to the subtask represented by the arc, and also on whether the execution is regular. Changing the control algorithm is the equivalent of varying the allocation strategy of the model, but not its structure. Finally, by representation as a team, a control algorithm is itself amenable to the same modelling.

Normal Critical Path concepts apply here, in determining overall execution time for a team, and in detection of a critical path. A cost function can be associated with the net resources, such as processors and memories, allotted to arcs. Slack time represents the time between an arc S completing execution, and the interaction of its results with the rest of the team. This is effectively storage of such results until all arcs which interact with S at its terminal node have completed. Thus slack time has an associated cost for information storage. Overall Completion time for Σ can be decreased by allocation of more powerful processors on critical arcs. This clearly raises the cost unless overhead and idle time created by such action are nonexistent.

We now develope the idea of hardware/software correspondence. This is based on the following premise: any function that can be done by software can also be done by hardware, and vice versa.

We add the proviso that obviously there must always be some quantal level of hardware present, else the function would never be physically executed. This is equivalent to saying that all computing activity is performed by a combination of hardware and software, and that the partition of the task between them is an arbitrary one; further, that this partitioning can be made at any level or part of the function, by building appropriate hardware. An extreme case is the performance of some task entirely by special purpose hardware, which is equivalent to reducing the software element to a single instruction. We suggest that the distinction between hardware and software is an artificial and fluid one. Consequently, in developing a model of computing activity we are concerned that it should take into account various possible hardware/software decompositions of the activity.

We now propose a graphical representation of a net, and consider under what circumstances it may be regarded as the dual of the team representation outlined above. A graph π will be considered a model of a net in the following way. Each node of π will correspond to a storage element of the net. Each arc will represent a possible data flow through a processor P between such storage elements. We make the remark that a processor P may be able to connect itself across more than one pair of nodes. Thus there will be an arc in π for every possible connection that P can make between a pair of memory elements, but at any instant there will be a flow on only one of these arcs. If P is P_I, the identity processor, then no transformation on the data flow will occur.

In the graph Π traversal of an arc P may be regarded as the execution of some program by the processor P, taking input data and status from the initial node (memory element) and producing output data and status at the terminal node. Regular execution on an arc P of the graph Π occurs if the program being executed by P remains attached to the arc for the period of time necessary for it to run to completion. For example paging is not regular execution. The previous remarks on model depth and the recursive properties of graphical representation apply equally to the graph Π, except that in this case a subgraph p represents, not subprograms, but sub-processors; the subgraph p must have the functional capability previously represented by the arc P.

We can regard the graph Π as operating in some environment from which programs are selected, attached to arcs at the initial node, and detached later at the terminal node, then to return to the environment which acts as a source and sink. A team operates in an analogous fashion except that in the case of a team the environment is a source and sink of processors. We see that in the case of a team the environment provides net elements, and in the case of a net it provides team elements. The process of attachment and detachment may be regarded as a control algorithm whose properties are symmetric between these two activities. In both cases arc traversal represents the execution of some stage of an overall task. We now define a particular graph Π in relation to a team represented by a graph Σ. In the graph Π there is an arc P for each member (arc S) of the team, which represents the processor drawn from

the environment of Σ to execute that member.

The arc P has as its initial node a memory element containing all data and status information needed by the member S to commence execution. The terminal node of the arc P is a memory element which will contain all output and status information produced by the team member S, after it has completed execution. Under these conditions it is quite clear that the graphs Π and Σ are isomorphic. The graph Π which exactly corresponds to the hardware needs of the team Σ is its hardware dual. A team Σ which exactly uses the net Π is the software dual of that net. The isomorphic graphs Π and Σ may be considered as a mapping of a computing function between two spaces which could be called <u>hardware space</u> and <u>software space</u>.

At any given instant the state of the computing activity represented by Π and Σ can be characterized as follows. Any arc in either graph which has a member of its environment attached to it is termed <u>active</u>. The point on the active arc S of Σ which has been reached by the processor P in its traversal, at the instant under consideration, is called the <u>contact point</u> of the arc P on S. The location of the contact point is an indication of how much of the process represented by arcs P and S has been completed. The point on the active arc P of Π reached by S is called the contact point of S on P. Its location represents the amount of the processor's allocated resources which have been used by the program S.

At any moment the only interaction taking place between hardware and software is at the contact points of the graphs Σ and Π .

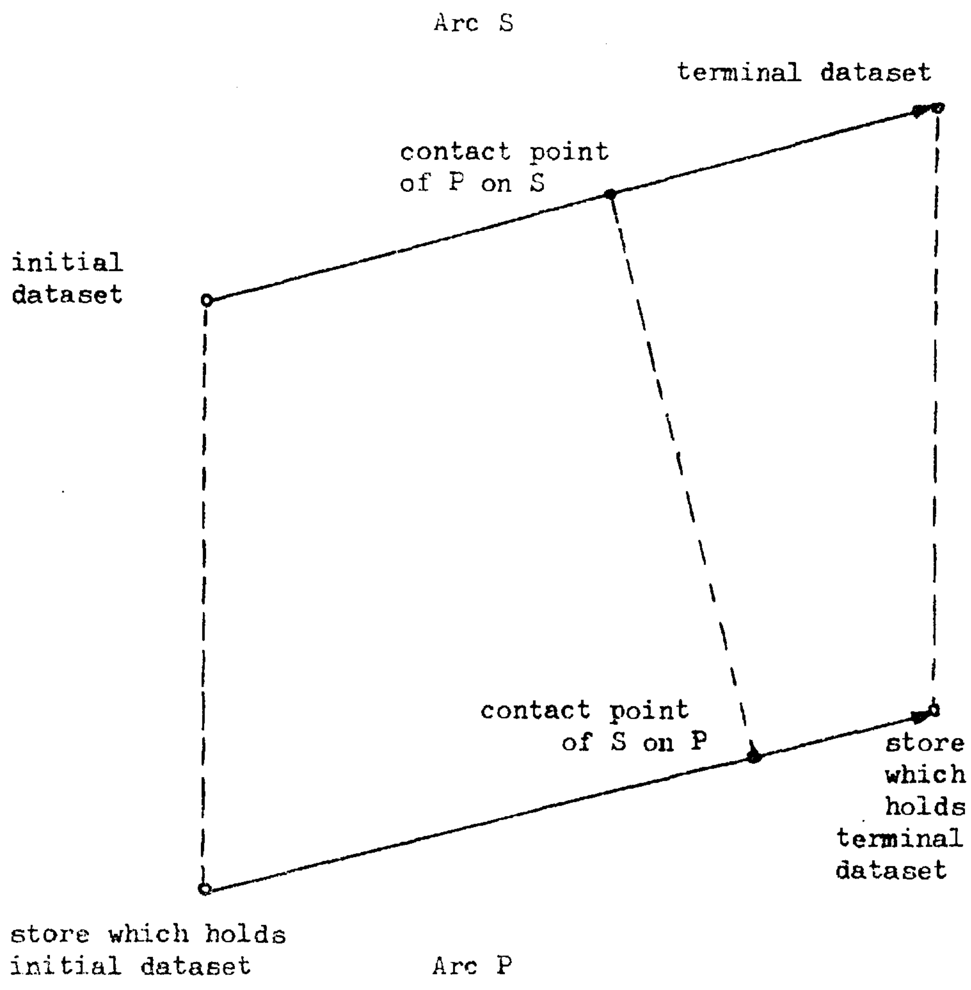
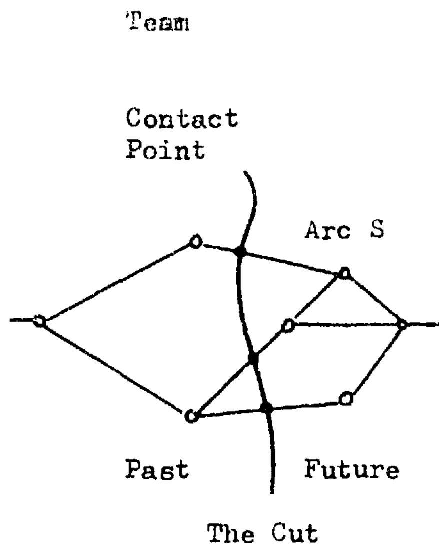
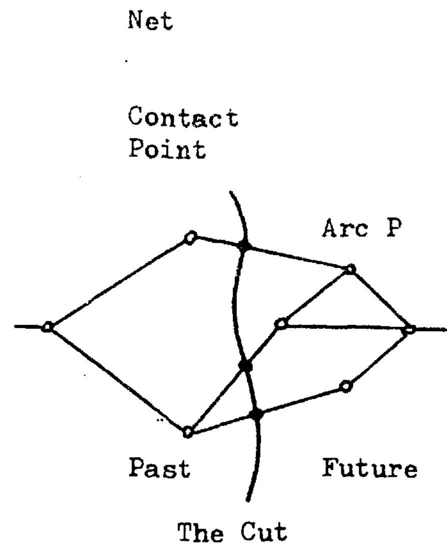


Fig. 3-2 Contact between arcs S and P.



Environment of Processors

Arc S represents a program of a team. Nodes represent events where programs begin, end, and exchange information. Processors from the environment are attached to and detached from programs at nodes.



Environment of Processes

Arc P represents a processor of a net. Nodes represent memory elements which receive, provide, and transfer information. Programs from the environment are attached to and detached from processors at nodes.

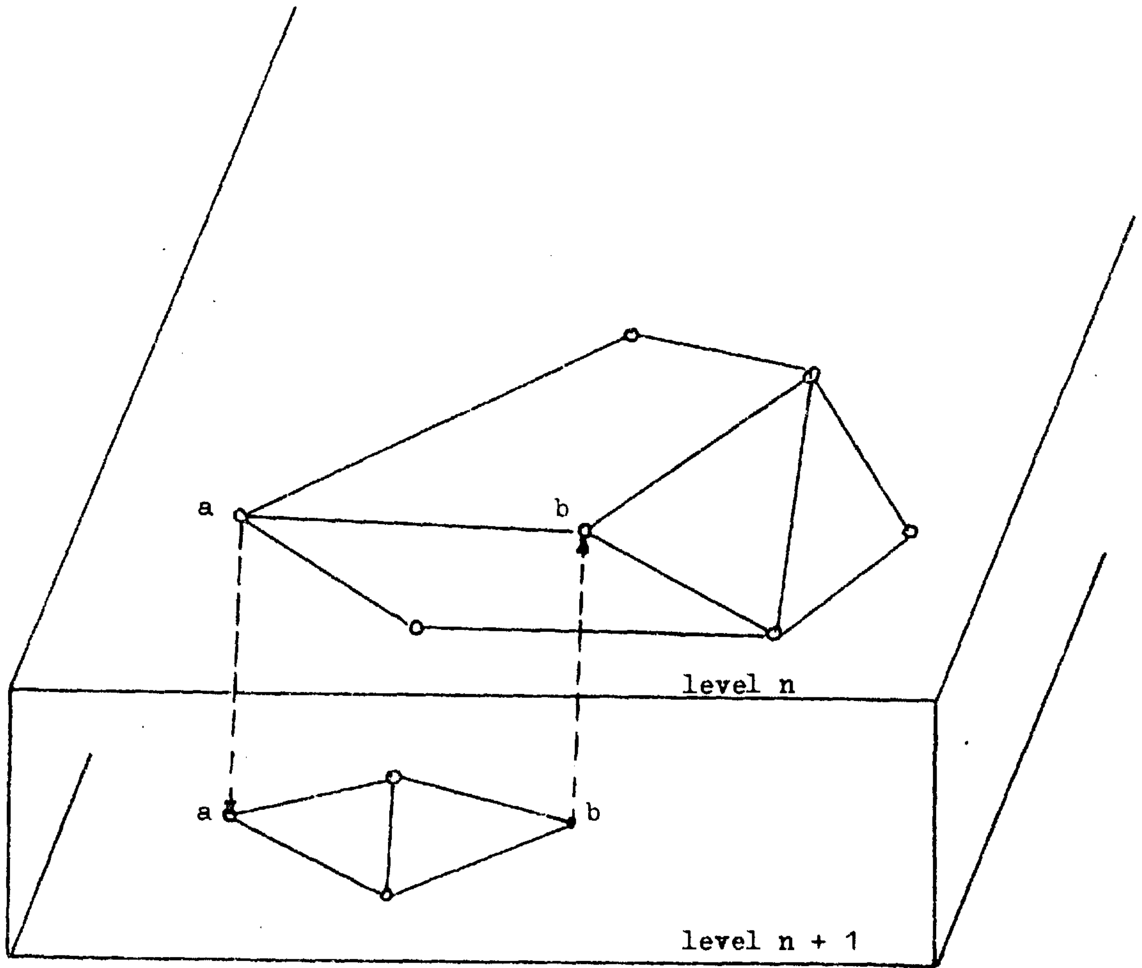
Fig. 3-3 Correspondence between software and hardware representations.

The remainders of the graphs are not in contact. The state of the computing activity is determined by the set of contact points. The performance of some task can be regarded as a traversal of both graphs by a set of contact points. This set can be thought of as a cut on the graphs, with past activity on one side and future activity on the other. Clearly not all of Π and Σ need to coexist at any moment, since all that is necessary for completion of the computing process is the existence of the parts of the graphs immediately required by the cut.

3.2 Recursive structure of SIGMA and PI graphs.

This section provides a recursive description of the Π and Σ graphs described above. Two types of recursion arise in connection with increasing the information stored in such graphs. The first is that the addition of nodes and arcs to an existing graph, where the nodes and arcs are of the same type as those already there, produces a new graph. The second is the replacement of an arc by a subgraph. The nodes and arcs of the subgraph need not have the same properties as those of the parent graph. This fact is indicated in the representation by entering the subgraph with a special type of arc name a down-arc, and leaving by an arc named an up-arc. The subgraph has no other topological connection with its parent, and is said to be one level of detail deeper, (down level and up level will be used interchangeably for down-arc and up-arc). The nodes at each end of a down or up-arc can be regarded as different views of the same event or information. In fact up-arc and down-arc are analogous to the block delimiters begin and end in ALGOL.

The highest level of the graph is level one. This level is regarded as being entered by a down-arc from level zero, which is the universe in which the system being represented is embedded. This may be shown as a single node at level zero. An example of how a graph might appear viewed at levels zero through three is shown in Fig. 3-4. In fact the levels can be regarded as horizontal planes containing graphs with a down or up-arc being a vertical line connecting superposed nodes in adjacent planes. This is shown in Fig. 3-5.



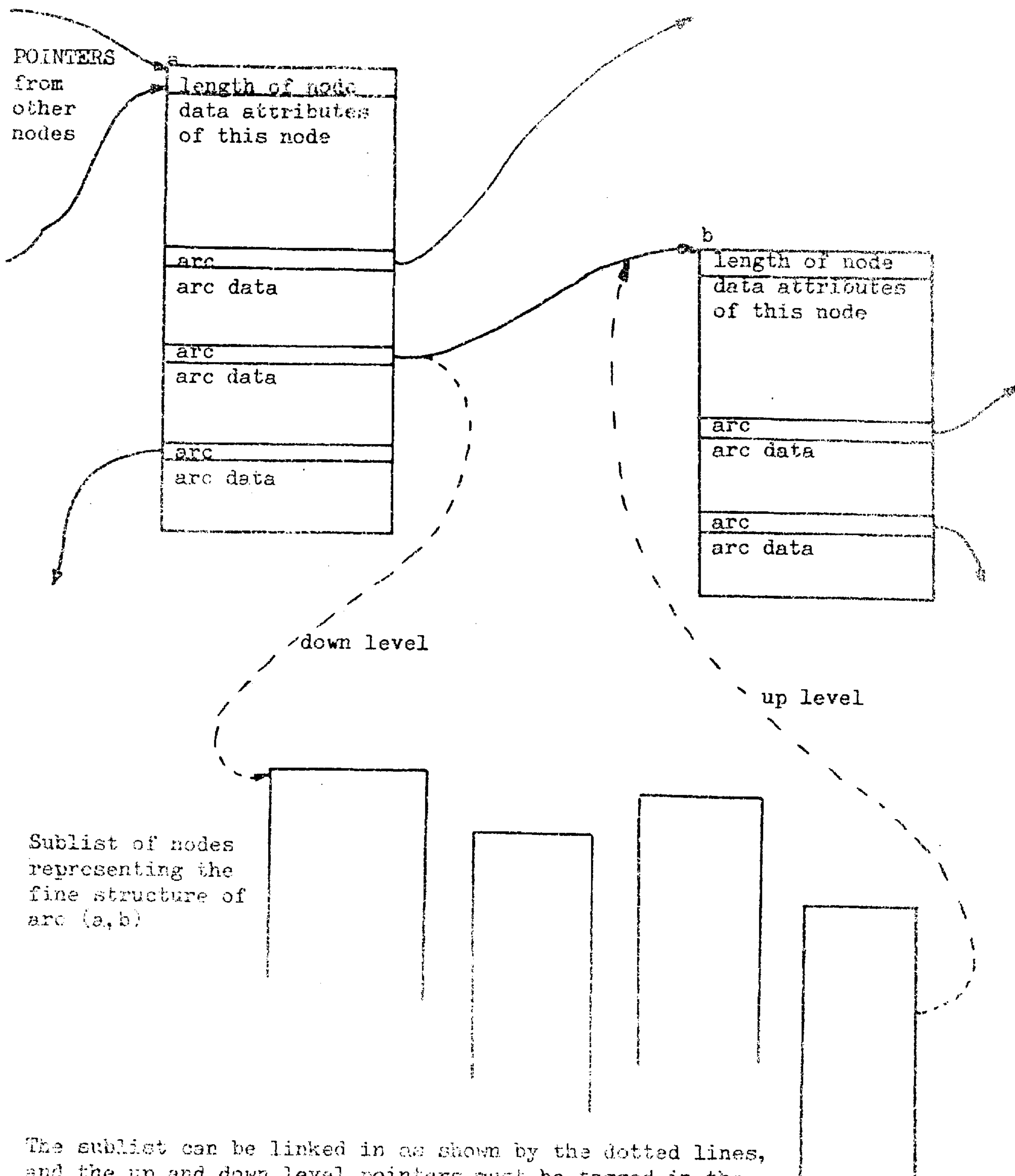
Level n + 1 holds the fine structure of arc (a,b)

Fig. 3-5 Planar representation of fine structure.

These types of graphs can be represented as list structures inside a computer, each node being a variable length table containing the data attributes of the node, followed by a variable number of pointers to other nodes (these are of course the arcs), each pointer being followed by the data attributes of the arc. Fine structure can be inserted or deleted by linking or delinking sublists at the appropriate level (up and down level pointers would be recognizably tagged). An example is shown in Fig. 3-6.

In a Σ graph there can be more than one down-arc pointing to the initial node of a subgraph, and corresponding to each of these, an up-arc to the appropriate terminal node in the higher level graph. In this situation the subgraph corresponds to a procedure or subroutine, and each down-arc-up-arc pair corresponds to a call on the procedure. Clearly when such a subgraph is activated during graph traversal, the controlling algorithm must retain records of the activations in order to return the cut to the upper level via the correct up-arc, as is indeed the case with a real procedure or subroutine call.

Furthermore if an arc S of Σ invokes Σ itself as the subgraph of S we have a recursive situation since the activation will continue down through an indefinite number of levels until an escape path through Σ, not including S, is activated. Such a Σ graph can represent a procedure which recursively calls itself. In this case the control algorithm will be required to produce and order the dynamically generated down-arcs and uparcs. This is shown in Fig. 3-7.



The sublist can be linked in as shown by the dotted lines, and the up and down level pointers must be tagged in the node tables in which they occur.

Fig. 3-6 List structure representation showing subgraph.

Graph

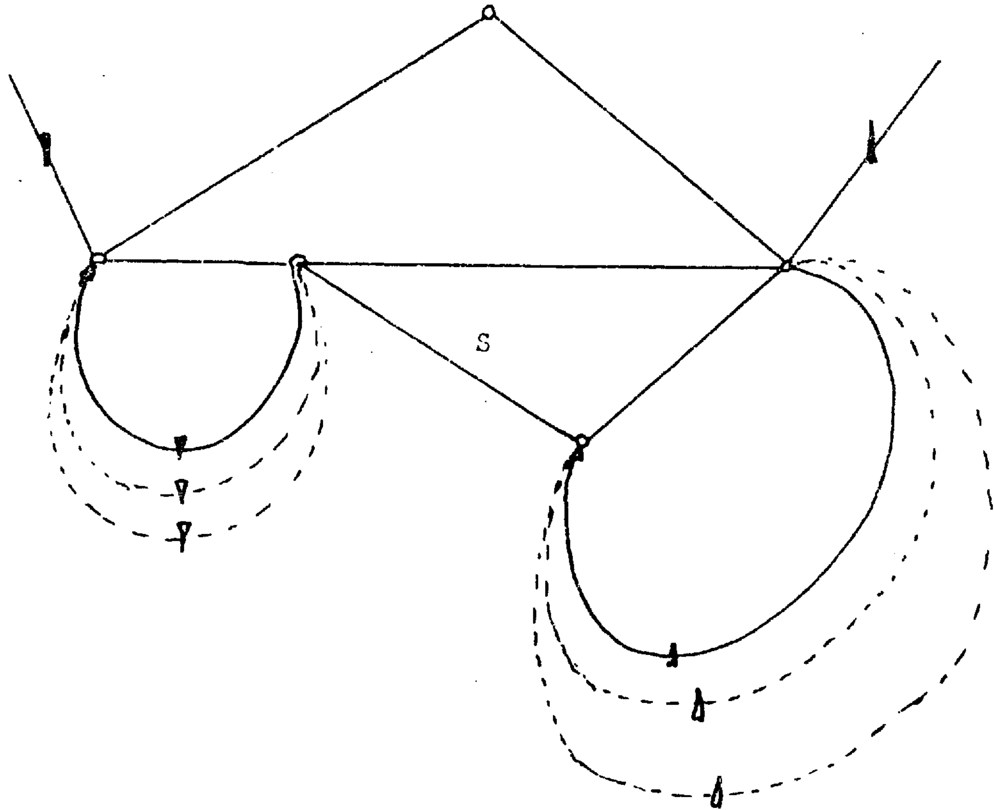


Fig. 3-7 Recursive call of Σ by itself.

A recursive definition of these types of graphs is provided below. A slightly modified Backus Naur form has been used, which uses set operators and substitutes a connection operator for the sequencing implicit in expressions such as <a><b>.

<deep graph>::=<down-arc>·<process graph>·<up-arc>

<process graph>::=<initial node>·<<edge>set>·<graph>·<<edge>set>
·<terminal node>|<initial node>·<edge>
·<terminal node>|<node>

<graph>::=<<node>set>·<<edge>set>·<graph>|<<node>set>

<edge>::=<deep graph>|<arc>

<<element>set>::=<<element>set>U<element>|<element>

<U>::=<set union operator>

<·>::=<many-to-many connection operator>

3.3 <u>Execution of a process by a processor.</u>

This section developes some functions which we suggest provide a description of the processing of a single element S of a team Σ. We shall presuppose the intuitive idea of computing power, and some measure of progress through a program. Such a measure of progress may be thought of as a quasi-distance s.

We approach the subject from the point of view of efficiency. Efficient use of a piece of hardware over a period of time, is the continuous use, over that period, of all externally visible functions of the hardware. For example, if a processor has the ability to perform twenty types of operations and is used by a program which involves only five, then three quarters of the hardware is idle while the program is executing. Inefficiency is the execution of a program by hardware of a greater computing power than that required by the program.

We now define an <u>ideal processor</u> P_O for a given program S. P_O has the property that its hardware varies in such a way that at any given point in the program, P_O consists of only that hardware needed for the program to advance at that point. This is equivalent to saying that the computing power of P_O varies along the arc S in such a way that P_O is completely efficient at all states s of the arc S (by stage we mean the quasi-distance s).

We define the computing power of P_O on S to be a function $p_O(s,u)$, not necessarily scalar, at any stage s; u is a parameter which determines the relative speed of the processor

concerned. Thus processors $P_1(u)$ and $P_2(2u)$ are identical in structure but all components of P_2 work twice as fast as those of P_1. Suppose P_0 operates for a small time δt at stage s and advances through S by δs. We then define the amount of computation done as

$$\delta w = p_0(s,u)\,\delta t$$

and the computation density as

$$j(s) = \delta w/\delta s = p_0(s,u)\,\delta t/\delta s$$

The δ notation here does not indicate the infinitesimals of calculus, but very small quantum jumps, which may be regarded as a step-wise approximation to such infinitesimals. The reason for this is that we are considering digital computers with discrete machine states. Because of their binary structure, transitions in such machines will have a quantal nature. Consequently the functions we shall deal with map onto integer, rather than real, spaces, and the processes involved may be regarded as atomic, or discrete, in their behaviour. In what follows the integration sign will be regarded as the analog in such spaces of the real integration operator.

We can now define the time taken by P_0 to execute S as

$$T_0 = \int_a^b \frac{j(s)}{p_0(s,u)}\,ds$$

and the total computation done on the arc S as

$$W = \int_a^b j(s)\,ds = \int_a^b p_0(s,u)\frac{\delta t}{\delta s}\,ds$$

We now suggest that any processor P other than P_0 will have a

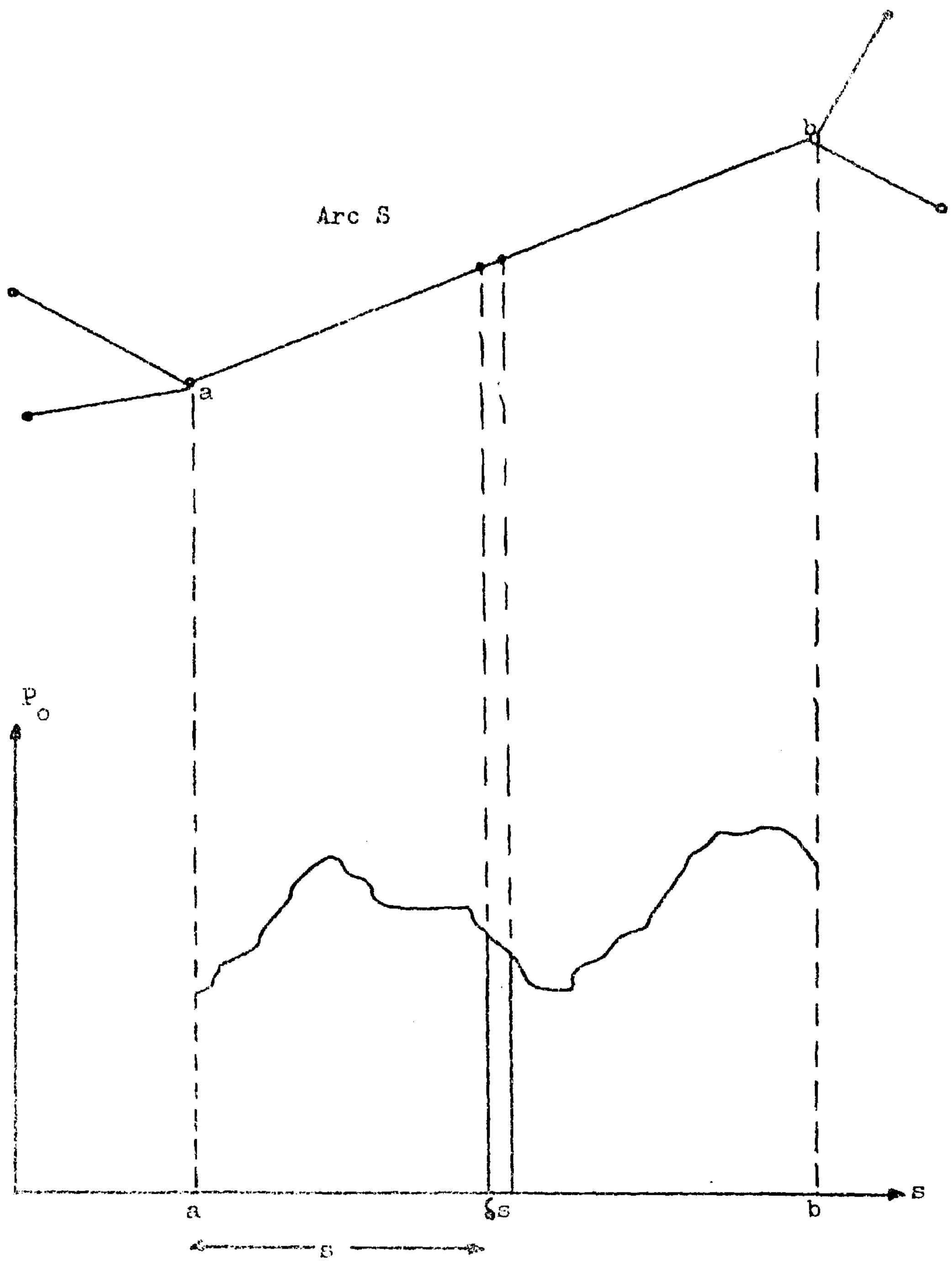


Fig. 3-8 Execution of an arc S by Processor P_o.

p-function, $p(s,u)$ which describes the computing power of P at stage s of the program S. This p-function is an expression of the hardware present which can be applied to the execution of S at the stage s. Hardware which cannot be used at this stage cannot contribute to the instantaneous computing power of P at s on S. We can now say that if,

$p(s,u) < p_O(s,u)$ P cannot execute at s,

$p(s,u) = p_O(s,u)$ P executes completely efficiently,

$p(s,u) > p_O(s,u)$ P executes faster than P_O, but not efficiently.

The last inequality indicates faster execution by P of stage s, else we would have extra hardware in operation producing no detectable differences from the behaviour of P_O. In the latter situation we cannot say that P has greater computing power than P_O. Furthermore, if P executes stage s faster than P_O, its hardware must differ from that of P_O, and cannot therefore be efficient in the way defined above. In general, if $p(s,u) \geqslant p_O(s,u)$ for $s \in (a,b)$, then the time T for P to execute S will be less than T_O, and P will execute inefficiently. We can see that for the class of processors with the same u, no processor can execute S more slowly than P_O. A program is strictly sequential if $T = T_O$ for all P such that $p(s,u) \geqslant p_O(s,u)$ on $s \in (a,b)$.

For the time being, we shall define the range $r(p,s)$ of a processor P at stage s as the distance through which P can advance along S, without the intervention of some controlling algorithm. Since $p_O(s,u)$ defines the minimum power to progress along S it also defines the minimum or quantal range $r_O(s)$. If the time taken by P_O to make the quantal transition from s to $s + r_O(s)$ is $\eta_O(u)$, we can write,

$$j(s)\ r_O(s) = p_O(s,u)\ \eta_O(u)$$

If $s = s_0$ and the succeeding quantal stages are s_1, s_2, s_3, and so on, then we can picture the range of a processor P_i as in the Fig. 3-9. The distance between points s_i and s_{i+1} represents the time $\eta_0(s_i, s_{i+1})$ for P_0 to advance from stage s_i to stage s_{i+1}. The distance from point s_i to s_j represents the time $\eta(s_i, s_j)$ taken by a processor P, whose range at s_i is to s_j, to advance to that stage. For example, the processor P_1, shown above, has a range $r(p_1, s_0) = (s_0, s_2)$ and takes a time $\eta_1(s_0, s_2)$ to reach s_2 from s_0. Since P_1 is more powerful than P_0,

$$\eta_1(s_0, s_2) < \eta_0(s_0, s_1) + \eta_0(s_1, s_2)$$

We can envisage a whole series of processors, or 'power levels', P_1, P_2, P_3, . . . which correspond to the quantal stages s_1, s_2, s_3, . . . The relations between them can be expressed as follows:

$$r(p_0, s) = (s_0, s_1) \qquad \eta_0(s_0, s_1)$$

$$r(p_1, s) = (s_0, s_2) \qquad \eta_1(s_0, s_2) < \eta_0(s_1, s_2) + \eta_0(s_0, s_1)$$

$$r(p_2, s) = (s_0, s_3) \qquad \eta_2(s_0, s_3) < \eta_0(s_2, s_3) + \eta_1(s_0, s_2)$$

until,

$$r(p_i, s) = (s_0, s_{i+1}) \; ; \eta_i(s_0, s_{i+1}) < \eta_0(s_i, s_{i+1}) + \eta_{i-1}(s_0, s_i)$$

Each processor P_{i+1} is more powerful than P_i, but less efficient over its range. When we say that $p_i > p_0$ we mean that there is some hardware of P_i which is not needed immediately at s for the next quantal stage, but that will allow P to reach $s + r(p_i, s)$ in a time η_i such that,

$$\eta_i < \int_s^{s+r(p1,s)} \eta_0 \; ds$$

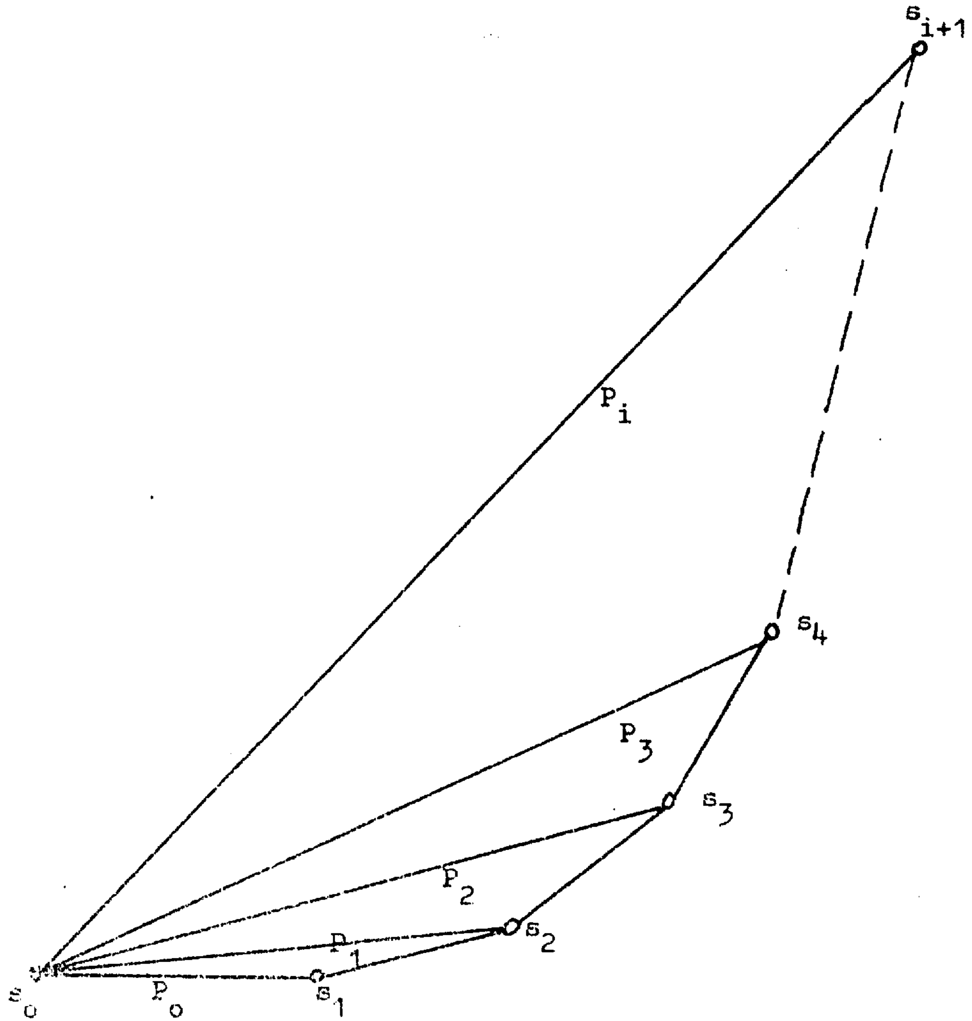


Fig. 3-9 Range of a processor P_i at s_o on S.

If $\eta_j = \int_s^{s+r(p_j,s)} \eta_O \, ds$ for all $j < i$,

then we say that S is strictly sequential from s over the range $r(p_i, s)$.

The extra hardware which allows P_i to reach $s + r(p_i, s)$ sooner than P_O, will be termed <u>redundant</u> hardware. The redundancy of P_i at s is

$$\alpha_i(s) = \frac{p_i(s,u)}{P_O(s,u)}$$

If P_j possesses hardware extra to that of P_i, $r(p_j, s) = r(p_i, s)$ and $\eta_j = \eta_i$ then the extra hardware in P_j is termed <u>superfluous</u>. For a processor P with range r at s we can write,

$$\int_s^{s+r} j(s) \; ds = w(s, s+r) < p(s,u) \; \eta(s, s+r)$$

equality occurring only if P is P_O. The <u>loss</u> L is a measure of the inefficiency of P over the range r which gives rise to the above inequality.

$$L = \int_s^{s+r} \frac{p(s,u) - p_O(s \cdot u)}{p_O(s,u)} \frac{\partial t}{\partial s} \; ds$$

$$= \int_s^{s+r} \left(\alpha_i(s) - 1\right) \frac{\partial t}{\partial s} \; ds$$

The expressions derived above represent an attempt to characterize the traversal of an arc S by a processor P. The concept of function dispersal is automatically dealt with, since if the hardware of P is oriented towards a function S then $p(s,u)$ will tend to $p_O(s,u)$. How closely $p(s,u)$ approaches $p_O(s,u)$ may be regarded as a measure of function dispersal.

We use these expressions to suggest that there are two distinct measures which characterize processor usage during arc execution. The first of these we call <u>utilization</u>. This is the proportion of the processor which takes part in the arc execution; that is, the proportion which is not superfluous. Naturally such a measure implies some means of quantifying the proportion, and we shall pursue this topic in the next section.

The second measure we propose to call <u>efficiency</u>. At any point s on the arc, or any instant in time, the efficiency is the proportion of the utilization which would be required by an ideal processor executing at that point; that is, the efficiency is the inverse of the redundancy. We can extend these definitions to cover sections or periods of arc execution. In this case the utilization consists of all elements of the processor required by the section; the efficiency will be the distance or time integral of the proportion of this utilization which is in use.

Multi-programming, or hardware sharing, can also be represented, since from the point of view of an arc S, a period when its processor is executing on some other arc may be regarded as having $= 0$ for that time. In fact pure storage and/or waiting time can be represented by dummy arcs with $p = 0$ for all s. Such an arc may nevertheless require memory elements and hence still represents a use of net facilities.

In its most general form an arc is a store (M) to store (N) transfer via a processor (P). Conventional execution can be shown as an arc with $M \equiv N$ and $P > P_0$; storage, as an arc with

$M \equiv N$ and $P = 0$; and data transfer as an arc with $M \not\equiv N$ and $P = P_I$, the rate of transmission being purely a function of u.

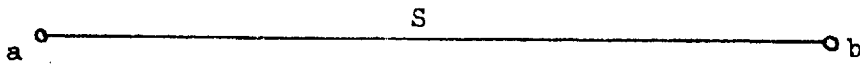
3.4 Allocation of a processor to a process.

This section deals with the problem of allocating one of a number of processors to a software task (process), and the derivation of some measures by which such allocations can be judged. The situation under consideration is the behaviour of a processor P when allocated to execute the software task represented by an arc S of a team graph Σ.

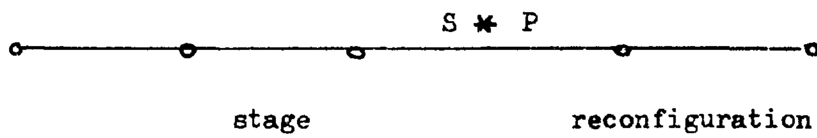
The execution of any software task is regarded as a chain of stages. Each stage is such that the processor can execute it as a single indivisible operation. That is to say that the processor can provide a hardware realization of the stages so that once the stage is initiated it will achieve its terminal state without further intervention. On completion of a stage the processor must be reconfigured to become a realization of the next stage. Thus the execution of an arc S will be a sequence of hardware realizations of stages of S, with each stage requiring a reconfiguration of P. The part of P responsible for the realization of software stages, i.e. reconfiguration, will be called the controller. (See Fig. 3-10)

It is clear that the division of S into stages will depend on P. The arc S is a description of a task to be performed, without reference to the processor allocated to it, i.e. machine independent. $S * P$ is the division of S into stages realizable by P, and is therefore machine dependent. The division will obviously be different for different processors, with only the initial and terminal nodes (a and b) remaining the same. This is the mechanism by which allocation of P to S makes S machine dependent.

Single arc of $\sum$ representing a software task.



Chain of stages representing the realization of the task by P.



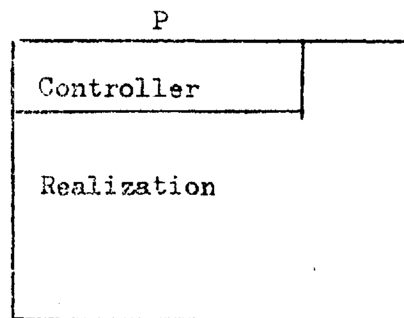


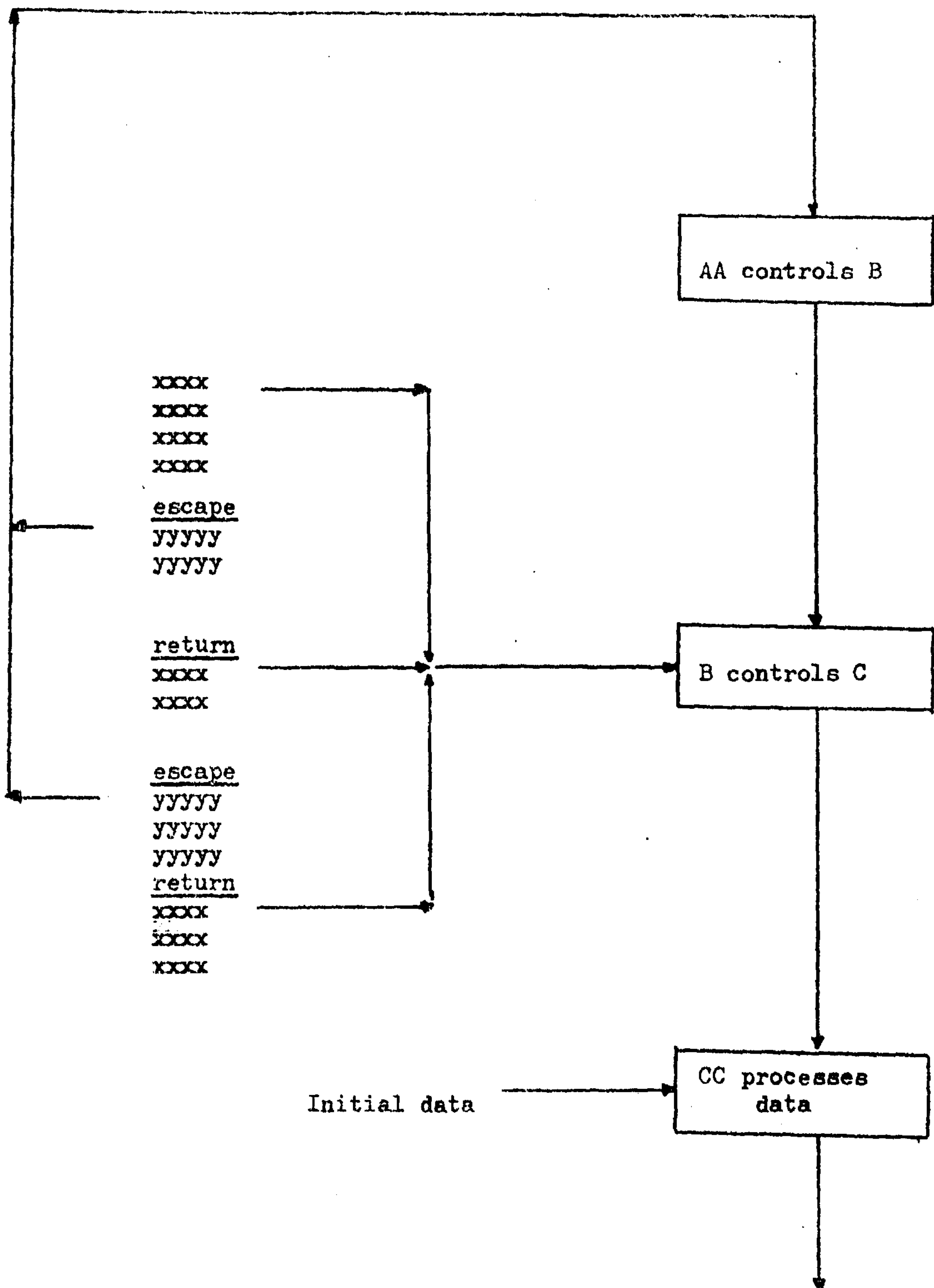
Fig. 3-10 Division of S when P is allocated.

It was stated earlier that a stage is executed as a single indivisible operation. What is meant here is that this is the case as long as we are working at the level of the subgraph of which S is an arc. It is not meant to deny the existence of a fine structure for S, which could be examined by the insertion of subgraphs at a lower level of detail.

The controller mentioned above may be regarded as a hardware program, whose input data is an instruction stream which acts on the rest of the processor as if it were a data structure. The controller may in turn be regarded as a realization of the manufactured processor, initiated by the ON/OFF switch of the computer. Reconfiguration of the controller (equivalent to changing the meaning of the instruction stream) is not general, but is possible on some machines where it is called micro-programming, and usually requires manual intervention.

Micro-programming under program control would clearly require another level of hardware controlling the reconfiguration of the controller. This level would need its own instructions, which could possibly be provided by the expedient of an escape code in the insturction stream. (See Fig. 3-11)

Whenever it is possible to allocate one of a group of processors to a process, it is desirable to make the "best" choice. The agent of the choice may be an operating system, systems analyst, or any other entity controlling the execution of software tasks. The criteria by which the goodness of an allocation is judged may vary from situation to situation, and in relative importance. For the purposes of investigating team execution on a net, an attempt is made below to develop criteria and procedures for evaluating processor allocations.



xxxx - normal instructions

yyyyy - microprogramming instructions

Fig. 3-11 Microprogramming under program control.

We do not claim that these are the only or the best possible criteria and procedures, but that they represent an interim solution, which allows the main investigation to go forward.

We will characterize the processing requirements of arcs in a subgraph σ by a number n (σ) of realizable software functions. The number and nature of these functions will be arbitrary except that they will be constant over σ, and sufficient (from the point of view of the entity controlling the allocation) to characterize all the software tasks which occur in σ.

Any processor P which is to be evaluated as a potential executioner of S will have associated with it data on its characteristics with respect to each of the software functions used in the description of σ. (See Fig. 3-12)

The behaviour of P in executing a particular function will be characterized by the measures listed in Fig. 3-13.

In addition two derived measures from the previous section will be used. These are the efficiency (Ef) and utilization (Ut). In terms of the measures defined they will be taken as

$$Ef = \sum_{j=1}^{B_j} t_j / T_i B_i$$

and,

$$Ut = B_i / B_p$$

We will not make use of Measures 3), 5), 7) in our initial allocation algorithm. It is remarked however that the time taken for a transfer of data between two sets of store cells will be an increasing function of the complexity of the gating pattern of

Data for S

No. of stages of this type

Arbitrary no. n(σ) of types of function.

Data for P

characteristics of P for this function

Fig. 3-12 Necessary correspondence between data for P and S.

1) Time (T_i) for P to execute the i th function. The unit of time will be taken as the time for P to transmit data between two store cells, when not limited by their speeds. This corresponds to the U parameter of the previous section.
2) The total number of store cells in P which can be used during data transformation (B_p). This is a constant for the processor.
3) The total number of data paths in P which can be used during data transformation (D_p). This is a constant for the processor.
4) The number of store cells required to realize the function (B_i).
5) The number of data paths required to realize the function (D_i).
6) The time t_j that a store cell j is in use during the realization of the function.
7) The time t_j that a data path j is in use during the realization of the function.
8) The cost of the processor P per unit of time (C_p).

Fig. 3-13 Measures characterizing function execution.

the pattern of the transfer. Thus the more complex the transformation occuring during transfer, the longer the sets of store cells will be in use. Consequently such complexity will affect the value of t_j to some extent. This perturbation of the t_j will be regarded as a sufficient interim measure.

The allocation of a processor P to an arc S of a subgraph described in terms of n (σ) software functions will be characterized as follows. Suppose that the i th function must be executed $\emptyset_i$ times. Then the total time to execute the arc is

$$T(S) = \sum_{i=1}^{n} T_i \emptyset_i$$

and the total cost is $C_p T(S)$. The average efficiency is

$$\sum_{i=1}^{n} Ef_i \emptyset_i T_i \Big/ \sum_{i=1}^{n} \emptyset_i T_i = \sum_{i=1}^{n} \sum_{j=1}^{B_i} \left(t_j \emptyset_i \big/ B_i \right) \Big/ \sum_{i=1}^{n} \emptyset_i T_i$$

From the definition of utilization we can say that the utilization for complete arc execution will be the union of the function utilizations. By this we mean that if a component of the processor takes part in the execution of a function it therefore takes part in the execution of the arc. Consequently we say

arc utilization = $B(S)/B_p$, where $B(S)$ is $\bigcup_{i=1}^{n} B_i$

As this last measure is somewhat unwieldy we may use Max (B_i) or Average (B_i) at times.

These measures will be calculated by a matching procedure. This will check that the processor P can in fact perform the task S, i.e. that for all functions for which $\emptyset_i$ is non-zero, T_i and B_i exist. In the case where simultaneous or overlapping demands for a function may be made by a process (or processes)

from P, a version of the matching procedure could be provided which would simulate the execution of the stages of S. The order of execution would be a function of a statistical distribution to be specified by parameters in the procedure call. This version would be used to derive measures similar to the above in hardware sharing situations. The choice of a processor can now be made using the following criteria:-

1) that P can in fact execute S
2) that T(S) satisfies any time constraint on S
3) that T(S) is minimized
4) that average efficiency be maximized
5) that arc utilization be maximized
6) that a cost function involving the above and also the total cost be minimized.

3.5 The hardware allocation problem in team execution.

This section deals with the manner in which hardware can be allocated to the various elements of a team, allowing it to progress to completion. We make the preliminary remark that there is no loss of generality in considering one team. If there are several teams within the same computing system, there is then an implicit graph at a level above, whose elements are the individual teams. Occurrence within the same net implies an interaction, if not a logical one then at least one of hardware requirement, between the several teams. Such an interaction and its associated controlling mechanism will appear as a process which can be described by the Σ-graph representation, and has the original teams as components, which in turn will be sub-levels of this graph. Analysis of this graph would then include analysis of the individual teams implicitly.

We now define the cut zone to be the set of arcs of Σ on which there are contact points, together with their initial and terminal nodes. We also add an extra chain to the graph named the idle path. This is essentially a dummy process which requires all unallocated hardware. We can picture the horizontal direction within Σ as a time axis, in which case progress occurs as the cut moves from left to right across Σ. We have stipulated that all reconfiguration (reallocation of hardware to software) takes place only at a node, and consequently there will be a node on the idle path for, and vertically below, every node in Σ. There will be a contact point on the idle path lying between the nodes corresponding

○ - node not bound to Π-graph

✱ - node in cut zone

● - contact point

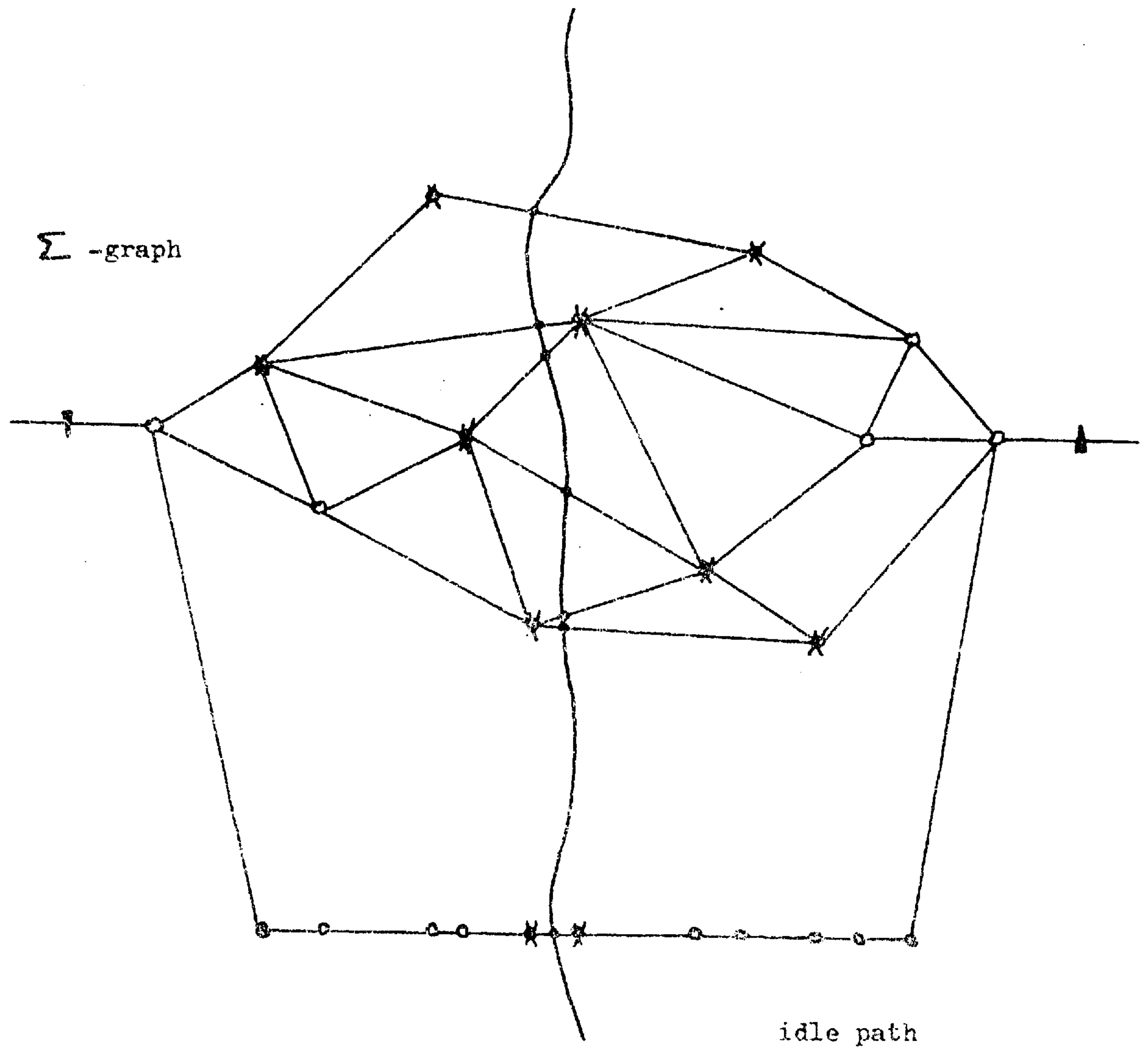


Fig. 3-14 Software graph showing cut zone.

to the most recent reconfiguration and the next one. This represents execution of the null program by all unused hardware of the net. Thus we can see that all hardware of the net is allocated in the cut zone, and conversely that an inventory of hardware across the zone will yield a sum equal to the total resources of the net, and will be constant in time.

The problem we wish to solve is how we arrived at an existing allocation in the cut zone, and how the cut will advance. The state of the cut zone is a direct result of the application of some allocation procedure at the preceding nodes of Σ. Thus a clear subproblem is how the state of the cut zone will alter as the cut crosses a node, and an allocation procedure K is applied. Analysis of successive applications of K at all the nodes in turn as the cut progresses through them, should provide a prediction of how the team Σ will execute under K with constraints Π (the nature of the available hardware, i.e. the net Π, is a parameter for K.

The following general comments may be made about K. The aim of the allocation procedure is to execute the team at least cost within some time constraints. These constraints may be the execution of the team as a whole within some time T, or the requirement that the cut reach certain nodes by certain times T_i. We assume that there will be a cost function associated with the elements of Π, which may be a function of economic cost, or of computing power (the p-functions mentioned previously). It is highly probably that K will have to deal with a priority structure when making its hardware allocations, since priority demands are not generally equivalent to completion

constraints. For example, requiring an arc to be executed as soon as possible is not a time constraint, and must be expressed in terms of a priority. The question also arises as to the distance ahead in Σ over which K will attempt to optimize its allocation. The minimal case is to consider only the cut zone, while at the other end of the spectrum an attempt can be made to optimize over a complete subgraph. This distance ahead will be termed the horizon of K. For reasons which will appear later, some of the data attributes or structure of Σ within the horizon may not be known at the time that K makes an allocation. Thus there must be facilities in K to perform a partial optimization with whatever data is available. In passing it may be noted that for a simple enumerative optimizing technique the computation performed by K goes up exponentially with the distance of the horizon.

There are two situations in which we can expect to use K. The first is as a part of the controlling mechanism of a real computer system. The second is the analysis of some given Σ graph to determine its behaviour when executed on π under K. The difference between these two situations are significant enough to warrant mention. In predictive use the potentially available horizon of K will probably be large, as data will be given for the whole graph at the beginning of the analysis. In control use there is likely to be much less data, a smaller horizon, and the description of the part of Σ within the horizon is likely to be incomplete. If the state of the cut zone proves unsatisfactory on some application of K, e.g. failure to meet some constraint becomes inevitable, then in

predictive use K can notify the analysis procedure A which may back track through Σ, reparameterizing and restructuring as far as necessary to correct the problem. This action is of course impossible in control use, and some means of escape must be provided when there is no allocation which will produce further progress through Σ. Furthermore, in control use Σ is being continuously created, both by input to the real system, and by the results of current processing.

We now consider what takes place as the cut crosses node K. Firstly it is necessary to deal with the memory associated with node k. This can be characterized by an $n \times m$ matrix, where m is the number of arcs (i,k) entering the node, and n is the number of arcs (k,j) leaving. We then have μ^k_{ij} as the amount of memory of node k containing data produced by (i,k) and used by (k,j).

The total memory of node k used by arc (k,j) will be written as $\mu^k_j = \sum^m_{i=1} \mu^k_{ij}$. Consequently an arc (k,j) will use a processor P_{kj} for a time T_{kj} and will need an amount of memory for this period equal to $\mu^k_j + \mu^j_k$. We shall leave aside the question of scratch memory for the time being, except to comment that it will be considered together with the allocation of P_{kj}, rather than μ^k_j and μ^j_k. This is appropriate, since firstly scratch memory may be reasonably considered as an extension of a processor, and secondly because the amount needed tends to vary with the processor allocated rather than the initial and final data sets.

In a CPM type representation nodes represent a strict logical dependence, i.e. all OUTarcs (outward arcs) require

Node table k.

Memory

element μ^k_{ij}

Inarcs (i,k)

i = 1,m

m = 3

Outarcs (k,j)

j = 1,n n = 4

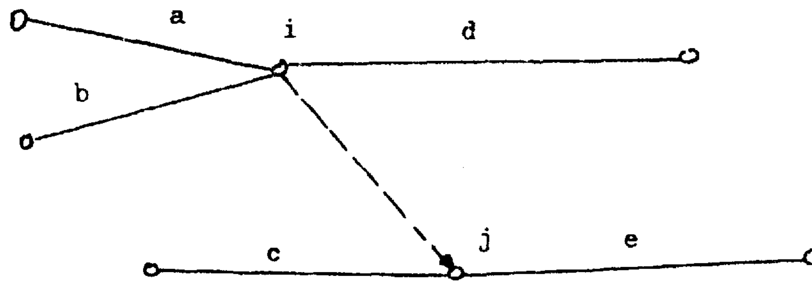
Fig. 3-15 Memory matrix for a node k.

all INarcs (inward arcs). The case where some OUTarcs require only certain of the INarcs is dealt with by the introduction of dummy arcs which specify logical dependence. An example is shown in Fig. 3-16. Here, e requires a, b and c, but d requires only a and b, so that a dummy arc (i,j) is introduced showing the logical relationship. This sort of treatment is equivalent to specifying $\mu_{ij} > 0$ for all i,j. For Σ-graphs we shall not make this restriction and will deal with logical dependence by means of the node table. If (k,j) is independent of (i,k) this will be indicated by writing $\mu^k_{ij} = 0$.

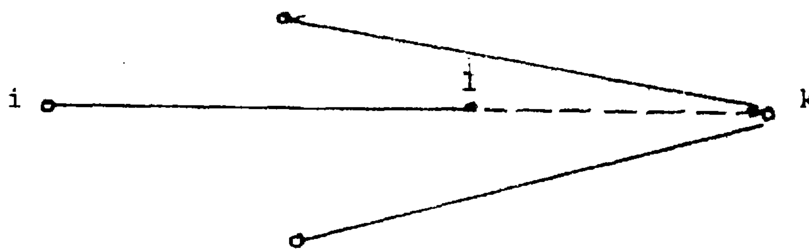
When the cut crosses node k all processors P_{ik} become available; all initial nodes of the arcs (i,k) leave the cut zone, all final nodes of the arcs (k,j) enter it; processors P_{kj} and memories μ^k_j are allocated; and all memories μ^i_k become available.

It is possible that the arcs (i,k) are not synchronized to end at the same time. Furthermore this may remain unknown until as late as the allocation of the last arc (i,k). Such situations will be dealt with by the introduction of a dummy arc and node for all but the last process/processes to end. These represent storage ($P = \emptyset$) of output data sets until all INarcs complete. An example is shown in Fig. 3-16. Clearly $\mu^l_{ik} = \mu^k_i = \mu^k_l$ and $P_{lk} = \emptyset$.

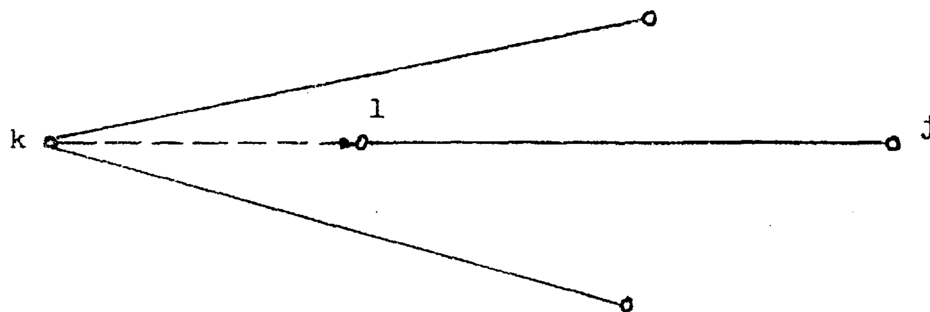
For the storage arc (l,k) we have $\mu^l_k \equiv \mu^k_l$, that is to say that they denote the same piece of physical memory. The dummy node l allows the freeing of P_{ik} and any scratch memory associated with it, and also of μ^i_k, at time t_l instead of t_k. The introduction of such a storage arc may be done as part of



CPM logical dependency



Dummy arc handling early completion of (i,k)



Dummy arc handling delay in allocating (k,j)

Fig. 3-16 Use of dummy arcs.

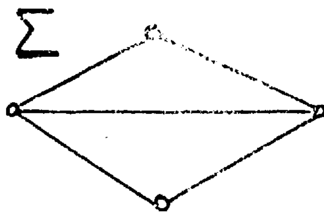
some analysis, or by K itself as part of its optimizing technique. Depending on cost function it may or may not be desirable to synchronize completion times of the INarcs (i,k). For a particular application of K it will not in general be possible to calculate the completion times of all nodes in the cut zone. A sufficient, though not a necessary, condition for the completion time of a given node to be calculable is that the cut has passed all its predecessor nodes.

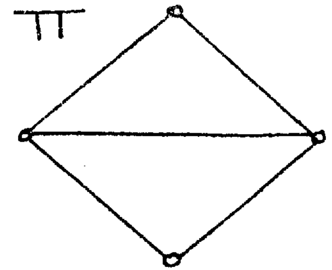
A similar procedure to the above can be followed if K cannot find a processor to allocate to some arc (k,j). This is to create a dummy node between k and j scheduled for the time of the next node on the idle path, and a dummy arc representing storage of the input data set until that time. Again $P_{kl} = \emptyset$ and $\mu^{l}_{kj} = \mu^{k}_{j} = \mu^{k}_{l}$ and $\mu^{l}_{k} \equiv \mu^{k}_{l}$

When we consider several levels within the Σ-graph the analysis becomes more complex. Firstly to reconfigure/reallocate at a level up from the one we are considering means scanning back to the last down-level and forward to the corresponding up-level, and reallocating for the subgraph. The allocated resource itself has a described fine structure so that we still have a non-trivial problem at the sublevel.

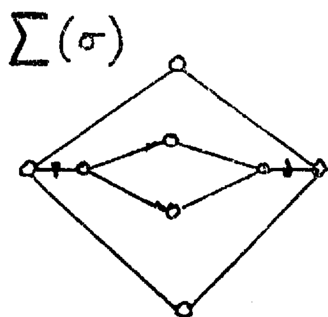
Traversal of a contact point at level n is the equivalent of traversal of σ by a cut, and the allocation analysis, at level n + 1.

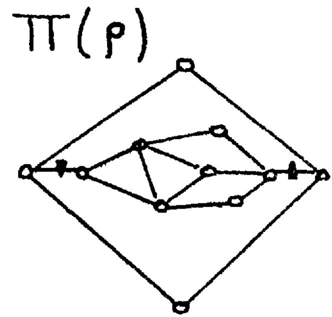
There is clearly a larger overhead in reconfiguring at the upper level, since this is not just a reallocation on the σ cut at level n + 1, but a change of total resource across it by reallocation at level n, and possible introduction of a



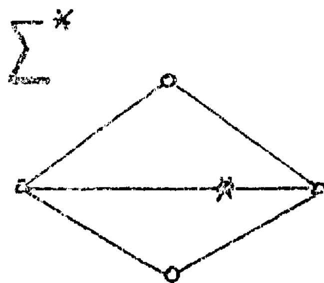


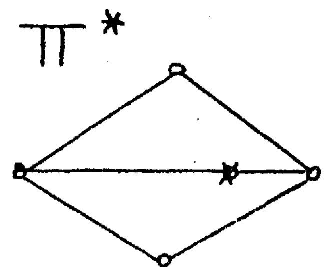
level n problem.





levels n and n + 1.





reallocation

at level n.

Fig. 3-17 Reallocation on more than one level.

dummy node at that level to allow this. Moreover a wider area of Σ is now affected.

A general scheme of predictive analysis might be as follows. A graph analysis algorithm A applies K to successive nodes of Σ, and accumulates the resulting information concerning completion times, loss, redundancy, superfluity, efficiency, and so on, throughout Σ. This will then provide a picture of how Σ will execute on Π under K. Improvement of such execution may be possible by modification of K, alteration of the net Π, or restructuring of Σ. The necessary changes will be determined by A after, and in some cases during, its pass through Σ. The process can then be repeated until some desired characteristic is achieved. An important procedure will be the arc analysis procedure S, which evaluates the execution of a single arc with a particular processor. This arc procedure may be regarded as the escape condition (in a recursive sense) of A. Consequently for a subgraph σ inserted in place of an arc (a,b), K will call S for each processor allocation it considers with parameters (a,b). S(a,b) will then find a down-level indication and consequently call A(σ). Only when A(σ) returns, can S(a,b) complete and A(Σ) progress.

Overhead at a node may be regarded as the amount of computation performed by K at that node. Consequently overhead will exist at all levels of Σ. In the case described above there will be two distinct overheads associated with node a. At the upper level, that of allocating resources to the OUTarcs of a, and at the sublevel, that of all the node allocations (calls to K) within σ (by A(σ)). At the highest level the allocation procedure becomes the attachment of Σ to some net Π.

3.6 Properties of nodes in SIGMA and PI graphs.

A way of matching the arcs of team and net graphs has been described above. This matching forms part of the overall process of binding a Σ-graph and a Π-graph. Binding establishes a correspondence between datasets and stores, processes and processors, in order to execute the function described as a Σ-graph.

A matrix representation of dataset requirements and repartitioning has been put forward. It is clear that each arc has an initial and terminal dataset. The initial dataset may be comprised of data from several sources, and the final one may supply data to several succeeding arcs. The logical dependence of one arc on another is equivalent to one arc requiring at least a part of the data produced by another as a part of its own initial dataset.

It is this logical dependence and interaction which a node represents, and which determines the arcs entering and leaving that node. With the above modelling it is therefore a truism to state that an arc has only one initial and only one final dataset, since by definition they contain all data required and produced by the arc. The initial dataset of an OUTarc is the product of repartitioning the datasets of at least some of the INarcs, and once created can be considered as a unit. A consequence of the repartitioning requirement is that the datasets must reside in the same storage medium. Otherwise repartition produces an initial dataset comprised of data on several storage media which conflicts with the model of arc execution developed so far.

This leads us to specify that an arc has only one terminal and one initial dataset, and that each dataset resides in only one store. In fact no loss of generality is involved since a process which uses data from more than one store can always be represented as a Σ-graph of arcs for which the above is true.

The assumption that interaction between processes takes place only at a node is equivalent to the independence of arcs. This independence leads us to require that the datasets of an arc are disjoint from those of other arcs. For example, if the terminal datasets of two arcs are not disjoint then the values of the data are not determined, since one arc may overwrite or alter a datum produced by the other. Further if the initial and terminal datasets are bound to the same area of physical storage the indeterminacy extends to initial datasets. This problem has been dealt with in real computer systems by an interlock on store areas preventing simultaneous writing by several processors. Read-only storage is of course not subject to a logical limitation of this type. Dijkstra provides a software version of this interlock by the use of P and V operators. Any computation where two or more processes ostensibly access the same dataset must in fact contain some interlock to ensure determinacy of the results. This can be modelled with a Σ-graph adhering to the criteria developed above.

It may be briefly mentioned that all data produced by a process is used in repartitioning. Data which was not used would be lost to the task in so far as no process would use

OUTARCS (k,j) j = 1, n n = 5

INARCS (i,k) i = 1, m m = 4

6	18	17	22	35	39
10	850	200	400	100	700
12	600		400	400	
31			900	500	
24		300		200	400

REP [1, 0] = nodenumber of i th inarc's initial node.

REP [0, j] = nodenumber of j th outarc's terminal node.

REP [0, 0] may be used to hold the nodenumber of this node.

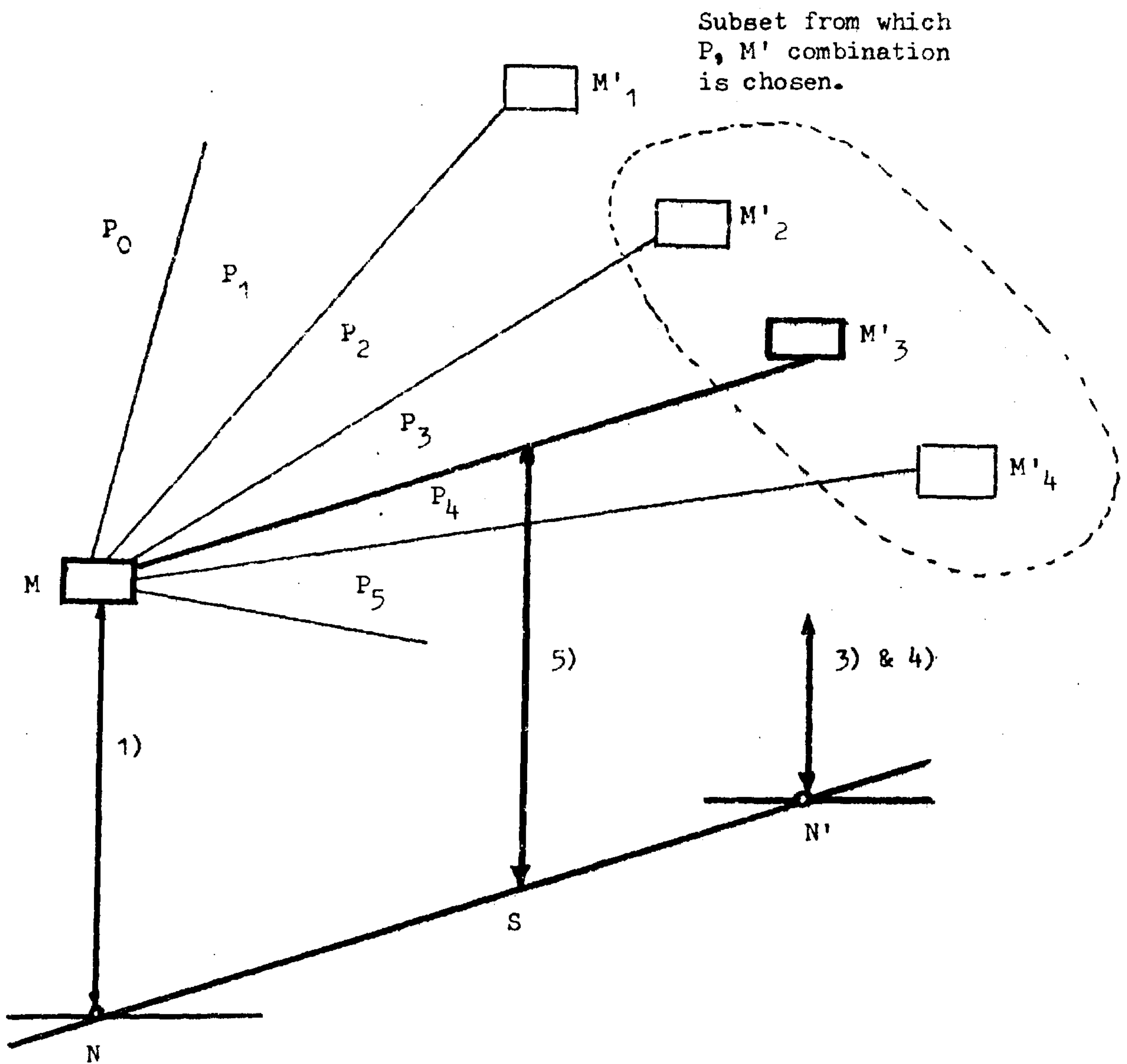
REP [i, j] amount of data produced by i th inarc which is used by j th outarc.

Fig. 3-18 <u>Repartition matrix REP of a node.</u>

it, the execution of the task would not be affected, and the corresponding storage would be discarded after repartition. Thus there would be no point in producing it. This is a way of saying that the row sum of the repartition matrix is equal to the size of the dataset produced by the INarc corresponding to that row.

We can now describe the way in which binding occurs. We have a task whose cut has reached a node N and an OUTarc S whose terminal node is N'. The node N is attached to a store M in a π-graph. The OUTarcs of M represent processors P which may read from M and produce datasets in stores M'. When all the non-zero REP (i,j), where j is the column of REP for OUTarc S, have been produced, we must choose one of the Ps and allocate it to S. In general we can choose only from the subset of the Ps which can execute S.

This subset in turn determines which M' we can use for the terminal dataset of S. Thus the particular software functions required by S constrain our choice of P and M'. Further we must reject any M' which cannot contain the datasets of N', since they may all be required simultaneously. However, we need not require that the chosen M' be empty. This is because the presence of data in M' only has the effect of delaying execution of S. On the other hand the existence of such a delay may make the choice of M' non-optimal. We can see that in fact the first INarc of a node to be attached to a processor will also determine the store to which the node is bound.



1) N is already bound to M.

2) S is ready to go (all parts of initial dataset produced).

3) The functions required by S resrtict us to only some of the Ps.

4) The total memory required at N' restricts us to only those Ps with M's of sufficient size.

5) Choose a particular P and its M' from the subset produced by steps 3) and 4).

Fig. 3-19 Binding of an arc S and its terminal node N'.

When the cut reaches a node then all the OUTarcs and their terminal nodes will undergo binding as above, and it is in this way that the computation progresses. We make the following comments about possible binding situations.

It is possible to have more than one arc between a pair of nodes N and N'. This represents two or more processes using data produced by one group of source arcs, and providing data for a single group of successors. The hardware dual is the existence in a net of more than one processor which reads from M and writes to M'.

It is also possible to have arcs with the same node as initial and terminal node. This represents an arc producing data required in its own initial node. This construction will be used later in this section. The hardware dual is a processor which reads from, and writes to, M.

If none of the processors P which are OUTarcs from M can execute the arc S, we can say that the program has failed. The failure is of the "impossible function" type, for example trying to rewind a card reader. This type of error arises because of faulty program specification, or a faulty allocation at some earlier stage. Such an allocation may have a variety of causes.

If the choice subsets described earlier are disjoint, then the allocation problem is greatly simplified because no processor is suitable for more than one OUTarc of N, i.e. there is no competition between the OUTarcs of N for any processor.

If the store M' has a smaller capacity than that required by the group of datasets of N', the team cannot be executed as

it stands at this point. However this may be circumvented (either automatically or by redesign) by reorganizing the Σ-graph at this point into a number of nodes of sufficiently small requirements. This is strongly analogous to the paging/ segmenting techniques used to solve this problem in actual computer systems.

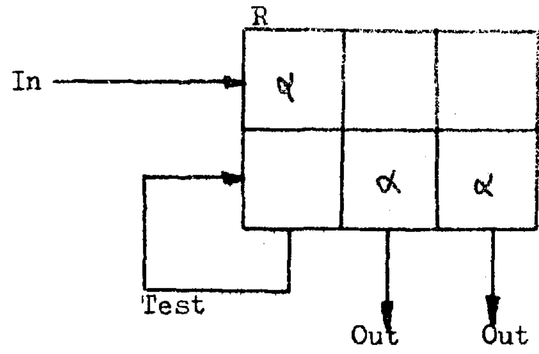
We suggest that an important criterion for the logical consistency of a program is that two (or more) arcs should not specify the same terminal node N' when their arc functions imply different terminal stores M'.

The rest of this section deals with loops and branching statements in programs. We will deal first with the representation of branching, since the description of a loop is trivial if an adequate versions of the former is available.

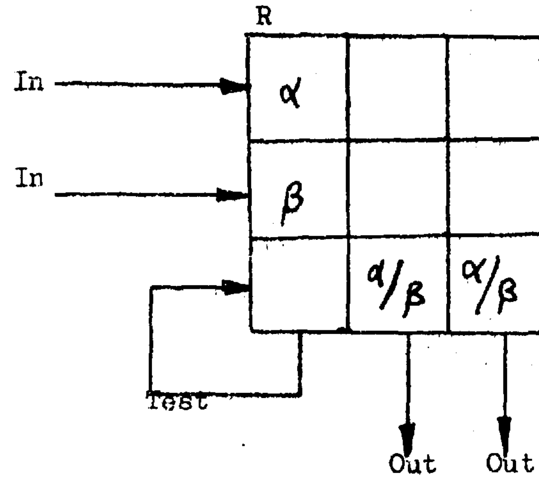
Branching statements will be represented as arcs which have the same node as initial and terminal node. This has been mentioned as a possible construction above. The essential aspect of branching is the performance of a test on a dataset (possibly consisting of only one bit) and the choice of some course of action from several as a result of the test. Clearly branching in its canonical form does not transform a dataset, though branching may be combined with transformation on a level macroscopic to the testing mechanism.

We shall allow that an arc representing a test will need only one of the parts of its initial dataset present to be initiated, and that only one of the parts of its terminal dataset will be produced as a result of this initiative. This terminal part will be logically the same as the initial

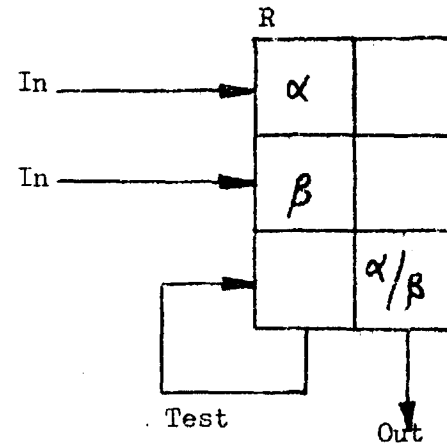
One initial part,
two possible branches.



Two Initial parts,
two possible branches.



Two initial parts,
one possible branch.



One initial part,
one possible branch.
(trivial case)

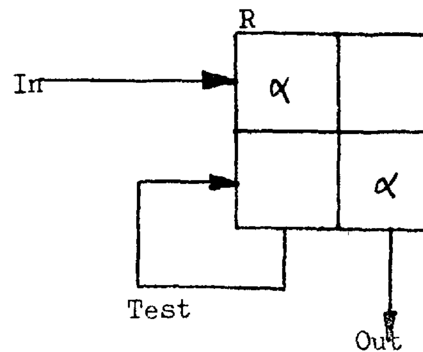
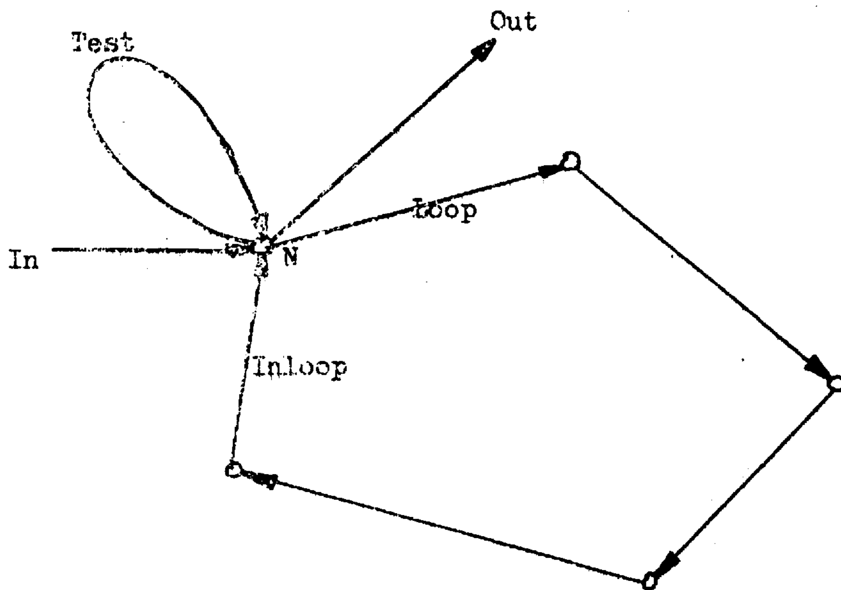


Fig. 3-20 Branching arcs.



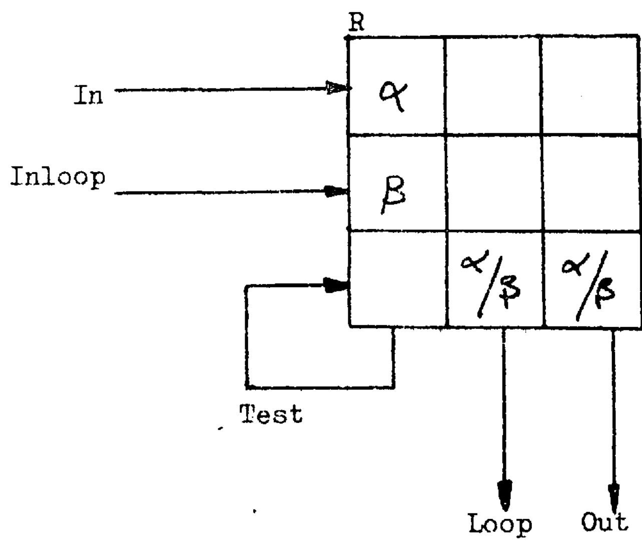


Fig. 3-21 Loop representation.

part which enabled the test. A delay may or may not be associated with the test, and in general this type of arc will be similar in all respects to an arc with distinct initial and terminal nodes.

A test will thus provide a part of the initial dataset of one of a subset of the OUTarcs of the node at which the test occurs. This OUTarc is the arc which will be executed (if possible) in the particular realization of the Σ-graph during which the test is made. Consequently we will provide a means of disabling the remaining OUTarcs of the subset, since they must not be executed unless re-enabled by some subsequent application of the test. Such a mechanism also allows us to distinguish between arcs whose initial datasets are disabled and those which are merely waiting for their production.

Thus all possible branches will be represented in the Σ-graph, but a particular realization will bind a unique selection of these while disabling the rest. This is equivalent to saying that for any given execution of a program only one of the possible paths through it will be taken.

The hardware dual of this situation is the ability of a processor P to write to several stores M'. Clearly when P is allocated to a process S, because the process has only one terminal dataset, which resides in one store, the other possible configurations of P will not be used. All possible configurations will be represented in the corresponding Π-graph as arcs between M and the stores M' to which P can

write. Only one will be used in any particular realization of S on the Π-graph, and the rest will be disabled for the period of realization.

In an actual computer system the choice made during any particular realization of a Σ-graph will be data dependent. Where an analysis is being carried out we have a number of mechanisms available for making the choice. Random choice, irrespective of which initial part enabled the test, random choice dependent on the initial part, and either independent or dependent presetting of the terminal part to be chosen prior to the analysis, are possible methods.

Loops can be represented by the use of a test arc as follows. The initial parts to the test are the first entry to the loop and a subsequent entry. The terminal parts are the exit from the loop or the body of the loop (i.e. a sequence of nodes and arcs which leads back to the subsequent entry). Loops which are a sequential representation of an inherently parallel computation can be represented by their parallel form. Loops which are iterated a given number of times will use the loop counter as the datum for the parts of the initial dataset of the test arc.

Finally we suggest that the REP matrix bears a strong kinship to the precedence matrix for the INarcs and OUTarcs of its node.

As an example of the use of loops, Fig. 3-22 gives the REP matrix for one process of the two process interlock algorithm below. The algorithm is described more fully in Cooperating Sequential Processes by Dijkstra.

```
"begin integer c1, c2, turn;
        c1:= 1; c2:= 1; turn:= 1;
        parbegin
        process 1: begin A1:c1:= 0;
                         L1: if c2 = 0 then
                                begin if turn = 1 then goto L1
                                      c1:= 1;
                                B1: if turn = 2 then goto B1
                                     goto A1
                                end;
                             critical section 1;
                             turn:= 2; cl:= 1;
                             remainder of cycle 1; goto A1
                   end;
        process 2: begin A2: c2:= 0;
                         L2: if c1 = 0 then
                                begin if turn = 2 then goto L2;
                                      c2:= 1;
                                  B2: if turn = 1 then goto B2
                                      goto A2
                                  end;
                             critical section 2;
                             turn:= 1; c2:= 1;
                             remainder of cycle 2; goto A2
                   end
        parend
end".
```

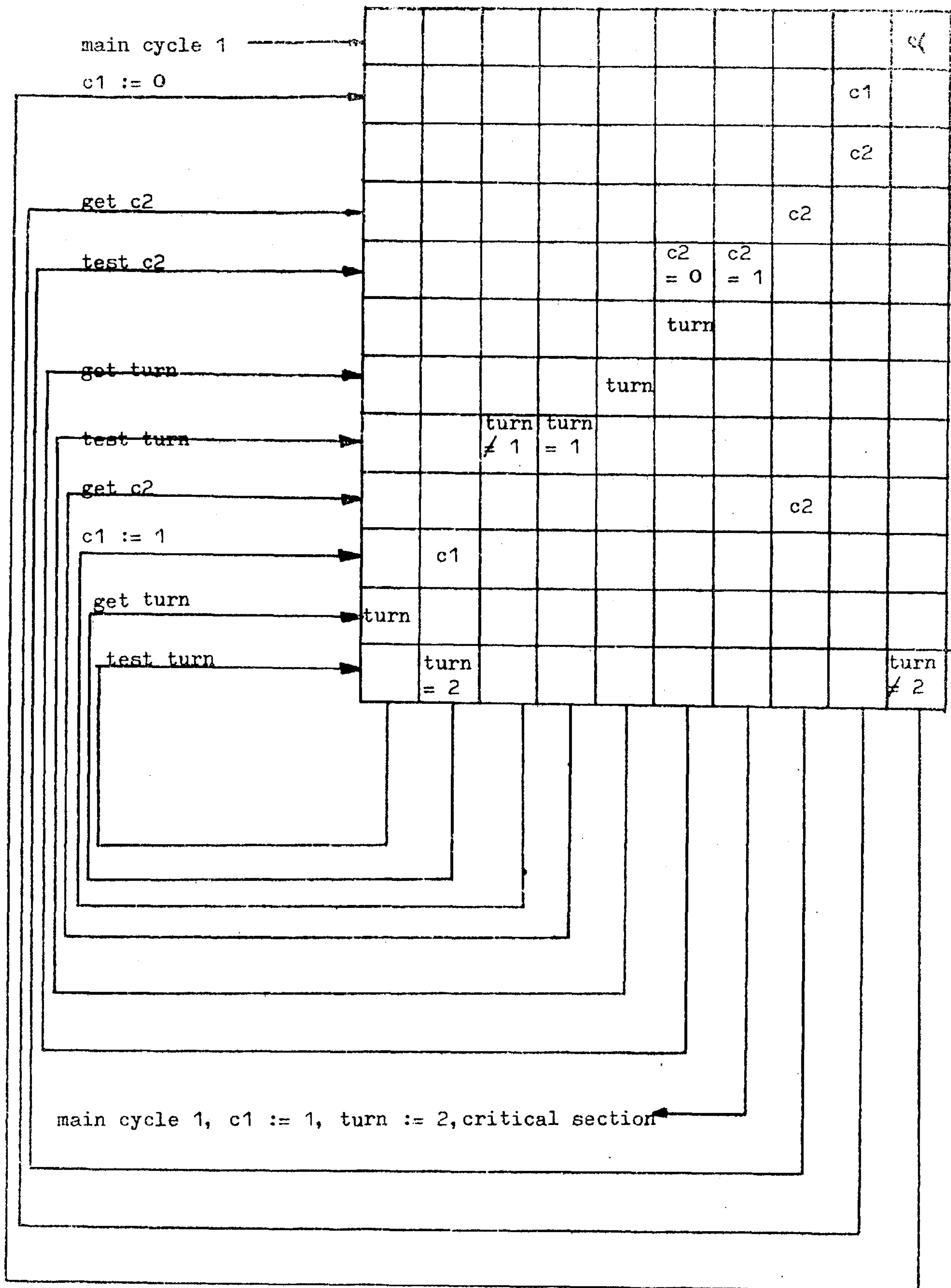


Fig. 3-22 R-matrix for process one of Dijkstra's interlock algorithm.

3.7 Data dependence and reentrance.

There are two ways in which data can influence the processing required by a program. These are by the size of a dataset, and by the value of a data item. For example, an input operation may be repeated until a special character is detected. In this case the amount of processing is clearly dependent on the size of the dataset for which the special character is a terminator.

Where alternative paths through a program exist the choice of path, and therefore the processing done, usually takes the form of testing the value of a program variable. In fact dataset size and variable value can often be expressions of the same thing. If the input operation above is counted then the value of the count will express the size of the input dataset. This count may be used in another part of the program to control the size of an output dataset, or to select a program path.

That is to say that any run of a program is provided with the values of the data for the run, and the size. The size may appear explicitly as a particular value, or implicitly as a delimiter. It is more general to say that the structure of the data influences the processing performed by a program. Currently, however, there are no widely implemented processing units which can operate directly on the structure as well as the value of data. Nor are there storage media capable of directly expressing any structure except linear sequential strings (of bits or characters).

Because of this much of the structure of a dataset is expressed as additional values within the linear sequential

mold, e.g. pointers, cross-references, subdelimiters, and so on. The structure then influences the processing which occurs by selecting the program path according to these additional values.

In the context of contemporary processors and storage media there is therefore no loss of generality in describing the factors which affect the processing of a particular run as dataset size and value.

In a Σ-graph (SIgraph; node called SInode) the data dependency of the OUTarcs of a node on its INarcs is specified in the repartition matrix of the node. That is to say that REP [i,j] gives the proportion of the terminal dataset of INarc i that is required by OUTarc j. These proportions apply to incoming datasets of unit size, e.g. one record or character. The actual size of the terminal dataset will be determined by a run-time attribute of INarc i, λ_i.
If D_i is the total size, then

$$D_i = \lambda_i \sum_j \text{REP } [i,j]$$

The total size of the initial dataset E_j of OUTarc j will now be,

$$\lambda_j = \sum_i \lambda_i \text{ REP } [i,j]$$

This will in turn determine the actual size of the terminal dataset of OUTarc j, when it is executed. The factor λ has no effect on the characteristics of arc execution such as utilization and efficiency. It is regarded as simply multiplying the amount of storage required to hold datasets, and the time required to process them. Consequently all PIarc-SIarc matching procedures, and comparison of the matches will be independent of the values of λ. Execution times will be λ times those

for SIarcs operating on unit datasets, and this clearly must have an effect on overall subgraph execution.

It is often the case that a dataset is left unchanged by a process using it. In such a case the dataset is still available for other processes, and still useful since its contents remain known. If the dataset is unaltered by a process, then it may be used concurrently by another. This type of data is called read-only and is often encountered in programming systems. On the other hand data which is altered by a process is left undefined on completion, and furthermore cannot be used concurrently by another process since the contents are unreliable. We call this data read-write data.

We would like a SIgraph to include differentiation between read-write and read-only datasets. This can be done by the sign of the REP matrix element which represents the dataset. If $REP[i,j] > 0$, then the dataset is read-write, and if $REP[i,j] < 0$, then the dataset is read-only. The current status of a SInode's datasets is held in a separate matrix ACT of identical dimension to REP. If $ACT[i,j] = 0$, the dataset represented by $REP[i,j]$ is inactive; if $ACT[i,j] > 0$, the dataset is active. ACT is called the <u>activity</u> matrix of the SInode.

With the introduction of read-only data it becomes clear that a necessary corollary is some means of deactivating datasets, else a read-only dataset, once activated, must remain so. We now extend the properties of a <u>SIarc</u> to include a function zero (a SIarc is an arc of a Σ-graph and is defined by functions one to n where n is a parameter of the graph). This function, PHI [0], is specified, like the rest, by the

modeller. If PHI [0] > 0, then the terminal datasets of the SIarc are to be activated, and if PHI [0] < 0, they are deactivated. We define PHI [0] = 0 as taking no action concerning the terminal datasets, and such an arc will thus bring the allocator to the terminal node without affecting it.

In this it is the modeller's equivalent to an allocator generated delay arc. Where PHI[0] is non-zero it is anticipated that the numerical value may be used to define other subclasses of terminal action.

Read-write/read-only datasets and activation/deactivation SIarcs are logically complementary and allow alternative dataset action at both initial and terminal nodes to be specified within a SIgraph.

The selection of alternative program paths according to the value of program variables cannot be accurately modelled short of duplicating program execution with real data. To correctly imitate the run-time choices, all variable values involved in them would have to be derived, by the same algorithms as used in the program, from the same data. Clearly one might just as well execute the program under investigation with some run-time monitoring to record all values and choices. Neither is such an effort particularly rewarding, since the results are relevant to only one run. Instead, we make the assumption, in common with most simulation models, that for sufficiently large numbers of runs the values of variables used in choosing program paths will be drawn from recognizable probability distributions.

This allows us to dispense with knowledge of the actual data values of a program run. At each point of choice we use

the estimated characteristics of the probability distribution to perform the choice by random drawing. In a SIGMA graph such choices are called IFloops (an arc with the same initial and terminal node).

An IFloop is regarded as having its initial datasets in column j of the repartition matrix, and producing terminal data in row i of the same matrix, and will be activated by only one of the elements of column j instead of all. On completion it will activate only one of the elements of row i, instead of all.

The IFloop can choose the element to be activated by one of several random drawing methods. The method to be used is part of the data which describes the IFloop. Because of the relation between dataset size and variable value, we include the facility for an IFloop to choose the value of λ by similar methods.

Since λ has a multiplicative effect on execution time, this is also equivalent to random drawing of the execution time of an IFloop. For completeness this too is included in the facilities provided in an IFloop, which can now be seen to provide an adequate means of expressing and emulating the data dependent aspects of subgraph execution.

We now turn to reentrance, which can be regarded as falling within the scope of a discussion on data dependence. Reentrance is the property of a hardware-software system which allows it to sustain concurrent executions of the same program.

We will call each such execution a transaction. A transaction is distinguished (and identified) by its data. If we assign a unique integer to each transaction as it is created,

then this integer will form part of the data, albeit only a read-only datum. In a SIgraph the arcs may execute in parallel. If the arcs are treated as representing reentrant programs, then we may have more than one transaction per arc.

A set of active arc executions in a subgraph are logically related if the are historically descended from the same activation of the initial node of the subgraph. Such a set has been called a cut, or cut zone, previously. Being descended from one activation of the initial node, the members of a cut represent a single realization of the task represented by the subgraph. That is to say the cut represents one transaction executing the arc whose structure is represented by the subgraph.

If more than one transaction is executing this higher level arc, then there will be a cut active in the subgraph for each transaction. If the executions of these transactions are to retain the qualities of reentrance, then their datasets must remain distinct and must not combine (by repartitioning at a node) to activate any arc, since such an arc would belong to both cuts, which would in consequence no longer be logically distinct.

Thus we can see that reentrant execution of a subgraph requires logical independence of each active cut. Since the execution of individual arcs is already logically independent, even with one cut, the requirement is that the datasets used in repartitioning at any SInode shall always belong to the same cut. In practice this demands that each cut carry its own status information about each of its active nodes.

We call this type of execution completely reentrant (mode 3).

It is possible to derive some more limited modes of graph binding as follows. If we introduce the condition that no cut may activate an arc until the previous cut has completed it, we are effectively introducing a first in first out discipline within the subgraph. This is equivalent to requiring that transactions at the upper level should always maintain the same ordering (namely that in which they were generated). We call this type of execution sequentially reentrant (mode 2). It can be realized by requiring the executing allocator to adhere to the condition stated above, and to queue (FIFO) terminal datasets at their terminal nodes in the event of any element of a prior one still being active at that node. That is to say an incoming transaction on arc i will be queued (i.e. will continue to require storage) until all REP [i,j] are inactive (all prior transactions on OUTarcs j completed). The first transactions of each INarc queue will be used to reactivate the node as soon as it has become inactive.

If we now eliminate the possibility of queueing datasets, we restrict the execution even further. We now require that not only will there be at most one execution of an arc taking place at any given time, but also that there will be only one realization of a dataset at any time. That is to say that successive transactions on an arc read from and write to the same dataset (i.e. there will be only one store image of the dataset at any time). This type of execution demands that an arc may not be activated by a transaction if the terminal dataset still has any active components produced by a previous

activation (mode 1). This restriction must be made since a process may access its initial and terminal datasets at any time while it is active.

Since transactions now use the same datasets, their behaviour now corresponds to that of cooperating sequential processes as described by Dijkstra. This is so because arc execution is a critical section with respect to the arc's initial and terminal datasets. The interlocking of arc access to datasets is performed by the allocator.

Thus though more than one cut may be initiated, all are subject to the same interaction constraints as those which operate between the members of a single cut. For this reason we call this type of execution non-reentrant.

We now consider some aspects of simultaneous allocation. It is sometimes required that two processes commence execution simultaneously. A typical case is that where one process times the other. Clearly the initial data of both processes must be present before either is allocated. This is equivalent to viewing the two processes as the subgraph of a SIarc whose initial data is their combined initial data. This SIarc is then subject to the normal condition that all its initial datasets must be present (active) before it may execute.

Furthermore the arcs which produce these datasets must be INarcs to its initial SInode. Consequently we see that, without loss of generality, when two processes are to be allocated simultaneously their initial datasets must appear in the same column of a SInode REP matrix. This column corresponds to the SIarc whose subgraph is formed by the two processes.

For convenience we develope a shorthand description of the situation which dispenses with the necessity for a subgraph. We describe both processes by SIarcs which appear in the OUTarc chain of the SInode, and add a further row to its REP matrix. The zero element of the column becomes the arc specifier for the first SIarc, and the last element of the column (which is an element of the new row) becomes the arc specifier of the second SIarc.

In order to perform a simultaneous allocation the allocator acts as follows. When the column j becomes ready the allocator attempts to allocate the first OUTarc (specifier is REP [0,j].

If REP [m + 1, j], (m Inarcs) is non-zero this indicates that there is a second OUTarc to be allocated simultaneously. The allocator will attempt to allocate this arc as well. If both allocations succeed then two <u>ties</u> (bound process-processor pairs) will be activated and simultaneous allocation is achieved. If one arc is allocated successfully and the other is not, the hardware resources for the first are reserved until the second arc can be allocated as well. In meantime it is marked as a delayed arc. Reservation is accomplished as follows. If the successfully allocated SIarc requires a fraction u of the processor P which was chosen, then a variable which represents the current usage of P is increased by u, u itself being recorded in a similar variable (SFRAC) of the SIarc. Thus u is unavailable for allocation to other SIarcs. Terminal storage, if it is required, is also allocated in the terminal <u>PInode</u> of the <u>PIarc</u>. These terms denote a node and arc of a PIgraph.

When the allocator returns to the column and is able to allocate hardware resources to the second SIarc successfully, it initiates ties for both SIarcs. Each tie will then release its resources on terminating. When a processor is reserved, an identifying attribute is recorded in the element of REP which specifies the OUTarc. This allows the allocator to know which processor, and consequently which store, was reserved, on a subsequent scan of the SInode.

This record of reservation is a special case of the fact that whenever a SIarc specified by REP [p,j], (p = 0 or m + 1), is allocated, the processor state identifier (SEQF) is recorded in ACT [p,j]. This means that ACT [p,j] always contains the identifier of the last processor to be allocated to the corresponding SIarc, and this facility is used for error handling and disabling of hardware, as well as simultaneous allocation. The delayed status of an arc j can be shown by setting ACT [p,j] negative. It is clear that by the provision of further arc specifiers per column simultaneous allocation of more than two processes can be described and executed using the method outlined above.

The ACT matrix separates the descriptive aspect of the SIgraph, from the binding time information. This is particularly helpful when we consider the implementation of mode 3 binding. Here binding is completely reentrant, so that each cut which traverses the SIgraph must carry all its status information with it. This can be achieved by allowing each cut to carry its own set of ACT matrices for nodes at which it is active. Since every completing tie activates the allocator which

generated it we now have the means of completely separating binding information from graph description, which is the essence of mode 3 binding. The effect is to make the SIgraph itself into read-only data for the allocator.

The ACT matrix also provides the counter function for DOloops (again helping to separate SIgraph description from dynamic variables). When a DOloop is activated it searches for the row i of the REP matrix for which it is the INarc. The condition for this is that SEQF equals REP [O,i]. If column j provided the initial data of the DOloop, then ACT [i,j] is used as the DOloop counter.

When the DOloop is activated it checks ACT [i,j] for zero. If if is zero the DOloop assumes that this is a first iteration and sets ACT [i.j] to the number of iterations required (specified in one of the parameters of the DOloop description). At the end of each iteration ACT [i,j] is decremented by one, and tested for zero. While it is positive the column of the first non-zero REP element in row i is chosen as IFCOL. When ACT [i,j] is zero after the decrement, then the column of the second non-zero REP element is chosen. If ACT [i,j] is negative this is regarded as a non-fatal error and IFCOL is set to -1. This signals the allocator not to activate any dataset, and effectively extinguishes the DOloop.

An example of the deliberate use of the last case occurs when a deactivation arc operates on the terminal row of the DOloop. In this case ACT [i,j] will be set to zero, and so the subsequent DOloop decrement will bring it to -1. Consequently the DOloop will be extinguished by the deactivation arc.

Such a deactivation, followed within one iteration time by an activation, presents a problem in reentrance. The second activation will find ACT [i,j] = 0 and set it to n, where n is the number of required iterations. The first DOloop will

now never find ACT [i,j] = -1 and so will not extinguish itself. Instead it too will decrement ACT [i,j] and continue to iterate. Now we have two DOloops iterating concurrently and, of course, decrementing ACT [i,j] twice as fast. The number of concurrent DOloops can build up to n in this way. The reentrance rules described previously would normally prevent this happening. While the first DOloop was active the second one would be queued (in mode 2) or not allocated (in mode 1) because ACT [i,j] would be detected as non-zero, thus showing the existence of an already active OUTarc j. The deactivating of row i sets ACT [i,j] to zero and effectively hides the existence of an active OUTarc.

It is clear that this problem extends to any case where one or more of the initial datasets of an active OUTarc are deactivated and then reactivated while the OUTarc is still active. To ensure behaviour appropriate to the execution mode we introduce a further check for ties which activate (rather than deactivate) their terminal datasets. The check is on the attribute SFRAC of SIarcs corresponding to the columns j containing the terminal datasets REP [i,j] of the tie. If SFRAC > 0 the SIarc is known to be allocated and executing (SFRAC holds the processor fraction allocated) and consequently the reentrance rules can then be applied. For SIarcs which use no processor functions we require that SFRAC be set to 1 on allocation.

We now make some comments on error handling. By error we mean a hardware error, i.e. a malfunction of some part of the PIgraph. A detected software error implies a different path

through the SIgraph from the point at which it was detected. A software error whose detection is not modelled in the SIgraph is a wrong result from the human point of view, but not from the algorithmic one.

A hardware error occurs during the execution of a tie. Typically it will be modelled by an IFloop which chooses the error path or the normal path by drawing from a statistical distribution. Errors which are not detected or not acted upon obviously do not concern us. When a tie is initiated there must be a subsequent moment at which it is decided whether the tie completed normally or in error. There is usually a finite time limit on this moment. The decision can only be finally taken by the initiator of the tie, since it is only the initiator who has the ability to directly reinitiate the tie, or go on to the next tie, or transaction. Furthermore an error can be of the type which renders the recipient unaware that a tie was ever initiated. That is to say that the only location where a record of tie initiation and the data for its reinitiation can be relied upon to exist, is at its initial SInode.

The error decision can be made in one of two ways. The return of an acknowledgment allows a decision depending on whether the acknowledgment was a good or bad one. If no acknowledgment is returned the arrival of the time limit allows a decision to be made depending on whether the expiry implies an error or a normal termination to the process. In either case we require that the decision shall correspond to the activation of one of two datasets in the initial SInode

of the tie. This SInode is the only place where there can be certainty of the decision being taken at all, and where there can be certainty of the retention of the initial datasets of the tie.

We do not concern ourselves with the dataset which represents normal termination since this is clearly only a matter of the deactivation of the initial data and/or the extinction of the process arising from the dataset. In the event that the dataset corresponding to error termination is activated there are usually two possible procedures. The process which was in error can be repeated (tie reinitiated), or the corresponding hardware made unavailable for future allocations.

Reinitiation can be modelled using normal SIgraph facilities. Suppose REP [1,j] represents the initial tie data and REP [2,j] is a ready flag, then on completion of the tie j, REP [1,j] will remain active (read-only dataset) and REP [2,j] will be deactivated. On normal completion the initial data will be deactivated and REP [2,j] activated. In the case of error termination the initial data will remain active and REP [2,j] will be activated, thus making OUTarc j ready again, and so the tie will be repeated.

A common method of treating errors is a fixed number of repetitions followed by disabling the hardware involved. The disablement is for a finite period whereupon the hardware is enabled and execution attempted again. In the SIgraph we provide a general facility for enabling and disabling hardware, i.e. one which can be used for other reasons besides error handling, in the form of two corresponding IFloops.

The first disables the hardware last allocated to the SIarc specified in a parameter (a) of the loop. It does this by a similar method to that used for reservation of hardware. The IFloop searches the SInode for the arc specifier REP [p,j] corresponding to a and extracts the value of the processor attribute SEQF from ACT [p,j]. The processor must be an OUTarc of the PInode to which the SInode is tied, if the initial tie data has been retained. Otherwise the PInode can be reached through a PIgraph node index.

In either case the PIarc is found and its inuse fraction is incremented by 1. This has the effect of making the processor unavailable for further allocation irrespective of the fraction currently allocated and its subsequent release. The attribute SEQF of the processor (which completely identifies it) is now placed in ACT [p,k] where REP [p,k] is the arc specifier for the disabling IFloop. The attribute SFRAC of the IFloop is set to one as usual for a SIarc which requires no processor. The tie which originally executed in error can now be reinitiated or deactivated as required.

If it is desired to make the processor available again after a delay, the disabling IFloop can be given the appropriate duration, and its tie can alter its associated processor from null to the disabled processor. On completion the allocator will release the disabled processor as part of its normal completion procedure since SFRAC of the disabling IFloop has been set to one and the tie now has an attached processor.

Alternatively the IFloop may be given a zero duration and its completion allowed to initiate a delay loop. This loop

can then initiate the second type of IFloop mentioned, namely an enabling loop. The enabling IFloop operates in a similar manner to the disabling IFloop, except that it subtracts one from the inuse fraction of the processor. The processor itself is obtained in an identical manner to that used by the disabling IFloop. It is clear the OUTarc specifier in an enalbing IFloop can refer to a disabling IFloop, so that the former can release the last processor disabled by the latter.

The duration of a tie will depend on the physical characteristics of the stores in which its initial and terminal datasets reside. We introduce a function V to represent this perturbation of tie duration. Clearly for normal stores p, V will be a function of the quantity of data being processed by the tie, so we write that the tie duration will be

$$t * \lambda + V(\lambda, p_1) + V(\lambda * E, p_2)$$

where E is the sum of the terminal REP elements, and t is the processing time per unit data, as provided by the allocator. A subset of store characteristics provided by PIgraph description might be delay, latency, block size, and block time. The delay is, for example, the average seek time during disc access. The latency is the rotational period of a disc or drum. The block size is the quantity of data moved in one transfer, and the block time is the time to move it.

The function V can be defined to suit the modeller, and we would choose the following as a default. One drawing from the uniform distribution between zero and delay, plus m drawings from the uniform distribution between zero and latency (where m is the number of blocks), plus m times the block time, i.e.

$V(\lambda) := m * \text{block time} + \text{uniform}(0, \text{delay}) + z;$

where <u>for</u> i : 1 <u>step</u> 1 <u>until</u> m <u>do</u>

z : z + uniform (0, latency);

m is defined as the smallest integer greater than λ/block size.

An example of the use of the facilities above is the way in which splitting the leading character from a message is modelled. We achieve the desired effect by simultaneous allocation of two IFloops, the first one resetting LAMBDA to LAMBDA minus one, the second one setting LAMBDA to one. The total memory requirement is exactly equal to the initial memory present and no account need be taken of the range of LAMBDA values.

Finally we mention a possible extension of SIgraph facilities. This is the addition of further variables which propagate with the cut. Such variables might be carried by ties. At each node a new value is generated for an outgoing tie from the values carried by its inarc ties. Values are set by IFloops and may or may not be altered by the node algorithm. The reason for propagating the values of these variables is either the collection of cut statistics or the fact that their values may be used to control the binding of the cut at locations, or under circumstances, specified by the modeller. An example occurs in modelling a message switching network, where a variable which might well be propagated would be the node number of the message destination. This would be operated on by a routing algorithm at each node and the result would determine the transmission line which would be allocated (i.e. binding is controlled by the result).

The method used for the LAMBDA variable can be extended indefinitely simply by the addition of carrier variables to the tie definition, and the addition of an appropriate node algorithm to produce the outgoing value.

In a general sense such variables represent the inclusion in the model of the variables of the real system. The reason for inclusion is that their values determine the behaviour of the real system sufficiently strongly to render the model inaccurate or even useless without them. If all variables are included then we end up with a replica rather than a model of the real system. Without them the model may not fulfil its purpose. The choice of variables to be included must therefore rest with the modeller. His judgment should be confirmed by a positive validation of the model.

CHAPTER IV

IMPLEMENTATION

4.1 General Criteria.

In the previous chapter we have described a system for modelling computational activity. This system was implemented as a program in the SIMULA language on a Control Data 6600 computer. The name of the program is SHAPE, which is an acronym standing for Software Hardware Allocation and Performance Evaluator. A brief introduction to SIMULA appears in Appendix II.

An equal emphasis was placed on the modelling of software and hardware to improve the evaluation of real performance. Furthermore, the basic interchangeability of hardware and software pointed the way to modelling and descriptive systems which were applicable to both, and minimised their differences.

Because software and hardware are regarded as similar and complementary, a correspondence occurs between the two. Basically this is the correspondence of store and dataset; processor and process, alternative connection and branching statements, parallel connection and concurrent processes, and so on. Wherever possible in the SHAPE system a single structure is used to model both hardware and software. The differences between them appear as different interpretations rather than changes in the structure.

For example the graphical representation is used throughout, the software interpretation being called a SIgraph, and the hardware one a PIgraph. This had led to nodes representing stores/datasets and arcs representing processor/processes. This seems a more useful graph model than earlier ones which have used the arcs only as a visual expression of the precedence relationships between computations. In these previous models the nodes were used to represent the computations, thus leaving the modeller with

no remaining structure to which datasets could be naturally ascribed. In the SHAPE system precedence relations are treated more explicitly from the point of view of data dependency, so that the computations which transform datasets are an inherent representation of these relations. A further consequence of the SHAPE interpretation of nodes and arcs is a simple expression of the binding situation at any moment by means of a cut across the graphs.

The SI and PI graphs used in SHAPE are of a general kind. There is no planarity restriction, arcs are allowed to have the same initial and terminal nodes, and multiple arcs between a pair of nodes are also permitted. The model has been provided with a recursive capability in order to allow areas of special importance to be investigated in greater detail, the submodel remaining embedded in the main structure as a subgraph.

In SHAPE, processor is used to denote any data-transforming piece of hardware, rather than a general purpose computer or Von Neuman machine. The reason for this is that it allows us to take into account specialized or restricted processors, and the great variety of special function hardware units which exist today, such as display controllers, multiplexors, disc controllers, etc. These must be modelled since they represent a dispersal of the intelligence and computing power of a utility, and can also be of considerable significance when overall performance is being considered.

A fundamental problem which arises in modelling a program is the representation of both the static and dynamic behaviour of the program. A static model of the program is one which shows the program as it might be written on paper, that is to say with all

paths, possibilities, and branches present. The dynamic model represents one particular execution of the program. A particular execution is obviously one where at each point of choice in the static model the choice has been made. Thus the dynamic model consists of a selection of the actions available in the static model.

In the case of SHAPE, a SIgraph shows all the possible computations which may take place during realization of the task represented by the graph. As the task is realized, as a binding of the SIgraph to a PIgraph, unselected alternatives are disabled. On completion the bound graph which remains gives us the dynamic model of that particular execution.

From this point of view branching statements are an online control device for programs, which allows the selection of alternatives to be postponed until the actual execution, and automates the process of selection (it is possible to imagine a very primitive program which referred the predicate data of every IF statement to the computer operator, who, flowchart in hand, would make the decision and then reactivate the machine at the appropriate instruction sequence).

Any attempt to model the execution of a task, and analyse the performance of that execution, must be able to handle this transition from static to dynamic representation. Some previous models have used branching probabilities, mean execution times, and so on to provide statistical results for overall execution measures. SHAPE allows for the use of these methods and also some others which are more data dependent, as well as making the insertion of predetermined decisions particularly easy.

The representation of IF statements was influenced by the fact that one of their more important functions is in the programming of loops. In SHAPE the loop entries and exits are handled by IF-type operations and the structure of these operations has been oriented to making loop representation as convenient as possible.

In the model as it stands today nearly all binding and allocation takes place as the cut crosses a node in the SIgraph. It is when this happens that nodes enter and leave the cut zone, nodes are bound to stores, and processors allocated to arcs. Consequently this is the area of prime interest in modelling the mechanisms which ensure continuing execution of the task.

It is intended that the SHAPE system will allow the trial of alternative binding strategies, and that the binding problems will be formulated in such a way that these strategies (that is to say the mechanisms mentioned above) can be easily inserted and removed. The problem is essentially that of optimizing the choice of m out of n processors to be allocated to m processes subject to various constraints (of course there may be fewer processors than processes as well). The optimization may be done for this choice alone, over the cut zone, or beyond the cut zone.

A recent result demonstrates the equivalence of preemptive scheduling and fractional allocation, [MUNT 70]. This leads us to expand the range of choice from integral allocations to fractional ones. The rationale for this is that optimization with fractional allocation seems far more amenable to solution than the corresponding situation with preemption.

The intention of the SHAPE system is to provide an evaluation of program realization for alternative allocation strategies, or to compare the behaviour of different realizations.

The main components of the SHAPE program consist of the graph input procedures, the allocator, and the procedures for matching and binding a PIarc to a SIarc. The binding of two graphs occurs in simulated time. A time scale is generated for each pair of graphs, so that where a pair of bound arcs have subgraphs these are bound in their own independent timescale, while that of the upper level is unaffected. This mechanism is used by the matching procedure to derive the time required for a PIarc to execute a SIarc when both have subgraphs.

A pair of bound arcs is called a <u>tie</u>. A tie is created by the allocator and exists for the duration calculated by the matching procedure. On terminating it activates the allocator which releases resources previously associated with the tie, and then creates ties for any processes now ready and able to execute.

The SHAPE program does not include all aspects of the model described in the previous chapter. This is due partly to limitations in the compiler and associated software (see Appendix III) and partly to insufficient time for programming a full implementation. The points of difference are described as they arise below.

In the following sections we describe the graph input procedures and then the operation of the allocator. After this we give a more detailed treatment of ties and IFloops, and then derive measures for hardware and software performance.

4.2 Graph input.

This section describes the way in which SI and PI graphs are input from a sequential storage medium such as cards or magnetic tape, to a random access medium such as core store. Such input is necessary because binding of the two graphs as performed in the SHAPE system, required the graphs to be in their topologically linked form.

In this form each node consists of a block of data about the node and a pointer to a chain of arcs. Each arc consists of a block of data about the arc and points to the next arc in the chain, and the terminal node of the arc; the initial node of all arcs in the chain is the one at the head of the chain, by definition.

The blocks of storage for the elements (arcs and nodes) of the graph may be situated anywhere in the available core store, and are linked by the pointers described above. The linkage so formed duplicates the topology of the graph. Clearly such a linkage can only exist in a random access type of storage medium, so that we have to provide a sequential form of the graphs for storage on sequential media. Such storage is desirable since we cannot keep the graphs permanently in core store, and private discs are not always available.

A normal SHAPE run will therefore be to input a SI and a PI graph from a sequential storage medium, set up the topological linkage, then perform the binding of the graphs, and output the results.

We shall now describe the topologically linked form of the SI and PIgraphs, starting with the structures common to both.

Both nodes and arcs have an attribute called POINTER. In the case of an arc this points to the next arc on the chain, and in the case of a node to the first arc on the chain. The arcs on a chain are called the OUTarcs of the node which heads the chain. Each node possesses three integer attributes besides its pointer. These are its node-number, the number of its INarcs, and the number of its OUTarcs (NODENUM, INARCS, OUTARCS).

Each arc has three other pointers besides that to the next arc. The first (NEXTNODE) points to the block of storage used to hold the data for the arc's terminal node, and the other two (FIRSTNODE, DX) are used when a subgraph exists for this arc. The first of these points to the first node of the subgraph, and the second points to the index for the subgraph. An index holds a double entry for each node in a subgraph. The entry consists of the node number, and its address in core store. Entries are ranked in order of increasing node number. An index also has its own length and that of the arc data vectors as attributes. An arc has one numerical attribute, SEQF. Entier (SEQF) is the node number of the arc's terminal node, and the fractional part of SEQF distinguishes between several arcs which have the same initial and terminal nodes. For example if there were three arcs between nodes 4 and 7, their respective values for SEQF might be 7.1, 7.2, 7.3.

The whole graph is referenced by a special arc called a graph header. In the graph header FIRSTNODE points to the first node of the graph and DX to its index. The graph header has two additional attributes which are the name of the graph, and its TYPE (SI or PI). The structure described so far is common to both types of graph.

In SHAPE we use SIMULA class definitions to provide arcs, nodes, indexes, graph headers, as shown in Fig. 4-1. Node and arc linkage is shown in Fig. 4-2. Where several arcs have the same fine structure only one subgraph is necessary and all the arcs will point to its first node and index. A subgraph linkage is illustrated in Fig. 4-3.

We now describe the data associated with nodes and arcs in SI and PI graphs, which depends on the type of graph. A PInode (representing a store element) has the following attributes: cost, latency, block size, blocks per track, and capacity. These are held in an array MU together with a random number seed for use in the generation of latency times. The run-time variables TOTUSE, FSTUSE, LSTUSE, INUSE, MOX, MUT, MEF, and MIT are used for gathering statistics during binding.

A SInode has as input data the repartition matrix (REP) described previously. In SHAPE the activity matrix (ACT) has not been implemented. Instead the allocator treats all datasets as read-write data (REP [i,j] initially positive), and the sign of REP [i,j] is used during binding to indicate its activity (negative for active, positive for inactive). Consequently all REP elements input to a SHAPE run are positive. One other data item of a SInode is the variable PNID which gives the node number of a a node in a PIgraph. If PNID is non-zero, then the SInode will be tied to the specified PInode during binding.

Binding time attributes are column vectors Q, LAM, BET, QD, QT, QS. During semi-reentrant binding for each row of the repartition matrix, the corresponding element of Q is the initial pointer to a queue of completed ties which have that row as their

```
        class  element (pointer) ;
        ref  (element) pointer ;;
element Class  arc (nextnode, firstnode, dx, seqf);
        ref (node) nextnode, firstnode;
        ref Iindx) dx;
        real seqf;;

element class  node (nodenum, inarcs, outarcs);
        integer  nodenum, inarcs, outarcs ;;

arc     class  graph (graphname, type);
        value  graphname;
        text  graphname;
        integer  type;

        class indx (indxl, adle);
        integer indxl, adle;
        begin
        integer array number [0:indxl] ;
        ref (element) array  address [0:indxl] ;
        number [0] :=1;
        number [indxl] :=indxl
        end;
```

Fig. 4-1 Node and Arc Class Definitions.

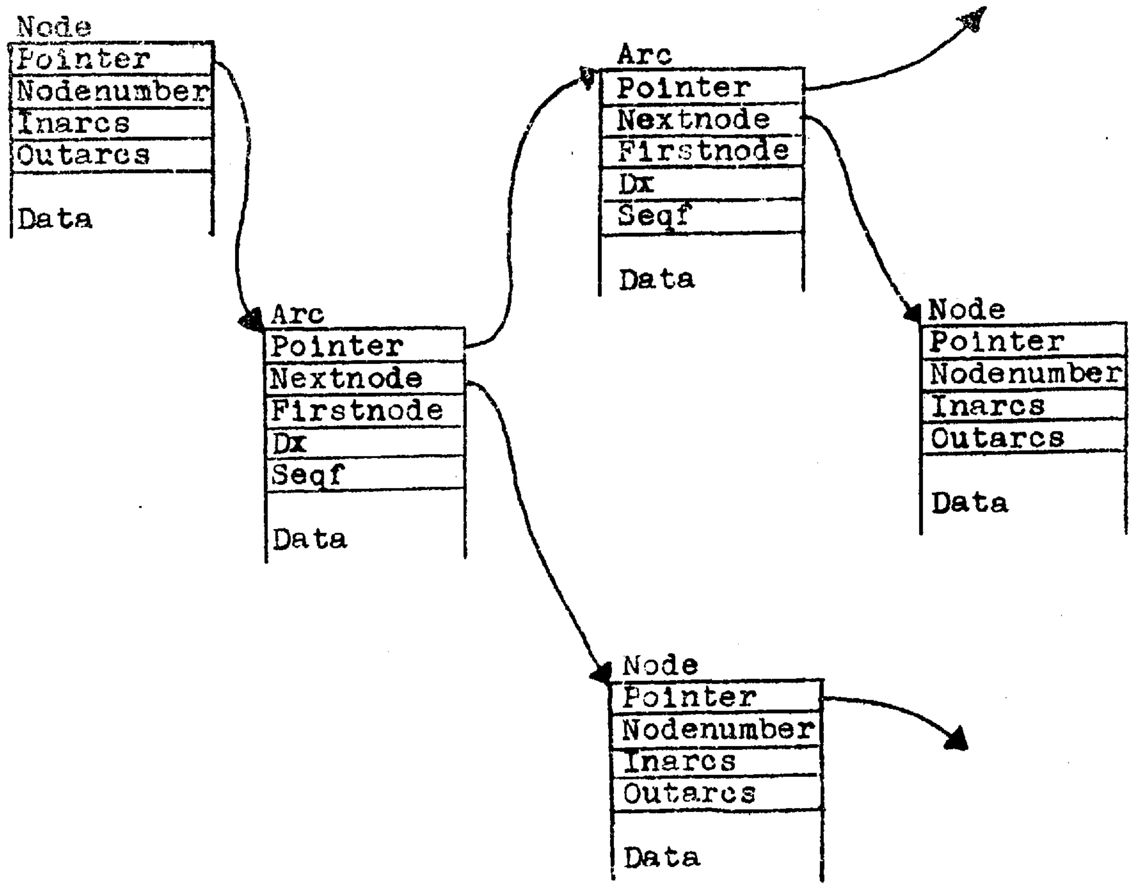


Fig. 4-2 Node and arc linkage.

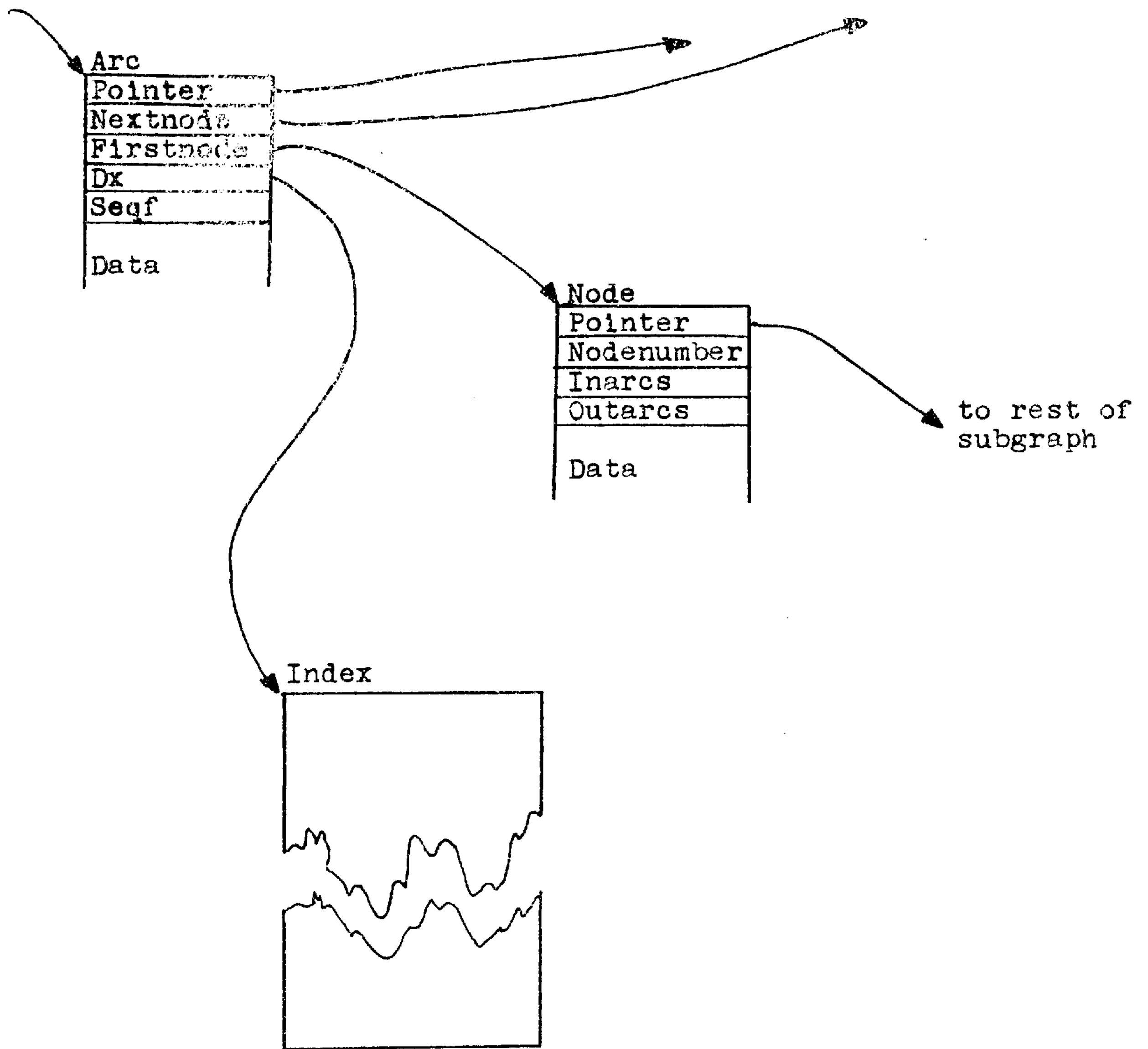


Fig. 4-3 Subgraph Linkage.

terminal data. The variables LAM [i] and BET [i] always contain the values of lambda and beta for the current activation of row i. QD, QT, QS are used for the collection of software statistics and are described in greater detail below.

The input data of a SIarc consists of two arrays PHI and IFF. PHI is the function frequency vector and has n elements where n is the number of software functions which characterize the graph. PHI [i] is the frequency of the i th function in the SIarc. N is also called the arc data length (equal to attribute ADLE of the graph index).

IFF is an array that has zero elements unless the SIarc is an IFloop. In this case IFF contains a random number seed, two ifcodes and their four parameters. A binding-time attribute of a SIarc is SFRAC, which is used to hold the fraction devoted to this SIarc of the currently allocated processor. The array STARC is used to accumulate software statistics as well as providing a counter for use by IFloops.

A PIarc has as input data the processor performance array PSI which consists of three vectors each with n elements. The first vector gives the times taken by the processor to perform the n software functions characterizing the graph, and the second and third give the processor utilization and efficiency for these functions. Other data attributes are the physical identifying number of the processor (ID) and the cost per unit time of using the processor. The attribute ID is required since many arcs in a PIgraph can refer to the same physical processor, which has the capability of reading from and writing to many memories. The run-time variable PFRAC provides the fraction

of the processor which is currently allocated, and six other variables are used for statistical purposes. SIMULA class definitions for the node and arc data structures are given in Fig. 4-4, and 4-5.

On a sequential medium SI and PIgraphs are stored as sequences of card images. Each element of the graph (node or arc) consists of a set of cards. The first card of the set uses columns one to twenty to define the element in the topology of the graph. The rest all have columns one to twenty blank. All fields consist of ten columns, and a card may have up to seven fields.

The first twenty columns mentioned above are the first two fields of the card. A node has its node number in field one, and field two is blank. An arc has the node numbers of its initial and terminal nodes in fields one and two. If an arc has a subgraph the remaining fields of its first card (arc card) contain information about the subgraph. This consists of the numbers of the first and last nodes, the number of elements in the function vectors (arc width), the number of nodes in the graph (graph size), and a factor which determines the size of the index relative to the number of nodes (GFACTOR). A graph header card also has this information about its graph.

If an arc has a subgraph, then the data cards for the subgraph immediately follow the set of cards for the arc. A graph header card has the name of the graph and its type in fields one and two. The card image formats are shown in Appendix IV, together with detailed description of all array usage.

```
node    class pinode (mu);
        real array mu;
        begin
        real totuse, fstuse, lstuse, inuse, max,
             mut, mef, mit;
        ref (sinode) pstie;
        integer u;
        u := mu [4] ;
        end;

node    class sinode (rep, pnid);
        value rep;
        real array rep;
        integer pnid ;
        begin
        ref (transaction) array q [0: inarcs] ;
        real array lam, bet [0: inarcs] ,
                   qd, qt, qs [0: inarcs, 1:4] ;
        ref (pinode) sptie;
        integer active;
        end;
```

Fig. 4-4 Node Class Definitions.

```
arc     class  siarc (phi, iff);
        real array phi, iff;
        begin
        real array store [0:7] ;
        real sfrac;
        integer ug;
        ug := iff [0];
        end;

arc     class piarc (psi, psid);
        real array psi, psid;
        begin
        real array phicap [0: psi [0, 1] ];
        real putpr, putmx, putav, pefpr, pefmx, pefav,
             pfrac;
        end;
```

Fig. 4-5 Arc Class Definitions.

The sets of cards for the graph elements are ordered as follows. Each node is followed by all its OUTarcs. The nodes may come in any order. The first card of the deck should be the graph header card, which provides information required by the input routines.

The type of a graph is an integer which gives the number of function vectors which appear in the arc data. Consequently a SIgraph is of type 1 and the PIgraphs currently used are of type 3.

The graphs are input by a set of procedures in the way described below. Firstly, the graph header is read, and an object of this type is generated. Then a procedure called SUBGIN is executed using some of the information from the graph header. These actions are performed by procedure GIN. The procedure GIN has one parameter (G) which references the graph header after GIN has been called to input a graph. The procedure SUBGIN inputs a subgraph. The highest level of a graph is regarded as being a subgraph of the graph header.

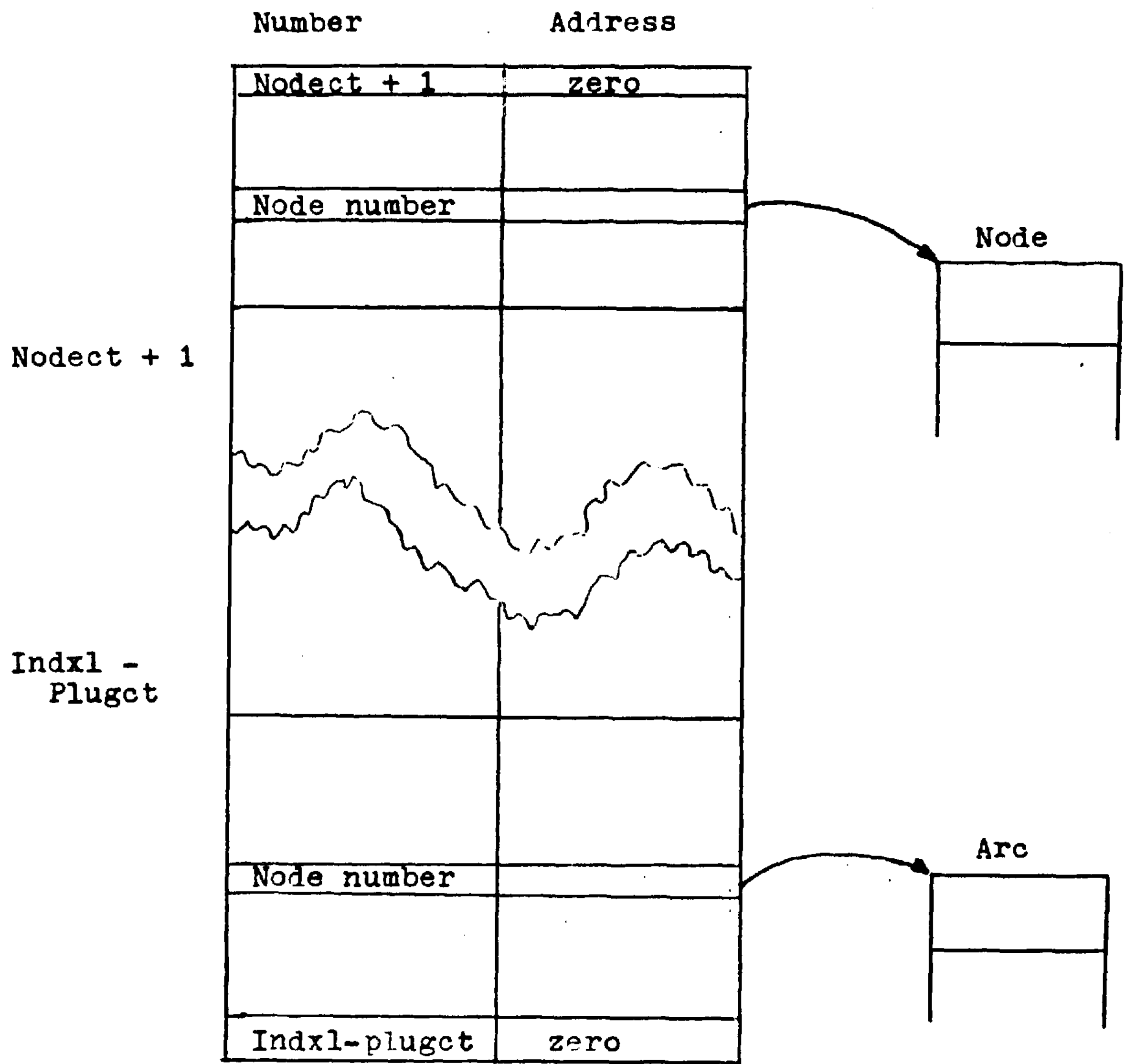
SUBGIN sets up the index for the subgraph being input. The number of entries is the size of the graph (number of nodes) times GFACTOR. Also set up is a scratch array for arc data. After this the procedure INNODE is called a number of times equal to the graph size. When the first node is found, it is linked to the arc

(or graph header) which heads the subgraph. This arc also has a pointer to the index for the subgraph.

The procedure INNODE creates a node of the appropriate type and enters its data. An entry is created in the index for this node which gives its number and address. The procedure INARC is then called a number of times equal to the number of OUTARCS of the node.

The procedure INARC reads in the data for one arc and then creates an arc object. The pointer from the arc to its terminal node is created by searching the index for that node number and thus accessing its address. If the terminal node has not yet been read in from the sequential file no entry will be found. In such a case a plug is created in the free space area of the index. The plug consists of the node number in question and the address of the arc requiring its address. When this node is read in, the procedure which enters it in the index also satisfies all the plugs requiring its address (see Fig. 4-6).

The arcs are chained as follows. Each call of INARC has a pointer to the arc created by the previous call as one of its parameters (in the case of the first call, by INNODE, this pointer points to the node at the head of the chain). This allows the linkage to be established from the previous to the currently created arc. After linking, the pointer is updated to point to



<u>if</u> nodect + 1 = indxl - plugct <u>then</u>

indexfull := <u>true</u>;

<u>comment</u> nodes are held in ascending order by

node number;

Fig. 4-6 <u>Index usage for graph input.</u>

the current arc, and so is ready for the next call, if any. The end of the chain is indicated by a null pointer.

If INARC finds that subgraph information occurs in the arc card, then a call to SUBGIN is made. This call will then input the subgraph which follows the data cards of the arc. In this way the process of subgraph input operates recursively.

Several arcs may have the same subgraph. In this case only one need be followed by the subgraph card deck, and the others may give the initial and terminal node numbers of this arc. Such a provision leads to a plugging mechanism for subgraphs similar to the one for nodes described above.

In this way the input procedures of the SHAPE system creates a topologically linked data structure of the type described earlier from a sequential file of card images. A list of procedures used is given in Appendix IV. It is often the case that several nodes or arcs have identical data. To allow the modeller to specify such replication compactly, rather than having to repeat the complete data each time, some facilities for data replication are included in the graph input formats.

If a node Q has identical data to a previous node P

then the nodenumber of P appears on the first card of the node Q description, after the number of OUTarcs. The rest of the data is then dispensed with, and picked up from node P by the graph input routines. In fact only one copy of such data is kept, and this is referenced by all the nodes to which it applies. For all arcs the parameter SEQF is held separately on the second card of the arc data. The IFloop parameters are held on the third card. For arcs (r,s) which have the same data as arc (p,q) the nodenumbers of p, and q, and SEQF of the arc (p,q) can be placed after SEQF on the second card of the data of arc (r,s). If this is so, no data follows, and the graph input routines link the arc (r,s) with the data of arc (p,q).

Each run of the SHAPE program is controlled by a run card which is the first data to be input. The card contains the following items: the number of graphs for this run (one or two), the binding mode, the debug parameter, codes specifying the type of hardware and software statistics required, and the binding time limit if any. The run card parameters are described in Appendix IV.

The facility for data replication is also used to provide a mechanism for interlocking the various states of a single physical processor. An interlock is needed primarily for allocation, so that the fraction of the processor allocated is always known by referencing a single variable, and can be altered by only one process at a time. To achieve this, all states (PIARCS) of a single physical processor have an attribute PSID which is a one dimensional array. There is only one copy of this array, and it is this which is accessed irrespective of which PIarc is being dealt with. The zero element of the array holds the fraction of the physical processor currently allocated, and thus an automatic interlock is provided. The remaining elements of the array are used for statistical purposes.

A procedure called TOPSCAN is also provided for use with the graph input routines. This procedure performs a topological scan through the graph listing the linkages which it finds. Its purpose is to check that the graph input routines have functioned correctly

before binding is initiated.

4.3 The allocator.

The current version of the SHAPE system performs binding of SI and PIgraphs using a SIMULA object of class allocator. The allocator has been constructed as a class definition since it is regarded as controlling the execution of a single cut or connection between the two graphs. Use of the class definition allows the generation of more than one allocator, the retention of local data describing the condition of its cut by each allocator, and the convenient use of SIMULA simulation facilities.

The execution rule (or block) of the allocator is prefixed by the predefined class SIMULATION, so that each allocator generated is effectively an independent simulation (system of quasi-parallel processes). In what follows some knowledge of the programming language SIMULA is assumed.

Within the execution rule a process of class tie is defined. This process is used to represent the allocation of a process (arc of a SIgraph, not simulation process) to a processor for a given period of time. During this period the process and processor are said to be tied.

When a tie is completed the allocator is called to free resources, update the cut status and initiate ready processes by binding to appropriate resources. An allocator operates only for one subgraph. When an arc is found which itself has a subgraph, the procedure which matches processes to processors generates a new allocator to provide the results of the matching.

In a real computer system binding is done either by hardware or software. If by software then this software requires at least intermittent use of system hardware.

Thus in the real system the resource allocation (binding) mechanism itself requires some of the resources it allocates. The one exception is the case when the hardware involved is special purpose hardware which cannot be used for any other activity. This case will be termed free resource allocation for obvious reasons (the resource allocation hardware is of course only free from the point of view of the allocator, for its purchased it is a resource permanently assigned to the allocator which is treated as another process). Where the resource allocation mechanism uses only a very small proportion of the resources it allocates, then it may be thought of as free.

In a real system resource allocation is performed, for example by various procedures in the operating system by the control unit of the central processor (this is very low level), by the control elements in a multi-plexor, and so on. These are the real analogs of the SHAPE allocator.

We note that different levels of task execution have different allocation mechanisms. This is reflected in the SHAPE system by generation of a new allocator when a subgraph is encountered. The allocator in the SHAPE system is a supervisory algorithm which advances task execution, as represented by the binding of SI to PI graphs, by reallocation of resources as various elements of the task terminate.

The allocator of the SHAPE system corresponds to the algorithm K, and the procedure match to the procedure S, which are described in the previous chapter. The general structure of the allocator is shown in Fig. 4-7.

```
class       allocator (parameters);
simulation  begin
            process class tie (params);
                    begin hold (duration of tie);
                    activate allocator after current;
                    end;
            release resources of completed arc;
            update cut status;
            determine number of arcs ready to proceed;

            for s:=1 stop 1 until number ready do
            begin

            for p:=1 stop 1 until processors available do
            begin
            match (s,p);
            if better match then save (s,p);
            end;

            activate new tie (s, best p);
            allocate resources;

            end;
            end;
procedure   match (s,p);
            begin
            if s ≡ subgraph then activate new allocator
                            else simple match;
            provide analysis of matching;
            end;
```

Fig. 4-7 Outline of class allocator.

This outline shows that the binding process is recursive in that it can deal with subgraphs nested to an arbitrary depth. In the SHAPE programming system a cut is represented by the set of all ties in the sequencing set of the corresponding allocator's simulation system.

Referring back to the previous chapter, we note that a cut consists of all arcs of a SIgraph which are currently being executed together with their initial and terminal nodes, and all elements of a PIgraph which are tied to these arcs and nodes. Each object of class tie represents the execution of a SIarc, and includes a pointer to this SIarc and the PIarc tied to it by the allocator. Each tie also has pointers to the initial and terminal nodes of its SIarc and PIarc. In this way the set of ties corresponds to the set of active arcs and their nodes, i.e. to the cut zone.

When a SIarc completes its execution the elements of the terminal nodes' repartition matrix which represent its output datasets are marked as active (set negative). The allocator then examines the updated matrix to see whether any of the OUTarcs now have all their initial datasets active. If this is the case such an arc is ready to proceed.

The actions and constraints involved in binding such an arc fall into two categories. The first category is the constraints, and consequently decisions, which can be derived directly from the nature of the SHAPE model.

The second category consists of decisions made between alternatives equally acceptable from the point of view of the model. The algorithms which make these decisions taken together form a

resource allocation strategy. For the prototype SHAPE system to operate some such strategy was required, and in fact was provided as a minimal set of simple rules. It should be emphasized that these rules are arbitrary, can be changed at will, and thus provide opportunities for investigating different strategies of resource allocation.

We shall now examine the detailed operation of the prototype allocator. The allocator parameters are shown in Fig. 4-8. These enable the initial conditions to be set up and the datasets of the first SInode to be activated by the first call of the allocator. We shall now follow a typical iteration commencing after a tie has terminated.

The allocator has six reference variables which point to the PIarc and SIarc of the tie which has just completed, and to the initial and terminal nodes of these arcs.

P - PIarc

PIN - initial node of PIarc

PINN - terminal node of PIarc

S - SIarc

SIN - initial node of SIarc

SINN - terminal node of SIarc

These pointers are set by the execution rule of the tie, just before it terminates, using an inspect statement. In this way the allocator is aware of the elements of the completed tie on entry.

The allocator uses a number of Boolean variables to give information about the tie, and later on in activating new ones. These are as follows.

```
class  allocator (fpn, fsn, pdx, sdx, t, gutmx, gutav);

fpn - pointer to first node of PIgraph.
fsn - pointer to first node of SIgraph.
pdx - pointer to index of PIgraph.
sdx - pointer to index of SIgraph.
t   - eventually holds total time to execute graph.
gutmx, gutav - performance measurement variables.
```

Fig. 4-8 Allocator Parameters.

PERT - true if no PIgraph, SIgraph evaluated as an activity graph with arc duration given by sum of elements of vector PHI.

FERST - true if allocator called to first node of graph. Condition is PIN = = SIN = = none.

LARST - true if allocator called to a terminal node. Condition is that the terminal SInode indicator of the first OUTarc be zero, i.e. SINN • REP [0,1] = 0

IPH - true if tie was an IFloop. This is a tie whose initial and terminal nodes are the same. The condition is SIN = = SINN and S = / = none.

DLAY - true if tie was a delay loop. This is an arc which is used when the allocator finds a ready arc at a SInode, but cannot bind it because there are no resources available. In this case a delay is activated to ensure that the allocator is called to this node at some future time to attempt to bind the ready arc again. The delay loop is like an IFloop, but not tied to any PIarc, and has no SIarc. The condition is SIN = = SINN and S = = none.

complete - set true if initial SInode has no active datasets (ACTIVITY = 0) after tie terminates

The first part of the allocator deals with the freeing of resources used by the completed tie. The repartition matrix (REP) of the initial SInode (SIN) is accessed, and the column for this SIarc (S) is found. The zeroth element of the column (REP [0, j]) is the terminal node indicator, and by this the column can be identified. The indicator value is the same as the sequence fraction of the SIarc.

Having found the column in the repartition matrix of the initial node which corresponds to the completed SIarc, the allocator proceeds to deactivate the datasets shown in the column as active (all if a normal OUTarc, one if an IFloop). At the same time the number deactivated is counted and their size (amount of memory required) is summed.

After this the active count for this SInode is decremented by the number deactivated, and the amount of memory in use in the tied PInode is decreased by the sum of the dataset sizes. Each SInode has a reference variable SPTIE which points to the store (PInode) in which the datasets of the SInode are resident. When there are no datasets active SPTIE has the value none, and the SInode is not tied. If the active count falls to zero, then the allocator sets SPTIE :- none.

The allocator now deals with the processor (PIarc) to be freed. Since a processor may transfer information between more than one pair of stores, we allow each such state to be represented as a separate PIarc in a PIgraph. These states all represent the same physical processor however, and so it is convenient, for resource allocation purposes, to know what fraction of a processor is in use, as a sum over all states. Each processor is given an identifying integer (ID) which stays constant over its states, i.e. every PIarc representing a state of a processor will have the same value for ID. This is held in an array PSID which is common to all states of the processor. This array is made common by declaring the corresponding variable as accessible by reference rather than value during PIarc generation by the graph input procedures. The first element of the array PSID contains the sum over all states

of the processor fraction currently allocated. The remaining elements are used for measurement purposes.

The attribute SFRAC of the tied SIarc gives the fraction of the processor which was allocated to this SIarc. The allocator will subtract this from the particular PFRAC attribute accessed by the method described above,and also set SFRAC to zero.

For a discussion of fractional allocation see section 4.4.

At this stage the allocator accesses the repartition matrix of the SIarc's terminal node and searches for the row which describes the terminal datasets of the SIarc. The zeroth element of each row (REP [i,0] is the initial node indicator. The indicator value is the node number of the initial SInode of the INarc, plus the sequence fraction of the SIarc, minus the node number of the terminal SInode.

The terminal datasets (non-zero elements of the row) are activated by making them negative. If the completed SIarc was an IFloop then only one element is activated. The column in which this element occurs is given by the allocator's local variable IFCOL. IFCOL is preset by the tie before it calls the allocator.

We can provide for more than one SIarc to activate the same row of a REP matrix by noticing that the allocator, when searching for the correct row of the REP matrix to activate, tries to match the INarc specifier to the following expression,

SEQF - SNN + SN

where SNN is the terminal nodenumber, SN is the initial node number, and SEQF is the sequence fraction of the incoming arc. Row i is selected as representing the terminal datasets of the INarc if

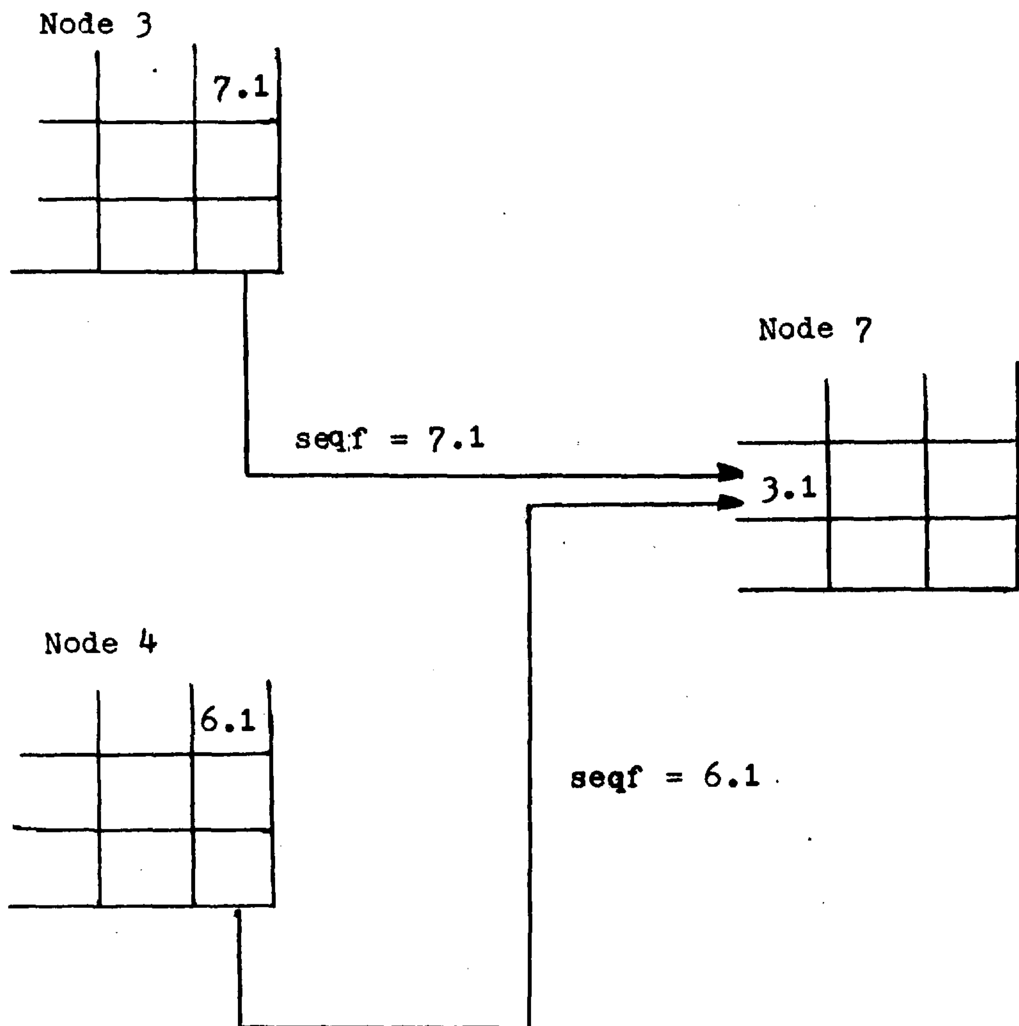
REP [i,0] = SEQF - SNN + SN

We usually require that entier (SEQF) = SNN, and that entier (REP [i,0]) = SNN, in which case SN is the official initial node for row i. However it is clear from the above expression, that if we wish to activate the row i by an INarc from some other (unofficial) node, we can do so as long as SEQF + SN has the same value as before. That is to say that row i of the REP matrix can be activated by an INarc from any SInode in the SIgraph so long as the value of SEQF of the INarc is suitably chosen. An example is shown in Fig. 4-9.

When the stores (PInodes) which hold the datasets of the initial and terminal nodes of the SIarc are different, storage allocated to the initial SInode is released. Storage allocated to the terminal SInode is not, since the terminal datasets of the SIarc must continue to exist, being the initial datasets of subsequent arcs.

When the stores are the same (this occurs if the SIarc is an IFloop, or if the initial and terminal SInodes have been allocated to the same store) the allocated storages of the initial and terminal nodes are regarded as being superimposed. Thus storage is only released if the initial allocation is larger than the terminal requirement.

This last is an allocation strategy and not a constraint of the model. It was chosen since it corresponds to the strategy followed by most operating systems. For example, when storage is allocated for execution of a FORTRAN program, terminal variables, i.e. ones which are used to hold results and have no initial value, are included in the initial allocation. If these results are to be preserved for a subsequent execution phase,



Node 3 is the official origin node for row one of Node 7.

Fig. 4-9 Multiple INarcs.

then only excess storage is discarded on completion of the program.

SHAPE does not require a model to contain a terminal node. If part of the cut is extinguished (for example by the terminal dataset of a SIarc having only zero elements), then the cut need not terminate if new active datasets are being generated elsewhere. It is possible to generate a fixed number of these using a DOloop, but a more flexible alternative was provided for SHAPE. This was the possibility of indefinite generation of active datasets, and termination of binding when a given time limit (variable from one run of the SHAPE program to the next) was exceeded.

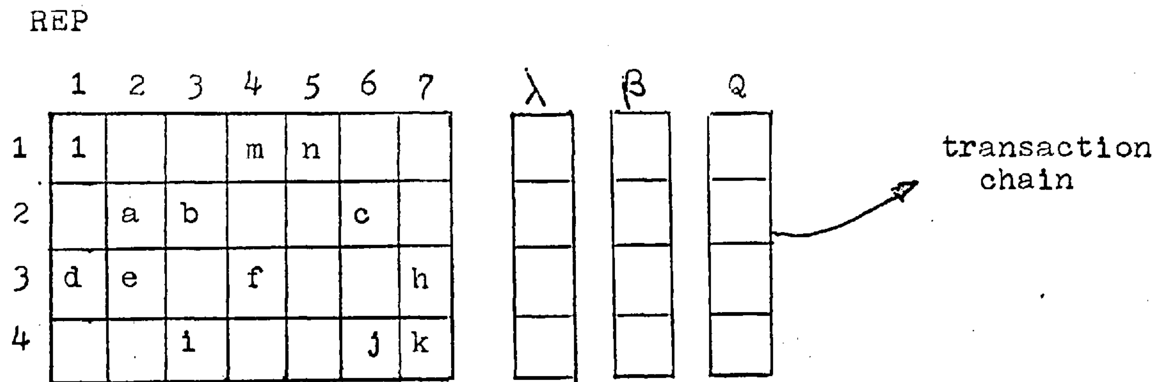
An additional feature is provided in SHAPE, namely multiple termination. A terminal node is one which has an OUTarc to node zero. This OUTarc is notional since node zero does not actually exist, and is represented by a column of the REP matrix which has its OUTarc specifier equal to zero. Clearly such a column can exist in more than one node of the SIgraph, thus allowing for the representation of more than one binding termination. When such a column becomes ready (all datasets active) it is immediately deactivated to represent the instantaneous execution of the OUTarc to node zero.

If there are then active ties still present, the cut is not considered to have terminated. That is, the presence of the ties implies there is further binding to be performed, and the allocator continues with this as usual. The same column may become ready again, at a later stage of binding, and the termination procedure repeated. This repetition will continue so long as active ties remain in the system, and is called multiple termination.

In SHAPE non-reentrant and semi-reentrant binding (modes 1 and 2) have been implemented. Completely reentrant binding is described in the preceding chapter, as are the differences between modes 1 and 2. From the point of view of the allocator there are two important distinctions, firstly that in mode 2 a ready SIarc may be allocated even if one or more of its terminal datasets is still active, and secondly that a completing tie may find a terminal dataset active, in which case it is queued. If such a tie is queued, it is called a transaction and has three main attributes, LAMBDA, BETA, and IFCOL. These are the LAMBDA, BETA and IFCOL of the completing tie. Transactions are chained and the head of the chain for each INarc of a REP matrix is pointed to by the corresponding member of a reference array Q, whose dimension is equal to the number of INarcs. The chains are processed on a FIFO basis.

The methods of queueing and reactivating the REP matrix by bringing in queued transactions are governed by the principle of keeping successive cuts distinct and allowing no interaction between them. Sequence is maintained by FIFO queueing, and separation by ensuring that a transaction is brought into a REP matrix only when the columns containing the elements it will activate are all inactive. An example is shown in Fig. 4-10.

When a tie terminates in mode 2, as it deactivates its initial datasets it also scans the rows of the REP matrix which contain those datasets. If it finds that such a row has no active datasets and there is a queued transaction for the row, it will bring in the transaction.



Transaction at head of Q[2] activates datasets a,b, and c.

Fig. 4-10 Transaction entry to REP matrix.

Suppose column two in Fig. 4-10 completes, then datasets a and e will be deactivated, and rows two and three scanned for activity. If datasets b and c are inactive the row is available for a queued transaction. If Q[2] is not null, the transaction at the head of the chain will be brought into the REP matrix. That is to say that datasets a, b, and c will be activated and the transaction values for LAMBDA and BETA will be inserted in the array elements LAMBDA [2], and BETA [2]. If IFCOL is greater than zero, then only the dataset for that column is activated, e.g. if IFCOL of Q [2] is equal to 6 then dataset c only, will be activated. In such a case the row scan does not require a and b to be inactive.

Separation of successive cuts is ensured, since we know that all the columns of the REP matrix affected by bringing in a transaction belonging to cut n + 1 have completed execution in cut n. Other columns may still be executing in cut n, but cannot interact with cut n+ 1 since they have no elements in common with the activated row, e.g. any of datasets e, d, m, f, r, h, k, may still be active when a transaction is brought in on row two. Similarly, should a tie whose INarc terminal datasets are represented by row two, find any of a, b, c, active on its termination, it will be placed in the queue defined by Q [2].

The number of datasets which are currently active in a REP matrix is called its activity. In mode two the activity includes the number of transactions queued at the node. If queues only are considered, activity is equivalent to queue size. If the REP matrix has only one element per row, the row is analogous to the server of the corresponding queue. When the activity at a SInode falls to zero there can be no storage requirement, and

consequently the SInode is freed from the PInode to which it was bound. This allows a subsequent reactivation of the node to bind it to any acceptable PInode.

Finally we add a postscript to reentrance restrictions where IFloops are involved. When an arc is allocated in mode 1, the allocation is allowed only if its terminal datasets are inactive. However, if any terminal dataset belongs to a column which is an IFloop, a further restriction becomes logically necessary. The aim of both restrictions is to prevent a SIarc being allocated while any columns of its terminal node which contain its terminal datasets are active. This ensures non-reentrant execution. As long as the column is not an IFloop, the first restriction is sufficient . If it is an IFloop, we bring in the further restriction that no dataset of the column may be active, as otherwise the IFloop might still be active due to a dataset in another row to the terminal row of the arc being allocated. Similarly in mode 2, a terminating tie is queued if it would otherwise activate a dataset in the column of an already active IFloop. With this rule some illegal side effects are also avoided which can arise when the IFloop is a DOloop. This ends the section of the allocator which deals with the freeing of resources.

We now describe the section of the allocator concerned with activating new SIarcs. Having processed a completed tie, the allocator examines the terminal SInode of the tie to see whether the activation of the terminal datasets provides any OUTarc of the SInode with a complete set of active initial datasets.

Effectively this means scanning the repartition matrix to see whether any column has all its non-zero elements negative. If the column represents an IFloop, it is sufficient to find one negative element. During the scan, whenever an OUTarc is found to be ready, an attempt is made to allocate a processor to it, and to allocate any storage required by its terminal node.

For each OUTarc (column of REP) the Boolean variables TIED, POSSIBLE, NOPE, PCAN and IPPH are used. These are initially set to <u>false</u>. If the OUTarc has the same terminal and initial node, then IPPH is <u>true</u>. The SHAPE system includes SIarcs which require no processor. Since matching of PIarc to SIarc is based on the function vector PHI of the SIarc, we allow arcs to have all elements of PHI equal to zero. We interpret this as a statement that the SIarc requires no hardware functions, therefore no processor. Such SIarcs are allocated as usual, except that they are not bound to any processor. Clearly they are of zero duration, and hardware dependent only for terminal dataset storage. In every way they are treated as regular SIarcs, and provide a convenient method of treating aspects of a model which are time or logic, rather than hardware, dependent. If such a SIarc is found by the allocator scan, the variable NOPE is set <u>true</u>. If the OUTarc can be executed (this is determined during the attempt to allocate resources to it), then POSSIBLE is set to <u>true</u>. If the OUTarc's terminal node is tied to a store, then TIED is set to <u>true</u>. If the SIarc is not only executable, but the appropriate resources are available, then PCAN is set <u>true</u>. If LARST is <u>true</u> and a ready OUTarc is found, this signifies that all the datasets of the last node have been completed.

Consequently the graph (or subgraph) is complete, and the allocator exits to its own completion procedure.

On finding a ready OUTarc the allocator attempts to allocate the resources it requires, and activate it, as follows. First the allocator searches down the chain of SIarcs from the node under consideration to reach the data block for the ready arc. This block holds a pointer to the terminal node of the arc, which is read to the variable SINN. The variables SIN and S already hold pointers to the SInode being scanned and the ready arc respectively. The variable PIN holds a pointer to the store to which the SInode is tied.

At this point we enter the hardware allocation loop of the allocator. This loop is traversed for each ready OUTarc found in the scan. For a processor (PIarc) to be able to execute the ready arc it must be able to read from the store to which the SIarc's initial node is tied, since it is in this store that the SIarc's initial datasets reside. That is to say we must restrict ourselves to OUTarcs of this store.

The allocator accesses the terminal SInode's repartition matrix and calculates the quantity of storage required by the SIarc. In the case of an IFloop the storage required is the size of the largest dataset which could be selected by the IFloop. A restriction introduced here is that the size of the largest dataset may not be greater than the size of the initial dataset. This is not a constraint of the model; the reason is that an IFloop is regarded as performing a test on its initial dataset, and consequently choosing an alternative rather than creating any new data.

For generality the implementation provides for the case where an IFloop has its first IFCODE (this code governs the selection of the IFloop's terminal dataset) set to zero. This condition is interpreted as meaning that though the SIarc has the same initial and terminal nodes, it is to be treated as a normal SIarc and all the datasets of its terminal row are activated.

The SHAPE implementation has the property that a SIarc which is active and allocated (tied to a PIarc and executing) is automatically protected from further (erroneous) allocation. This could occur since all elements in its column of the REP matrix of its initial node remain negative while the tie executes. Should the allocator scan such a column it would appear ready and consequently a candidate for allocation. However, on completing a tie the allocator scans only columns which contain a dataset activated by the completion of the tie. Such a column could not have been previously ready (and also, therefore, not previously allocated) since at least one of its elements was inactive. This ensures that any OUTarc allocated by the allocator has become ready on that call of the allocator and is therefore not already allocated. The exception to this is the case of delayed columns, but these are known to be unallocated since their OUTarc specifier is set negative. In brief, if a terminating tie activates row i of its terminal REP matrix, then the allocator scans only columns j for which REP [i,j] < 0, and columns which have been marked as delayed in the manner described below.

The allocator now chains down the OUTarcs of the initial PInode performing the following tests. If the ready arc's terminal node is tied to a store, a check is made that this is

also the terminal store of the PIarc. If not, the PIarc is not considered.

If the terminal SInode is not tied, the PIarc's terminal store is checked to see that its capacity is sufficient to provide the maximum storage the SInode may require. This restriction is not a constraint of the model; it is an allocation strategy aimed at preventing system deadlocks. If the restriction is not satisfied, then the PIarc is not considered.

If the SIarc is an IFloop, then the PIarc's terminal store must be the same as the initial one, since all datasets of a SInode must reside in the same store. The model provides the facility to specify that a SInode be tied to a specific store of a PIgraph. Each SInode has an attribute PNID. If this is non-zero, the allocator will only tie the SInode to a PInode whose node number is equal to PNID. As each node must be uniquely numbered, there will only be one such node in any graph. Use of this facility requires that the SIgraph be used with PIgraphs known to have appropriately numbered nodes, decreasing the independence of the team description.

If all the above tests have been successfully negotiated, the allocator will now proceed to assess the performance of the processor in executing the SIarc we have been dealing with. This it does by calling procedure MATCH. Procedure MATCH requires pointers to the two arcs, and the length of the performance vectors, as parameters.

It provides in return the time the processor will take to execute the SIarc, together with certain measures of performance of such an execution.

If the processor is incapable of executing this SIarc, MATCH returns a negative value for the execution time.

MATCH derives its results from the software function frequency vector of the SIarc (PHI [i]), and the three performance vectors (PSI [i,1], PSI [i,2], PSI [i,3]) which give the time used the utilization, and the efficiency in execution of the i th function.

If MATCH finds that the SIarc has a subgraph then it checks that the PIarc being matched also has one. If not, an error is logged. Otherwise MATCH generates a new allocator to bind the two subgraphs, and thus provide the required performance measures. Control passes to this allocator and remains there until this sub-simulation is completed. MATCH then extracts the results it needs and exits back to the original allocator.

Here we check the time provided by procedure MATCH. If positive, the Boolean variable POSSIBLE is set to true. The allocator then checks that the processor's terminal store has sufficient storage available to accomodate the terminal datasets of the SIarc. It also checks that the processor or a fraction thereof is set free to be allocated. If both these conditions are satisfied, PCAN is set to true and the allocator proceeds to compare the performance measures of this processor with the best found to date. If the comparison is favourable the new processor replaces the old as the best choice for this SIarc.

At present the comparison is made on the time taken to execute the SIarc. The reasons for this strategy (again such a choice is not a constraint of the model) are as follows.

The performance measures currently in use are not definitive. One of the purposes of the prototype system is to examine their validity. Their use in allocation decisions would distort the behaviour of the system and therefore severly interfere with any such assessment. The choice of execution time as an allocation criterion is prompted by its frequent appearance (sometimes implicit) in existing systems, and by its widespread use as the variable to be optimized in theoretical treatments of processor allocation.

The algorithm used for obtaining an OUTarc LAMBDA from the LAMBDA values of its INarcs makes the new LAMBDA equal to the scalar product of the OUTarcs REP matrix column and the LAMBDA vector, that is,

$$\lambda_j := \sum_{i=1}^{INARCS} REP\,[i,j]\,\lambda_i$$

This means that the LAMBDA value of a tie now gives the total amount of data being processed by the tie. This allows the modeller to specify the quantitative aspects of data repartitioning, and to incorporate absolute quantities as well as relative ones.

The derivation of an OUTarc BETA from the BETA values of its INarcs will depend on the interpretation given to the variable BETA. This was introduced as a modelling aid for the collection of cut statistics. It is expected to be used mainly to record generation times for cuts or parts of cuts, and so the following algorithm was chosen as being the most useful for such recording.

$$\beta_j := \underset{i}{Max}\,[e_{ij}\,\beta_i]$$

where $e_{ij} = 1$ if REP [i,j] $\neq 0$ and zero if REP [i,j] $= 0$
With this algorithm cut age is regarded as being the age of the youngest cut member, in the event that more than one age is produced, and allows BETA to record the most recent value produced by a SIgraph specified change.

The duration of a tie is now LAMBDA * T where T is the execution time of the SIarc per unit data. This is the T provided by the procedure MATCH, and is adjusted to reflect the fraction of processor allocated to the tie.

The allocator performs the steps outlined above for each PIarc on the chain of OUTarcs of the initial PInode. On reaching the end of the chain the best choice, whose address and characteristics have been saved, is allocated to the SIarc.

This is done by setting the attribute PSID [0] of the PIarc to the previous fractional allocation plus the fraction currently being allocated. Any storage required for the terminal datasets of the SIarc is allocated and the change recorded. Finally an object of class tie is generated, with an execution time derived from that provided by procedure MATCH.

The allocator may arrive at the end of a chain of PIarcs without finding one which it can allocate to a ready SIarc. This can occur for two reasons. The first is that no processor was found which was able (this includes terminal store suitability) to execute the SIarc. The class of circumstances which lead to this situation correspond to what are usually called run-time errors. Such errors may sometimes imply a logical error in the SIgraph being executed, for example, a missing job control card, or they may imply that the graph cannot be executed on the given

PIgraph, i.e. configurational limitations. An example of the latter might be the generation of more data by a program than could be accomodated on a physical storage device. In these cases the allocator ceases to bind the two graphs and takes an error exit. The allocator has a number of tests which check for error conditions throughout the iteration. When an error exit is taken, an error code is output which identifies the condition which has arisen. A list of error codes and their meanings is given in Appendix IV.

The second reason for not allocating resources to a ready PIarc is that all resources are in use. In such cases the arc is marked as ready by setting its terminal node indicator (REP [0,j]) negative, and a delay is generated for this node. The delay is a type of SIarc which does not require hardware but ensures that the allocator is recalled to the desired node at a later time, when resources are again free. When the allocator returns to a node due to a delay arc, it performs no freeing of resources, but scans the zeroth elements of the columns of the matrix REP to find delayed ready OUTarcs. It then attempts to activate these OUTarcs in the normal way. There is never more than one delay associated with a SInode, and this propagates as long as delayed SIarcs remain unallocated.

Delays are scheduled by the allocator to reactivate when resources become available. If no such occurrences are found in the list of future events, then a deadlock situation has arisen, and the allocator terminates binding with an error message.

Every call of the allocator checks whether the system time has exceeded the binding time limit, and if so halts binding and exits to the statistic processing procedures which operate on the data accumulated during the run.

A debugging option has been included in the SHAPE implementation to output extensive tracing information during each iteration of the allocator. In particular all software to hardware matchings (successful and unsuccessful) are output, together with the appropriate reasons.

For further details the reader is referred to listings of the SHAPE program in INDRA Note 286.

4.4 Ties and IFloops.

We now discuss some aspects of the SHAPE implementation which are not explicitly prescribed by the modelling system presented in the previous chapter. The first of these if the representation in a directed graph of processors which can read from and write to more than one store. Such a processor would seemingly require a PIarc with several initial and terminal nodes. Below we argue that this is a misleading picture of the situation, and put forward a description using the PIgraph as currently defined. In the implementation itself this method is compressed by the use of a processor state for each potential configuration.

The next aspect of the SHAPE program dealt with is its ability to represent preemptive scheduling. An arc at any level of a SIgraph is the indivisible process at that level. Consequently the question must arise as to how the implementation will model a preemptive event occuring during arc execution without violating that property. By the introduction of fractional allocation, and using the results of Muntz [MUNT 70] we argue that allocator variation of the fraction is equivalent to preemptive scheduling. The latter part of this section then deals with branching arcs, called IFloops in the implementation.

Within the simulation block of class allocator a process is defined with the name TIE. This process has a duration equal to the product of a time TIM and the tie's datasize LAMBDA. Both variables are parameters of the process and have values provided by the allocator which activates the tie. If the SIarc of the tie is an IFloop, then either parameter may be changed by the tie itself.

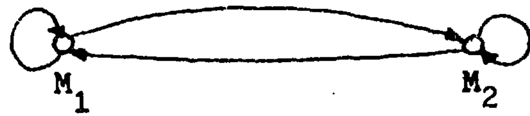
When the tie terminates it sets pointers in the data area of of the allocator (which is global to the tie) to reference the initial and terminal nodes of the SIarc and PIarc which constitute the tie. Pointers are also set to reference the arcs themselves. The tie then reactivates the allocator, terminating itself in the process.

The PIarc which is allocated to a SIarc to form a tie may represent one state of the processor involved. In the SHAPE system, a PIarc is used to describe each possible configuration of a processor. These PIarcs are referred to as states of the processor, since they all refer to the same physical processor.

This is not a fundamental attribute of the PIgraph method, but a shorthand for the basic, but more unwieldy representation of such processors. A processor which can read from and write to more than one store does so by having a data path (in some sense separate) to each store. For any given configuration only one pair of data paths is in use. Both read and write data paths use storage internal to the processor (usually one or more registers), and data transformation occurs when the processor proper operates on this internal storage.

We can represent each data path by a PIarc, internal storage by a PInode and the processor by a loop at this node. Thus a one to one correspondence is retained between hardware items and PIgraph elements. We use the many state representation as a shorthand in situations where more detail is not required, so reducing the processing required, for a run. An illustration of the two representations is shown in Fig. 4-11.

Multi-state



One-to-one

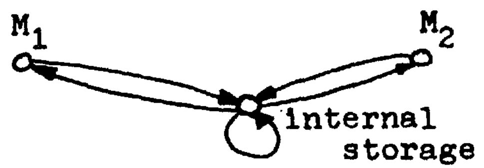


Fig. 4-11 Four state representation of a processor.

In the SHAPE system a task may be allocated a fraction of a processor as well as a complete one. In a real system it is not usual to find true fractional allocation. Where it does occur, closer examination reveals it to be unitary allocation of sub-assemblies of the processor, or preemptive allocation invisible to the allocatee (preemptive allocation usually occurs in its most elementary form, namely time-slicing).

We use various results of [MUNT 70] to justify the use of fractional allocation to protray preemptive scheduling in the SHAPE system. In their paper Basic Scheduling (BS) discipline is defined as one in which once a processor is assigned to a task it must work continuously on this task until it has been completed. If processors can be interrupted before a task is completed and reassigned to a new task, the discipline is called Preemptive Scheduling (PS).

An alternative variation of the BS discipline is to allow fractional allocation of a processor to a task. If the fraction assigned is w then it is considered to increase the computation time of the task by a factor of 1/w. If the fraction allocated to a task is allowed to change during its execution the discipline is called General Scheduling (GS).

[MUNT 70] shows that a General Scheduling discipline is equivalent to a Preemptive Scheduling discipline. As remarked above, real systems usually use some form of PS. The reallocation of resources can only occur when an individual task completes. It need not occur if completion does not make any other task ready. That is to say that task completion is a necessary but not sufficient condition for reallocation.

(unless we include return of the resource to the idle chain, in which case completion is also sufficient).

The allocator of a real (preemptive) system is either alerted to the completion of a task by the setting of flags, or is automatically activated by an interrupt. The essential purpose of the interrupt mechanism is in fact to activate the system allocator (interrupt identification and housekeeping) which preempts resources (the processor) for a higher priority task (interrupt handling). Handling the interrupt may itself generate new tasks which are generally of lesser priority. Such tasks compete for resources with those already in the system, without preemptive priority, i.e. are added to tables or queues.

The SHAPE allocator is able to duplicate the behaviour described above. A completing task (tie) sets allocator variables with identifying information before activating it. The allocator will then update the status of the tie's terminal datasets, free resources used, and has the capability to preempt a processor for a higher priority task which is now ready.

Such preemption can be achieved by altering the existing fractional allocations of the processor to provide the necessary resource. When a task completes, the freed processor fraction may be allocated amongst other tasks already tied to the processor, because these are chained (the chain starting with the attribute PSTIE for each PIarc) and consequently available to the activated allocator.

From the above remarks we see that the SHAPE allocator can meet the requirements of a scheduler for a General Scheduling discipline, since it is able to allocate a fraction of a processor,

and to vary this fraction when the system changes state. From the equivalence of a GS and PS we contend that the SHAPE allocator can adequately represent preemptive scheduling, and can also duplicate its dynamic behaviour.

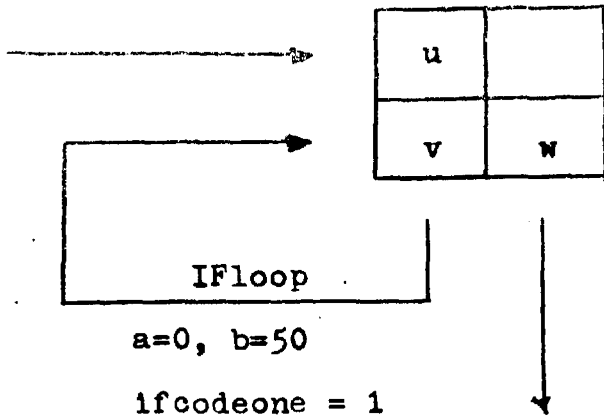
In the SHAPE system an IFloop is a SIarc which has the same initial and terminal node. Such an arc is allowed to perform some functions which are not made available to arcs with different initial and terminal nodes, and we now describe these functions.

An IFloop description consists of six real numbers which are stored in an array called IFF, at run time. There are two IFcodes, and each IFcode has two parameters, say A and B. If both IFcodes are zero then no special action is taken when the IFloop is activated. The array IFF is an attribute of all SIarcs, but we make the restriction that only an IFloop may have non-zero IFcodes.

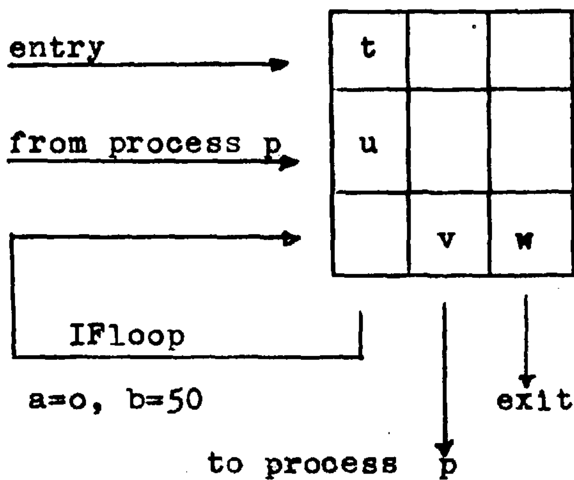
This restriction excludes arcs with different initial and terminal nodes from executing IFloop functions. The restriction is arbitrary and has been made only to test the hypothesis that modelling computational activity does not require IFloop functions to be available on other arcs.

At present all IFloop functions are executed as soon as the IFloop is activated. There is then a delay of duration T * LAMBDA before the IFloop terminates and activates its terminal dataset(s); T is the arc execution time per unit data, and LAMBDA is the data size.

The first IFcode (IFCODEONE) controls the choice of terminal datasets to be activated. If IFCODEONE equals zero, then all the terminal datasets are activated, otherwise a choice is made.



The incoming arc activates dataset u. This is sufficient to initiate execution of the IFloop. It chosses to activate dataset v, reinitiating its own execution, until the counter reaches 50. It then chooses dataset w, activating the outarc.



On entry the dataset t is activated and initiates execution of the IFloop. This will choose to activate dataset v and so execute process p. On completion process p activates dataset u which executes the IFloop again. This has the effect of executing process p fifty times before exiting through dataset w.

Fig. 4-12 DOloop examples.

The method of choice depends on the value of IFCODEONE (which is an integer between 0 and 7). If IFCODEONE equals one then the IFloop behaves as a DOloop, i.e. it adds one to a counter held in array element STARC [0], and activates the first dataset in its terminal row. As soon as the counter equals parameter B, the second dataset is chosen for chosen for activation, and the counter is reset to the value of parameter A. Use of DOloops is shown in Fig. 4-12.

If IFCODEONE equals two then a random choice is made between the first and second datasets of the row, with probability of choosing the second equal to parameter A.

If IFCODEONE equals three then the k th dataset of the row is activated, k being a random integer between A and B.

IFCODEONE equal to four is used for setting the BETA parameter of the IFloop to its termination time. All terminal datasets are activated as in the case IFCODEONE equal to zero. BETA, like LAMBDA, is a variable which propagates with the cut, and is currently used to retain the cut creation time. Its age is then available at any stage of its history.

The second IFcode, IFCODETWO, is concerned with providing new values for T or LAMBDA. If it is positive T is set to the new value, if negative then LAMBDA is reset. The new value itself is chosen by a method corresponding to the numeric value of IFCODETWO (an integer 1 to 7). If IFCODETWO equals zero then no action is taken and both T and LAMBDA are left as provided by the allocator. If either is reset, this alters the duration of the IFloop appropriately. IFCODETWO has its own pair of parameters in the array IFF, which we will again call A and B.

If IFCODETWO equals one then the new value used is a linear function of the old one, namely A times the previous value plus B.

If IFCODETWO equals two then a random choice is made between retaining the old value and replacing it by B. The probability of replacement is A.

If IFCODETWO equals three then the new value is a random integer between A and B.

If IFCODETWO equals four, then the new value is a random real number between A and B.

If IFCODETWO equals five the new value is randomly chosen from a normal distribution of mean A and variance B/1.96. If the new value is greater than B it is set to B, which removes the five per cent tail of the distribution.

IF IFCODETWO equals six the new value is randomly chosen from a negative exponential distribution of mean 1/A. Should the chosen value exceed B, it is set to B. If however B is zero, then this rule is not applied.

If IFCODETWO equals seven the new value is randomly chosen from a Poisson distribution of mean A. The new value is set to B if it exceeds B, and B is greater than zero.

Should an IFcode be out of range, or a specified dataset not found in the terminal row, then the IFloop passes a signal to the allocator not to activate any terminal datasets. This effectively extinguishes the IFloop passes a signal to the allocator not to activate any terminal datasets. This effectively extinguishes the IFloop since no further activity occurs (apart from deactivation of its initial dataset).

This facility may be used deliberately to terminate an unwanted process if desired, since it does not cause the

allocator to halt the binding of the two graphs.

A further facility implemented in the SHAPE program compensates for the absence of mode 3 binding. This allows an IFloop to deactivate its own initial dataset immediately after activation. Since this dataset is the only indication in the graph structure that an IFloop is executing, the effect is to allow several reentrant executions of the IFloop to occur concurrently. The facility is involved by changing the sign of IFCODEONE, making it negative.

The actions taken according to the numerical values of the IFcodes are summarized in Fig. 4-13.

ifcode value	dataset activated
0	all
1	<u>if</u> counter < b <u>then</u> first: counter + 1 <u>else</u> second counter:= a
2	random choice - prob (first) = 1-a prob (second) = a
3	k th where k:= random integer (a,b)
4	all: beta := termination time
5	illegal
6	illegal
7	illegal

ifcode value	new value for lambda or beta
0	no action
1	newval:= a * oldval + b
2	prob (newval:= oldval) = 1 - a prob (newval:= b) = a
3	newval:= random integer (a,b)
4	newval:= random real (a,b)
5	newval:= normal (a,b/1.96) if b > 0 <u>then</u> newval ≤ b
6	newval:= negexp (a) <u>if</u> b > 0 <u>then</u> newval ≤ b
7	newval:= poisson (a) <u>if</u> b > 0 <u>then</u> newval ≤ b

Fig. 4-13 <u>Summary of IFcode actions.</u>

4.5 Hardware measurements.

In this section we develope performance measures for hardware usage during computation. The purpose of a measure is to distinguish quantitatively if possible, between alternative courses of action. Performance arising from a particular course of action is judged good or bad by criteria expressed in terms of measures. For a measure to show different alternatives without bias, its derivation and operation should be independent of them; its value is then an accurate reflection of the alternatives.

The modelling system described in Chapter III is recursive. We argue below that measures used in it should also be capable of recursive application. Among the aims of the system is the comparison of different software graphs executing on the same hardware and vice versa, as well as the investigation of alternative allocation strategies. Consequently we require that any measures used in the SHAPE system are independent of allocation strategy and graph features which can be varied by the modeller.

Our choice of performance measures attempts to satisfy these conditions. We concentrate on two elements underlying many existing measurement systems, and which were first put forward in Chapter III. To recap briefly, two measures for the performance of a processor P in executing an arc S were used. These were the utilization (ut) and efficiency (ef). Utilization may be thought of as that fraction of the processor which is needed by the task S, i.e. 1 - ut is the fraction which is never used. Efficiency is the weighted average fraction of the utilization which is is use during the execution of S.

We regard a processor P to be made up of n components weighted with a cost function c_j for the j th component. The processor can perform any of m functions f_i in time t_i. When a particular function is being performed not all components are used. The fraction (weighted by the cost function c_j) used is the utilization ut_i for the function f_i. Each component is in use for a time $t_{ij} \leqslant t_i$, that is to say that the utilized components may not be in use for all of the time taken to perform the function f_i. The efficiency ef_i in performing function f_i is the weighted fractional time in use of those components which are utilized. This then leads to the following definitions.

$$C = \sum_j c_j$$

$$ut_i = \sum_j c_j \text{ sign } (t_{ij}) / \sum_j c_j = \sum_j c_j \text{ sign } (t_{ij}) / C$$

$$ef_i = \sum_j c_j t_{ij} / t_i \sum_j c_j \text{ sign } (t_{ij}) = \sum_j c_j t_{ij} / ut_i t_i C$$

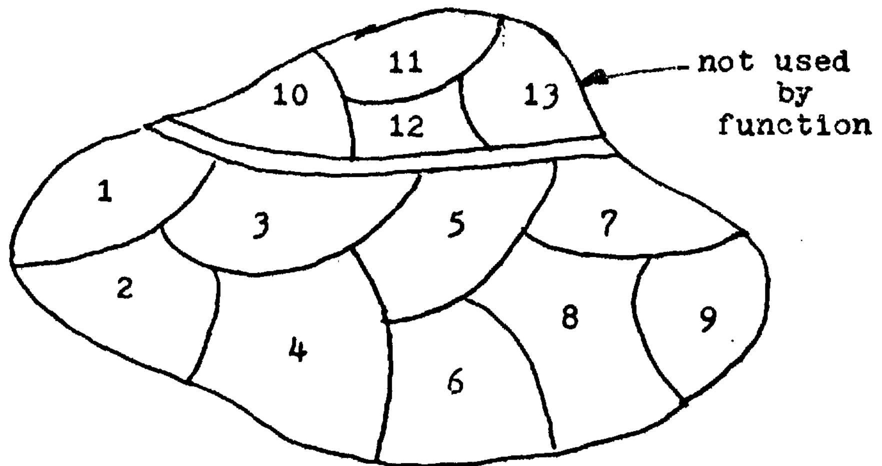
An example of the use of these definitions is shown in Fig. 4-14.

Since the SIgraph model is structured recursively, as is the SHAPE allocator, it is clear that we would like some form of performance measure which was also defined in a recursive fashion. Such definition would allow statistics to be uniformly derived at any level of the model, irrespective of the depth at which the SHAPE run was executing. Using the measures utilization and efficiency, we would like relations between levels k and k+1 of the type,

$$ef^k = f\ (ef^{k+1}, ut^{k+1})$$

$$ut^k = g\ (ef^{k+1}, ut^{k+1})$$

Processor P has thirteen components.



Components one to nine are used by function f.

$$ut = \sum_{j=1}^{9} c_j \Big/ \sum_{j=1}^{14} c_j$$

$$ef = \sum_{j=1}^{9} c_j t_j \Big/ \sum_{j=1}^{9} c_j$$

function	mix	time	c_1	c_2	c_3	. . .	c_n	ut	ef
f_1	ϕ_1	t_1	t_{11}	t_{12}	t_{13}	. . .	t_{1n}	ut_1	ef_1
f_2	ϕ_2	t_2	t_{21}	t_{22}	t_{23}		.	ut_2	ef_2
f_3	ϕ_3	t_3	t_{31}				.	ut_3	ef_3
.	.	.					.	.	.
.	.	.					.	.	.
.	.	.					.	.	.
f_m	ϕ_m	t_m	t_{m1}	.	.	. . .	t_{mn}	ut_m	ef_m

Fig. 4-14 Processor utilization and efficiency.

The reason that utilization can be less than one is the existence of a minimal unit of allocation in most of the systems under discussion. That is to say that a certain unit, or amount, of the resources available must be allocated, or none at all. In such a situation, if a task is to be executed, then the allocated resource will generally exceed the task requirement (it is infrequent for the requirement to be an exact multiple of the allocation unit). In the case of a processor it is clear that it is possible to allocate only the whole processor at any given time. Since few tasks require the complete range of functions which the processor can perform, there will be unused components in most task executions.

If we apply this point of view to a SIgraph of many levels, we see that there will be a minimal unit of allocation at each level, determined by the resources which can be described at that level. When resources are allocated for an arc at level k - 1, which has a subgraph at level k, not all components of the resource may be needed in executing the subgraph. The components will be described by the k th level of the PIgraph on which execution is taking place. If we assign a cost c_j to the j th component at the k level then the utilization at level k - 1 will be the sum of the c_j for components used during the execution, divided by the sum of c_j for all components belonging to the unit allocated at level k - 1. That is to say,

$$ut^{k-1} = \sum_j c_j \text{ sign } (t_j) / \sum_j c_j$$

where t_j is the time for which the j th component was in use.

Following a similar line of reasoning, we can say that the efficiency at level k - 1 will be the fractional usage of those components actually used in executing the subgraph at level k. We use the words fractional usage to denote fractional usage in both component space, and time. That is to say, if the processing unit c_j spends a total time t_{ja} allocated to arc a, its utilization during such allocation is ut_{ja}. If we sum over components and arcs of the k-level subgraph, we get

$$ef^{k-1} = \sum_j c_j \sum_a t_{ja} ut^k_{ja} \;/\; t^{k-1} \sum_j c_j \text{ sign } (t_j)$$

$$= \sum_j \sum_a c_j t_{ja} ut^k_{ja} \;/\; t^{k-1} ut^{k-1} C^{k-1}$$

where t^{k-1} is the time taken to execute the subgraph at level k.

At the lowest level of a PIgraph we are, by definition, unaware of the fine structure of the processing units being allocated at that level, and of the task being executed. Without this knowledge we can at most know the time for which an individual component is used (t_j). This does not affect the derivation of ut^{k-1} for the arc (task).

Assuming that the graph model is constructed to a depth at which the addition of further levels (greater depth) will not affect the results being sought we can take the ut^k_{ja} to be one at the deepest level. That is to say, that the omission of a fine structure (i.e. subgraph) on the part of the modeller implies that the ut^k_{ja} are negligibly different to one (negligibly in the sense that taking $ut^k_{ja} = 1$ introduces a negligible error in the behaviour being investigated). This action can at most affect ef^{k-1}; ef^{k-2} is calculated using ut^{k-1} which is unchanged.

Taking $ut^k_{ja} = 1$ we get $\sum_a t_{ja} ut^k_{ja} = t_j$ and consequently

$$ef^{k-1} = \sum_j c_j t_j \;/\; t^{k-1} ut^{k-1} C^{k-1}$$

which agrees with the earlier expression for ef_i where, it should now be clear, this approximation was implicitly made.

In the SHAPE system we use a mix of functions f_i to characterize an arc. This is a shorthand for describing an arc as a chain of arcs with the arc representing function f_i being repeated $\varnothing_i$ times.

We are now concerned to derive suitable formulae for ut and ef of an arc as characterized in the SHAPE system. Applying results for ut^{k-1} and ef^{k-1} where level k is a chain as mentioned above, we get,

$$ut^{ch} = \sum_j c_j \text{ sign } (\sum_i \varnothing_i t_{ij}) / C$$

since component c_j will be utilized if any of the products $\varnothing_i t_{ij}$ is non-zero, and

$$ef^{ch} = \sum_i \varnothing_i \sum_j c_j t_{ij} / \sum_i \varnothing_i t_i \sum_j c_j \text{ sign } (\sum_i \varnothing_i t_{ij})$$

using the deepest level approximation. This is appropriate since the chain-mix analysis is only performed when no fine structure is given for the arc. The expression simplifies to,

$$ef^{ch} = \sum_i \varnothing_i t_i ut_i ef_i / ut^{ch} \sum_i \varnothing_i t_i$$

In the SHAPE characterization the quantities $\varnothing_i$, t_i, ut_i and ef_i are given as input data. Clearly the problem is to find an expression for ut^{ch} without knowing the c_j and t_{ij} (i.e. without knowing the fine structure of the processing unit).

At the time of writing it does not seem possible to obtain such an expression without making further assumptions. The validity of any assumption will depend on the context in which the model is being used. We now put forward three possibilities. Firstly

$$ut^{ch} = \underset{i}{\mathrm{Max}} \; (ut_i \; \mathrm{sign} \; (\emptyset_i))$$

This might seem appropriate when the components used by the f_i tend to be subsets of the set used by f_m where $ut_m = \mathrm{Max}\ (ut_i)$.
Secondly we suggest

$$ut^{ch} = \sum_i \emptyset_i t_i ut_i \; / \sum_i \emptyset_i t_i$$

which is the expected utilization during execution of the chain. Thirdly we present a possible derivation if it is assumed that at the deepest level all c_i are equal (to one, with no loss of generality). Such an assumption can be made when the components are identical, or when the ut_i and ef_i data which has been provided reflects such a situation. In this case we can say,

Prob $\{t_{ij} = 0\} = 1 - ut_i$ for all j,

Prob $\{c_j$ not used in chain execution$\}$

$$= \prod_i (1 - ut_i) \text{ for i such that } \emptyset_i \neq 0.$$

$$= \prod_i (1 - ut_i \; \mathrm{sign} \; (\emptyset_i))$$

Prob $\{c_j$ is used$\}$ $= 1 - \prod_i (1 - ut_i \; \mathrm{sign} \; (\emptyset_i))$

and consequently,

$$ut^{ch} = 1 - \prod_i (1 - ut_i \; \mathrm{sign} \; (\emptyset_i))$$

We call these three possible approximations to ut^{ch}: utmx, utav, utpr respectively.

They give rise to three possible values for ef^{ch}, depending on which one is used in the expression. These will be called efmx, efav and efpr respectively.

We can show the operation of these three definitions by a numerical example. Suppose an arc requires one execution of each of two functions $\emptyset_1$ and $\emptyset_2$. If $\emptyset_1$ and $\emptyset_2$ have durations of 1 and 3, and utilizations of 0.9 and 0.5 respectively, then we can see that

utmx = 0.9

utav = 0.6

utpr = 0.95

If a set of functions required by a SIarc can be ordered such that each function includes its predecessors, then clearly utmx is the appropriate measure. For example, the first function may be the no operation function of a central processor which simply advandes to the next instruction; the second may be a register transfer; the third a register transfer with an airthmetic operation. Each of these requires the components of the processor used by its predecessors.

The second measure, utav, is a statistic which corresponds to the expected value of the utilization during arc execution. This is not necessarily a value which could actually arise in arc execution, but provides the time weighted average of such values.

If the set of functions of a SIarc is such that they use groups of processor elements which are effectively independent, that is to say as if chosen at random, then utpr will be the most suitable measure.

The current SHAPE system is designed to produce all these statistics. To summarize, for each allocation we get,

$$\text{time} := \sum_i \emptyset_i t_i$$

$$\text{utpr} := 1 - \prod_i (1 - ut_i \text{ sign } (\emptyset_i))$$

$$\text{utav} := \sum_i \emptyset_i t_i ut_i \ / \ \text{time}$$

$$\text{utmx} := \text{Max}_i \ (ut_i \text{ sign } (\emptyset_i))$$

$$\text{efpr} := \sum_i \emptyset_i t_i ut_i ef_i / \ \text{utpr} * \text{time}$$

$$\text{efav} := \sum_i \emptyset_i t_i ut_i ef_i / \ \text{utav} * \text{time}$$

$$\text{efmx} := \sum_i \emptyset_i t_i ut_i ef_i / \ \text{utmx} * \text{time}$$

Extending these ideas to deriving appropriate measures for a subgraph (i.e. for the arc of which the subgraph represents the fine structure) we introduce a new variable, utcurrent (t). This is the weighted sum of component in use at time t during execution of the subgraph. If C_g is the total sum of components available and T_g is the time taken to execute the subgraph, then we have,

$$\text{gutpr} = \sum_j c_j \text{ sign } (t_j) / \ C_g$$

$$\text{gutmx} = \text{Max}_{0<t<T_g} \ (\text{utcurrent } (t)) / \ C_g$$

$$\text{gutav} = \int_0^{T_g} \text{utcurrent } (t) dt / \ C_g * T_g$$

It is of course not necessary to make approximations in the case of a subgraph, the expression gutpr is the correct one from the point of view of the previous derivations.

The measures gutmx and gutav are included in order to provide consistency at the level above the subgraph, in which there may be arcs without a fine structure. Thus the gutpr, gutmx and gutav of level k provide the values of utpr, utmx and utav for the arc at level k - 1 whose fine structure is represented by the subgraph.

In determining the efficiency we use the three types of arc utilization to provide

$$gefpr = \sum_j \sum_a c_j t_{ja} utpr_{ja} / gutpr * C_g * T_g$$

$$gefmx = \sum_j \sum_a c_j t_{ja} utmx_{ja} / gutpr * C_g * T_g$$

$$gefav = \sum_j \sum_a c_j t_{ja} utav_{ja} / gutpr * C_g * T_g$$

gutpr is used throughout in the denominator, since it correctly represents ut^{k-1}, and means that the numerators are being compared to a common standard.

In order to provide these statistics for a subgraph the SHAPE system maintains a running sum, for each processor, of the three expressions,

$$utpr_a * t_a$$

$$utmx_a * t_a$$

$$utav_a * t_a$$

This allows as a byproduct, the production of statistics for each processor of the type described above. For completeness a running maximum of $utmx_a$ is held, and also a cumulative function frequency vector Ø [1:n]. Ø [i] holds the total number of times the processor has executed function f_i. This allows us to define,

$$putpr = 1 - \prod_i (1 - ut_i \text{ sign } (\emptyset_i))$$

$$putmx = \underset{a}{Max} (utmx_a)$$

$$putav = \sum_a t_a \; utav_a / T_g$$

$$pefpr = \sum_i \emptyset_i t_i ut_i ef_i / putpr * T_g$$

$$pefmx = \sum_i \emptyset_i t_i ut_i ef_i / putmx * T_g$$

$$pefav = \sum_i \emptyset_i t_i ut_i ef_i / putav * T_g$$

In addition, for historical reasons we keep a running total of t_a, allowing us to define,

$$ptime = \sum_a t_a / T_g$$

We now apply the arguments above to memory elements. If we regard a processor as made up of memory elements and data paths, the expressions arrived at above apply to the data paths (as processing elements of weight c_j). Suppose each memory element is assigned a weight m_j and is in use for a time t_j, we can say for function i,

$$ut_i = \sum_j m_j \text{ sign } (t_{ij}) / \sum_j m_j$$

and, $M = \sum_j m_j$

and so, $$ef_i = \sum_j m_j t_{ij} / t_i \sum_j m_j \text{ sign } (t_{ij})$$

$$= \sum_j m_j t_{ij} / ut_i t_i M$$

This expression, like that derived for data path components in a processor, implicitly assumes that the utilization of memory at the level m_j is one. Where the m_j are memory components in a subgraph we say,

$$ut_m^{k-1} = \sum_j m_j \text{ sign } (t_j)/ M^{k-1}$$

where t_j is the time for which the memory (or store) was in use. A store is in use when all or part of it is allocated to the initial or terminal datasets of an active arc, or to the storage of an initial dataset of an arc which is not yet active.

The question now arises of what expression to use for ut_j. We shall use $mu_j(t)$ to denote the level of usage of element m_j at time t. That is to say that $mu_j(t)$ is in some unit of memory measurement, so that

$$0 \leqslant mu_j(t) \leqslant maxmu_j$$

where $maxmu_j$ is the capacity of m_j. We use the product $\sum_p t_{jp} ut_{jp}$ as an expression for the usage of m_j over the execution of the subgraph, since a summation over arc executions will not include dataset waiting times, and a memory can hold data for many active and inactive arcs at any given time. t_{jp} is the length of the period p in which $mu_j(t) > 0$ and $ut_{jp} = \max [mu_j(t)]\ maxmu_j$ so that

$$ef_m^{k-1} = \sum_j m_j \sum_p t_{jp} ut_{jp} / t^{k-1} \sum_j m_j \text{ sign}(t_j)$$

$$= \sum_j m_j \sum_p t_{jp} ut_{jp} / t^{k-1}\ ut_m^{k-1} M^{k-1}$$

Since memory is a homogeneous resource we can say that the union of all parts of the memory used during a given interval is equal to the maximum usage in that interval (this assumes a memory compaction mechanism which uses negligible resources). Since the union of used components in task execution is the utilization for that task, this allows us to write,

$$ut_{jp} = \max_{\tau_p \leqslant t \leqslant T_p} (mu_j(t))/ \, maxmu_j$$

where τ_p and T_p are the starting and ending times of period p respectively, so that,

$$ut_j = \max_p (ut_{jp})$$

Using this expression for ut_j in the equation for ef^{k-1} gives us consistency with the equivalent expression for processing unit usage. Following through for the efficiency of memory m_j we get,

$$ef_j = \int_0^{T_g} mu_j(t)dt/ \, ut_j T_g maxmu_j$$

If we stipulate that the component weights m_j and c_j are in the same units then we are able to combine processing unit and memory unit usage as follows:

Total resources allocated $= C^{k-1} + M^{k-1}$

Total resources utilized $= C^{k-1}ut_c^{k-1} + M^{k-1}ut_m^{k-1}$

Total resource usage $= C^{k-1}ut_c^{k-1}ef_c^{k-1} + M^{k-1}ut_m^{k-1}ef_m^{k-1}$

dropping the superscript, we have,

Overall utilization $U = (C\ ut_c + M\ ut_m)/ \, (C + M)$

Overall efficiency $E = (C\ ut_c ef_c + M\ ut_m ef_m)/ \, (C\ ut_c + M\ ut_m)$

$= (C\ ut_c ef_c + M\ ut_m ef_m)/ \, (u(C + M))$

In the SHAPE system the data on processor characteristics for each of the n functions at any level is assumed to consist of the values of U and E for each function.

It can be seen, by examining the expressions aoove, that utilization is independent of idle time in task execution. In fact the utilization will reflect how well the allocation mechanism for a task (subgraph), and its choice of allocatable

entities, is suited to the task in hand. The efficiency expressions tend to be an expression of resource usage (and therefore of idle time) of allocation units, and components within these units, for a sequence of allocations.

4.6 Software measurement.

We describe below the types of statistics which are produced by the SHAPE system concerning SIgraph binding. These fall into three categories, statistics for nodes, arcs, and cut(s).

Node statistics are held in three two-dimensional arrays, QD, QT, QS, all of dimension [0: INARCS, 1:4]. The zero row holds overall statistics for the node, while if binding takes place in semi-reentrant mode, the other rows contain statistics for the transaction queues of the corresponding rows of the REP matrix.

For the node as a whole we keep statistics of the amount of active data associated with the node in QD [0,j], j = 1,4. These are the time integral of the associated data, and its maximum value. In array QT [0,j], j = 1,4 we retain the number of activations of the node, the sum of their durations, and the minimum and maximum duration. In QS [0,j], j = 1,4 we hold the time integral of the node activity and its maximum value. Node activity is the number of currently active elements of the REP matrix plus the number of queued transactions, if any.

Similar statistics are kept for the individual INarc queues if binding is semi-reentrant. Node activation becomes queue activation, i.e. the number of transactions which enter the queue is counted. Duration becomes queue waiting time. In order to record this item transactions possess a scratch variable which is set to current system time on entry into the queue; on exit the waiting time is current time minus the scratch value. Activity becomes queue size and is recorded as for the whole node. The usage of arrays AD, AT, and QS is shown in Fig. 4-15.

	QD [0]	QT [0]	QS [0]
1	time integral of node data	activation counter	time integral of node activity
2	last time changed	sum of activation times	last time changed
3	current associated data	min activation times	current activity
4	max associated data	max activation times	max activity

	QD [1]	QT [1]	QS [1]
1	time integral of Q data	transaction counter	time integral of Q size
2	last time changed	sum of waiting times	dead time
3	current associated data	min waiting times	current Q size
4	max Associated data	max waiting times	max Q size

Fig. 4-15 Node Statistics in arrays QD, QT, QS.

However, if the INarc specifier of the row, REP [i,0] is negative the corresponding rows of QD, QT, and QS are not used for queue statistics but to accumulate counts, sums, maxima and minima of the LAMBDA and BETA factors of the incoming arc activations.

These are cut statistics, and may be collected at any node. If the node is the terminal node of the SIgraph, then the values collected will reflect the values of LAMBDA and BETA associated with the cut on its completion. At other nodes they will reflect intermediate stages of the cut history. BETA is a variable which records a time value and propagates with the cut. When one of the OUTarcs of a SInode is allocated, the value of BETA given to the tie is equal to the largest BETA associated with the INarcs which provided the initial data of the OUTarc. At the moment BETA is set to the current time when a cut is generated. On completion of the binding which this cut represents BETA will still have this value (unless deliberately reset by an IFloop) and thus provides the age of the cut. At nodes other than the terminal one BETA can be used to provide the cut age at an intermediate point of its history. In semi-reentrant mode, LAMBDA and BETA values of ties are retained in transaction attributes TL and TB when the ties are queued. A use for the BETA factor occurs when a cut represents the transmission of a message in a switching network. In this case the cut age is the overall transmission time from source to destination. Array usage for cut statistics is shown in Fig. 4-16.

The following statistics are recorded for SIarcs in array STARC [1:6], namely the number of times the arc was allocated, the sum of the execution times and utilizations, and the maximum

	QD [i]	QT [i]	QS [i]
1	cut counter	cut counter	
2	sum of lambdas	sum of betas	
3	min " "	min " "	sum of squares of betas
4	max " "	max " "	sum of squares of lambdas

Arrays QD, QT, QS are used in this way when REP $[1,0] < 0$

Fig. 4-16 <u>Cut statistics.</u>

and minimum execution time and utilization. This is shown in Fig. 4-17.

On graph completion in addition to the above statistics, the fraction of time (activity) for which nodes and arcs were active is printed. Averages are also printed for arc execution time, node activity, queue size, associated data, and in all cases these are averages over the whole graph time rather than the active time of the elements concerned. If the second type of average is required it can be obtained from division by element activity.

The items described above have been implemented as being a simple but sufficient and useful set of statistics for present use with the SHAPE system.

STARC

1	execution count
2	sum of execution times
3	max " "
4	min " "
5	sum of utilizations
6	max " "
7	min " "

Fig. 4-17 Arc statistics in array STARC.

CHAPTER V

VALIDATION

5.1 The choice of validation.

For any system which attempts to model a large class of computational processes there must be many candidates for the role of validation. In choosing a model it is advantageious to select the simplest one which still tests all the facilities of the modelling system. In our case a further consideration was the type of problem the system would be applied to after its validation. A validation based on a related problem would have the double advantage of ensuring the adequacy of the system for the subsequent work, and providing relevant experience in this area of its use.

One of the most stimulating of current developments has been the research and construction of computer netwroks. The initial problems have been the very basic ones of implementing suitable communication systems between the node computers, and their clusters of terminal users. Once such communications are implemented the connected user can access not only the facilities available at his own node computer, but those throughout the network. For this reason such networks have been called resource sharing networks.

Computer networks have also been constructed for other reasons. Message switching systems make the solution of the communication problem their prime objective. Real-time networks (of which military and airline ones are the best examples) have been implemented to conduct operations beyond the capability of a single computer.

This leads us to expand the remarks in Chapter I which assert that at any given time there must be tasks which require

a degree of computing power that can only be provided through parallelism. Real-time networks have been a response to such tasks. It is to be anticipated that as the operational difficulties of resource sharing networks are solved their facilities will not only be shared but also used cooperatively in the solution of computational problems of a new order of magnitude.

The common prerequisite of computer networks has been a communications system between the nodes. A very frequent solution has been store and forward transmission of messages as a series of packets. This has been the choice of the implementors of the Advanced Projects Research Agency (ARPA) and National Physical Laboratory (NPL) networks.

The intended application of the SHAPE system was to an extension of the ARPA network to Norway and London. For this reason the validation test chosen was the modelling of a small store and forward communication system. This model provides tests of the major functions of the SHAPE system. The creation of messages (i.e. the traffic) to give particular distributions of frequency and length uses IFloop facilities for random numbers generation, and delay, and dataset size setting. The dispatch of messages to their destinations requires the allocation of processors (the transmission channels) and memory along the route. Accumlation of messages at intermediate nodes uses the queueing ability of mode two binding. Measurement facilities are used to derive the validation test statistic, and so on.

In the following sections of this chapter we discuss the store and forward system to be modelled, present the model itself,

and derive the test statistic. Lastly, the results from a number of computer runs of the model are given and examined for confirmation of validity.

5.2 Store and forward networks.

In a circuit-switched network of communication channels two subscribers who wish to exchange information must first establish a circuit or path, between their terminal equipments. This path is static once established and remains in existence for the duration of the dialogue. The channels which make up the path are consequently dedicated for this period. Telephone systems are an example of circuit-switched networks.

A store-and-forward network transmits information between subscribers without establishing a fixed path between them, and without dedicating channels for the duration of the dialogue. This is achieved by formatting information as messages with an address or destination. A message is then transmitted along the route to its destination, with one channel being allocated at a time. Channels transmit between exchanges or nodes which are able to store messages and usually have several incoming and outgoing channels. When a message arrives at a node the outgoing channel is selected using the message destination, and routing information possessed by the node. If the channel is free the message is transmitted immediately, otherwise it is stored at the node and forwarded later, giving rise to the name for these networks. The routing may be fixed or vary with conditions in the network. An example of this type of network is the postal service. For this reason the phrase packet-switched network is sometimes used.

The nature of computer-to-terminal, or computer-to-computer, dialogue makes store-and-forward communications a more economic choice for computer networks than circuit-switching.

The dialogue typically has long pauses while a terminal user prepares his next input or a computer produces a reply. Nevertheless a high data rate is required when transmission does occur in order to provide good response times in interactive systems. Such usage inevitably incurs a high overhead in idle time when channels are dedicated, as is the case with a circuit-switched network. In contrast store-and-forward networks are an attempt to ensure that messages use the minimum channel capacity which is required for delivery. However, the storage facilities and the necessity of routing procedures now introduce a new overhead which must in turn be assessed.

The type of store-and-forward network which is used for this validation is that treated by Kleinrock in his book Communication Nets. This class of network is characterized by the following properties.

Each node in the network may be both a source and a sink of messages. The channels and nodes are assumed to be noiseless and reliable. Delays at nodes due to routing procedures and other housekeeping operations are assumed to be negligible. Messages are considered to have only one destination and must reach it to leave the network. This implies unlimited storage capacity at the nodes. Messages may not be transmitted out of a node until they have been completely received. Messages are generated at a node with exponentially distributed interarrival times, i.e. their generation is a Poisson process. Message length is also assumed to be exponentially distributed, and both processes are considered stationary with respect to time.

The first five of these properties are not unrealistic. The last three represent reasonable assumptions of great mathematical usefulness, and Kleinrock refers to telephone traffic data which supports their plausibility. The performance measure of this type of network is the average message delay. This is the mean over all messages of the total time spent in the network by a message. For ease of reference we use the same notation as Kleinrock, which is summarized below.

γ_{jk} = average number of messages entering network per second with origin j and destination k.

λ_i = average number of messages entering i th channel per second.

$1/\mu_{jk}$ = average length of messages which have origin j and destination k, in bits.

C_i = capacity of i th channel, bits/second.

γ = total arrival rate of messages from external sources.

λ = total arrival rate of messages to channels within the net.

$\bar{n}$ = average path length over all messages.

$1/\mu$ = average message length from all sources.

C = sum of all channel capacities in the net.

p = network load, i.e. ratio of average arrival rate of bits into the net from external sources to total capacity of net.

Z_{jk} = average message delay for messages with origin j and destination k.

T_i = average delay for a message passing through channel i, queueing plus transmission time.

T = average message delay.

The definitions lead to the relations,

$$\bar{n} = \lambda/\gamma$$

$$1/\mu = \sum_{j,k} \gamma_{jk}/(\gamma \mu_{jk})$$

$$C = \sum_i C_i$$

$$\gamma = \sum_{j,k} \gamma_{jk}$$

$$p = \gamma/(\mu C)$$

$$\lambda = \sum_i \lambda_i$$

$$T = \sum_{j,k} \gamma_{jk} \, Z_{jk}/\gamma = \sum_i \lambda_i \, T_i/\gamma$$

The average message delay is the performance measure to be optimized, and Kleinrock has derived analytic results for the allocation of channel capacities which achieves the optimal delay.

Firstly he shows that for the class of nodes with N outgoing channels of capacity C/N (the total capacity C is a constant) the average message delay is a minimum when N = 1. This result is used to develope the optimal channel assignment for a net of N independent nodes each with a single output channel. The assignment (subject to the constraint that the sum of the channel capacities is constant)

which minimizes the message delay averaged over the set of N nodes is given by,

$$C_i = \lambda_i/\mu_i + \left(C - \sum_{j=1}^{N} \lambda_j/\mu_j\right)\sqrt{\lambda_i/\mu_i} \bigg/ \sum_{j=1}^{N} \sqrt{\lambda_j/\mu_j}$$

Using this assignment gives,

$$T = \left(\sum_{i=1}^{N} \sqrt{\lambda_i/\lambda\mu_i}\right)^2 \bigg/ C(1-p)$$

Finally, for the general case of an interconnected net, with $\mu_i = \mu$ for all i, the optimal channel assignment is given by:

$$C_i = \lambda_i/\mu + C(1-\bar{n}p)\sqrt{\lambda_i} \bigg/ \sum_{j=1}^{N} \sqrt{\lambda_j}$$

This gives,

$$T = \bar{n} \left(\sum_{i=1}^{N} \sqrt{\lambda_i/\lambda_i}\right)^2 \bigg/ \mu C(1-\bar{n}p)$$

The assignment can be interpreted as follows. Each channel is first apportioned just enough capacity to satisfy its average required flow of λ_i/μ bits/sec. After this the capacity is,

$$C - \sum_{i=1}^{N} \lambda_i/\mu = C(1-p)$$

which is then distributed amongst the channels in proportion to the square root capacity assignment. It is this last case which we have chosen for the validation of the SHAPE program. A model of an interconnected net is described using the SHAPE system and the channel capacities are calculated as shown above. The mean message delay is then measured and compared with the calculated value of T, using given confidence limits. For detailed background to this subject the reader is referred to [KLEI 64].

5.3 The validation model.

The model used is a simple one from the first part of Kleinrock's book. It describes the hypothetical message flow between five cities of the United States. The topology of the network is shown in Fig. 5-1. The traffic matrix is based on a conjecture of Zipf that the flow between two cities of population P_i and P_j a distance D_{jk} apart is given by

$$\gamma_{jk} = \alpha P_j P_k / D_{jk}$$

where α is a constant of proportionality. This leads to the proportional traffic matrix given in Fig. 5-2. Kleinrock chooses the total capacity to be equal to the total proportional network traffic (38.33) and $\mu = 10$. The routing procedure is fixed and consists of the set of shortest routes. This leads to the mean path length

$$\bar{n} = \lambda/\gamma = 1.31$$

The routing is shown in Fig. 5-3. Variable values for individual links of the network are shown in Fig. 5-4, the total link traffic λ being 50.23. From the individual channel delays and the traffic matrix, we can calculate the delay for each type of message, and these are given in Fig. 5-5. Using the intermediate results that,

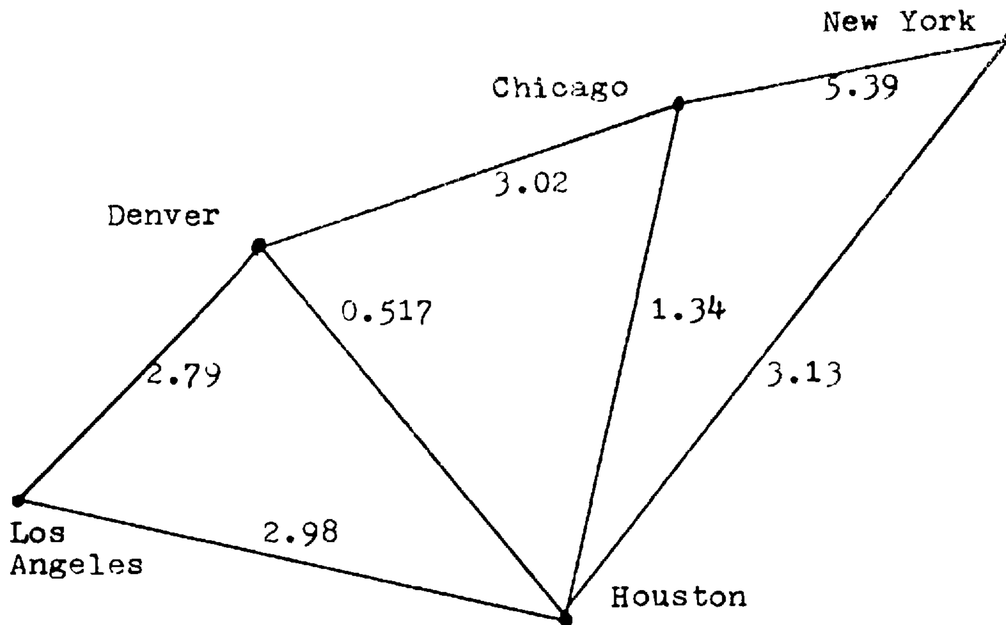
$$C(1-\bar{n}p)/\sum\sqrt{\lambda_i} = 1.39286$$

$$\sum\sqrt{\lambda_i}/2 = 11.95698$$

$$p = 0.1$$

We find that the average message delay

$$T = 0.0447767$$



The channels shown are full duplex, so that total channel capacity in the net is 38.33.

Fig. 5-1. Validation Network.

	NY	CH	HO	DE	LA
NY	-	9.34	0.935	0.610	2.94
CH	9.34	-	0.820	0.628	2.40
HO	0.935	0.820	-	0.131	0.608
DE	0.610	0.628	0.131	-	0.753
LA	2.94	2.40	0.608	0.753	-

Total Traffic γ = 38.33

Mean message length $1/\mu$ = 0.1

Fig. 5-2. <u>Proportional Traffic Matrix.</u>

	NY	CH	HO	DE	LA
NY	-	1	1	2-CH	2-HO
CH	1	-	1	1	2-DE
HO	1	1	-	1	1
DE	2-CH	1	1	-	1
LA	2-HO	2-DE	1	1	-

Key 1 path length is one, routing direct

2-X " " " two, routing through X.

Mean path length $\bar{n}$ = 1.31

Fig. 5-3 Message routing.

Link	λ_i	$\sqrt{\lambda_i}$	C_i	T_i	$1/C_i$
NY/CH	9.950	3.15436	5.38858	0.0227605	0.185578
NY/HO	3.875	1.96850	3.12934	0.0364718	0.319556
CH/DE	3.638	1.90735	3.02046	0.0376411	0.331074
CH/HO	0.820	0.90554	1.34329	0.0792839	0.744441
HO/DE	0.131	0.36194	0.51723	0.1933610	1.933371
HO/LA	3.548	1.88361	2.97340	0.0381155	0.335750
LA/DE	3.153	1.77567	2.78856	0.0404325	0.358608

Total link traffic = 50.23

Fig. 5-4. Link Traffic, Delay, and Capacity.

Z	NY	CH	HO	DE	LA
NY	-	0.0227605	0.0364718	0.0604016	0.0745873
CH		-	0.0792839	0.0376411	0.0780736
HO			-	0.1983610	0.0381155
DE				-	0.0404325
LA					-

The matrix elements Z_{jk} give the delay for messages with origin j and destination k.

Fig. 5-5. Message Delay Matrix Z.

The top right and bottom left halves of the traffic matrix define two identical and non-interacting systems. Consequently it is only necessary to model one of them, and we in fact choose the top right system. This system is shown in Fig. 5-6 with node numbers assigned to the cities. The topologies of the SIgraph and PIgraph are shown in Fig. 5-7, but without the IFloops of the SIgraph. Nodes of the SIgraph, examples of its arcs, and examples of the PIgraph data are shown in Figs. 5-8, 5-9, 5-10 and 5-11 respectively.

Node one generates messages to four destinations, namely nodes two to five. The generation of each is caused by an INarc of the node (e.g. that entering the row with arc specifier 1.21). Termination of the INarc activates two matrix elements. The first of these reactivates the INarc itself for a delay drawn from a negative exponential distribution, while the second activates a further arc (e.g. that of column with specifier 1.22) which draws a value for the dataset size from a similar distribution. This second arc has a zero duration.

The dataset size is effectively equivalent to the message length in our model, since the function which is executed by SIarcs between two SInodes is the transmission of one bit between them. Such SIarcs have a non-zero element in their function vector, and so must be tied to PIarcs capable of executing the function. These PIarcs represent the communication channels of the network, and their function vectors give the time required to transmit one bit from the initial to the terminal node, i.e. the inverse of the channel capacity. The nodes of the PIgraph therefore correspond to the storage available for messages at each of the five cities.

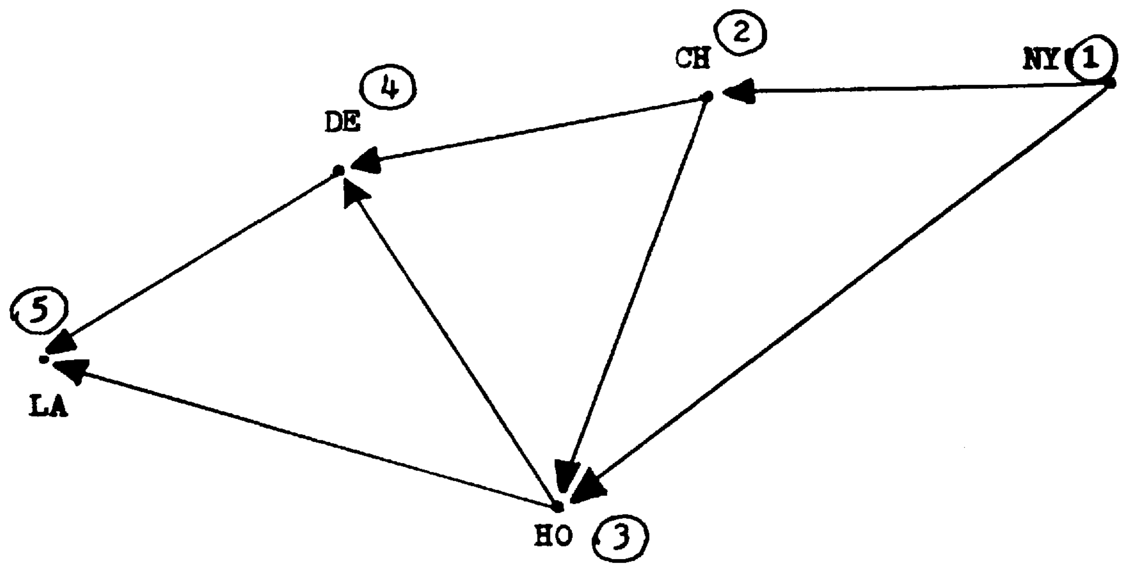
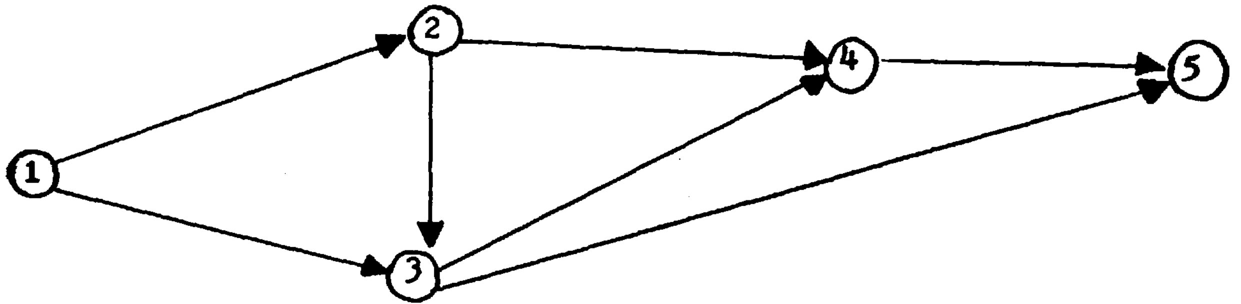


Fig. 5-6. Directed Semi-Network.

PIgraph



SIgraph (loops and node 0 not shown)

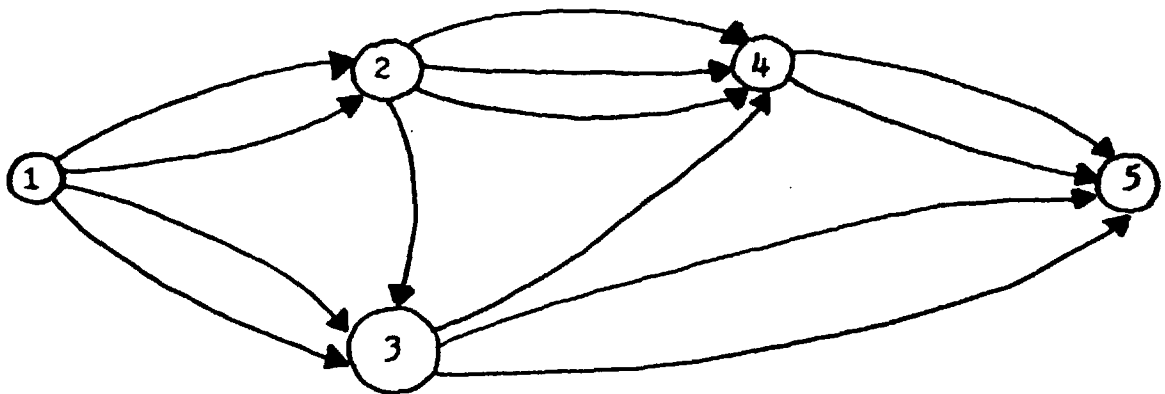


Fig. 5-7. SIgraph and PIgraph topologies.

(1)	1.21	1.22	2.20	1.31	1.32	3.30	1.41	1.42	2.40	1.51	1.52	3.50
1.21	1	1										
1.22			1									
1.31				1	1							
1.32						1						
1.41							1	1				
1.42									1			
1.51										1	1	
1.52												1
↓			(2)			(3)			(2)			(3)

(2)	2.31	2.32	3.30	2.41	2.42	4.40	2.51	2.52	4.50	4.10	2.20
2.31	1	1									
2.32			1								
2.41				1	1						
2.42						1					
2.51							1	1			
2.52									1		
→ 1.40										1	
→ 1.20											1
2.20											
↓			(3)			(4)			(4)	(4)	

Fig. 5-8. Nodes 1 and 2 of model SIgraph.

3	3.41	3.42	4.40	3.51	3.52	5.50	5.10	3.30
<u>3.41</u>	1	1						
3.42			1					
<u>3.51</u>				1	1			
3.52						1		
→ 1.50							1	
→ -1.30								1
→ -2.30								1
3.30								
			↓ 4			↓ 5	↓ 5	

(4)	4.51	4.52	5.50	5.20	4.40
<u>4.51</u>	1	1			
4.52			1		
2.50				1	
-2.10					1
-2.40					1
-3.40					1
4.40					
			5	5	

(5)	5.50
→ -3.10	1
→ -3.50	1
→ -4.20	1
→ -4.50	1
5.50	

Fig. 5-9. <u>Nodes 3,4, and 5 of the model SIgraph.</u>

Transmission arc

```
1     2
          2.20
          0     0     0     0     0     0
          1
          0
          0
          0
```

Generator delay arc

```
1     1
          1.21
          4     6     0     γjk   0     0
          0
          0
          0
          0
```

Generator message length arc

```
1     1
          1.22
          0    -6     0     μ     0     0
          0
          0
          0
          0
          0
```

Termination arc

```
5     5
          5.50
          0     0     0     0     0     0
          0
          0
          0
          0
```

Fig. 5-10. Example arc data of the SIgraph.

PIgraph nodes

1	0	2				
	1'20	0	0	0	0	0

PIgraph arcs

1	2		
		2	
		11	0
		$1/C_i$	1
		-1	-1
		-1	-1
		-1	-1

Fig. 5-11. Example node and arc data of the PIgraph.

We have used the SHAPE facility for specifying that a SInode be tied to a particular PInode to ensure that messages are generated in the correct store. The fixed linkage between the SInodes and PInodes is required since the SIgraph contains the routing pattern of the network. As each message is generated the current time is recorded in the BETA variable of the generating tie. This is then propagated with the message until its destination node is reached. Here cut statistics collection is involved by setting the incoming arc specifier negative. This has the effect of measuring the message delay which is accumulated in scratch variables as described in Chapter IV. After this the message is destroyed by the use of an IFloop whose terminal row has only zero elements.

The distribution used for the delay between generation of successive messages with the same destination is negative exponential so that the generation is a Poisson process. The mean of this distribution then determines the average rate of message generation for this destination. In the model these means are taken from the proportional traffic matrix. Similarly the message length is generated using drawings from a negative exponential distribution of mean $1/\mu$ (=10). In order to approximate unlimited storage at the PInodes, we have given each one a capacity of ten to the power twenty.

The model as a whole is started by activating the terminal datasets of each generator arc. After this initial activation the generation proceeds automatically as described above. The initial activation is produced by null arcs from node zero. This is possible since the SHAPE system allows activation of the same REP-matrix row by arcs from different SInodes.

In the run card for the model we specify node zero as the initial node and consequently graph binding commences with the activation of the row of the node which has a zero INarc specifier.

The run then continues until the time limit specified on the run card is reached. The seed of each random drawing stream is taken from the arc data, so that different runs can be produced by altering the seeds. Details of the data, and seeds of individual runs, can be found in INDRA note 285, Institute of Computer Science, 1973.

5.4 The statistical test.

We have ten distinct message types in the validation model, each with a theoretically calculated mean message delay of z_i where i is the message type. We shall call the delay for the j th message of type i, x_{ij}, and the number of these messages n_i.

If we now consider the variables,

$$x_k = x_{ij} - z_i$$

then these have a theoretical mean of zero for every message type. Because all the ten groups of the x_k have the same mean, we can combine their variables by simple addition so that,

$$Ns^2 = \sum_i x_k^2$$

where $N = \sum_i n_i$

and s^2 is the variance of all the x_k taken together. This allows us to apply a t-test to the whole sample of the x_k. The hypothesis for the test is then that

$$\bar{x}_k = 0$$

The advantage of treating the data in this way is that we have now developed a single test which utilizes all the observations produced during a run of the model. We now derive the statistic t as follows.

$$t = \frac{\bar{x}_k - \mu}{s/\sqrt{N-1}}$$

where, in this case, $\mu = 0$. Also we can write:-

$$s^2 = \left(\sum x_k^2 - N\,\bar{x}_k^2\right) \Big/ (N-1)$$

so that,

$$t = (N-1)\bar{x}_k \Big/ \sqrt{x_k^2 - N\,\bar{x}_k^2}$$

Now,

$$\bar{x}_k = \frac{1}{N}\sum_i\left(\sum_j x_{ij} - n_i z_i\right)$$

$$= \frac{1}{N}\sum_i n_i(\bar{x}_i - z_i)$$

and,

$$\sum_k x_k^2 = \sum_{ij}(x_{ij} - z_i)^2$$

$$= \sum_{ij} x_{ij}^2 - 2\sum_i z_i \sum_j x_{ij} + \sum_i n_i z_i^2$$

$$= \sum_{ij} x_{ij}^2 - 2\sum_i n_i z_i \bar{x}_i + \sum_i n_i z_i^2$$

$$= \sum_{ij} x_{ij}^2 - \sum_i n_i z_i (2\bar{x}_i - z_i)$$

also,

$$s_i^2 (n_i - 1) = \sum_j (x_{ij} - \bar{x}_i)^2$$

$$= \sum_j (x_{ij}^2 - 2x_{ij}\bar{x}_i) + n_i \bar{x}_i^2$$

$$= \sum_j x_{ij}^2 - n_i \bar{x}_i^2$$

In the model runs $N >> 30$ so that the t - distribution is very close to the normal distribution. That is to say we use the bottom row of the table in Fig. 5-12. For this case the probability that $|t| > 1.96$ is 0.05. Thus if we get such a value from a run we reject the hypothesis that $\bar{x}_n = 0$ with 95% confidence. Otherwise we accept the hypothesis.

We can obtain 95% confidence limits for the true mean μ of the x_k by writing

Fig. 5-12. VALUES OF t CORRESPONDING TO GIVEN PROBABILITIES*

Degrees of freedom n	Probability of a deviation greater than t						Probability of a deviation greater than t					
	.005	.01	.025	.05	.1	.15	.2	.25	.3	.35	.4	.45
1	63.657	31.821	12.706	6.314	3.078	1.963	1.376	1.000	.727	.510	.325	.158
2	9.925	6.965	4.303	2.920	1.886	1.386	1.061	.816	.617	.445	.289	.142
3	5.841	4.541	3.182	2.353	1.638	1.250	.978	.765	.584	.424	.277	.137
4	4.604	3.747	2.776	2.132	1.533	1.190	.941	.741	.569	.414	.271	.134
5	4.032	3.365	2.571	2.015	1.476	1.156	.920	.727	.559	.408	.267	.132
6	3.707	3.143	2.447	1.943	1.440	1.134	.906	.718	.553	.404	.265	.131
7	3.499	2.998	2.365	1.895	1.415	1.119	.896	.711	.549	.402	.263	.130
8	3.355	2.896	2.306	1.860	1.397	1.108	.889	.706	.546	.399	.262	.130
9	3.250	2.821	2.262	1.833	1.383	1.100	.883	.703	.543	.398	.261	.129
10	3.169	2.764	2.228	1.812	1.372	1.093	.879	.700	.542	.397	.260	.129
11	3.106	2.718	2.201	1.796	1.363	1.088	.876	.697	.540	.396	.260	.129
12	3.055	2.681	2.179	1.782	1.356	1.083	.873	.695	.539	.395	.259	.128
13	3.012	2.650	2.160	1.771	1.350	1.079	.870	.694	.538	.394	.259	.128
14	2.977	2.624	2.145	1.761	1.345	1.076	.868	.692	.537	.393	.258	.128
15	2.947	2.602	2.131	1.753	1.341	1.074	.866	.691	.536	.393	.258	.128
16	2.921	2.583	2.120	1.746	1.337	1.071	.865	.690	.535	.392	.258	.128
17	2.898	2.567	2.110	1.740	1.333	1.069	.863	.689	.534	.392	.257	.128
18	2.878	2.552	2.101	1.734	1.330	1.067	.862	.688	.534	.392	.257	.127
19	2.861	2.539	2.093	1.729	1.328	1.066	.861	.688	.533	.391	.257	.127
20	2.845	2.528	2.086	1.725	1.325	1.064	.860	.687	.533	.391	.257	.127
21	2.831	2.518	2.080	1.721	1.323	1.063	.859	.686	.532	.391	.257	.127
22	2.819	2.508	2.074	1.717	1.321	1.061	.858	.686	.532	.390	.256	.127
23	2.807	2.500	2.069	1.714	1.319	1.060	.858	.685	.532	.390	.256	.127
24	2.797	2.492	2.064	1.711	1.318	1.059	.857	.685	.531	.390	.256	.127
25	2.787	2.485	2.060	1.708	1.316	1.058	.856	.684	.531	.390	.256	.127
26	2.779	2.479	2.056	1.706	1.315	1.058	.856	.684	.531	.390	.256	.127
27	2.771	2.473	2.052	1.703	1.314	1.057	.855	.684	.531	.389	.256	.127
28	2.763	2.467	2.048	1.701	1.313	1.056	.855	.683	.530	.389	.256	.127
29	2.756	2.462	2.045	1.699	1.311	1.055	.854	.683	.530	.389	.256	.127
30	2.750	2.457	2.042	1.697	1.310	1.055	.854	.683	.530	.389	.256	.127
∞	2.576	2.326	1.960	1.645	1.282	1.036	.842	.674	.524	.385	.253	.126

The probability of a deviation *numerically* greater than t is twice the probability given at the head of the table.

* This table is reproduced from "Statistical Methods for Research Workers," with the generous permission of the author, Professor R. A. Fisher, and the publishers, Messrs. Oliver and Boyd.

$$\mu = \bar{x}_k \pm 1.96\ s/\sqrt{N-1}$$

$$= \bar{x}_k \pm 1.96\sqrt{\sum x_k^2 - N\bar{x}_k^2} \Big/ (N-1)$$

This allows us to examine the range of μ which falls within the confidence limits of every run.

5.5 Validation results.

Altogether eight runs of the model were executed, the last of these having an order of magnitude longer run time. Each run produced the normal SHAPE statistics, as described earlier. The cut statistics were used to calculate the t-test values as described in section 5.4. For each message type the terminal node's row corresponding to the message arrival INarc was tagged (the INarc specifier set negative) for cut statistics accumulation. Consequently for each message type the cut statistics COUNT, AVGDUR, and DURVAR (corresponding to n_i, $\bar{x}_i$, and $\sum_j x_{ij}^2$) were output.

From these the t value for the run can be calculated and these are given in Fig. 5-13. In all the runs the t value fell within the acceptance limits at the 95% confidence level ($|t| < 1.96$). If we had observed a run which gave a value of t outside these limits we should be forced to reject the null hypothesis H_0 that

$$\bar{x}_k = 0$$

The t values shown in Fig. 5-13 allow us to accept it. This is equivalent to accepting the hypothesis that the mean message delay observed in the model is equal to

$$T = 0.0447767$$

from the derivation of $\bar{x}_k$. Consequently we consider the runs described as constituting a validation of the SHAPE model. The confidence limits of $\bar{x}_k$ can be written as

$$\bar{x}_k \pm L$$

and we have given $\bar{x}_k$ and L for each run in Fig. 5-13. Since we accepted H_0 for each run the value zero lies within the confidence limits of every run. We have plotted these limits in Fig. 5-15, and shown the interval (a,b) common to all of them.

	t	$\bar{x}_k \times 10^{-3}$	$L \times 10^{-3}$
RUN 1	0.9036	0.7257	1.5740
RUN 2	-0.7993	-0.8514	2.0877
RUN 3	0.1156	0.0933	1.5809
RUN 4	-1.0713	-1.1047	2.0212
RUN 5	0.5919	0.6949	2.3013
RUN 6	-1.0364	-1.1365	2.1493
RUN 7	-1.5115	-1.5652	2.0297
RUN 8*	0.9515	0.3500	0.7209
MEAN		-0.3492	

*long run

Fig. 5-13. Values for t-test and $\bar{x}_k$.

MESSAGE TYPE	NY/CH	NY/HO	CH/HO	NY/OE	CH/DE	HO/DE	NY/LA	HO/LA	CH/LA	DE/LA
Z_1	.02276	.03647	.07928	.06040	.03764	.19836	.07459	.03812	.07807	.04043
RUN 1	.02239	.03623	.07831	.06468	.03379	.23200	.07954	.03653	.07791	.04200
RUN 2	.02242	.03618	.07958	.05125	.03429	.19990	.07225	.03756	.08000	.03516
RUN 3	.02359	.03631	.07738	.06186	.03594	.23590	.07871	.03674	.06957	.04228
RUN 4	.02329	.03536	.06920	.05837	.03342	.19250	.07427	.03549	.07398	.03927
RUN 5	.02299	.03561	.06863	.05944	.03599	.20050	.08553	.03878	.07191	.04251
RUN 6	.02204	.03301	.07187	.05046	.03364	.17340	.07607	.03337	.08085	.03881
RUN 7	.02333	.04011	.08070	.05401	.03247	.16980	.06779	.03421	.07396	.04250
RUN 8*	.02238	.03558	.08051	.06118	.03723	.17550	.07770	.03695	.07948	.04152
MEAN	.02280	.03605	.07576	.05766	.03460	.19744	.07648	.03620	.07596	.04051

*long run

Fig. 5-14. Mean Message Delay by Message type for eight runs.

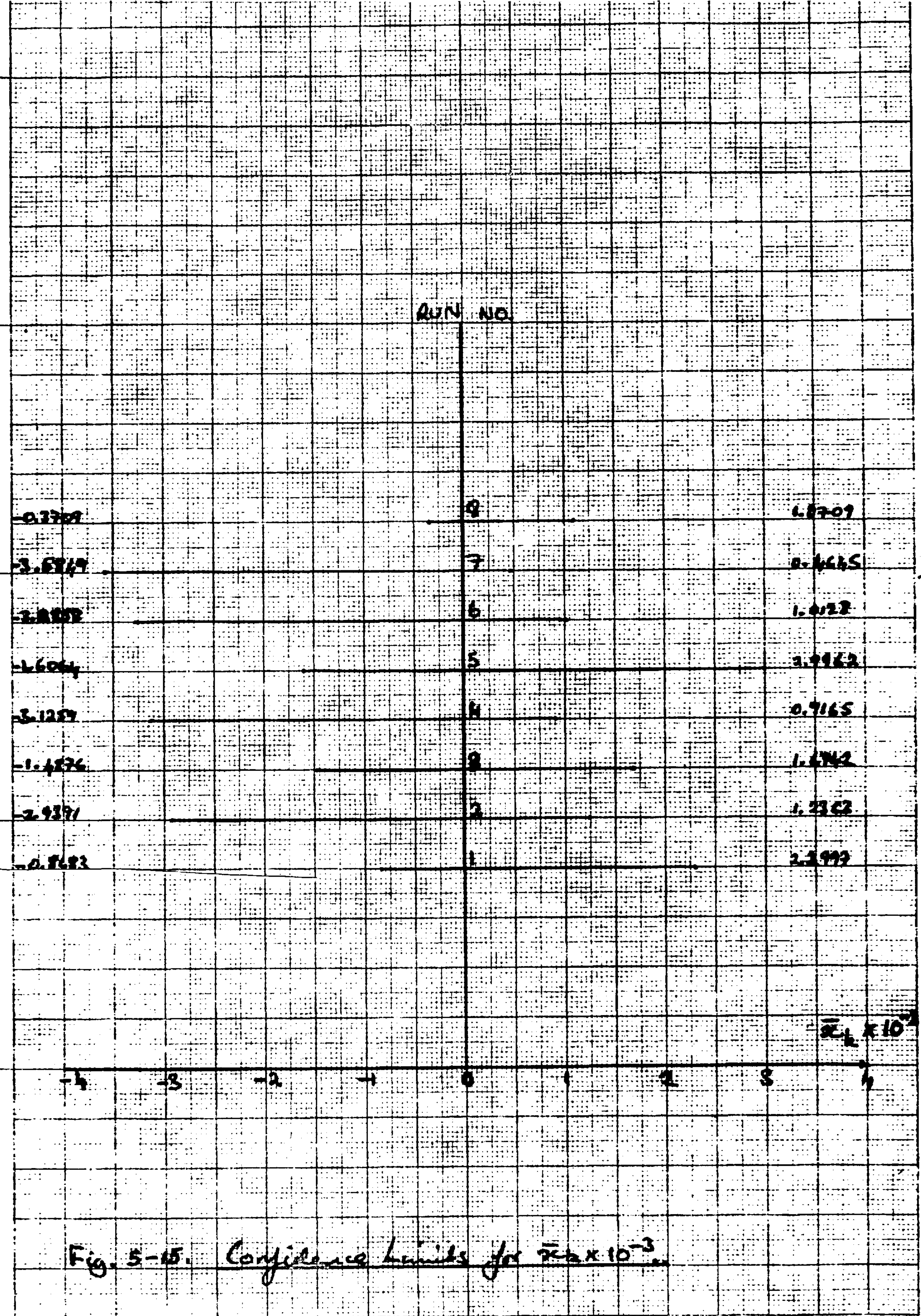


Fig. 5-15. Confidence limits for $\bar{x}_k \times 10^{-3}$.

The relative smallness of a and b further supports the null hypothesis.

In Fig. 5-14 we have given the observed mean message delay for each type of message. These show a correspondence with the theoretically expected values which is closest for the most frequent messages. We have also shown the mean over the eight runs for each type.

Using the delay and the COUNT for each message type we can calculate the overall observed mean message delay.

$$T = \sum_i AVGDUR_i \times COUNT_i / \sum_i COUNT_i$$

However, since the message lengths are drawn in a random fashion, the average observed message length is usually slightly different from 0.1 for each type. In consequence the mean message transmission time will differ from the expected valye. This is a component of the mean message delay and so perturbs the delay from what it would have been if the average length of each message type was 0.1.

If the queueing time is not large compared with its transmission time we can make a first order correction for this effect by normalizing the mean delay of each type of message with respect to its mean length, giving

$$T_m = \sum_i COUNT_i \times AVGDUR_i \times \frac{0.1}{AVGLAM_i} \Bigg/ \sum_i COUNT_i$$

A similar problem occurs because the number of messages of each type generated during a run will differ from the theoretically expected number (which we will call $ECOUNT_i$). We can again make a correction for this by using $ECOUNT_i$ instead of $COUNT_i$ in the above expression giving a mean delay

$$T_n = \sum_i ECOUNT_i \times AVGDUR \times \frac{0.1}{AVGLAM_i} \Bigg/ \sum_i ECOUNT_i$$

We give values for T, T_m and T_n in Fig. 5-16. The last run of the model had a considerably longer run time than the others. In this run the values of T and $\bar{x}_k$ were much closer to the expected values, and the confidence interval smaller. This demonstrates the convergence towards expected values with longer run time and supports the claim to validity based on the t-test results.

	$T \times 10^2$	$T_m \times 10^{-2}$	$T_n \times 10^1$
RUN 1	4.506	4.437	4.486
RUN 2	4.313	4.541	4.631
RUN 3	4.492	4.431	4.428
RUN 4	4.370	4.650	4.644
RUN 5	4.588	4.538	4.492
RUN 6	4.508	4.576	4.429
RUN 7	4.325	4.528	4.529
RUN 8*	4.489	4.517	4.542
MEAN	4.449	4.527	4.523

*long run

Fig. 5-16. Mean Message Delay.

CHAPTER VI

APPLICATION

6.1 A UK link to the ARPA network.

This chapter deals with the application of the SHAPE system to a particular design problem, namely the behaviour of a linkage between computers in the United Kingdom and the ARPA network. The choice of an example from the field of computer networking is a natural consequence of our belief that this is the direction which the mainstream of computing will take in the future. In Chapter I we suggested that the search for greater computing power must sooner or later require coordination of dispersed facilities in order to solve problems too large for a single computer to undertake. The ARPA netwrok is certainly a first step towards this goal, since it provides both communication between computers and user access to all resources available in the network. Its extension to the United Kingdom via Oslo is therefore of great interest to us. Application of the SHAPE system to this link is also particularly attractive since we have been closely associated with the research team working on this project. The association has given us an intimate knowledge of the design and operation of the ARPA network and its extension to the UK.

In the sections which follow we describe and analyse the characteristics of the link. A model is constructed with the SHAPE system and used to observe the behaviour of the link under various conditions.

The ARPA network provides store and forward communications between the set of computers shown in Fig. 6-1. The computers located at the various nodes are drawn from a variety of manufacturers, and most are incompatible both in hardware and

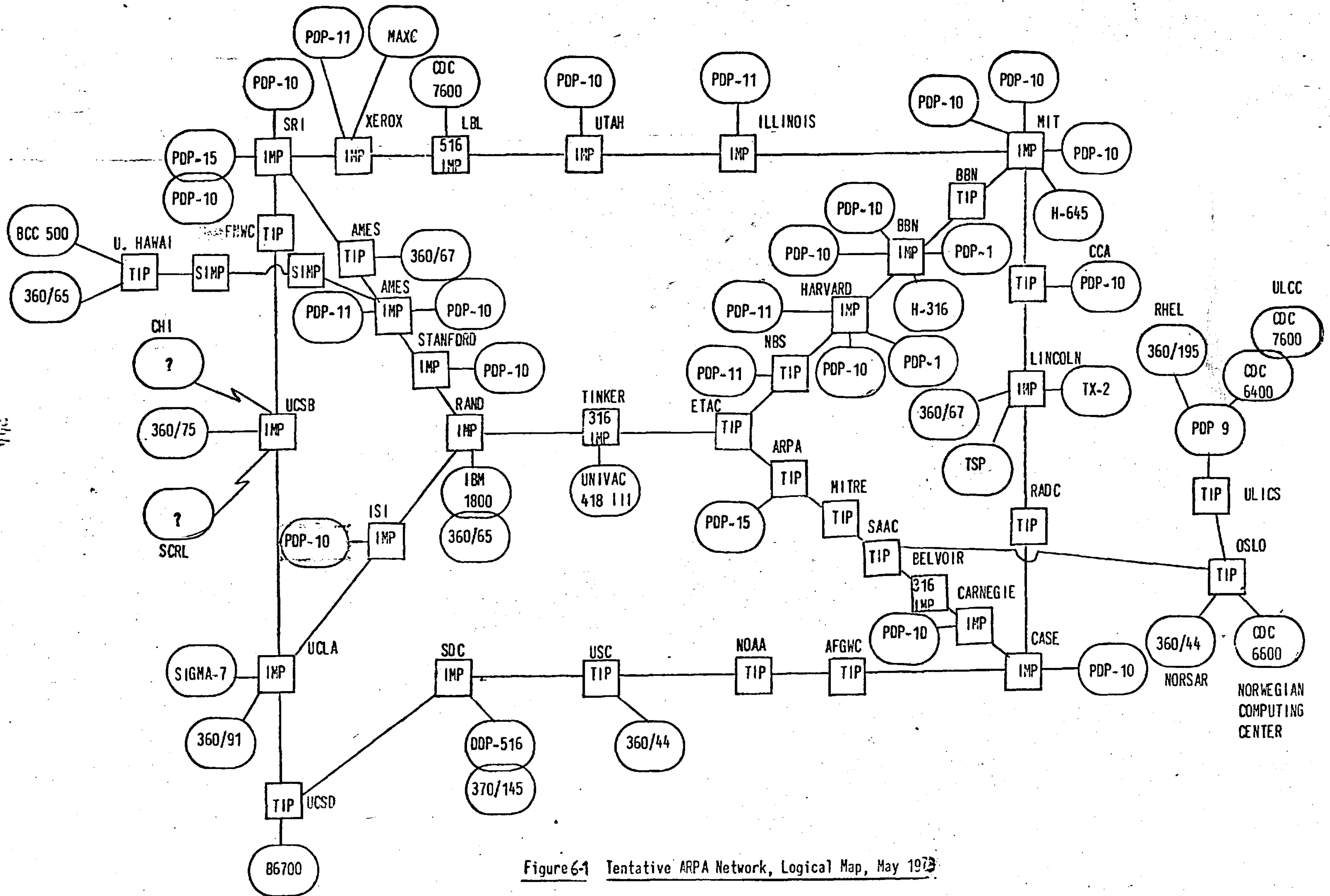


Figure 6-1 Tentative ARPA Network, Logical Map, May 1973

software. The network has to provide communications for this set of machines, and allow effective use to be made of them from any point in the network.

An underlying constraint placed on the design of the network was that its operating procedures would not interfere in any significant way with the operation of the existing facilities which were to be connected. Consequently the message handling tasks are carried out in a dedicated Interface Message Processor (IMP) situated at the site of the computer to be connected (HOST). In most cases the communications channels are 50 kilobit per second full duplex telephone lines and these run between IMPs. An IMP modified to directly support terminals is called a TIP.

In order to provide reliability there are at least two paths through the network for every origin-destination pair. A 24 bit cyclic checksum is provided for each block of data, and the IMP is a ruggedized computer with a mean time between failures of 10,000 hours. TIPs, however, are not currently ruggedized. Messages which flow between HOSTs are broken up into packets, each of maximum size approximately 1,000 bits. There can be up to eight packets in a message, which is assembled and disassembled by the IMPs. The packets make their way individually through the IMP network where appropriate routing procedures direct the traffic flow.

A positive acknowledgment is expected within a given time period for each inter-IMP packet transmission. In the absence of an acknowledgment the transmitting IMP will repeat the transmission (perhaps over the same channel or over a suitable alternative). This process is repeated a number of times after which the communication channel is regarded as unavailable.

Absence of an acknowledgment may indicate, for example, that the message contained errors on receipt or that no more buffer space is available in the receiving IMP.

There may be up to 64 dialogues occuring at any one HOST. The dialogues take place along two logical communications channels called links. A HOST will send a message along the outward link of a dialogue and then await a Request For Next Message (RFNM) on the inward link.

In those cases where a user is making more or less direct use of a remote software system, the network is intended to provide a total round-trip delay which does not exceed the human short term memory span of one to two seconds. In the design of the network it was also considered desirable that the response should be comparable, if possible, to using a remote display console over a private voice grade line where a 50 character line of text can be sent in 0.2 seconds.

The linkage to Europe consists of a telecommunications channel between a TIP in the United States and one in Oslo, which is in turn connected to a TIP in London. The Oslo TIP will have at least one HOST and the London one will have a PDP9 computer as a pseudo-HOST. It is intended to interface two other computers in the UK to the network via the PDP9, which is situated at the Institute of Computer Science. These are a CDC 6400 computer at the University of London Computing Centre, and an IBM 360/195 at the Rutherford High Energy Laboratory. Each computer is expected to support a cluster of interactive users, as well as performing some file transmers. The configuration is shown if Fig. 6-1. Delivery of the TIPs is currently scheduled for the third quarter of 1973,

and the link is expected to be operational by the end of the year. The transatlantic channel will be via satellite.

6.2 Analysis of the link.

In this section we discuss the detailed structure and operation of the link, and so extract the features we wish to include in the model. One of our main concerns is the average response time as seen by a European interactive user of the ARPA network. This is partly made up of the message transmission delays introduced by the link channels. The structure of the link is shown in Fig. 6-2. The capacity of channel i is C_i full duplex and the number of interactive users at the terminal node is N_i.

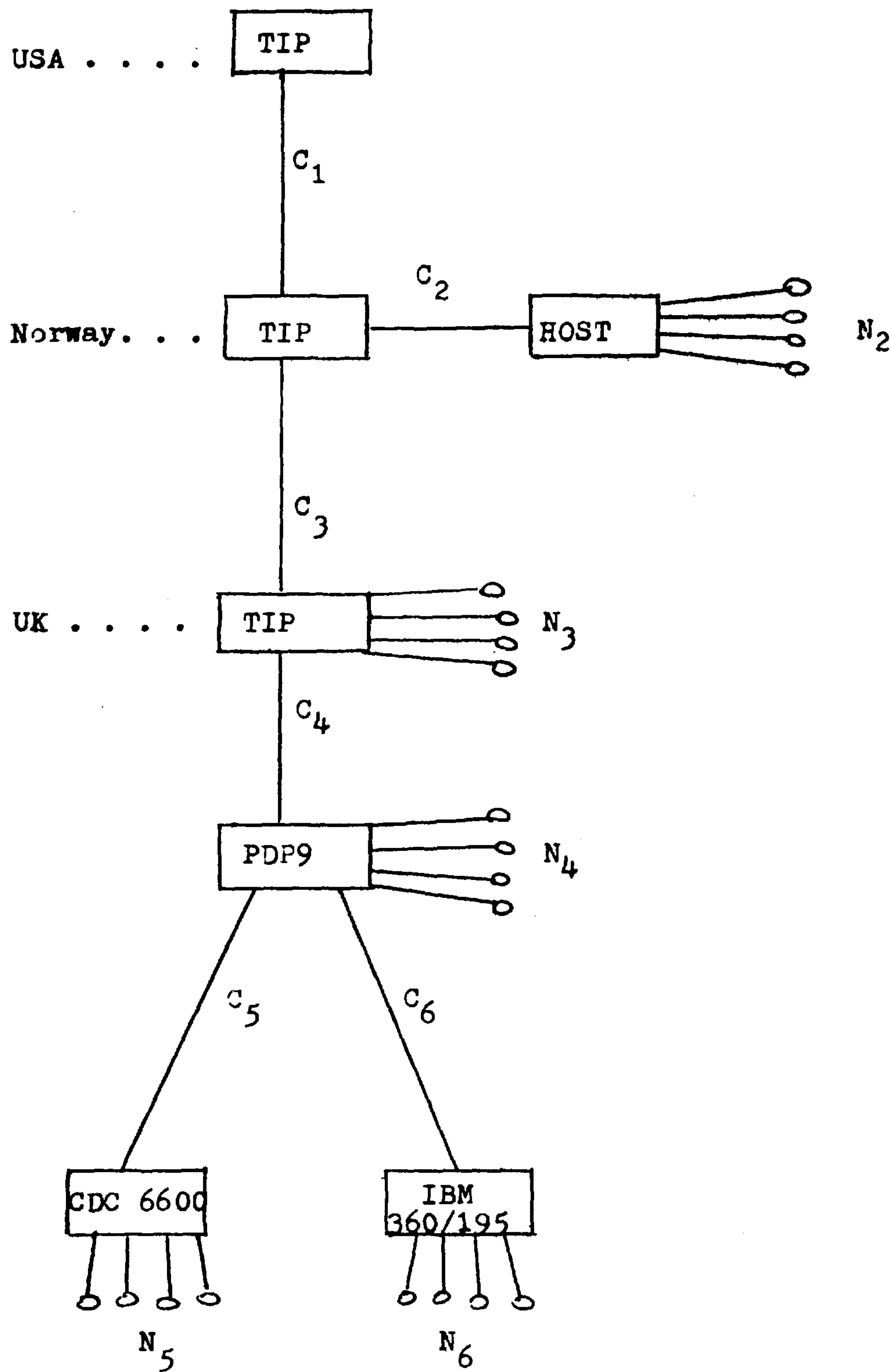
The ARPA network is designed to give a mean message delay of 0.2 seconds. Satellite transmission on channel 1 gives a propagation delay of approximately 0.25 seconds. If the remote HOST in the United States gives an immediate reply (e.g. an echo) to a user message, then the average response time as seen by a European user will be

$$R = 0.9 + 2T_a + K$$

where T_a is the average delay in the link subnet for a sample of messages of mean length a, and K is the sum of the time for a teletype to transmit carriage return to its node plus the time for the first character of the reply to reach the teletype from its node. Response time is therefore the interval between the user typing carriage return, and the first character of his reply being printed. K is approximately 0.2 seconds if the teletype line operates at 110 bits per second so that

$$R = 1.1 + 2T_a$$

This suggests that a reasonable design range for T_a would be 0.1 to 0.8 seconds, making R between 1.3 (good) and 2.7 (tolerable) seconds.



C_i is the capacity of channel i (full-duplex).

N_i is the number of interactive users in a cluster.

Fig. 6-2. UK-ARPANET Linkage.

We define a as the mean length of user messages in the subnet. Inter-TIP acknowledgments (ACK) and requests from the destination for the next message from a user (RFNM) are 150 bits long. A message originating in Clusters 5 or 6 will be transmitted to the PDP9 (node 4). Here the PDP9, behaving as a HOST, will introduce the message into the ARPA network. When the message is successfully received at its destination a RFNM will be transmitted to the PDP9. This in turn must request the next message from the node which produced the original one. In this way there will be RFNM-like traffic on channels 5 and 6. We estimate that these pseudo-RFNMs will have a length of 100 bits.

Since the users are interactive we can say that, with very few exceptions, the length of a user message will be less than 1,000 bits. Consequently they will be transmitted as single packets within the ARPA network. Each packet carries a total overhead of 150 bits, so that on channels 1 and 3 a user message will have length a + 150.

We now consider the traffic pattern in the subnet, that is to say the number of messages per second between each cluster and the US TIP. We have assumed that the quantity of traffic moving between nodes of the subnet itself will be negligible. On the basis of current knowledge, the best estimates for the average number of active users at the nodes are $N_2 = 6$, $N_3 = 12$, $N_4 = N_5 = N_6 = 4$. These figures determine the proportional traffic. The total volume of messages (assuming all users to exhibit similar behaviour) depends on the number of messages per second (L) that a user will generate. In what follows we deal with the case $L = 1/30$, although in the model L is a variable parameter.

When an interactive user sends a message to the US, he will normally be taking part in a dialogue with a remote computer. Consequently his message will give rise to a message from this computer in reply. Therefore we expect the traffic pattern inward to be symmetrical with that outward from the US. In that case the numbers of RFNMs and messages per second travelling inwards will be the same. Similarly, on channels 1 and 3 there will be ACKs travelling inward for each outward bound message and RFNM. For example, if the subnet generates 30 L messages per second to the US, we would also expect 30 L RFNMs and 60 L ACKs inward per second on channel 1. If L_i is the number of packets (messages, RFNMs and ACKs) inwards per second on channel i, then these are tabulated in Fig. 6-3.

On channels 1 and 3 for each message (length a + 150) we have a RFNM and two ACKs each of length 150 bits so that the average length is (a + 600)/4 bits. On channels 2, 4, 5 and 6 for each message (length a) we have a RFNM of 100 bits, so that the average packet length is (a + 100)/2 bits. We define the channel loading P_i as the average number of bits per second transmitted on the channel divided by its capacity.

$$P_i = \text{avg. packet length} \times L_i/C_i$$

We define T_i to be the mean message delay on channel i. As in Kleinrock's treatment [KLEI 70A] we regard T_i as having two main components. The first is the mean message transmission time, namely mean message length divided by channel capacity. The second component is the mean waiting time for a message. This is derived from the true total loading of the channel, i.e. including ACKs and RFNMs. If m_i and s_i are the mean message and packet lengths respectively we can write,

Channel	L_1	L_1 (L = 1/30)
1	120 L	4
2	12 L	2/5
3	96 L	16/5
4	24 L	4/5
5	8 L	4/15
6	8 L	4/15

Fig. 6-3. Traffic on each channel of the subnet.

$$P_i = s_i \times L_i/C_i$$

$$T_i = m_i/C_i + s_i/C_i \times P_i/(1-P_i)$$

These variables are tabulated for the case L = 1/30 in Fig. 6-4. We can calculate the mean user message delay in the subnet when the messages have mean length a from

$$T_a = (30T_1 + 6T_2 + 24T_3 + 12T_4 + 4T_5 + 4T_6)/30$$

If all the C_i are variable this gives us a six dimensional solution space, or five dimensional if the C_i have a constant sum. While a solution is feasible it may require a considerable computation. In the case we are considering, four of the six channels already have fixed capacities allocated. C_5 and C_6 are 2.4 Kb, and C_2 and C_4 are 50 Kb. This leaves C_1 and C_3 to take on one of the following possible values, namely 4.8, 7.2, 9.6 or 50 Kb. We now choose a hypothetical mean message length for the purposes of investigation. It has been observed that the mean message length of actual traffic in the ARPA network is close to 600 bits. Without foreknowledge it seems most probably that European traffic will be similar, and so we take this value as our starting point. If a = 600, we can calculate T_a from the T_i, and these results are summarized in Fig. 6-5. They show that all available combinations of C_1 and C_3 fall within the design range advanced above, but that the good response case requires values of 50 Kb for each.

The square root channel capacity assignment which is optimal for regular store and forward networks, can be instructively applied to the case we are considering when all the C_i are allowed to vary. The square root assignment is optimal so long as the packet traffic on each channel has the same mean length.

Channel	P_i
1	$(a + 600)/C_1$
2	$(a + 100)/5C_2$
3	$4(a + 600)/5C_3$
4	$2(a + 100)/5C_4$
5	$2(a + 100)/15C_5$
6	$2(a + 100)/15C_6$

Channel	T_i
1	$(a + 150)/C_1 + \frac{1}{4}(a + 600)\ P_1/(1-P_1)C_1$
2	$a / C_2 + \frac{1}{2}(a + 100)\ P_2/(1-P_2)C_2$
3	$(a + 150)/C_3 + \frac{1}{4}(a + 600)\ P_3/(1-P_3)\ C_3$
4	$a / C_4 + \frac{1}{2}(a = 100)\ P_4/(1-P_4)C_4$
5	$a / C_5 + \frac{1}{2}(a + 100)\ P_5/(1-P_5)C_5$
6	$a / C_6 + \frac{1}{2}(a + 100)\ P_6/(1-P_6)C_6$

Fig. 6-4. Values of T_i and P_i for $L = 1/30$.

C_1 \ C_3	4.8	7.2	9.6	50
4.8	.37	.33	.30	
7.2	.33	.27	.25	
9.6	.29	.25	.21	.17
50			.15	.10

Fig. 6-5. Values of T_{600} for various combinations of C_1 and C_3.

Examining the average packet lengths we see that when a = 400 all of them are equal to 250 bits.

Consequently we can get some idea of the channel capacities which would be required for a mean message length of 600 bits by applying square root assignment to the network for messages of average length 400 bits (mean packet length 250 bits). If a is the mean message length in the subnet we define b as the corresponding mean packet length, giving

$$b = (2a + 875)/6.7$$

This is plotted in Fig. 6-6. We calculate the mean path length $\bar{n}$ as the average path length of all packet journeys, weighted by number of packets. This gives us

$$\bar{n} = (4\times 8L + 4\times 8L + 3\times 8L + 2\times 24L + 1\times 48L + 1\times 60L)/168L$$

$$= 67/42$$

The network loading is $P = 168/30C$ when $L = 1/30$, giving

$$\bar{n}P = 6700/3C$$

If we define S as the sum of the L_i, $S = \sum_i L_i$

then we can use the following of Kleinrock's results:

$$T_b = \bar{n}\ (\sum \sqrt{L_i/S})^2 b/\ C(1-\bar{n}P)$$

$$C_i = bL_i + C(1-\bar{n}P)\sqrt{L_i}/\sum\sqrt{L_i}$$

This allows us to write

$$T_b = \frac{67}{42} \cdot \frac{1210}{268} \cdot b/(C - 6700/3)$$

or

$$C = 250\ (268L + 605/84T_b)$$

and

$$C_i = 250\ (L_i + \frac{605}{84\times 35} \cdot \frac{1}{T_b} \cdot \sqrt{L_i/L})$$

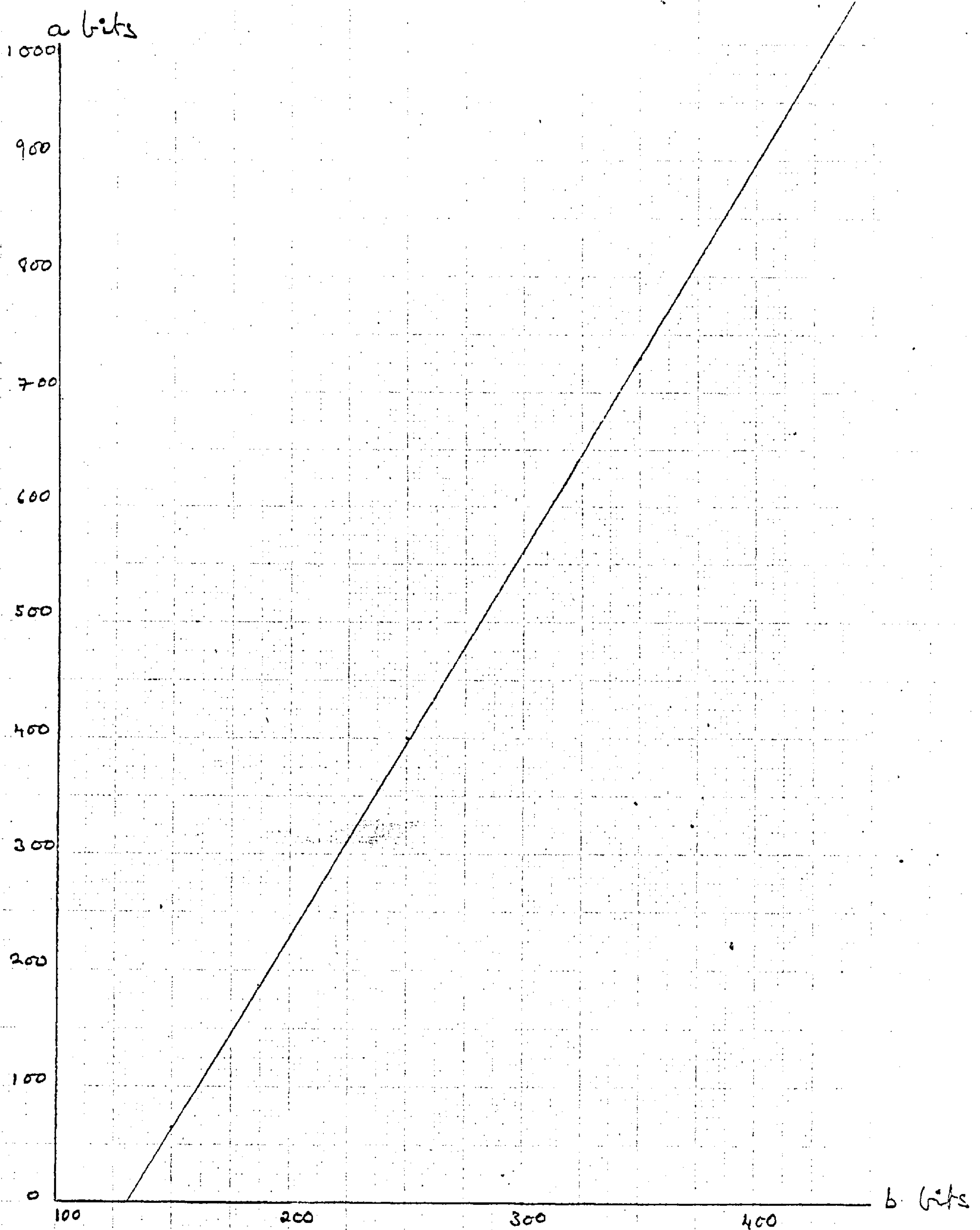


FIG. 6-6. GRAPH OF MEAN MESSAGE LENGTH AGAINST MEAN PACKET LENGTH IN SUBNET.

For any T_b we have a corresponding value of C and consequently of the C_i. From these we can calculate T_a using the equations of Fig. 6-4. We have tabulated these values in Figs. 6-7 and 6-8. A graph of T_b and T_a against is shown in Fig. 6-9.

From these values we can see that if T_a is to fall in the range 0.1 to 0.8 seconds when a = 400 the corresponding values of T_b, C_1 and C_i are as shown below. We may also notice that as C becomes large the ratio T_a/T_b approaches a constant (approx. 3.6) as we would expect.

$$0.1 < T_a < 0.8$$
$$0.03 < T_b < 0.24$$
$$62.4 \text{ Kb} < C < 9.75 \text{ Kb}$$
$$19.8 \text{ Kb} < C_1 < 3.35 \text{ Kb}$$
$$6.0 \text{ Kb} < C_2 < 0.84 \text{ Kb}$$
$$17.6 \text{ Kb} < C_3 < 2.90 \text{ Kb}$$
$$8.7 \text{ Kb} < C_4 < 1.26 \text{ Kb}$$
$$4.9 \text{ Kb} < C_5 < 0.67 \text{ Kb}$$
$$4.9 \text{ Kb} < C_6 < 0.67 \text{ Kb}$$

We can see from these figures that a = 400 we will get at least acceptable response as long as all the C_i have values greater than those shown in the right hand column. In the case being considered C_2 and C_4 may have values up to 50 Kb which is certainly adequate. C_5 and C_6 are 2.4 Kb which is sufficient for acceptable response, though not enough for good response. This lack can of course be compensated for by increasing C_1 and C_3 above their left hand column figures. We see that C_1 and C_3 should be of comparable size, with C_1 slightly greater than C_3.

While the ranges for the C_i are based on a = 400 we can make some estimate of the capacities required to give the same response

TB	TA	CT
.1000E-01	.3635E-01	.1826E+06
.2000E-01	.7229E-01	.9243E+05
.3000E-01	.1078E+00	.6237E+05
.4000E-01	.1430E+00	.4733E+05
.5000E-01	.1779E+00	.3831E+05
.6000E-01	.2124E+00	.3230E+05
.7000E-01	.2465E+00	.2800E+05
.8000E-01	.2804E+00	.2478E+05
.9000E-01	.3139E+00	.2228E+05
.1000E+00	.3472E+00	.2027E+05
.1100E+00	.3802E+00	.1863E+05
.1200E+00	.4129E+00	.1727E+05
.1300E+00	.4454E+00	.1611E+05
.1400E+00	.4776E+00	.1512E+05
.1500E+00	.5095E+00	.1426E+05
.1600E+00	.5413E+00	.1351E+05
.1700E+00	.5727E+00	.1284E+05
.1800E+00	.6040E+00	.1226E+05
.1900E+00	.6351E+00	.1173E+05
.2000E+00	.6660E+00	.1125E+05
.2100E+00	.6966E+00	.1082E+05
.2200E+00	.7271E+00	.1043E+05
.2300E+00	.7574E+00	.1008E+05
.2400E+00	.7875E+00	.9750E+04
.2500E+00	.8174E+00	.9449E+04
.2600E+00	.8472E+00	.9171E+04
.2700E+00	.8768E+00	.8914E+04
.2800E+00	.9062E+00	.8676E+04
.2900E+00	.9355E+00	.8454E+04
.3000E+00	.9646E+00	.8246E+04

Fig. 6-7. Values of T_a and C for various T_b.

C1	C2	C3	C4	C5	C6
.5741E+05	.1794E+05	.5127E+05	.2569E+05	.1463E+05	.1463E+05
.2921E+05	.9020E+04	.2604E+05	.1295E+05	.7349E+04	.7349E+04
.1980E+05	.6047E+04	.1762E+05	.8697E+04	.4921E+04	.4921E+04
.1510E+05	.4560E+04	.1342E+05	.6573E+04	.3708E+04	.3708E+04
.1228E+05	.3668E+04	.1089E+05	.5298E+04	.2979E+04	.2979E+04
.1040E+05	.3073E+04	.9212E+04	.4449E+04	.2494E+04	.2494E+04
.9059E+04	.2649E+04	.8010E+04	.3842E+04	.2147E+04	.2147E+04
.8052E+04	.2330E+04	.7109E+04	.3387E+04	.1887E+04	.1887E+04
.7268E+04	.2082E+04	.6408E+04	.3032E+04	.1685E+04	.1685E+04
.5641E+04	.1884E+04	.5847E+04	.2749E+04	.1523E+04	.1523E+04
.6128E+04	.1722E+04	.5388E+04	.2517E+04	.1391E+04	.1391E+04
.5701E+04	.1587E+04	.5006E+04	.2324E+04	.1280E+04	.1280E+04
.5339E+04	.1472E+04	.4682E+04	.2161E+04	.1187E+04	.1187E+04
.5030E+04	.1374E+04	.4405E+04	.2021E+04	.1107E+04	.1107E+04
.4761E+04	.1289E+04	.4165E+04	.1899E+04	.1037E+04	.1037E+04
.4526E+04	.1215E+04	.3954E+04	.1793E+04	.9768E+03	.9768E+03
.4318E+04	.1149E+04	.3769E+04	.1700E+04	.9232E+03	.9232E+03
.4134E+04	.1091E+04	.3604E+04	.1616E+04	.8756E+03	.8756E+03
.3969E+04	.1039E+04	.3456E+04	.1542E+04	.8330E+03	.8330E+03
.3821E+04	.9920E+03	.3323E+04	.1475E+04	.7947E+03	.7947E+03
.3686E+04	.9495E+03	.3203E+04	.1414E+04	.7600E+03	.7600E+03
.3564E+04	.9109E+03	.3094E+04	.1359E+04	.7285E+03	.7285E+03
.3453E+04	.8756E+03	.2994E+04	.1308E+04	.6997E+03	.6997E+03
.3351E+04	.8433E+03	.2903E+04	.1262E+04	.6733E+03	.6733E+03
.3257E+04	.8136E+03	.2819E+04	.1220E+04	.6491E+03	.6491E+03
.3170E+04	.7861E+03	.2741E+04	.1180E+04	.6267E+03	.6267E+03
.3089E+04	.7607E+03	.2669E+04	.1144E+04	.6059E+03	.6059E+03
.3015E+04	.7371E+03	.2603E+04	.1110E+04	.5867E+03	.5867E+03
.2945E+04	.7152E+03	.2540E+04	.1079E+04	.5687E+03	.5687E+03
.2880E+04	.6947E+03	.2482E+04	.1050E+04	.5520E+03	.5520E+03

Fig. 6-8. Values of C_i corresponding to T_b shown in Fig. 6-7.

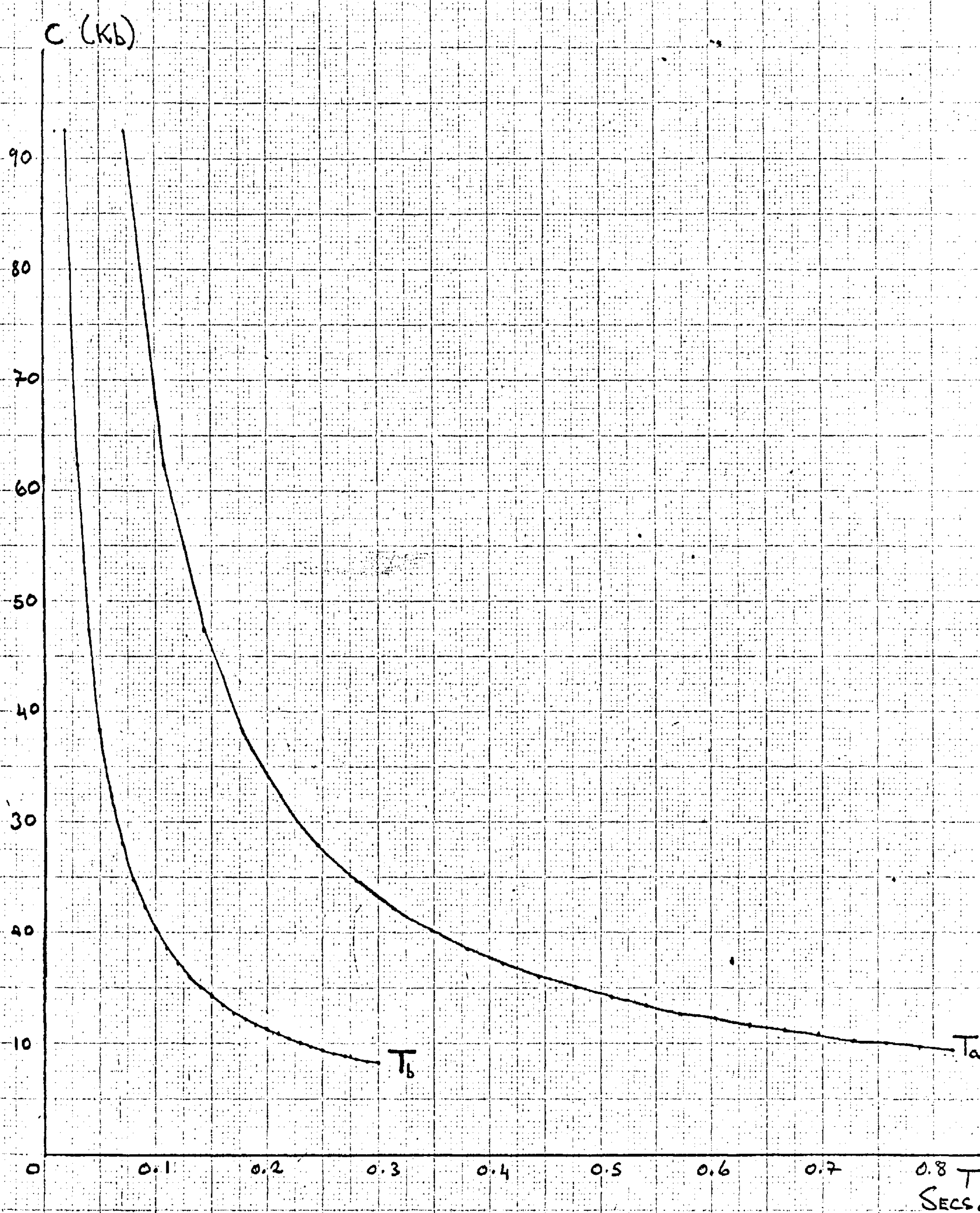


Fig. 6-9. Graph of T_a and T_b against C.

times for a = 600. The estimate would be an increase of not more than 50 per cent, since if a increases from 400 to 600 then b increases from 250 to 310, and we would not expect too severe a perturbation from the regular store and forward situation to which Kleinrock's equations apply. A fifty per cent increase in the C_i would ensure that mean packet transmission times, and therefore queueing delays, were no larger than in the original situation.

If C_2 and C_4 are 50 Kb, well above what is required for optimal assignment in the sense of minimizing the total C required for a given response time, then T_2 and T_4 will be very small indeed. Consequently other C_i may be assigned smaller than the optimal values but still be sufficient to achieve the required mean message response over the subnet.

The relative importance of these factors can be seen from the fact that in the subnet under consideration C_1 and C_3 must be 50 Kb to give a response T_a of 0.1 seconds (see Fig. 6-5). This is well over the 50 per cent increase which might have been expected if C_5 and C_6 were 7.35 Kb (1.5×4.9 Kb). In these last few pages we have tried to show that even where the square root capacity assignment is not strictly applicable, its use as an approximation can provide insight into the factors affecting a subnet.

6.3 <u>The link model.</u>

This section describes the model of the subnet that was developed using the SHAPE system. We shall deal with the hardware and software graphs and give reasons for the structure of each. Finally we examine how the model parameters can be varied.

The hardware graph is shown in Fig. 6-10. It contains a node for each of the computers involved in the subnet. These nodes were initially given a very large storage capacity (9'10 bits) to be effectively infinite. We use x'k to mean x times 10 to the power k. The remaining node data was set to zero, since its effects were not required for the investigation undertaken.
Each full duplex communication channel was represented by two logically and physically distinct PIarcs, running in opposite directions between the nodes at the ends of the channel.
By physically distinct we mean that each PIarc can be separately and simultaneously allocated, as required by the full duplex nature of the channel. This is achieved by giving each PIarc a distinct processor number. Only one set of function characteristics was defined in the PIarcs. This was the transmit function. The execution time corresponds to the time for the channel to transmit one bit, and utilization and efficiency were set to one since the channel must be allocated as a unit, and then transmits at a fixed rate. Typical data is shown in Fig. 6-11. Each line corresponds to a data card, and the formats are described in Appendix IV. The software graph has a set of nodes dealing with traffic from each terminal cluster, and one set which represents the flow of ACKs and RFNMs. The group of packets which corresponds to activity in cluster i is called stream i, so that the behaviour

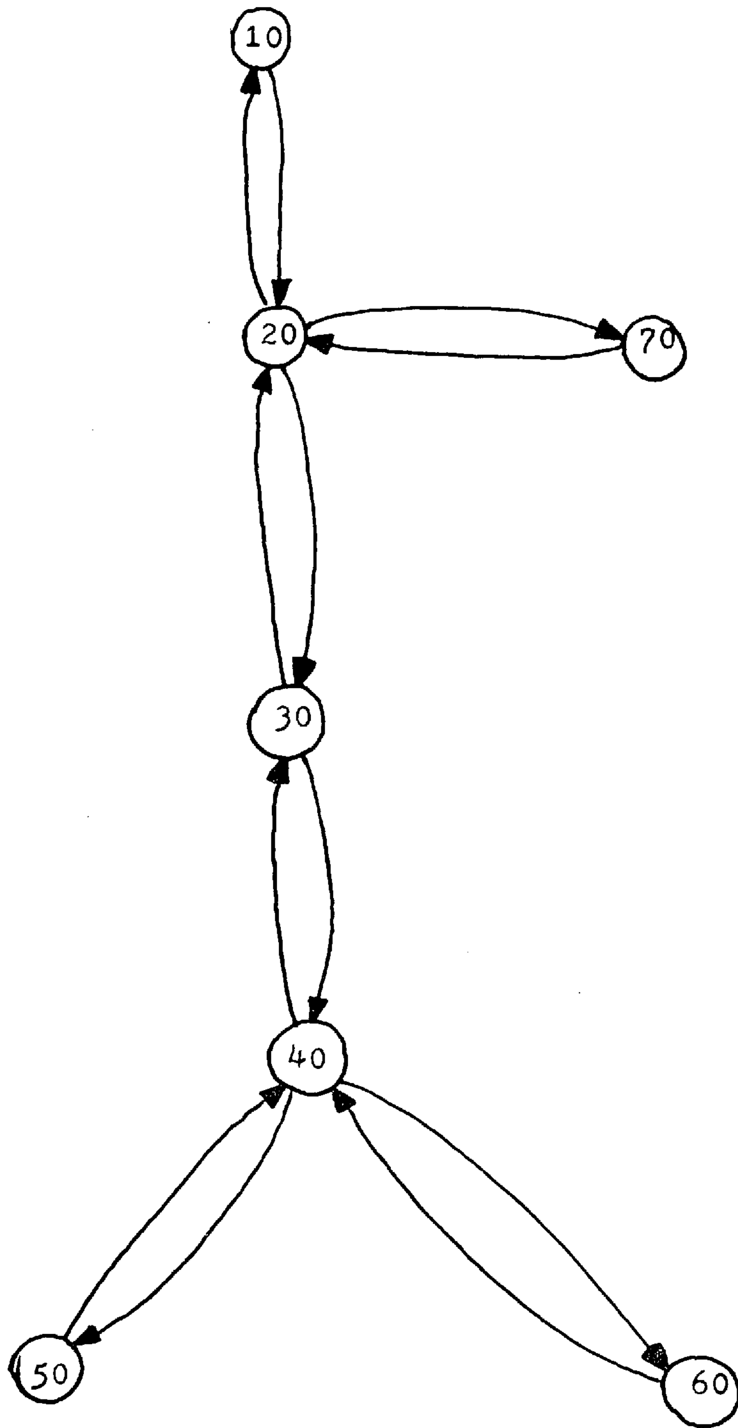


Fig. 6-10. Hardware graph of subnet.

PInode data : 3 inarcs and 3 outarcs, capacity is 9'10 bits.

```
40          3     3
            9'10  0     0     0     0     0
```

PIarc data: processor number is 45, channel capacity is 2.4K = (1/4.167'-4) bits per second, utilization and efficiency are 1, functions 2,3,4 not specified.

```
40    50
            50
            45          100
            4.167'-4     1     1
            -1          -1    -1
            -1          -1    -1
            -1          -1    -1
```

Fig. 6-11. Typical data for hardware graph.

of each stream is modelled by arcs between its set of nodes. Since most of the ACK and RFNM handling is the same for all five streams, this is modelled within a common set of nodes which we may call stream zero. We show the graph structure for streams zero, five, three and seven in Figs. 6-12, 13, 14, 15, 16 and the initialization node (1) in Fig. 6-17. Streams six and four have the same structures as streams five and three respectively.

It would have been possible to combine the activity of all streams at a PInode in a single corresponding SInode. However, this type of node would have been very large, with a high proportion of zero elements. The method we have chosen uses much less storage for the REP matrix elements. As well as this it is a good deal clearer, and more flexible.

The separate modelling of the streams arises as follows. One of the characteristics we wished the model to include was that the flow of messages would be circular. That is to say that a message leaves its cluster, travels through the subnet to the US TIP, is transformed into a reply, returns to the cluster where there is a delay corresponding to the user's think time, and recommences the cycle. This is in effect a cycle of queues, some in common, for each stream and leads to a fixed number of customers within the system once it has been activated. This model corresponds more closely to the real situation in which a number of interactive users participate in dialogues with remote computers. Usually a user will not send a message until he receives a reply to the previous one, often because his next action depends on the reply. Consequently, if response worsens the effect is to decrease the number of messages generated per second.

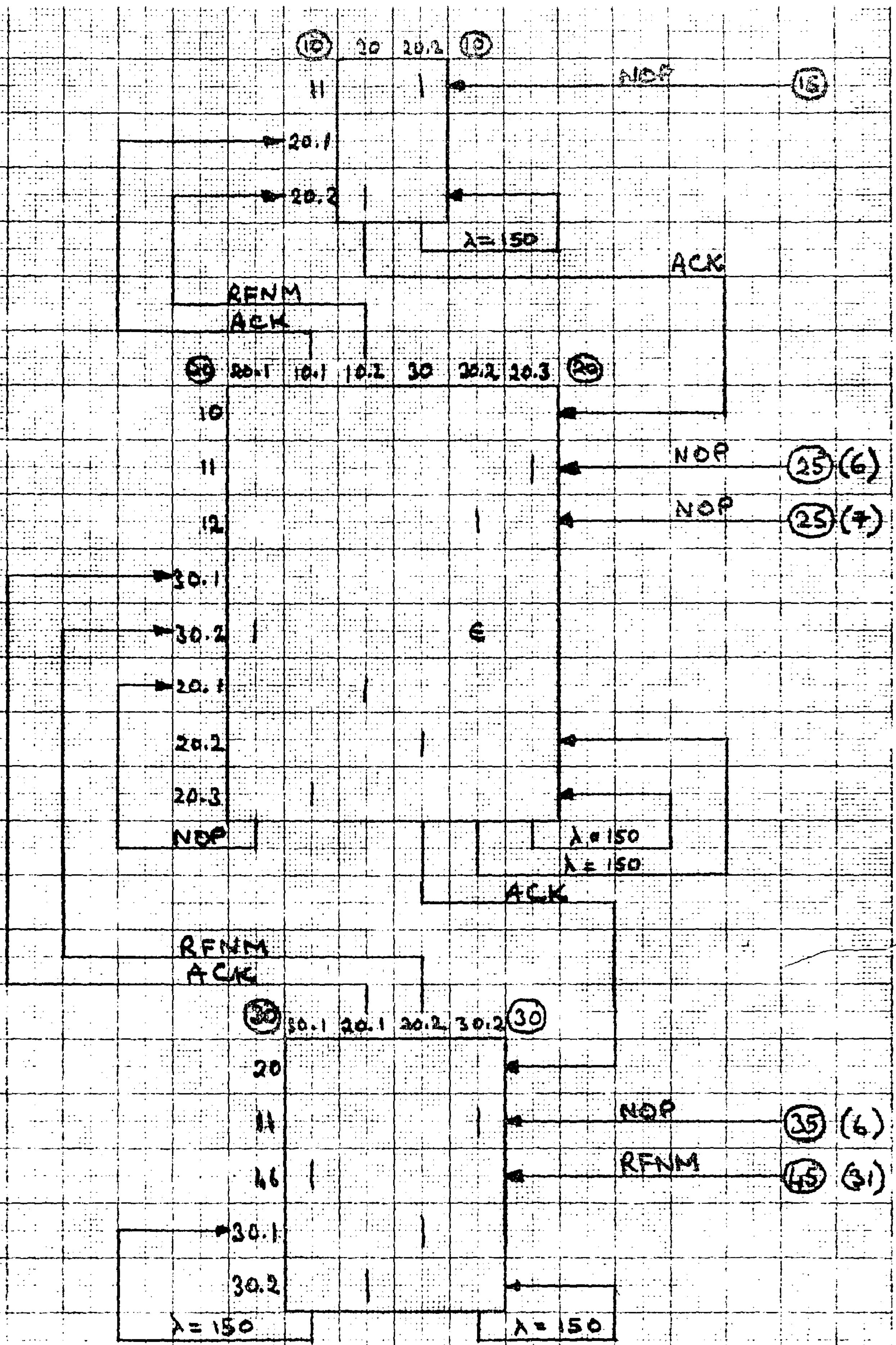


FIG. 6-12. STREAM ZERO. (ϵ IS NEGLIGIBLY SMALL)

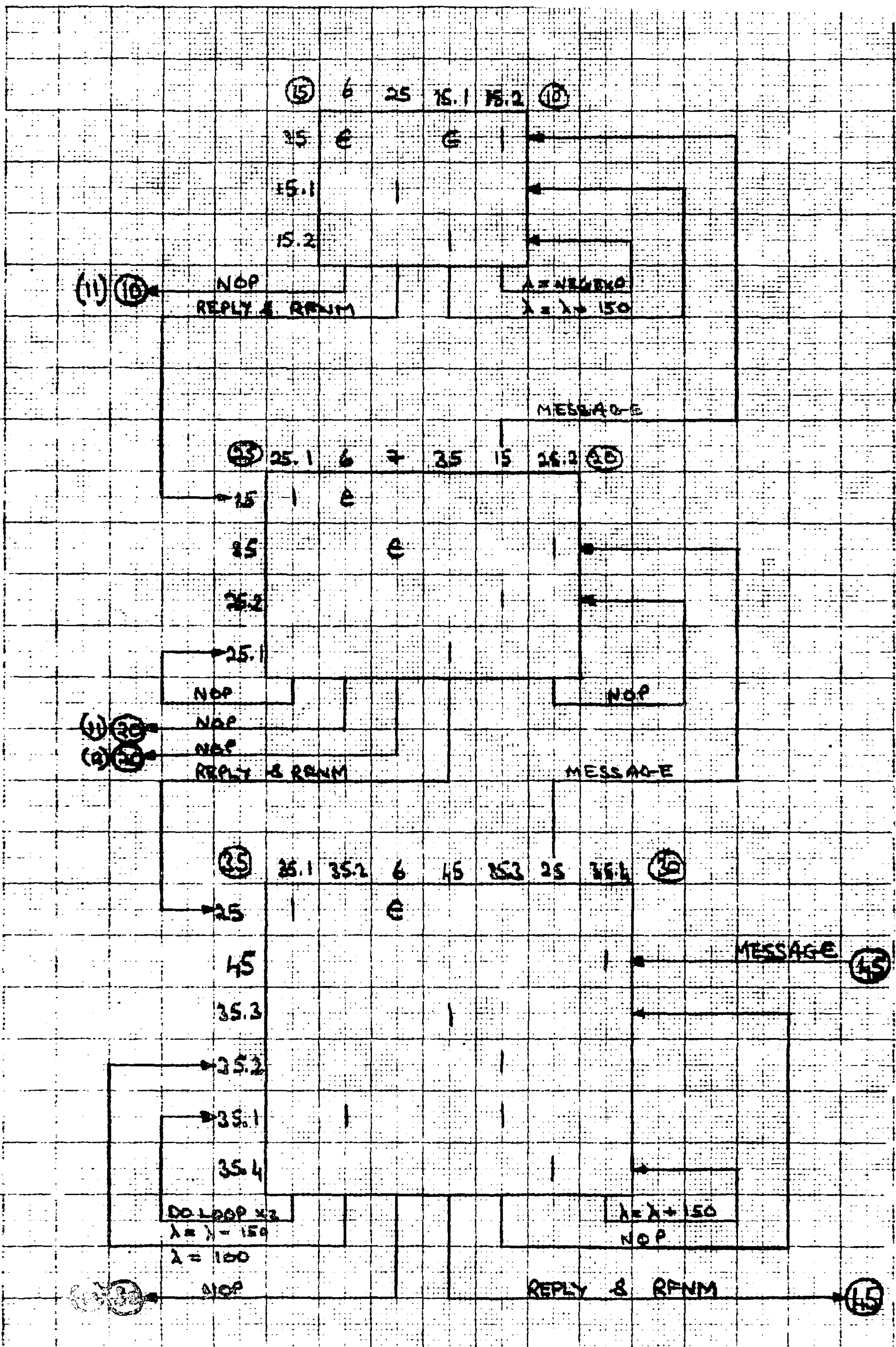


Fig. 5-13. Program Five; Nodes 15, 25, 35.

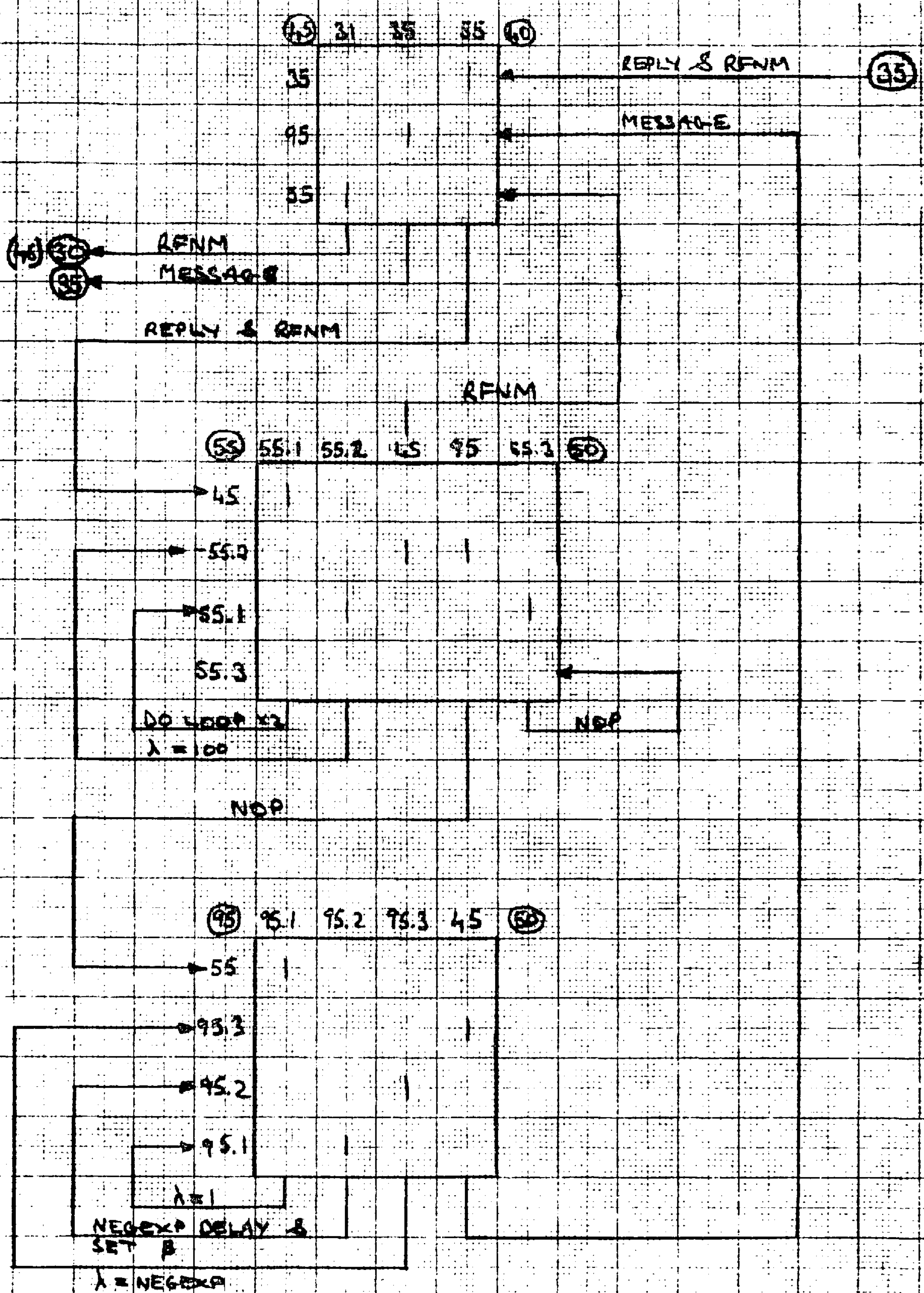


FIG. 6-14 STREAM FIVE: NODES 45, 55, 95.

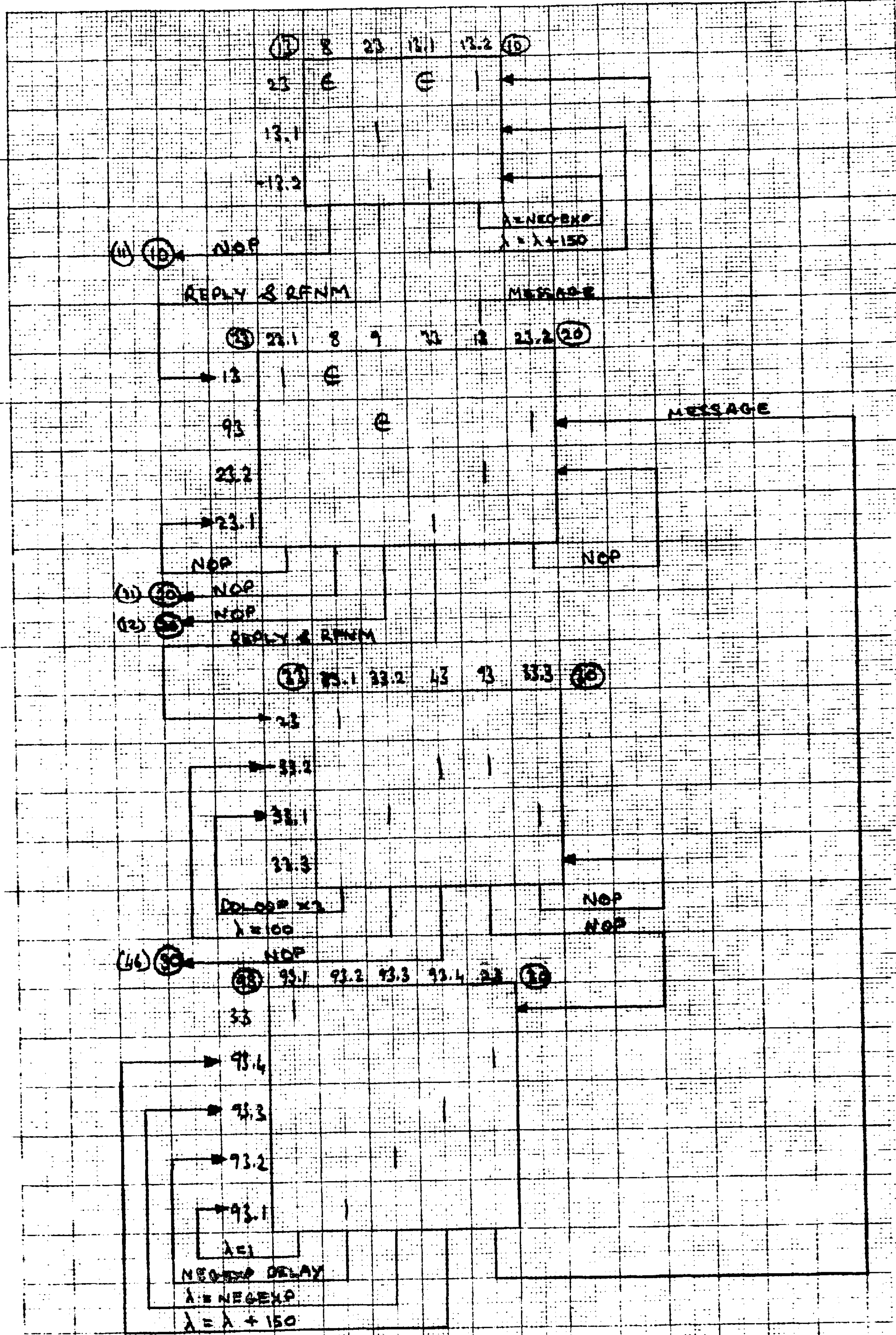


FIG. 6-15. STREAM THREE.

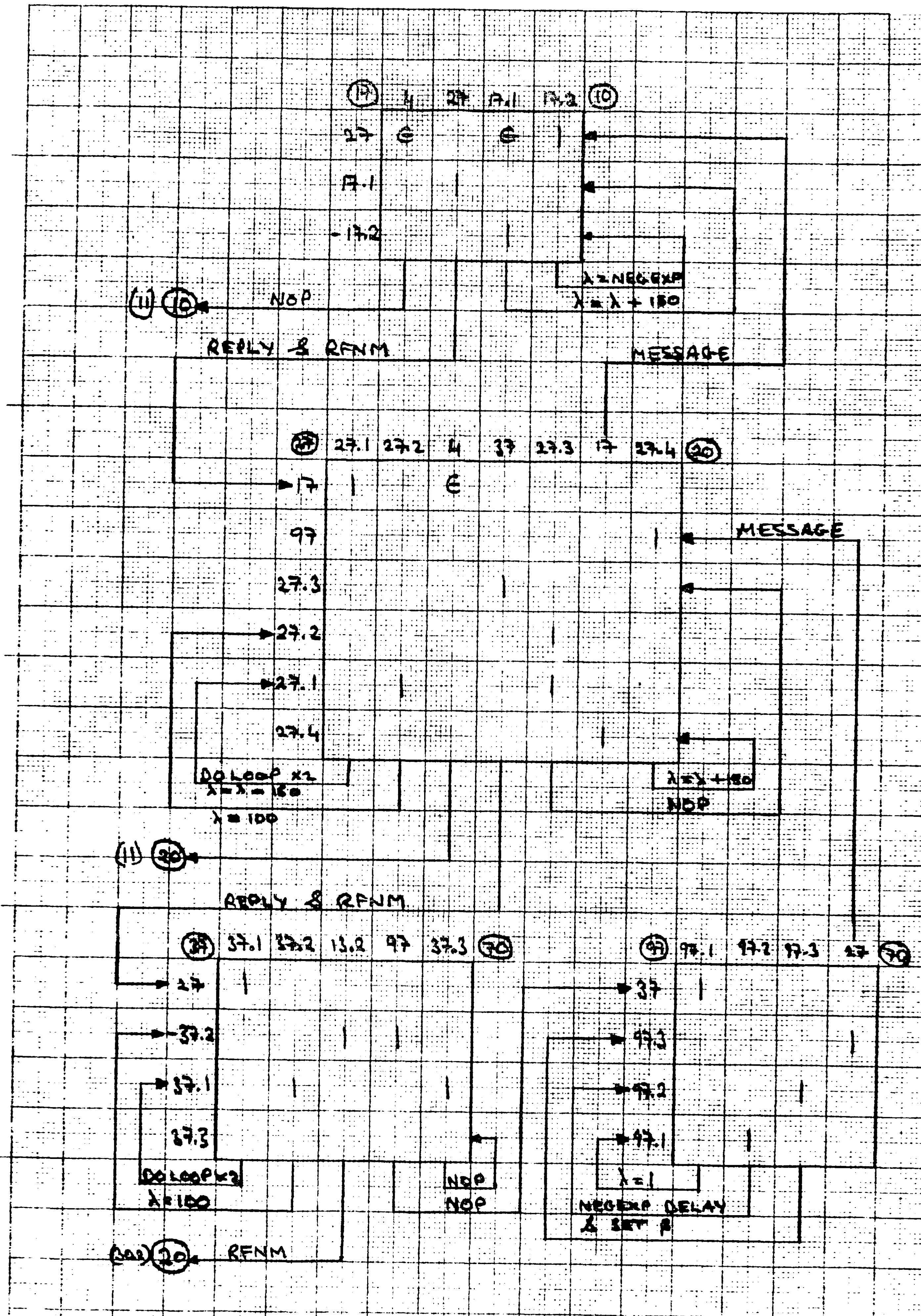


Fig 6-16. STREAM SEVEN.

CHART WELL

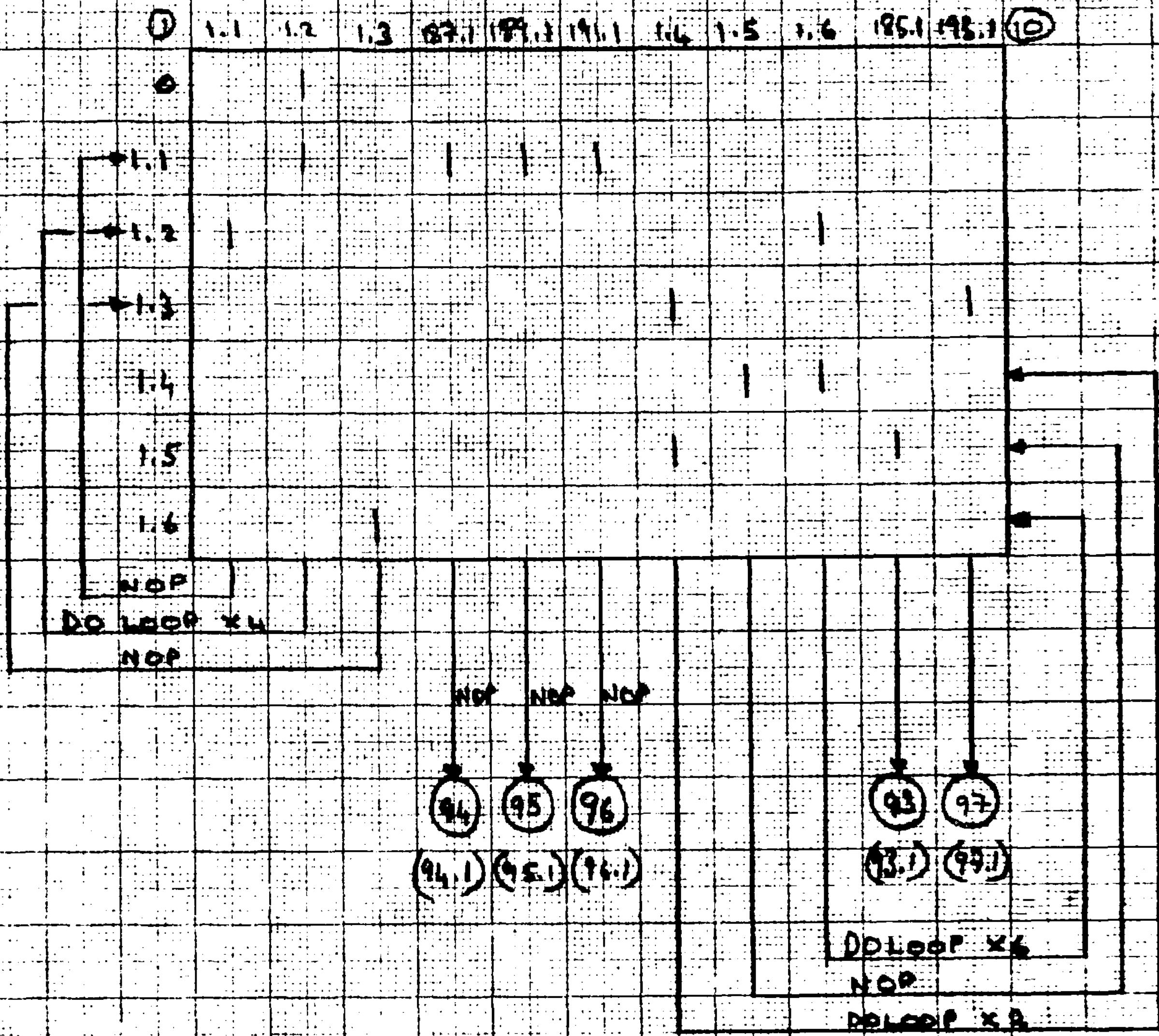


FIG. 6-17 INITIALIZATION NODE ONE.

This would not be the case if the more usual procedure of independent generation of messages was adopted.

In order to correctly represent cyclic message generation we must ensure that a reply returns to the same cluster which produced the corresponding message. This means that the outgoing reply streams, as well as the incoming messages, must be kept separate, for if they became intermingled and were activating the same dataset in a SInode, we would have no means of knowing the originating cluster, and therefore no way to route them to that cluster. The use of a separate set of nodes for each stream does not have any effect on the number of arc executions, but does mean that very similar data is replicated in each stream.

One of the conclusions which emerge from the application is that we could specify a much more compact model (without separate stream data) if the SHAPE system provided for more variables than LAMBDA and BETA to be carried by a cut. In that case such a variable might show cluster of origin and be used as a routing indicator for the returning reply.

We now follow the progress of a typical message from cluster five through the subnet. The user think time is represented by a delay arc (95.2) at SInode 95. This uses the IFloop facility for setting BETA to the arc termination time (IFCODEONE = 4). In fact IFCODEONE is set to -4 so that the delays can execute concurrently. On completion a value of LAMBDA is chosen (arc 95.3). The distribution is a negative exponential one, and any value drawn which is greater than 1,000 is truncated to that size. This is done to conform to the ARPANET limit on packet size. For this truncated distribution to have a mean (MU) of 600 bits,

we must select the correct mean (LAMBDA) of the exponential distribution. This is derived as follows. The probability density function of the exponential distribution is,

$$\text{p.d.f.} = \lambda e^{-\lambda x}$$

If the truncation level is k (= 1,000) we can write

$$= \int_0^k x\lambda e^{-\lambda x} dx + k\int_k^{\infty} \lambda e^{-\lambda x} dx$$

$$= [-xe^{-\lambda x}]_0^k + \int_0^k e^{-\lambda x} dx + k[-e^{-\lambda x}]_k^{\infty}$$

$$= -ke^{-k\lambda} + [-\frac{1}{\lambda} e^{-\lambda x}]_0^k + ke^{-k\lambda}$$

$$= (1 - e^{-\lambda k})/\lambda$$

Values of LAMBDA for various MU are tabulated in Figs. 6-18, 19.

Having chosen a value for the message length, the packet is transmitted to SInode 45. SInode 95 was tied to PInode 50 by having the appropriate field in its data set to this value. SInodes 45, 35, 25, 15 are similarly tied to PInodes 40,30, 20, 10. At SInode 45 the message is queued if necessary and then transmitted to SInode 35. Here the message length is increased by 150 bits (its ARPANET overhead) by arc 35.4. The message is again queued if necessary and transmitted to SInode 25. Here it activates two elements in the row marked 35. One (of magnitude one) queues the message for further transmission to SInode 15, and the second (of negligible size ϵ) acts via a no operation (NOP) arc to initiate transmission of an ACK from SInode 20 to SInode 30.

Since the outgoing LAMBDA is the product of the incoming arc and the REP element activated, the NOP has a LAMBDA which is

SOLUTION OF MU = (1. - EXP(-LAMDA*K))/LAMDA FOR K = 1000

MU	LAMDA	1/LAMDA
100	.0099995	100.0
110	.0090899	110.0
120	.0083313	120.0
130	.0076888	130.1
140	.0071372	140.1
150	.0066581	150.2
160	.0062378	160.3
170	.0058657	170.5
180	.0055336	180.7
190	.0052351	191.0
200	.0049651	201.4
210	.0047194	211.9
220	.0044947	222.5
230	.0042881	233.2
240	.0040974	244.1
250	.0039207	255.1
260	.0037563	266.2
270	.0036028	277.6
280	.0034591	289.1
290	.0033241	300.8
300	.0031971	312.8
310	.0030771	325.0
320	.0029637	337.4
330	.0028561	350.1
340	.0027539	363.1
350	.0026566	376.4
360	.0025639	390.0
370	.0024753	404.0
380	.0023906	418.3
390	.0023094	433.0
400	.0022316	448.1
410	.0021569	463.6
420	.0020850	479.6
430	.0020158	496.1
440	.0019491	513.1
450	.0018847	530.6
460	.0018226	548.7
470	.0017625	567.4
480	.0017044	586.7
490	.0016482	606.7
500	.0015936	627.5
510	.0015407	649.0
520	.0014894	671.4
530	.0014396	694.7
540	.0013911	718.8
550	.0013440	744.0
560	.0012981	770.3
570	.0012535	797.8
580	.0012100	826.4
590	.0011676	856.4

Fig. 6-18. Means of truncated negative exponential distributions.

SOLUTION OF MU = (1. - EXP(-LAMDA*K))/LAMDA FOR K = 1000

MU	LAMDA	1/LAMDA
600	.0011263	887.9
610	.0010859	920.9
620	.0010465	955.5
630	.0010080	992.0
640	.0009704	1030.5
650	.0009337	1071.0
660	.0008977	1113.9
670	.0008626	1159.3
680	.0008282	1207.5
690	.0007944	1258.7
700	.0007614	1313.3
710	.0007291	1371.6
720	.0006974	1434.0
730	.0006663	1500.9
740	.0006358	1572.9
750	.0006059	1650.5
760	.0005765	1734.6
770	.0005477	1825.9
780	.0005193	1925.5
790	.0004915	2034.4
800	.0004642	2154.2
810	.0004374	2286.5
820	.0004110	2433.4
830	.0003850	2597.4
840	.0003595	2781.9
850	.0003343	2990.9
860	.0003096	3229.7
870	.0002853	3505.0
880	.0002614	3826.2
890	.0002378	4205.6
900	.0002146	4660.8
910	.0001917	5217.0
920	.0001691	5912.0
930	.0001469	6805.5
940	.0001251	7996.6
950	.0001053	9500.0
960	.0001042	9600.0
970	.0001031	9700.0
980	.0001020	9800.0
990	.0001010	9900.0
1000	.0001000	10000.0

Fig. 6-19. Means of truncated negative exponential distributions (continued).

very small and variable. Consequently we need an arc (20.2) in SInode 20 to set the ACK size to 150 bits. This is then transmitted to SInode 30 where it is destroyed since the terminal dataset elements are all zero.

The message arrives at SInode 15 which is its final destination. Here it initiates transmission of an ACK from SInode 10 to SInode 20, in the same way as outlined above. Additionally a RFNM is immediately transmitted back to the originating node, followed by the reply which is simply the message with a new length. This is chosen in the same way as the original one. Both these are transmitted to SInode 25 using the same arc. Here ACKs are generated, and the packets forwarded to SInode 35.

At this point the packets leave the ARPANET so that the message is reduced in length by 150 bits, and the RFNM is changed to 100 bits. They are both transmitted to SInode 45 and on to SInode 55. Here the RFNM is separated from the reply using a DOloop which completes on every second activation (arc 55.1). The RFNMs are destroyed by arc 55.3 and the messages are used to create ingoing RFNMs of length 100 bits, as well as activating a new think period in SInode 95 (after reduction of LAMBDA to 1 by arc 95.1).

The ingoing RFNM is transmitted to SInode 45 and forwarded to SInode 30. Here its length is changed to 150 (RFNM length in the ARPANET) and it is transmitted to SInode 20, where it generates an ACK and is forwarded to SInode 10. Here the RFNM is turned into its own ACK by returning it to SInode 20 along the ACK transmission arc.

The other streams exhibit the same basic pattern with some small variations. The whole graph is initially activated by NOP arcs from SInode 1 which go to think time dataset of each stream. The number of times these are activated corresponds to the number of terminals in the cluster, and the activations are produced by appropriate DOloops at SInode 1.

The parameters of the model can be altered for each stream individually. In each one the think time for a user can be changed by altering the mean delay in, for example, arc 95.2. Similarly the mean message length can be reset, and need not be the same for messages as replies. The number of users active in a cluster can be altered by changing the DOloop limits in the initializing node.

It is very easy to convert the model from cyclic message generation to independent generation as a series of Poisson events. Firstly it is necessary to change the REP matrix element of the reply receiving node which activates the think time node from 1 to 0 (for example in SInode 55 we would alter the element at the intersection of row 55.2 and column 95). Then the think time delay is made to propagate itself by having it activated each time a message length is chosen (for example in SInode 95 we alter the element at the intersection of row 95.3 and column 95.1 from zero to one). A Poisson series of events is obtained by setting IFCODEONE positive in arc 95.2. This ensures that successive message generations take place at intervals drawn from a negative exponential distribution.

We have outlined a few elementary ways in which the model can be altered, and many more are possible. For example the next section includes a description of the addition of a background

of file transfer traffic to the model described above.

6.4 Results.

In this section we present the results obtained by executing the link model with various parameter values. The mean user think period delay was initially set to thirty seconds. The number of terminals activated in each cluster was as described in the previous analysis. This gives a total of thirty active terminals, so that the subnet generates approximately one message per second.

Three possible values were considered for C_1 (4.8, 9.6 and 50 Kb)* and in each case C_3 was made equal to C_1. The response observed at each terminal was recorded and statistics accumulated by setting an INarc specifier negative in the reply receiving node for each stream (e.g. -55.2 for stream five). The mean response at node i is $2T_i$, as defined earlier. Consequently an execution of the SHAPE model gives us the variables required to calculate T_a, the mean message delay in the subnet.

To examine the effects of increased loading the mean think period was decreased, thus increasing the overall message generation rate. A series of execution runs was performed for each of the three channel capacities considered, in which the load on the subnet was gradually increased. The results of each series are calculated in Figs. 6-20, 21, 22. The first five columns give the observed response times of the clusters and the last two give T_a calculated with observed and expected numbers of messages generated during the run.

The first series run was that with $C_1 = C_3 = C = 9.6$ Kb. For each value of the mean think time P at least three runs were performed. In order that they should provide independent results

* Kb stands for kilobits/sec.

CYCLIC MESSAGE GENERATION WITH C = 4.8 AND UNLIMITED MEMORY

THINK PERIOD	RESPONSE UK TIP	RESPONSE NORW TIP	RESPONSE PDP9	RESPONSE CDC 6600	RESPONSE IBM 360	RESPONSE NET MEAN	NORMALIZD NET MEAN
30.0	0.7157	0.4027	0.7923	1.1980	1.1430	0.3805	0.3911
30.0	0.7200	0.4137	0.7827	1.2480	1.1750	0.3939	0.3978
30.0	0.7423	0.4099	0.8026	1.2410	1.1950	0.4020	0.4041
26.0	0.7572	0.4245	0.8762	1.2060	1.1620	0.4118	0.4089
26.0	0.7344	0.4156	0.9163	1.2290	1.1640	0.4126	0.4077
22.0	0.7922	0.4523	0.8563	1.2350	1.2140	0.4348	0.4225
18.0	0.7540	0.4420	0.8961	1.3200	1.2140	0.4366	0.4214
14.0	0.9092	0.5205	0.8925	1.3580	1.3330	0.4711	0.4701
10.0	0.9632	0.6019	1.1240	1.4060	1.3450	0.5001	0.5079
6.0	1.6300	0.9692	1.2900	1.9690	1.6420	0.7158	0.7427
6.0	1.1030	0.8048	0.9993	1.7600	1.4600	0.5983	0.5763
3.0	1.4660	1.2170	1.0380	2.3260	1.8680	0.7173	0.7478
3.0	2.4770	1.2940	1.7220	2.2170	1.9700	0.9539	0.9971
3.0	5.0560	1.8270	2.0900	2.9160	3.0480	1.5552	1.5890
3.0	5.2490	1.8000	2.2160	2.7540	2.8190	1.5583	1.5901
1.0	2.3010	1.9130	2.1850	3.5960	3.2030	1.2164	1.2098
1.0	5.3940	3.6030	2.9210	3.3130	3.4210	1.8858	1.9865
1.0	7.5980	4.2790	3.0480	3.4560	3.3780	2.2043	2.3133

Fig. 6-20. Subnet response when C = 4.8.

CYCLIC MESSAGE GENERATION WITH C = 9.6 AND UNLIMITED MEMORY

THINK PERIOD	RESPONSE UK TIP	RESPONSE NORW TIP	RESPONSE PDP9	RESPONSE CDC 6600	RESPONSE IBM 360	RESPONSE NET MEAN	NORMALIZD NET MEAN
30.0	0.3577	0.2150	0.4291	0.8247	0.8282	0.2195	0.2309
30.0	0.3529	0.1971	0.4093	0.8575	0.8283	0.2168	0.2290
30.0	0.3450	0.2073	0.3928	0.7935	0.7737	0.1990	0.2196
30.0	0.3414	0.2004	0.4238	0.8262	0.7623	0.2200	0.2217
30.0	0.3462	0.2090	0.3894	0.8558	0.8265	0.2197	0.2273
26.0	0.3524	0.2210	0.4052	0.8605	0.8065	0.2311	0.2297
26.0	0.3518	0.1993	0.4045	0.8350	0.8462	0.2389	0.2283
26.0	0.3562	0.2022	0.4158	0.8789	0.8197	0.2259	0.2313
22.0	0.3479	0.2022	0.4216	0.8364	0.8236	0.2263	0.2273
22.0	0.3538	0.2229	0.4094	0.8376	0.8320	0.2202	0.2305
22.0	0.3423	0.2131	0.4099	0.8726	0.8296	0.2349	0.2293
18.0	0.3535	0.2086	0.4179	0.8689	0.8455	0.2412	0.2321
18.0	0.3527	0.2254	0.4041	0.9259	0.8503	0.2410	0.2367
18.0	0.3504	0.2053	0.4009	0.8705	0.8195	0.2224	0.2285
14.0	0.3759	0.2168	0.4126	0.8524	0.8632	0.2379	0.2368
14.0	0.3690	0.2242	0.4226	0.9590	0.8812	0.2465	0.2447
14.0	0.3702	0.2160	0.4283	0.8535	0.8712	0.2360	0.2372
10.0	0.3806	0.2367	0.4362	0.9020	0.8850	0.2382	0.2451
10.0	0.3737	0.2146	0.4444	0.9175	0.8806	0.2410	0.2426
10.0	0.4089	0.2507	0.4487	0.8978	0.8401	0.2512	0.2502
6.0	0.4089	0.2402	0.4916	1.1220	0.8448	0.2564	0.2636
6.0	0.4259	0.2297	0.5074	0.9140	0.8959	0.2546	0.2581
6.0	0.4754	0.2547	0.5124	0.9042	0.8599	0.2625	0.2686
3.0	0.6727	0.4374	0.6380	1.1090	1.1300	0.3500	0.3623
3.0	0.7050	0.5338	0.6243	1.1550	0.9725	0.3510	0.3726
3.0	0.6849	0.4259	0.6630	1.1810	1.1610	0.3631	0.3708
3.0	1.0800	0.4457	0.6865	1.2330	1.2010	0.4460	0.4570
3.0	1.0690	0.6008	0.8276	1.3540	1.3080	0.4999	0.4980
1.0	0.9013	0.7309	0.9235	1.8010	1.4690	0.5290	0.5063
1.0	1.7420	0.8612	1.4550	1.8560	1.6810	0.7314	0.7374
1.0	2.5820	1.3890	1.1410	1.8760	1.7490	0.8876	0.9183
1.0	3.8480	1.8400	1.3700	1.6190	1.6070	1.0830	1.1065

Fig. 6-21. Subnet response when C = 9.6.

CYCLIC MESSAGE GENERATION WITH C = 50 AND UNLIMITED MEMORY

THINK PERIOD	RESPONSE UK TIP	RESPONSE NORW TIP	RESPONSE PDP9	RESPONSE CDC 6600	RESPONSE IBM 360	RESPONSE NET MEAN	NORMALIZD NET MEAN
30.0	0.06586	0.05601	0.09723	0.5773	0.5479	0.0912	0.0995
30.0	0.06529	0.05527	0.09765	0.5609	0.5368	0.0880	0.0975
30.0	0.06478	0.05685	0.09761	0.5534	0.5356	0.0900	0.0970
30.0	0.06567	0.05499	0.09921	0.5748	0.5469	0.0944	0.0992
26.0	0.06513	0.05532	0.09694	0.5722	0.5635	0.1029	0.0998
26.0	0.06529	0.05557	0.09944	0.5781	0.5550	0.1122	0.0998
22.0	0.06501	0.05454	0.09477	0.5778	0.5775	0.0973	0.1006
18.0	0.06556	0.05531	0.09787	0.6142	0.5704	0.1188	0.1027
14.0	0.06547	0.05554	0.10050	0.5631	0.5284	0.0834	0.0966
10.0	0.06571	0.05687	0.09636	0.5821	0.5866	0.0994	0.1007
6.0	0.06658	0.05575	0.09845	0.6451	0.5773	0.1088	0.1025
3.0	0.06948	0.05798	0.10240	0.6041	0.5936	0.0880	0.0985
1.0	0.09042	0.07496	0.12560	0.7775	0.6426	0.0848	0.1042

Fig. 6-22. Subnet response when C = 50.

rather than being duplicate runs, all random number seeds were altered for each run. This is more satisfactory statistically, than running the model for different durations with the same set of seeds. For heavier loading the model took longer to reach a steady state, and provide convergent results. The two values of P where this had a significant effect were P = 3.0 and P = 1.0. In these cases more than three runs were made, and they appear in order of increasing duration. For P = 3.0 the value of T_a in the final run was 0.4999 secs with double the original run time. We would expect the steady state to be between 0.5 and 0.6 seconds. The result was not pursued further since the runs of that duration were already consuming considerable computer time, and the loading was at the limit of the range we were considering, namely one order of magnitude greater than the starting estimate.

The runs with P = 1.0 were executed to obtain an indication of behaviour for loading with P less than 3.0 There is little convergence apparent in the results, and we can see that the response time is considerably greater than the think time. Under these circumstances the cyclic nature of the model plays an important part by preventing message generation until responses are received. In contrast we can see that with P = 3.0 all responses are still less than half the think time. We conclude that with C = 9.6 and P = 30 we are approximately an order of magnitude from a load which causes the subnet to explode.

A similar set of runs was executed with $C_1 = C_3 = 4.8$ and it can be seen that the response time for P = 30 is a good deal longer, and that rapid deterioration begins at a larger value

of P (6.0) than with C = 9.6 Kb. Setting $C_1 = C_3 = C = 50$ Kb gives us the results in Fig. 6-22 which show that in this case there is little or no variation of response time in the range of loads we have examined. We note in passing that the results agree well with the values of T_a calculated earlier for P = 30 and shown in Fig. 6-5.

After these runs the model was altered to give non-cyclic generation of messages. The method of doing this has been described at the end of the previous section, and allows a message generation node to operate independently of the subsequent fate of its messages. The results are shown in Fig. 6-23. For the lower loadings mean response time is not significantly different from the cyclic case. However, for the runs P = 6.0 and P = 10.0 we see that T_a is slightly greater than for the cyclic case. The fact that this difference does not increase in the runs for P = 3.0 and P = 1.0 we attribute to insufficient run-time to reach a steady state. We would expect that in a steady state, with comparable loads on the subnet, the non-cyclic case would produce more messages per second, and consequently longer response time.

The question of comparable load is effectively that of the rates at which the five independent generators should produce messages. For the runs in Fig. 6-23 we derived mean delays as follows. If r_i is the mean response observed for cluster i when the think time delay is P, we can say that the mean cycle time for a message in the i th cluster is

$$\text{Cycle } i = P + r_i$$

$$\text{No. of messages/sec in } i \text{ th cluster} = n_i/(P + r_i)$$

NON-CYCLIC MESSAGE GENERATION WITH C = 9.6

THINK PERIOD	RESPONSE UK TIP	RESPONSE NORW TIP	RESPONSE PDP9	RESPONSE CDC 6600	RESPONSE IBM 360	RESPONSE NET MEAN	NORMALIZD NET MEAN
30.0	0.3649	0.2084	0.4305	0.8767	0.8026	0.2304	0.2345
26.0	0.3605	0.2006	0.4008	0.8423	0.8344	0.2285	0.2307
22.0	0.3554	0.2107	0.4314	0.8371	0.8218	0.2300	0.2315
18.0	0.3631	0.2177	0.4085	0.8452	0.8511	0.2308	0.2347
14.0	0.3782	0.2206	0.4073	0.8475	0.7961	0.2333	0.2344
10.0	0.4000	0.2389	0.4633	0.9106	0.9198	0.2589	0.2568
6.0	0.5189	0.3250	0.5377	1.0040	0.9684	0.2990	0.3036
6.0	0.4965	0.3129	0.4825	0.9923	0.8603	0.2792	0.2862
3.0	0.8306	0.4873	0.6889	1.2050	1.3740	0.4458	0.4327
3.0	0.8407	0.3868	0.5718	1.1800	1.2120	0.4031	0.4044
1.0	1.0100	0.5171	1.0870	1.5610	1.2740	0.5095	0.5152
1.0	2.9360	1.4500	1.3770	1.7610	1.3430	0.9271	1.0309

Fig. 6-23. Non-cyclic generation of messages.

$$\text{Mean message delay} = \tfrac{1}{2} \quad \frac{r_i n_i}{p+r_i} \qquad \frac{n_i}{p+r_i}$$

For each value of P in the set of runs, the terminals of cluster i were set to generate $1/(P + r_i)$ messages per second.

A further series of runs was undertaken which gave results for T_a for various mean message lengths (cyclic generation). These are shown in Fig. 6-24. The runs were for values of P = 3 and P = 30 and covered the range 300 to 900 bits for each. The purpose of these runs was to examine how sensitive T_a is to variations in a. For P = 30 we can see that there is a smooth increase, and that even with a very high (900) mean message length the value of T_a falls within the design range set previously. For P = 3 the results suffer from insufficient runtime for the higher message lengths. The sets of computer runs so far described are to be found in INDRA Notes 287 to 291 inclusive. The tabulated results are shown in graphical form in Figs. 6-25, 26.

We can see that for purely interactive traffic at the level estimated (P = 30), all three values of C would be sufficient. If C = 50 Kb the response time will be good ($T_a = 0.1$) for this level of loading or more. If C = 9.6 Kb, the response time is still fairly good ($T_a = 0.22$), and remains within the desired limits for loads up to an order of magnitude larger. For C = 4.8 Kb the value of T_a is about a half of the acceptable limit, and the load increase available is less than an order of magnitude. From these results we recommend that for operation with interactive traffic, the most appropriate values of C_1 and C_3 in the subnet would be 9.6 Kb.

We now discuss the addition of a background of file transfer traffic to the model. This type of traffic consists of full ARPA

CYCLIC MESSAGE GENERATION WITH BLOCK LENGTH VARIATION, C = 9.6, P = 3 AND 30

BLOCK LENGTH	THINK PERIOD	RESPONSE UK TIP	RESPONSE NORW TIP	RESPONSE PDP9	RESPONSE CDC 6600	RESPONSE IBM 360	RESPONSE NET MEAN	NORMALIZD NET MEAN
300	3.0	0.3343	0.2224	0.3142	0.6765	0.5825	0.1858	0.1906
400	3.0	0.5705	0.3306	0.5934	0.8322	0.9552	0.2988	0.3006
500	3.0	0.6470	0.4133	0.6124	1.1760	1.0560	0.3479	0.3520
700	3.0	1.1980	0.7290	0.8710	1.5090	1.5560	0.5524	0.5644
800	3.0	1.2960	0.7968	1.0980	1.6430	1.6300	0.6140	0.6201
900	3.0	1.2390	0.7213	0.9239	1.8400	1.6990	0.5905	0.6008
300	30.0	0.2211	0.1472	0.2271	0.4799	0.4504	0.1318	0.1359
400	30.0	0.2734	0.1806	0.2897	0.5797	0.6334	0.1698	0.1725
500	30.0	0.3143	0.1822	0.3273	0.7879	0.8431	0.2100	0.2107
700	30.0	0.4081	0.2560	0.4292	0.9344	1.1000	0.2665	0.2700
800	30.0	0.4424	0.2736	0.4755	1.2420	1.1500	0.2945	0.3049
900	30.0	0.4869	0.2956	0.5300	1.2430	1.2320	0.3229	0.3252

Fig. 6-24. Variation of response with packet length.

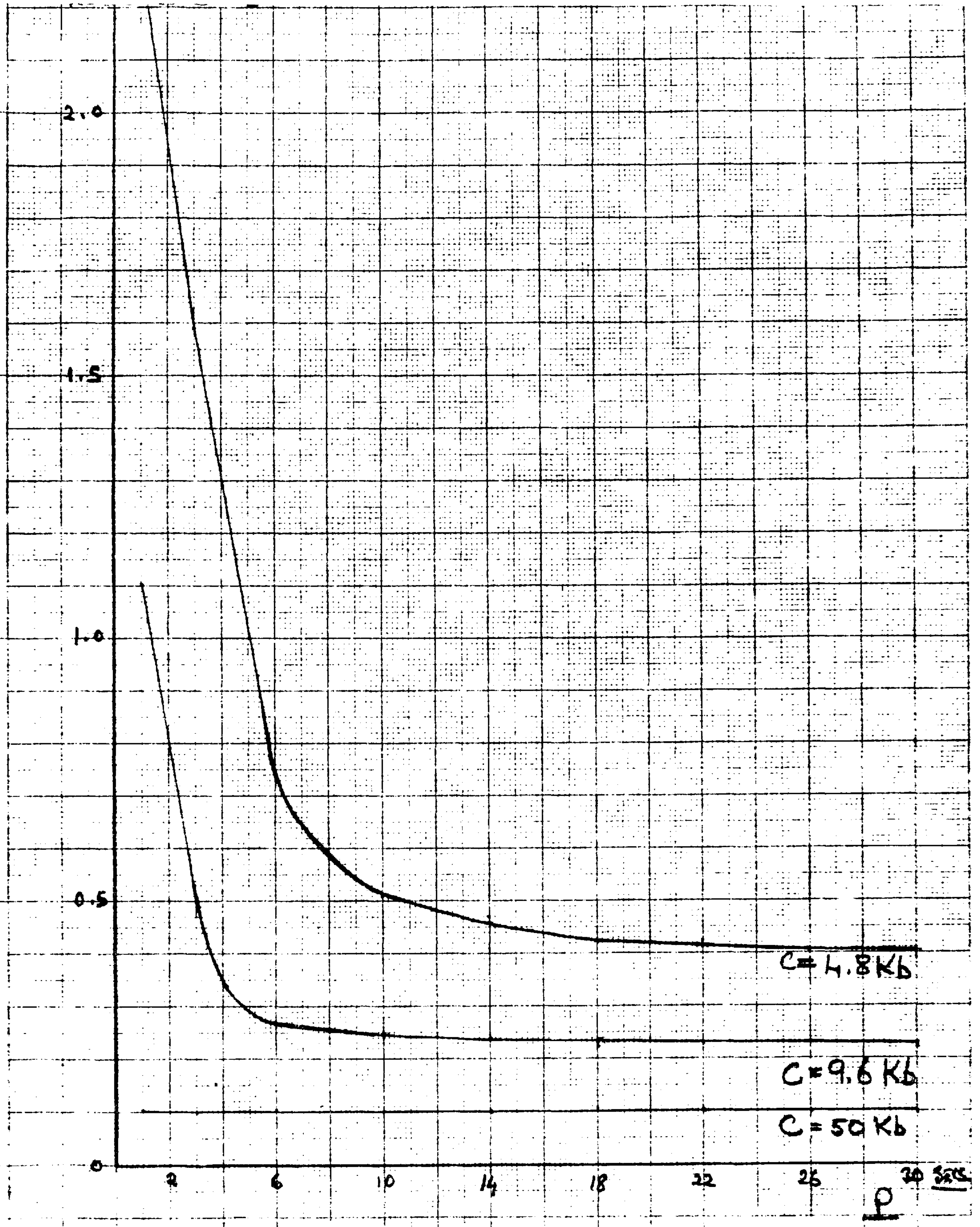


FIG. 6-25. GRAPHS OF RESPONSE AGAINST THINK PERIOD FOR C=4.8, 9.6, 50 Kb.

NORMALIZED
RESPONSE (SECS.)

P = 3.0

P = 30.0

0.6
0.5
0.4
0.3
0.2
0.1

200 300 400 500 600 700 800 900 BITS

PACKET LENGTH

FIG. 6-26. VARIATION OF RESPONSE WITH BLOCK LENGTH.

messages, that is to say eight packets each of 1,000 bits. In order to provide this background we have introduced a separate generating node for each stream (for example node 85 for stream 5). This node generates eight-packet messages at intervals drawn from a negative exponential distribution. We show the structure of the node in Fig. 6-27.

Since the traffic is a background it is sufficient to generate only the inward bound blocks, and examine the effect on inward interactive traffic. By arguments from the symmetry of the traffic inward and outward, we include a RFNM for each block and inward ACKs corresponding to those of the blocks themselves. The addition of the inward traffic is to stream zero, and the modified stream is shown in Fig. 6-28. The message generation is initialized by an extension to node one which gives the block traffic the same proportional pattern as the interactive. Of course this need not be the case in actual operation, but without foreknowledge it is the most reasonable estimate. The extended node one is shown in Fig. 6-29.

The average interactive message delay is observed by measuring the mean elapsed age of the messages from each stream as they reach the US TIP. As before, T_a is calculated from these values weighted by cluster size.

Several runs were executed with background traffic, but we have not included the results since the number of message trips completed within the run limits was insufficient to provide meaningful statistics. The amount of computer time required to produce significant results would not have been available without special arrangements and consequently longer runs were not attempted.

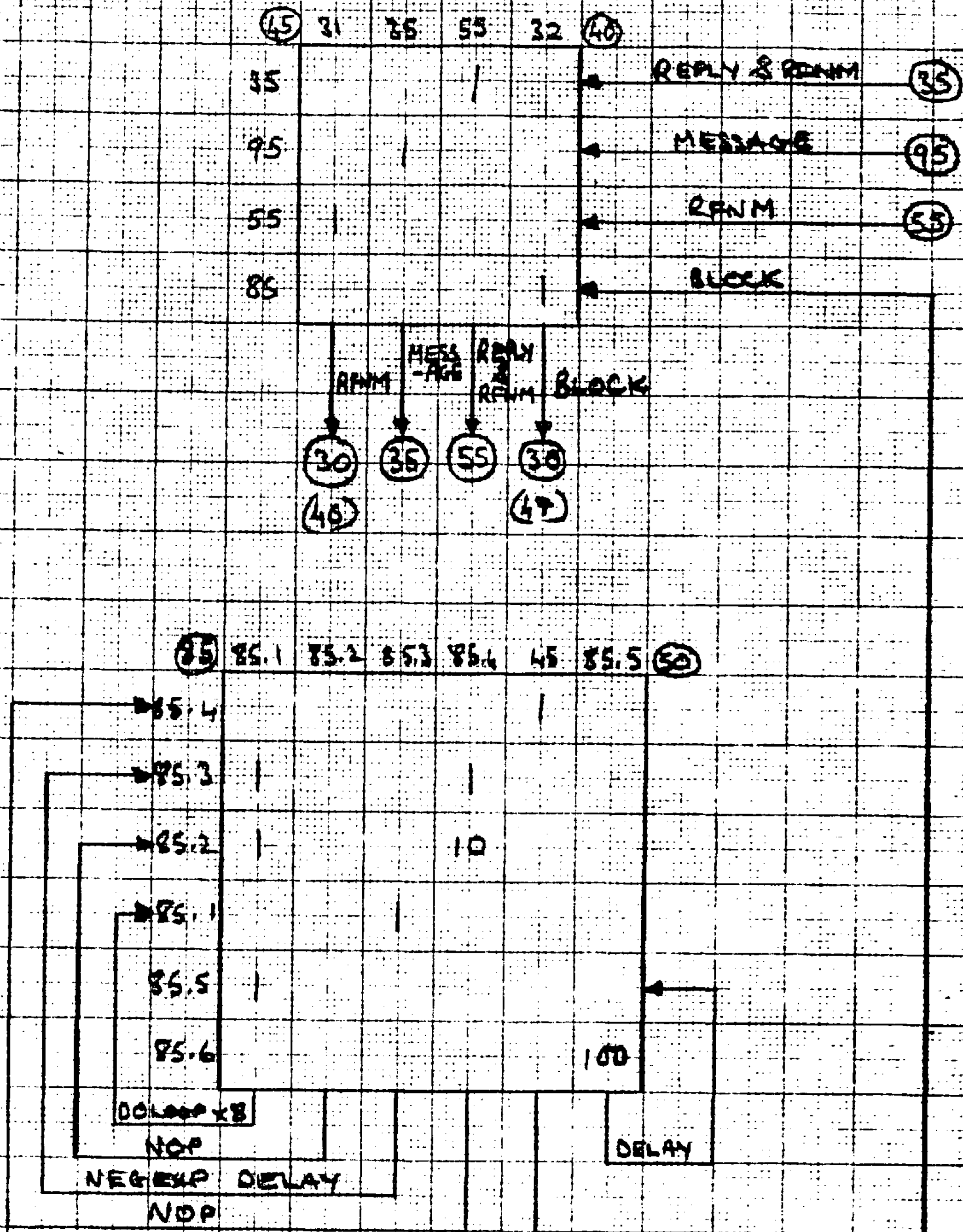


FIG. 6-27. BACKGROUND TRAFFIC GENERATION.

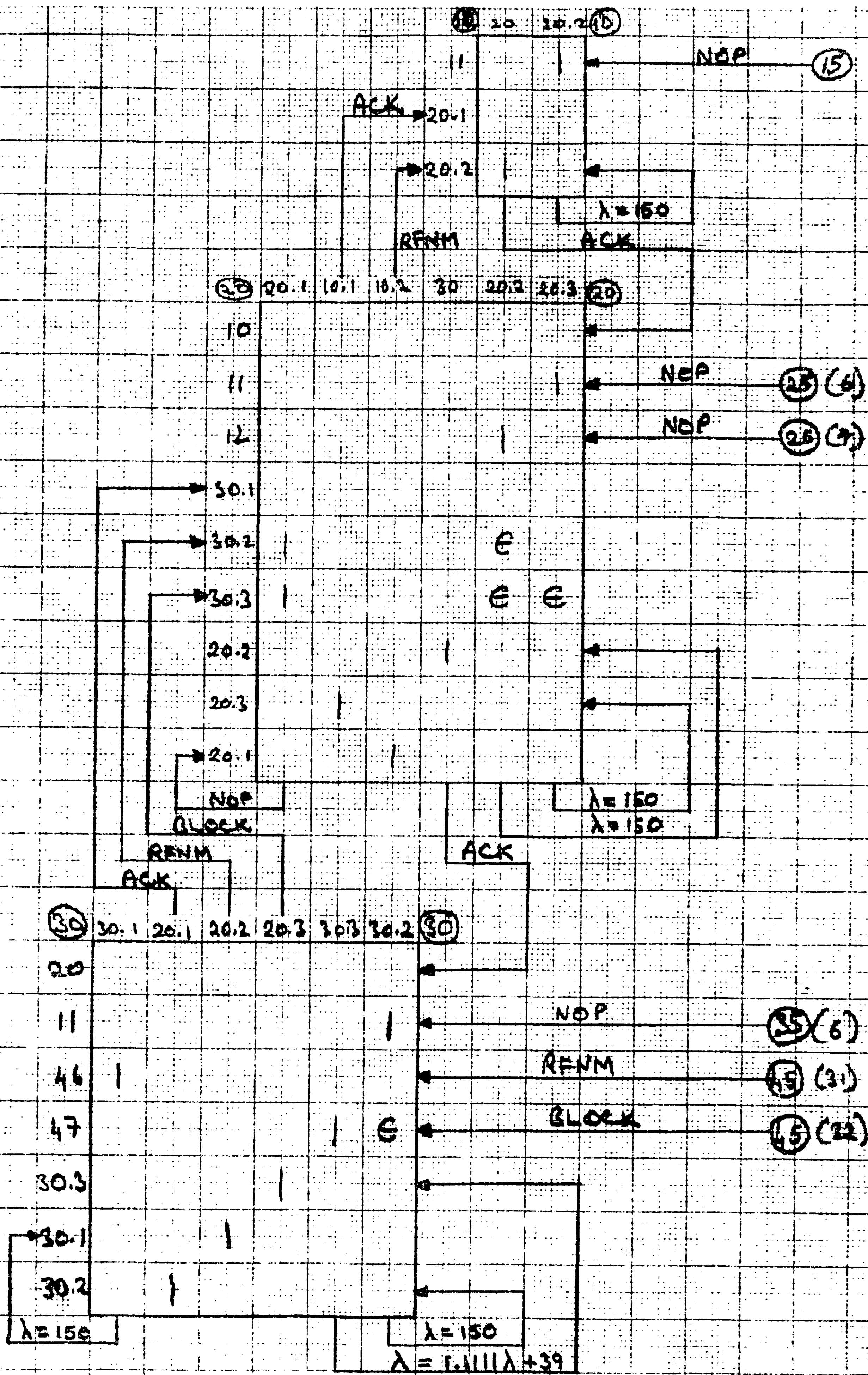


FIG. 6-28. STREAM ZERO WITH BACKGROUND.

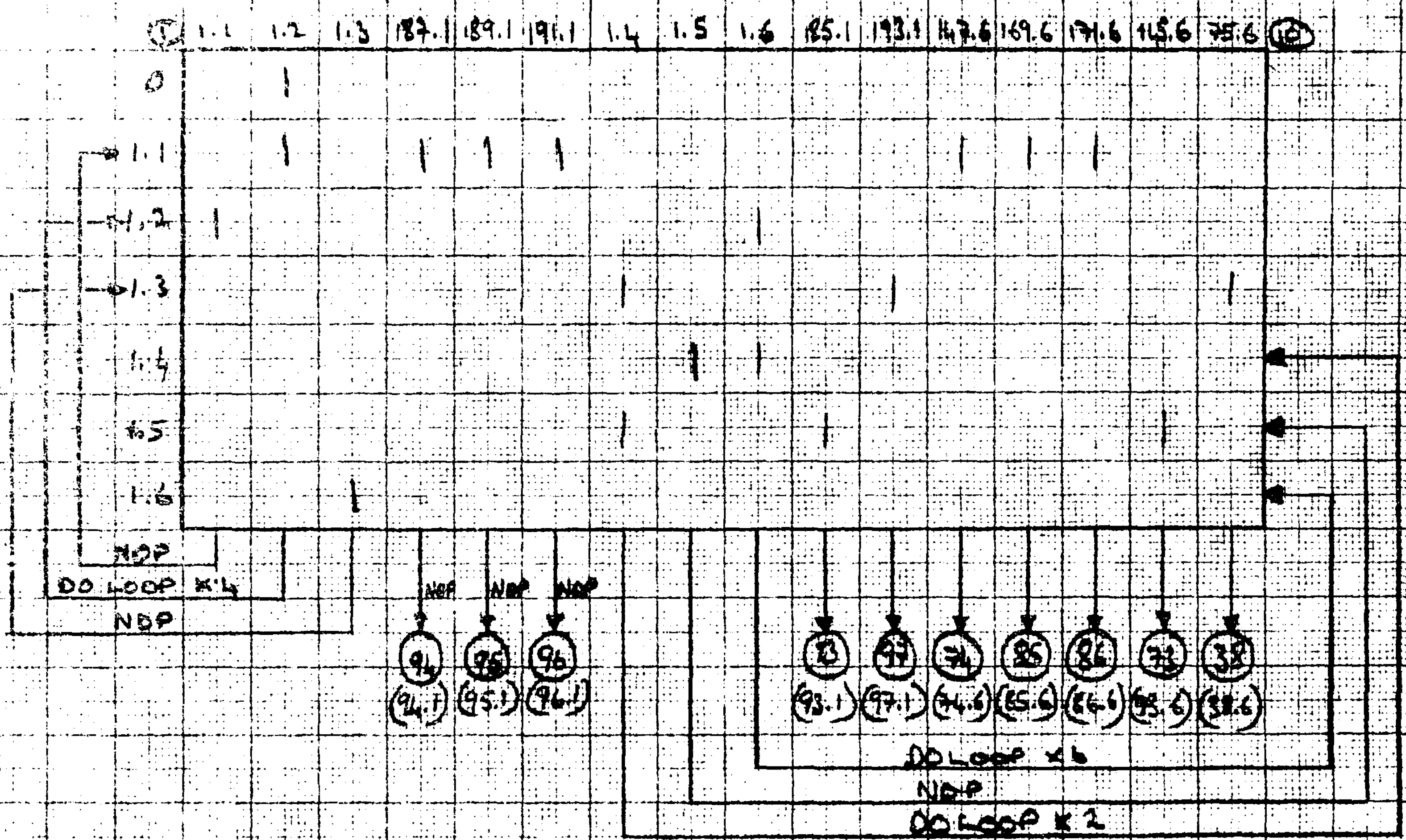


FIG. 6-29. INITIALIZATION NODE ONE INCLUDING BACKGROUND.

Finally a number of runs were made with the original model to examine the effect of varying C_1 and C_3 when the load was held constant. All possible combinations of the values 4.8, 7.2, 9.6 and 50 Kb for the channels C_1 and C_3 were run with a load on the network produced by setting P = 30. We show the values of T_a obtained in tabular form in Figs. 6-30, 31 and in graphical form in Fig. 6-32. The T_a are calculated using expected numbers of messages.

It is clear from the structure of the link that if one of C_1 and C_3 is to be increased in order to reduce the value of T_a, then the greater reduction is obtained by increasing C_1. The degree of this advantage is shown in Fig. 6-31. In Fig. 6-30 we can see that an increase in C_1 or C_3 benefits most those terminal clusters in whose message paths the channel is the most significant component. The results show that if a particular message delay is to be obtained, then the sum of C_1 and C_3 is least when they are approximately equal. Assuming that channel cost is related to capacity, enhancement divided equally between channels one and three will provide the greatest improvement in performance for a specific cost.

CHANNEL ONE	CHANNEL THREE	RESPONSE UK TIP	RESPONSE NORW TIP	RESPONSE PDP 9	RESPONSE CDC 6600	RESPONSE IBM 360	RESPONSE NET MEAN
4.8	4.8	0.7200	0.4137	0.7827	1.2480	1.1750	0.3977
4.8	7.2	0.6133	0.3935	0.7348	1.1460	1.0810	0.3595
4.8	9.6	0.5636	0.4040	0.6783	1.0380	1.0300	0.3362
4.8	50.0	0.4345	0.4081	0.5176	0.9386	0.8846	0.2838
7.2	4.8	0.5987	0.2656	0.7001	1.1210	1.0510	0.3378
7.2	7.2	0.4792	0.2703	0.5850	0.9739	0.9307	0.2888
7.2	9.6	0.4249	0.2648	0.4929	0.9249	0.8927	0.2655
7.2	50.0	0.3048	0.2789	0.3474	0.7811	0.7389	0.2133
9.6	4.8	0.5362	0.2054	0.6398	1.0320	0.9805	0.3046
9.6	7.2	0.4131	0.2069	0.4925	0.8912	0.8599	0.2529
9.6	9.6	0.3462	0.2090	0.3894	0.8558	0.8265	0.2257
9.6	50.0	0.2277	0.2131	0.2740	0.7193	0.6771	0.1782
50.0	4.8	0.4108	0.0551	0.4778	0.9104	0.8538	0.2371
50.0	7.2	0.2776	0.0554	0.3254	0.7709	0.7184	0.1820
50.0	9.6	0.2033	0.0552	0.2487	0.6985	0.6595	0.1533
50.0	50.0	0.0653	0.0553	0.0977	0.5609	0.5368	0.0983

Fig. 6-30 Subnet response for various combinations of C_1 and C_3 when P = 30.

T_a

C_1 \ C_3	4.8	7.2	9.6	30
4.8	0.3977	0.3595	0.3362	0.2838
7.2	0.3378	0.2888	0.2655	0.2133
9.6	0.3046	0.2529	0.2257	0.1782
30	0.2371	0.1820	0.1533	0.0983

Fig. 6-31 T_a for combinations of C_1 and C_3 when $P = 30$.

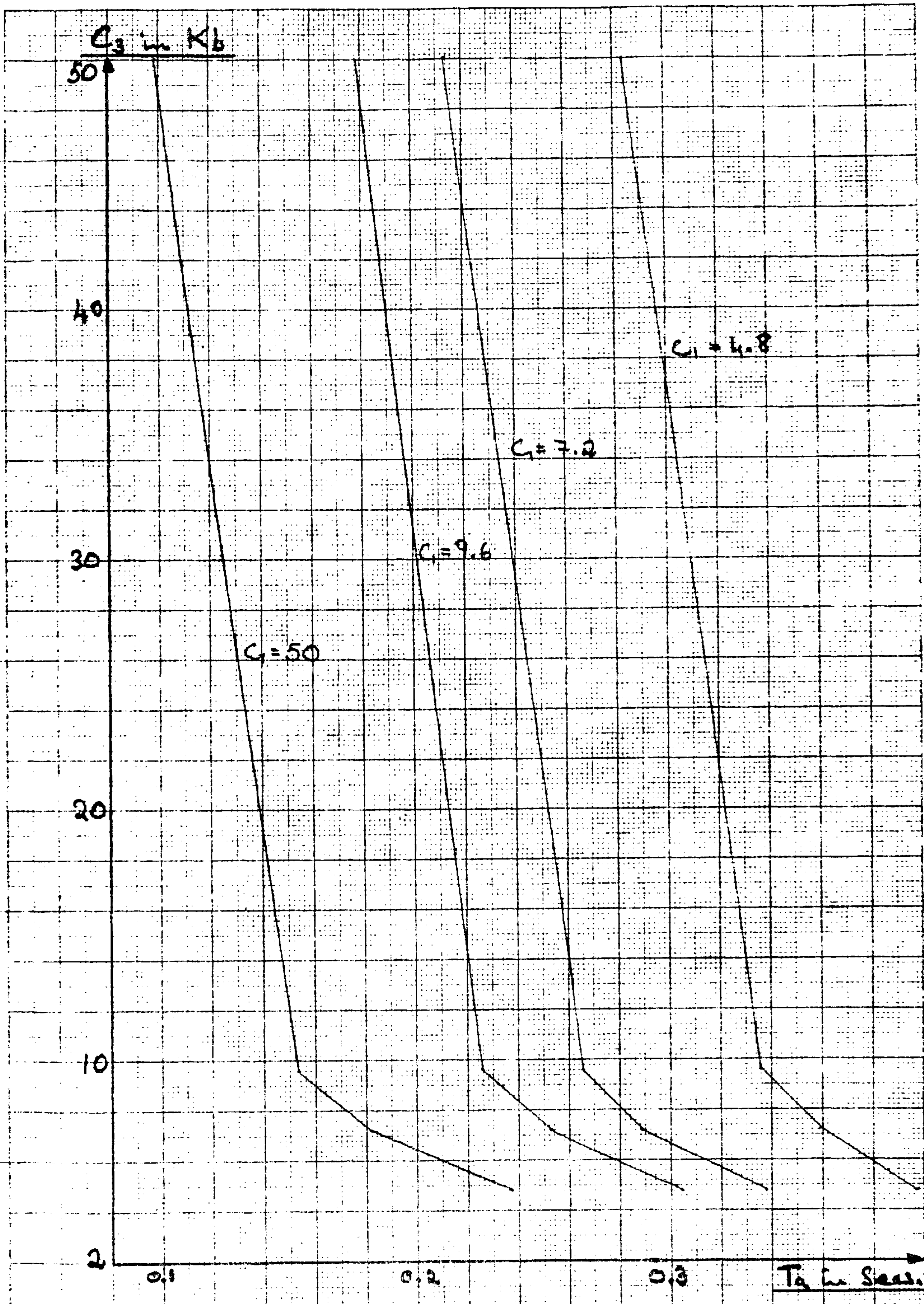


Fig. 6-32 VARIATION OF Ta WITH C_3 FOR VARIOUS C_1.

CHAPTER VII

CONCLUSION

7.1 Summary of research aims achieved.

In the preceding chapters we have described research which attempted to provide a system for modelling computational activity and to demonstrate its practicality in a real situation. We now summarize the results and draw some conclusions. In the next section we make suggestions for further research.

The SHAPE system uses directed graphs whose elements have associated numerical data in order to describe both the hardware and software of a computational process. This combination may be thought of as a type of notation which can not only perform a descriptive role, but is also capable of execution in the sense of enacting the computation described. We have found the notation useful in its own right as a means of clearly and unambiguously specifying hardware and software. Its graphical nature is particularly suited to the expression of parallelism in software and the hardware counterpart of parallelism, namely multi-processing. In the case of software the dependences of processes are expressed by the use of arcs and nodes. The interaction of processes via the data they produce is described by the repartition matrix associated with each node.

It seems clear that a computation can be hardened or softened to an arbitrary degree, that is that the proportion expressed in hardware can vary from next to nothing to the entire computation. This point of view has led us to search for representations whose structure is applicable to hardware and software, and which maximize the number of aspects common to both. To some extent we have succeeded in this. The hardware and software representations are both graphical in form.

The nodes of SIgraphs and PIgraphs correspond to the software and hardware aspects of data. Similarly the arcs of each type of graph correspond to the software and hardware aspects of data transformation.

Of course the correspondence is not as good as we would wish it to be. Nevertheless it has allowed us to develop, without undue difficulty, an algorithm which models the execution of a computation in fairly general terms. We have called the algorithm's action the binding of a SIgraph and Pigraph. This simple interpretation of computation is possible because of the correspondence between the two graphs. The algorithm is at the heart of the SHAPE implementation, in which it is called the allocator.

Our use of nodes and arcs to model data and data transformation has allowed us to give a simple and consistent account of data transmission and storage. Data transmission is treated as the operation of an identity processor since no alteration occurs, and data storage is equivalent to the operation of the null processor. The structure of the graphs is recursive, allowing a process to be modelled not only by an arc, but also by a functionally equivalent subsidiary graph. This allows a model to span an arbitrary number of levels of detail. The graphs can potentially model a recursive structure, and consequently the SHAPE implementation has the ability to bind the two graphs recursively.

By implementing the modelling system, and using it in the context of a real design situation, we have tried to verify that it could be programmed, used, and provide accurate results.

A high level language, SIMULA 67, was chosen for the implementation because of the need for list processing, class definition, simulation facilities, and recursion. Class definition was used to provide the elements of the system such as arcs and nodes, and their linkage in a graph model was reproduced within the computer with list processing primitives. The most convenient way to program the binding algorithm was through the use of simulation facilities, since an obvious fundamental property of a computation is that it advances with time. We did not succeed in implementing the full facilities of the system because of insufficient time and limitations in the software that was used. However, enough was completed to proceed with a validation of the SHAPE program.

A store and forward network was chosen as the validation example for two reasons. Firstly, our knowledge of the subject was very detailed, and this helped to ensure an accurate model and thorough comprehension of the results. Secondly, the modelling experience was directly relevant to the intended application of the SHAPE system to the European linkage with the ARPA network. The results produced by the store and forward model did not require us to reject the validity of the SHAPE system at the ninety-five per cent confidence level. In fact a good agreement with theoretically expected results was obtained.

We then turned to using the modelling system in a real design situation, namely the choice of channel capacities in two of the channels forming part of the European ARPANET. A model was successfully developed and used to predict mean interactive response times under various conditions. This work, and the design recommendations it led to, are described in the previous

chapter. In addition to their immediate utility, these results demonstrate the capacity of the SHAPE system to describe and model the opening of an application which is neither trivial nor artificial.

Graphs have been used previously to describe programs, and to discover properties of programs so described, but no equivalent schemes were produced for hardware description.

Our approach has been novel in the recognition of hardware software equivalence, which led us to provide a system of graphical description applicable to both, and in the requirement that the descriptions should be directly usable for modelling computational activity. As a result the SHAPE system is original in its use of the same elements to provide both types of description, and in leading to a new view of computational activity, namely the binding together of a software and a hardware graph.

Earlier the graph descriptions have been called a type of notation. This notation is not only descriptive but also executable in the same way as a high-level programming language. We believe that the SHAPE system is perhaps the first to provide an executable graphical notation for modelling computational activity, and the means for executing it.

7.2 Suggestions for further research.

As is necessarily the case in research, we have not been able to attempt all we would have wished to, nor even achieve all we attempted. Our suggestions for further research inevitably stem from this situation. In working towards our main aims we have sometimes had to make a choice or accept an assumption without sufficiently deep investigation. These are also areas of potential research.

For example, the correspondence between the elements of hardware and software graphs is not as good as it might be. In particular the IFloop, while adequate for modelling, does not correspond well with a PIarc whose initial and terminal nodes are the same. It might perhaps have been better to include the two IFcodes as normal software functions in the PHI vector. The choice of possible IFcode actions was not deeply investigated and could perhaps be rationalized. In addition, we would have liked to give more time to the representation of processors which can transfer data between more than one pair of stores. While their representation as a number of PIarcs with an allocation interlock is well justified, further consideration might provide a more elegant model.

An area which we have not touched upon at all, and which should be of some interest, is the investigation of a computation described by a SIgraph and PIgraph by analytical techniques rather than by actually binding the two graphs. Furthermore, while we have presented a model of the binding process, this has not included any techniques by which the binding algorithm might optimize some aspect of the computational process.

For example, the algorithm might attempt to minimize the total binding time for the graph pair, either by complete optimization or by a sub-optimal technique such as limited look-ahead. In addition to these subjects, we would have wished to give more attention to the equivalence between preemptive and fractional allocation strategies.

Turning now to the SHAPE program, we must make the comment that although the features of SIMULA 67 are very well suited to the implementation of the modelling system, the compiler and the programs it produces are also very inefficient. This had several undesired effects on our work. Firstly, it was not possible to implement the full set of modelling facilities. In particular, the ACT matrix attribute of a SInode was not included, and consequently neither were the related functions of mode 3 binding and error modelling. Further work we would suggest in this area would be the creation of an efficient implementation that provided these facilities.

In the SHAPE implementation the allocator binds a ready SIarc to the PIarc which minimizes the duration of the resulting tie. A useful feature which might be added to the system would be the insertion by the modeller of alternative binding strategies with his run-time data. Lastly, an aspect of the SHAPE program which is capable of improvement is the activation of ready SIarcs that are waiting for the PIgraph resources they need to become available. The existing method using delay arcs is primitive and not very efficient.

Finally, it must be clear that one validation cannot be an exhaustive test of the SHAPE program, and one application

cannot confirm its utility in a wide range of computations. Consequently we would wish to see the modelling system used in other areas beside the one chosen for this thesis. Where theoretically expected results were known further validation would be possible, and where none were available such use would provide additional demonstrations of the system's wider applicability.

APPENDIX I

BIBLIOGRAPHY

ABAT 68 Queuing Analysis of IBM 2314 Disc Storage Facility. ABATE, DUBNER & WEINBERG. Journal of the ACM. Vol.15, No.4, 1968.

ABLO 68 Irreducible Decompositions of Transformation Graphs by Assignment Techniques. ABLOW et al. IEEE Transactions on Electronic Computers. Vol.17, No.4, 1968.

ABRA 70 The ALOHA System - Another Alternative for Computer Communications. N. ABRAMSON. AFIPS Papers. FJCC, 1970.

AOKI 63 A Probabilistic Analysis of Computing Load Assignment in a Multiprocessor Computer System. AOKI, ESTRIN & MANDELL. AFIPS Papers. FJCC, 1963.

BAER 68 Graph Models of Computations in Computer Systems. BAER. Ph.D. thesis. Dept. of Eng., U. of Cal., Los Angeles, 1968.

BAER 69 Bounds for Maximum Parallelism in a Bilogic Graph Model of Computations. BAER & ESTRIN. IEEE Trans. on Elec. Comp. Vol.18, No.11, 1969.

BAER 70 Legality and Other Properties of Functional Programs. BAER, BOVET & ESTRIN. Journal of the ACM. Vol.17, No.3, 1970.

BART 68 Transmission Control in a Local Data Network. BARTLETT. IFIP 68 Papers, 1968.

BART 69 A Note on Reliable Full-Duplex Transmission over Half-Duplex Links. BARTLETT, SCANTLEBURY & WILKINSON. Communications of the ACM. Vol.12, No.5, 1969.

BASK 69 A Modular Computer Sharing System. BASKIN et al. Communications of the ACM. Vol.12, No.10, 1969.

BERN 66 Analysis of Programs for Parallel Processing. BERNSTEIN. IEEE Trans. on Elec. Comp. Vol.15, No.5, 1966.

BLUM 67 A Machine Independent Theory of Complexity of Recursive Fns. BLUM. Journal of the ACM. Vol.14, No.2, 1967.

BOVE 68 Memory Allocation in Computer Systems. D. P. BOVET. Ph.D. thesis. U.C.L.A., June 1968.

BOVE 70A A Dynamic Memory Allocation Algorithm. BOVET & ESTRIN. IEEE Trans. on Elec. Comp. Vol.19, No.5, 1970.

BOVE 70B On Static Memory Allocation in Computer Systems. BOVET & ESTRIN. IEEE Trans. on Elec. Comp. Vol.19, No.6, 1970.

BOWD 69 Priority Assignment in a Network of Computers. BOWDON. IEEE Trans. on Elec. Comp. Vol.18, No.11, 1969.

BRUN 71 A Theory of Asynchronous Control Networks. BRUNO & ALTMAN. IEEE Trans. on Elec. Comp. Vol.20, No.6, 1971.

BUDN 71 The Organization and Use of Parallel Memories. BUDNIK & KUCK. IEEE Trans. on Elec. Comp. Vol.20, No. 12, 1971.

BURN 70 A Study of Interleaved Memory Systems. BURNETT & COFFMAN. AFIPS Papers. SJCC, 1970.

CARR 70 HOST-HOST Communications Protocol in the ARPA Network. CARR et al. AFIPS Papers, SJCC, 1970.

CERF 71 Measurement of Recursive Programs. V.G. CERF & G. ESTRIN. IFIP 71 Papers, 1971.

CERF 72 Multiprocessors, Semaphores, and a Graph Model of Computation. V.G. CERF. Ph.D. thesis. U.C.L.A., April, 1972.

CHEN 72 Memory Requirements in a Multiprocessing Environment. CHEN & EPLEY. Journal of the ACM. Vol.19, No.1, 1972.

COFF 67 Bounds on Parallel-Processing of Queues with Multiple-Phase Jobs. COFFMAN. Naval Research Logistics Quarterly. Vol.14, No.3, 1967.

COFF 70 Waiting Time Distributions for Processor Sharing Systems. COFFMAN, MUNTZ & TROTTER. Journal of the ACM. Vol.17, No.1, 1970.

COHE 68 A Parallel Process Definition and Control System. COHEN. AFIPS Papers. FJCC, 1968.

COHL 64 A Bit-Access Computer in a Communications System. COHLER & RUBINSTEIN. AFIPS Papers. FJCC, 1964.

COLE 71 Computer Network Measurements: Techniques & Experiments. G.D. COLE. Ph.D. thesis. Dept. of Eng., U. of Cal., Los Angeles, October, 1971.

CONS 68 Control of Sequence and Parallelism in Modular Programs. CONSTANTINE. AFIPS Papers. SJCC, 1968.

CORN 70 An Efficient Algorithm for Graph Isomorphism. CORNELL & GOTLIEB. Journal of the ACM. Vol.17, No.1, 1970.

CRES 70 A Language for Treating Graphs. CRESPI-REGHIZZI & MORPURGO. Communications of the ACM. Vol.13, No.5, 1970.

DARR 69 Description, Simulation, and Automatic Implementation of Digital Computer Processors. DARRINGER. Ph.D. thesis, Dept. of Elec. Eng., Carnegie-Mellon U., Pittsburgh, May, 1969.

DAVI 68 The Principles of a Data Communications Network for Computers and Remote Peripherals. DAVIES. IFIP 68 Papers, 1968.

DEME 72A The Synthesis of Computer-Communications Networks. J. DEMERCADO & K. TOTH. Terrestial Planning Branch, Dept. of Communications, Ottawa, May, 1972

DENN 68 Resource Allocation in Multiprocess Computer Systems. Ph.D. P.J. DENNING. Dept. of Elec. Eng., M.I.T., May, 1968.

DENN 68 Programming Generality, Parallelism, and Computer Architecture. DENNIS. IFIP 68 Papers, 1968.

DIJK 66 Cooperating Sequential Processes. E.W. DIJKSTRA. Proceedings of the NATO Advanced Study Institute Summer School on Programming Languages, Villard-de-Lans, 1966. Academic Press.

DIJK 68 Structure of THE Multiprogramming System. DIJKSTRA. Communications of the ACM. Vol.11, No.5, 1968.

DIJK 72 A Class of Allocation Strategies Inducing Bounded Delays Only. DIJKSTRA. AFIPS Papers. SJCC, 1972.

DONN 69 Some Techniques for Using Pseudorandom Numbers in Computer Simulation. DONELLY. Communications of the ACM. Vol.12, No.7, 1969.

EARN 72 Analysis of Graphs by Ordering of Nodes. EARNEST, BALKE, & ANDERSON. Journal of the ACM. Vol.19, No.1, 1972.

EISN 62 A Generalized Network Approach to the Planning and Scheduling of a Research Project. H. EISNER. Operations Research No.10, January-February, 1962.

ESTR 63A Parallel Processing in a Restructurable Computer System. ESTRIM et al. IEEE Transactions on Electronic Computers. Vol.12, No.6, 1963.

ESTR 63B Automatic Assignment of Computations in a Variable Structure Computer System. ESTRIN et al. IEEE Trans. on Elec. Comp. Vol.12, No.6, 1963.

ESTR 72 Modelling, Measurement, and Computer Power. G. ESTRIN, R.R. MUNTZ, R. UZGALIS. AFIPS Papers. SJCC, 1972.

FENI 69 An Analytic Model of Multiprogrammed Computing. FENICHEL & GROSSMAN. AFIPS Papers. SJCC, 1969.

FIFE 66 An Optimization Model for Time-Sharing. FIFE. AFIPS Papers. SJCC, 1966.

FOLE 67 A Markovian Model of the U. of Michigan Executive System. FOLEY. Communications of the ACM. Vol.10, No.9, 1967.

FRAN 69 Analysis & Optimization of Disc Storage Devices for Time-Sharing Computer Systems. FRANK. Journal of the ACM. Vol.16, No.4, 1969.

FRAN 70 Topological Considerations in the Design of the ARPA Computer Network. FRANK et al. AFIPS Papers. SJCC, 1970.

FRAN 72 Computer Communication Network Design - Experience Experience with Theory and Practice. H. FRANK, R.E. KAHN, L. KLEINROCK. AFIPS Papers. SJCC, 1972.

FULL 72 An Optimal Drum Scheduling Algorithm. FULLER. IEEE Trans. on Elec. Comp. Vol.21, No.11, 1972.

FULT 72 Adaptive Routing Techniques for Message Switching Computer-Communication Network. G.L. FULTZ. Ph.D. thesis. U.L.C.A. July, 1972.

GARN 68 Mathematical Models of Information Systems. H.L. GARNER. Michigan U., Ann Arbor. July, 1968. U.S. Gov't R. & D. Reports. Vol.68, pp. 65-66 (A) AD 673 386 CFSTI.

GILB 72 Interference Between Communicating Parallel Processes. GILBERT & CHANDLER. Communications of the ACM. Vol.15, No.6, 1972.

GONZ 69 A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs. GONZALES & RAMAMOORTHY. AFIPS Papers. FJCC 1969.

GOSD 66 Explicit Parallel Processing Description and Control in Programs for Multi and Uni-processor Computers. GOSDEN. AFIPS Papers. FJCC, 1966.

GOTO 65 Memory Systems. GOTO. IFIP 65 Papers, 1965.

GRAH 66 Bounds for Certain Multiprocessing Anomalies. R.L. GRAHAM. Bell Systems Tech. Journal. Vol.45, pp. 1573-1581. November, 1966.

HABE 69 Prevention of System Deadlocks. HABERMANN. Communications of the ACM. Vol.12, No.7, 1969.

HART 65 Classifications of Computations by Time and Memory Requirements. HARTMANIS, LEWIS & STEARNS. IFIP 65 Papers, 1965.

HEAR 70 The Interface Message Processor for the ARPA Network. HEART et al. AFIPS Papers. SJCC, 1970.

HEBA 71 A Graph Model for Analysis of Deadlock Prevention in Systems with Parallel Computations. P.G. HEBALKAR. IFIP 71 Papers, 1971.

HELL 61 Sequencing Aspects of Multiprogramming. HELLER. Journal of the ACM. Vol.8, No.3, 1961.

HELL 72 A Measure of Computational Work. HELLERMAN. IEEE Transactions on Electronic Computers. Vol.21, No.5, 1972.

HEMM 69 Generating Pseudorandom Numbers on a Two's Complement Machine. HEMMERLE. Communications of the ACM. Vol.12, No.7, 1969.

HOAR 69 An Axiomatic Basis for Computer Programming. HOARE. Communications of the ACM. Vol.12, No.10, 1969.

HOLT 71 Comments on Prevention of System Deadlocks. HOLT. Communications of the ACM. Vol.14, No.1, 1971.

HOPC 68 Relations between Time & Tape Complexities. HOPCRAFT & ULLMAN. Journal of the ACM. Vol.15, No.3, 1968.

HU 61 Parallel Sequencing and Assembly Line Problems. T.C. HU. Operations Research. Vol.9, No.6, November, 1961.

HUTC 65 Computer Systems Design and Analysis through Simulation. HUTCHINSON & MAGUIRE. AFIPS Papers. FJCC, 1965.

IRAN 71 On Network Linguistics and the Conversational Design of Queuing Networks. IRANI & WALLACE. Journal of the ACM. Vol.18, No.4, 1971.

KARP 66 Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing. KARP & MILLER. SIAM Journal of Applied Mathematics. Vol.14, pp. 1390-1411, November 1966.

KATZ 66 Simulation of a Multiprocessor Computer System. KATZ. AFIPS Papers. SJCC, 1966.

KLEI 64 COMMUNICATION NETS. L. KLEINROCK. McGraw Hill, Inc. 1964.

KLEI 66 Sequential Processing Machines (S.P.M.) Analysed with a Queueing Theory Model. KLEINROCK. Journal of the ACM. Vol.13, No.2, 1966.

KLEI 67 Time Shared Systems: A Theoretical Treatment. KLEINROCK. Journal of the ACM. Vol.14, No.2, 1967.

KLEI 69A Comparison of Solution Methods for Computer Network Models. KLEINROCK. Proceedings of the Computers and Communications Conference, September, 1969.

KLEI 69B Models for Computer Networks. KLEINROCK. Proceedings of the International Communications Conference, 1969.

KLEI 70A Analytic and Simulation Methods in Computer Network Design. KLEINROCK. AFIPS Papers. SJCC, 1970.

KLEI 70B A Continuum of Time-Sharing Scheduling Algorithms. KLEINROCK. AFIPS Papers. SJCC 1970.

KLEI 71 Tight Bounds on the Average Response Time for Time Shared Computer Systems. L. KLEINROCK, R.R. MUNTZ & J. HSU. IFIP 71 Papers, 1971.

KLEI 72 Processor Sharing Queueing Models of Mixed Scheduling Disciplines for Time-Shared Systems. KLEINROCK & MUNTZ. Journal of the ACM. Vol.19, No.3, 1972.

KLEI 68 Certain Analytic Results for Time-Shared Processors. KLEINROCK. IFIP 68 Papers, 1968.

KOTO 68 Transformation of Sequential Programs into Asynchronous Parallel Programs. KOTOV & NARINYANI. IFIP 68 Papers, 1968.

LASS 69 Productivity of Multiprogrammed Computers. LASSER. Communications of the ACM. Vol.12, No.12, 1969.

LEWI 71 A Cyclic-Queue Model of System Overhead in Multi-Programmed Computer Systems. LEIWS & SHEDLER. Journal of the ACM. Vol.18, No. 2, 1971.

LOWE 70 Automatic Sequentation of Cyclic Program Structures Based on Connectivity of Processor Timing. LOWE. Communications of the ACM. Vol. 13, No.1, 1970.

LUCO 68 Asynchronous Computational Structures. F.L. LUCONI. Ph.D. thesis. M.I.T. January, 1968.

MANA 67 Production and Stabilization of Real-Time Task Schedules. MANACHER. Journal of the ACM. Vol.14, No.3, 1967.

MANN 70 Termination of Programs Represented as Interpreted Graphs. MANNA. AFIPS Papers. SJCC, 1970.

MART 66 Automatic Assignment and Sequencing of Computations on Parallel Processor Systems. D.F. MARTIN. Ph.D. thesis. Dept. of Eng., U. of Cal., Los Angeles, 1966.

MART 67A Experiments on Models of Computations and Systems. MARTIN & ESTRIN. IEEE Trans. on Elec. Comp. Vol.16, No.1, 1967.

MART 67B Models of Computational Systems - Cyclic to Acyclic Graph Transformations. MARTIN & ESTRIN. IEEE Trans. on Elec. Comp. Vol.16, No.1, 1967.

MART 67C Models of Computations and Systems-Evaluation of Vertex Probabilities in Graph Models of Computations. MARTIN & ESTRIN. Journal of the ACM. Vol.14, No.2, 1967.

MART 69 Path Length Computations on Graph Models of Computations. MARTIN & ESTRIN. IEE Trans. on Elec. Comp. Vol. 18, No.6, 1969.

MORG 71 Representation and Analysis of Computer Systems and Processes. D.E. MORGAN. Ph.D. thesis. Dept. of Comp. Sci., U. of Waterloo, June, 1971.

MUNT 69A Scheduling of Computations on Multiprocessor Systems: the Preemptive Assignment Discipline. MUNTZ. Ph.D. thesis. Princeton U., Princeton, April, 1969.

MUNT 69B Optimal Preemptive Scheduling on Two-Processor Systems. MUNTZ & COFFMAN. IEEE Trans. on Elec. Comp. Vol.18, No.11, 1969.

MUNT 70 Preemptive Scheduling of real Time Tasks on Multi-Processing Systems. MUNTZ & COFFMAN. Journal of the ACM Vol.17, No.2, 1970.

NAKA 71 A Feedback Queueing Model for an Interactive Computer System. G. NAKAMURA. AFIPS Papers. FJCC, 1971.

NEWE 67A Closed Queueing Systems with Exponential Servers. G.F. NEWELL & W.J. GORDON. Operations Research No. 15, 1967.

NEWE 67B Cyclic Queueing Systems with Restricted Length Queues. G.F. NEWELL & W.J. GORDON. Operations Research No.15, 1967.

NIEL 66 Analysis of General Purpose Computer Time-Sharing Systems. NIELSON. Ph.D. thesis. Stanford Computation Center, Stanford U. 1966.

NIEL 67 An Approach to Simulation of Time-Sharing Systems. NIELSON. AFIPS Papers. FJCC, 1967.

ORNS 72 The Terminal IMP for the ARPA Network. S.M. ORNSTEIN, F.E. HEART, W.R. CROWTHER, H.K. RISING, S.B. RUSSELL, A. MICHEL. AFIPS Papers. SJCC 1972.

PARN 67 SODAS and a Methodology for System Design. PARNAS & DARRINGER. AFIPS Papers. FJCC, 1967.

PARN 69A More on Simulation Languages and Design Methodology for Computer Systems. PARNAS. AFIPS Papers. SJCC, 1969.

PARN 69B On Simulating Networks of Parallel Processes in which Simultaneous Events May Occur. PARNAS. Communications of the ACM. Vol.12, No.9, 1969.

PETR 62 Kommunikation mit Automaten. C.A. PETRI. Schriften des Reinisch-West falischen Inst. Instrumentelle Math. und der Universität Bonn, 1962.

PFAL 72 Graph Structures. PFALTZ. Journal of the ACM. Vol.19, No.2, 1972.

RAMA 72 Optimal Scheduling Strategies in a Multiprocessor System. RAMAMOORTHY, CHANDY & GONZALEZ. IEEE Transactions on Electronic Computers. Vol.21, No.2, 1972.

RAND 68 Dynamic Storage Allocation Systems. RANDALL. Communications of the ACM. Vol. 11, No.5, 1968.

RAND 69 Note on Storage Fragmentation & Program Segmentation. RANDALL. Communications of the ACM. Vol.12, No.7, 1969.

RASC 70 A Queueing Theory Study of Round Robin Scheduling of Time Shared Computer Systems. RASCH. Journal of the ACM. Vol.17, No.1, 1970.

REIT 68 Scheduling Parallel Computations. REITER (ICS). Journal of the ACM. Vol.15, No.4, 1968.

ROBE 70 Computer Network Development to Achieve Resource Sharing. ROBERTS. AFIPS Papers. SJCC, 1970.

RODR 69 A Graph Model for Parallel Computations. RODRIGUEZ. Ph.D. thesis. Electronic Systems Lab., Dept. of Elec. Eng., M.I.T., Cambridge, Mass. September, 1969.

ROSE 69 A Case Study in Programming for Parallel Processors. ROSENFELD. Communications of the ACM. Vol.12, No.12, 1969.

RUSS 69A Automatic Program Analysis. RUSSELL. Ph.D. thesis. U.C.L.A., 1969.

RUSS 69B Measurement Based Automatic Analysis of Fortran Programs. RUSSELL & ESTRIN. AFIPS Papers. SJCC, 1969.

SCAN 68 The Design of a Message Switching Centre for a Digital Communications Network. SCANTLEBURY, WILKINSON & BARTLETT. IFIP 68 Papers, 1968.

SCAR 68 The Basic Language Project. SCARROT & ILIFFE. IFIP 68 Papers, 1968.

SHCE 67 An Analysis of Time-Shared Computer Systems. A.L. SCHERR. M.I.T. Research Monograph No.36, 1967.

SCHW 61 An Automatic Sequencing Procedure with Applications to Parallel Programming. SCHWARTZ. Journal of the ACM. Vol.8, No.4, 1961.

SHEM 67 Some Mathematical Considerations of Time Shared Systems' Scheduling Algorithms. SHEMER. Journal of the ACM. Vol.14, No.2, 1967.

SHOS 69 Synchronization in a Parallel-Accessed Data Base. SHOSHANI & BERNSTEIN. Communications of the ACM. Vol.12, No.11, 1969.

SLUT 68 The Flow Graph Schemata Model of Parallel Computation. D.R. SLUTZ. Dept. of Elec. Eng., M.I.T., September, 1968.

SMIT 66 An Analysis of Time-Sharing Computer Systems Using Markov Models. SMITH. AFIPS Papers. SJCC, 1966.

STEV 68 System Evaluation of the CDC 6600. STEVENS. IFIP 68 Papers, 1968.

STIM 69 Some Criteria for Time-Sharing Performance. STIMLER. Communications of the ACM. Vol.12, No.1, 1969.

STRE 70 An Analysis of the Instruction Execution Rate in Certain Computer Structures. W.D. STRECKER. Ph.D. thesis. Dept. of Comp. Sci., Carnegie-Mellon U., Pittsburgh, June, 1970.

TESL 68 A Language Design for Concurrent Processes. TESLER & ENEA. AFIPS Papers. SJCC, 1968.

VANH 66 Computer Design for Asynchronously Reproducible Multiprocessing. E.C. VAN HORN. Dept. of Elec. Eng., M.I.T. Ph.D. thesis. November, 1966.

VOLA 70 Graph Model Analysis and Implementation of Computational Sequences. S.A. VOLANSKY. Ph.D. thesis. Dept. of Eng., U. of Cal., Los Angeles, June 1970.

WALD 72 A System for Interprocess Communication in a Resource Sharing Computer Network. WALDEN. Communications of the ACM. Vol.15, No.4, 1972.

WALL 66 Markovian Models and Numerical Analysis of Computer Systems Behaviour. WALLACE & ROSENBURG. AFIPS Papers. SJCC, 1966.

WEBE 64 UNISIM - A Simulation Program for Communications Networks. WEBER & GIMPELSON. AFIPS Papers. FJCC, 1964.

WILK 68 The Control Functions in a Local Data Network. WILKINSON & SCANTLEBURY. IFIP 68 Papers, 1968.

WINO 67 On the Time Required to Perform Multiplication. WINOGRAD. Journal of the ACM. Vol.14, No.4, 1967.

ZEIG 71 Nodal Blocking in Large Networks. J.F. ZEIGLER. Ph.D. thesis. U.C.L.A., October, 1971.

ZEIG 72 Towards a Formal Theory of Modelling and Simulation: Structure Preserving Morphisms. ZEIGLER. Journal of the ACM. Vol.19, No.4, 1972.

Computer Network Design.

ABRA 70 The ALOHA System - Another Alternative for Computer Communications. N. ABRAMSON. AFIPS Papers. FJCC, 1970.

ANSL 72 Implementation of International Data Exchange Networks. N. G. ANSLOW & J. HANSCOTT. First Int. Conf. on Computer Communications, 1972.

ASSN 72 Analytical Results for the ARPANET Satellite System Model Including the Effects of the Retransmission Delay Distribution. ARPANET Satellites System Note 12. L. KLEINROCK & S.S. LAM. August, 1972.

BARA 64A On Distributed Communications: Introduction to Distributed Communication Networks. P. BARAN. The Rand Copporation, Memorandum, RM-3420-PR. August, 1964.

BARB 69 Experience with the Use of the British Standard Interface in Computer Peripherals and Communication Systems. D.L.A. BARBER. ACM Symposium on Problems in the Optimization of Data Communication Systems, Pine Mountain, Georgia, pp. 173-178. October, 1969.

BARB 72 The European Computer Project. D.L. BARBER. First Int. Conf. on Computer Communications, 1972.

BART 68 Transmission Control in a Local Data Network. BARTLETT. IFIP 68 Papers, 1968.

BEER 71 Tymnet - A Serendipitous Evolution. M.P. BEERS & N. C. SULLIVAN. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Comm. Systems. Oct., 1971.

BELY 69 On the Structure of a Heterogenous Computing System, Controlled by a Large Digital Computer. V.I. BELYAKOV-BODIN & Y.I. TORGOV. Foreign Tech. Division, Wright-Patterson AFB Rept. FTD-HT-23-1450-68, AD699640. Oct.'69.

BENE 66 Programming and Control Problems Arising from Optimal Routing in Telephone Networks. V.E. BENES. B.S.T.J. 45: 1373-1438. November, 1966.

BENV 69 System Load Sharing Study. A.A. BENVENUTO et al. The MITRE Corporation, Rept. MTR-5062. 1969.

BINA 71 Design Aspects of a Circuit Switching System for a National Network. R.A. BINA. Proceedings of the Int. Conf. on Communications, pp. 23.12-23.17. June, 1971.

BOEH 64 Digital Simulation of Hot Potato Routing in a Broadband Distributed Communication Network. S. BOEHM & P. BARAN. Rand Corp., Memorandum RM-3103-PR. August, 1964.

BOEH 66 Adaptive Routing Techniques for Distributed Communication Systems. B.W. BOEHM & R.L. MOBLEY. Rand Corp. Memorandum RM-4781-PR. 1966.

BOLT 71 Interface Message Processor: Specifications for the Interconnection of a Host and an IMP. BOLT, BARANEK & NEWMAN, Inc. Rept. 1822. February, 1971.

BOOT 72 The Use of Distributed Data Bases in Information Networks. E.M. BOOTH. 1st Int. Conf. on Computer Communications, 1972.

BOWD 69 Priority Assignment in a Network of Computers. E.K. BOWDEN, Sr. IEEE Trans. on Elec. Comp. Vol. C-18, pp. 1021-1026. November, 1969.

CARR 70 HOST-HOST Communications Protocol in the ARPA Network. CARR et al. AFIPS Papers, SJCC, 1970.

COHE 70 Control of Data Processor Networks. I. COHEN. 1970 IEEE Int. Conf. on Communications. pp. 19.28-19.34, 1970.

COLE 71 Computer Network Measurements: Techniques and Experiments. G.D. COLE. Ph.D. thesis. Univ. of Calif. Los Angeles, California. 1971.

COX 70 General System Organisation of Multi-Processor Configurations. P.R. COX. Software 70, Sheffield, England. pp. 33-40. April, 1970.

CROC 72 Function Oriented Protocols for the ARPA Computer Network. S.D. CROCKER, J. HEAFNER, J. METCALFE & J. POSTEL. AFIPS Papers, SJCC, 1972.

DAVI 67 A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals. D.W. DAVIES, K.A. BARTLETT, R.A. SCANTLEBURY & P.T. WILKINSON. ACM Symposium on Operating System Principles, Gatlinburg, 1967.

DAVI 68 The Principles of a Data Communications Network for Computers and Remote Peripherals. DAVIES IFIP 68 Papers, 1968.

DAVI 71 The Control of Congestion in Packet Switching Networks. D.W. DAVIES. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Comm. Systems. October, 1971.

DAY 68 Rio Grande Message Switching/Transportation System. W.J. DAY. Proc. of the 23rd National ACM Conf. pp. 307-320. 1968.

DEME 72A The Synthesis of Computer-Communications Networks. J. DEMERCADO & K. TOTH. Terrestial Planning Branch, Dept. of Communications, Ottawa. May, 1972.

DEME 72B The Canadian Universities Computers Network - Topological Considerations. J. DEMERCADO et al. 1st Int. Conf. on Computer Communications, 1972.

DESP 72 A Packet Switching Network with Graceful Saturated Operation. R.F. DESPRES. 1st Int. Conf. on Computer Communications, 1972.

DOLL 69 Efficient Allocation of Resources in Centralized Computer Communication Network Design. D.R. DOLL. Ph.D. thesis. University of Michigan, Systems Eng. Lab., Tech. Rept. 02641-1-T. June, 1969.

DUBN 70 TICKETRON - A Successfully Operating System without an Operating System. H. DUBNER & J. ABATE. AFIPS Conf. Proc., 36: 143-155, SJCC, 1970.

EVAN 67 Experience Gained from the American Airlines SABRE System Control Program. J. EVANS. Proc. of the ACM National Meeting, pp. 77-85. August, 1967.

EVER 57 SAGE: A Data Processing System for Air Defense. R.R. EVERETT, C.A. ZRAKET & H.D. BANNINGTON. EJCC, pp. 148-155. 1957.

FARB 71 The System Architecture of the Distributed Computer System - An Informal Description. D.J. FARBER & K.C. LARSON. Univ. of Calif., Irvine Tech. Rept. No.11, September, 1971.

FARB 72 The Structure of a Distributed Computer System - The Distributed File System. D.J. FARBER & F.R. HEINRICH. 1st Int. Conf. on Computer Communications, 1972

FISH 71 Introduction to the DATRAN Switched Digital Network. C.R. FISHER. Proc. of the Int. Conf. on Communications, pp. 23.1-23.3. June, 1971.

FRAN 69 Design of Economical Offshore Natural Gas Pipeline Networks. H. FRANK, B. ROTHFARB, D. KLEITMAN & K. STEIGLITZ. Office of Emergency Preparedness, Rept. No. R-1. Washington, D.C. January, 1969.

FRAN 70A Topological Considerations in the Design of the ARPA Network. H. FRANK, I.T. FRISCH & W.S. CHOU. 1970 Spring Joint Computer Conf. AFIPS Proc. Vol.36, pp. 581-587, 1970.

FRAN 70B Optimal Design of Centralized Computer Networks. H. FRANK, I.T. FRISCH, W.S. CHOU & R. VAN SLYKE. Proc. 1970 IEEE International Communications Conf. pp. 19.1-19.10. 1970.

FRAN 71 Optimal Design of Centralized Computer Networks. H. FRANK, I.T. FRISCH, R. VAN SLYKE & W.S. CHOU. Networks, Vol.1, No.1, pp. 43-57, 1971. (Same paper as previous entry)

FRAN 72 Computer Communication Network Design - Experience with Theory and Practice. H. FRANK, R.E. KAHN, L. KLEINROCK. AFIPS Papers. SJCC, 1972.

FRED 71 A Computer Network Interface for OS/MVT. D. FREDERICKSEN & R.W.RYNIKER. IBM Research Dept. RC3317. April, 1971.

FULT 71 Adaptive Routing Techniques for Store-and-Forward Computer-Communication Networks. G.L. FULTZ & L. KLEINROCK. Proc. of the Int. Conf. on Communications. pp. 39.1-39.8. 1971.

FULT 72 Adaptive Routing Techniques for Message Switching Computer-Communication Networks. G.L. FULTZ. Ph.D. thesis. U.C.L.A. July, 1972.

GAIN 71 The Emergence of National Networks. E.V. GAINS, Jr. & J.M. TAPLIN. Telecommunications. December, 1971.

GUND 63 Engineering Design and Implementation of a Multi-Computer Data Processing System for a Navy Command and Control Center. R.C. GUNDERSON & J.O. JOHNSON. Proc. 7th Int. Convention on Military Electronics. Western Periodicals, N. Hollywood, Calif. 1963.

HAMA 70 Distributed Computer Systems. R.M. HAMAKER. Telecommunications. Vol.4, pp. 25-30, March, 1970.

HAMS 68 Communication System Engineering Handbook. M. HAMSHER (Ed.). McGraw-Hill. 1968.

HANS 71 Reliability Considerations in Centralized Computer Networks. E. HANSLER, G.K. MCAULIFFE, R.S. WILKOV. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communications Systems. Oct. 1971.

HAYE 71 Traffic and Delay in a Circular Data Network. J.F. HAYES & D.N. SHERMAN. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Comm. Systems. October, 1971.

HEAR 70 The Interface Message Processor for the ARPA Computer Network. F.HEART, R.KAHN, S. ORNSTEIN, W. CROWTHER & D. WALDEN. 1970 Spring Joint Computer Conf. AFIPS Proc. Vol.36, pp.551-567, 1970.

HOWE 71 Control Concepts of a Logical Network Machine. W.G. HOWE & T.R. KIBLER. IBM Research Rept. RC3331. April, 1971.

HUST 72 Current and Near Future Data Transmission via Satellites of the Intelsat Network. J.M. HUSTED. 1st Int. Conf on Comp. Communications, 1972.

ISSA 68 Chrysler Message Switching Today and Tomorrow. L.R. ISAACS & D.C. BUZZELLI. Proc. of the 23rd National ACM Conf. pp. 321-327, 1968.

KAHN 71A Flow Control in a Resource-Sharing Computer Network. R.E. KAHN & W.R. CROWTHER. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communication Systems. pp. 108-116. Oct., 1971.

KAHN 71B A Study of the ARPA Network Design and Performance. R.E. KAHN & W.R. CROWTHER. Bolt, Beranek & Newman, Inc., Cambridge, Mass. Rept. BBN-2161. August, 1971.

KIRS 72 On the Development of Computer and Data Networks in Europe. P.T. KIRSTEIN. 1st Int. Conf. on Computer Communications, 1972.

KLEI 64 Communication Nets. L. KLEINROCK. McGraw-Hill, Inc. 1964.

KLEI 69A Comparison of Solution Methods for Computer Network Models. KLEINROCK. Proc. of the Computers and Communications Conf., September, 1969.

KLEI 69B Models for Computer Networks. KLEINROCK. Proc. of the Int. Communications Conf., 1969.

KLEI 70A Analytic and Simulation Methods in Computer Network Design. L. KLEINROCK. 1970 Spring Joint Computer Conf. AFIPS Proc. Vol.36, pp. 569-579. 1970.

LAWR 71 A Proposed Computer Network for the Australia National University. D.E. LAWRENCE. Computer Centre, Australian National Univ., Canberra. Rept. No.38. August, 1971.

LEIN 59 PILOT, A New Multiple Computer System. A.L. LEINER, W.A. NOTZ, J.L. SMITH & A. WEINBERGER. Journal of the ACM. Vol.6, pp. 315-335. July, 1959.

LICH 66 Tentative Specifications for a Network of Time-Shared Computers. ARPA Document, M-7. Sept. 9, 1966.

MARI 66 Toward a Cooperative Network of Time-Shared Computers. T. MARILL & L.G. ROBERTS. 1966 Fall Joint Computer Conference. AFIPS Proc. Vol.29, pp. 425-432, 1966.

MART 69 Telecommunications and the Computer. J. MARTIN. Prentice-Hall, Englewood Cliffs, N.J., 1969.

MCKA 71A Network/440 - IBM Research Computer Sciences Dept. Computer Network. D.B. MCKAY & D.P. KARP. IBM Research Rept. RC3431. July, 1971.

MCKA 71B A Network/440 Protocol Concept. D.B. MCKAY & D.P. KARP. IBM Research Rept. RC3432, July 1971.

MCKA 71C Exploratory Research on Netting at IBM. D.B. MCKAY, D.P. KARP, J.W. MEYER & R.S. NACHBAR. IBM Research Rept. RC3486. June, 1971.

MCKE 72 The Network Control Centre for the ARPA Network. A.A. MCKENZIE et al. 1st Int. Conf. on Computer Communications, 1972.

MEIS 71 Optimization of a New Model for Message Switching Networks. MEISTER, MULLER & RUDIN. Proc. Int. Conf. on Communications, 1971.

MEIS 72 On the Optimization of Message-Switching Networks. B. MEISTER, H.R. MUELLER & H.R. RUDIN, JR. IEEE Transactions on Communications, COM-20(1):8-14, Feb., 1972.

MEND 71 The Lawrence Radiation Laboratory Octopus. S.F. MENDICINO. Lawrence Radiation Lab. Rept. UCRL-73149. April, 1972.

MILL 68 DCS Autodin Trunking Transmission Between Switching Centers. J.Z. MILLAR. Invitational Workshop on Networks of Computers, Proc. National Security Agency, Fort Meade, Maryland, NOC-68:221-224. October 14-18, 1968.

MIT 69 An Experimental Computer Network. M.I.T. Lexington Rept. ESD-TR-69-74. March, 1969.

NAC 70A Analysis and Optimization of Store-and-Forward Computer Networks. N.A.C. 1st Semiannual Tech. Rept. for the Project. Defense Documentation Center. Alexandria, Va. June, 1970.

NAC 70B N.A.C. 2nd Semiannual Tech. Rept. for the Project: Analysis and Optimization of Store-and-Forward Computer Networks. Defense Documentation Center, Alexandria, Va. December 1970.

NAC 71A N.A.C. 3rd Semiannual Tech. Rept. for the Project: Analysis and Optimization of Store-and-Forward Computer Networks. Defense Documentation Center. Alexandria, Va., June, 1971.

NAC 71B N.A.C. Fourth Semiannual Tech. Rept. for the Project: Analysis and Optimization of Store-and-Forward Computer Networks. Defense Documentation Center. Alexandria, Va. December, 1971.

ORNS 72 The Terminal IMP for the ARPA Network. S.M. ORNSTEIN, F.E. HEART, W.R. CROWTHER, H.K. RISING, S.B. RUSSELL, A. MICHEL. AFIPS Papers. SJCC, 1972.

PATI 70 Coordination of Asynchronous Events. S.S. PATIL. M.I.T. Project MAC Rept. MAC-TR-72. June, 1970.

PECK 69 The Implications of ADP Networking Standards for Operations Research. P.L. PECK, MITRE Corp., McLean, Va., Rept. MTP333, AD696675, June 1969.

PLUG 61 American Airlines SABRE Electronic Reservations System. W.R. PLUGGE & M.N.PERRY. Proceedings WJCC, pp. 593-602. May, 1961.

PORT Comparison of Switched Data Networks on the Basis of Waiting Times. E.PORT & F. CLOS. IBM Rept. RZ 405, IBM Research Labs., Zurich.

PROS 62A Routing Procedures in Communication Networks - Part I: Random Procedures. R.T. PROSSER. IRE Transactions on Communication Systems. CS-10: 332-329. 1962.

PROS 62B Routing Procedures in Communication Networks - Part II: Directory Procedures. IRE Transactions on Communication Systems. R.T. PROSSER. CS-10:329-335, 1962.

RAYM 71 A Queueing Theory Approach to Communications Satellite Network Design. H.G. RAYMOND. Proc. of Int. Conf. on Communications, 1971.

REDD 71 Computer Network Simulator. J.L. REDDING. Naval Ship Research and Development Center Rept. NSRDC 3650. September, 1971.

ROBE 70 Computer Network Development to Achieve Resource Sharing. L.G. ROBERTS & B.D. WESSLER. 1970 Spring Joint Computer Conf. AFIPS Proc. Vol.36, pp. 543-549. 1970.

ROBE 72 Extensions of Packet Communication Technology to a Hand-Held Personal Terminal. AFIPS 1972 Spring Joint Computer Conference. L. ROBERTS. 1972.

RUSS 66 Communication and Systems Development intthe C.S.I.R.O. (Commonwealth Scientific & Industrial Research Organization) Network. J.J. RUSSELL & D.C. KNIGHT. Proc. 3rd Australian Computer Conf. pp. 384-386. 1966.

RUTL 69 An Interactive Network of Time-Sharing Computers. R.M. RUTLEDGE, A.L. VAREHA, L.C. VARIAN, A.H. WEIS, S.F. SEROUSSI, J.W. MEYER, J.F. JAFFES & M.A.K. ANGELL. Proc. 24th National Conf. ACM Publication. p.69, pp. 431-42, 1969.

SCAN 68 The Design of a Message Switching Centre for a Digital Communications Network. SCANTLEBURY, WILKINSON & BARTLETT. IFIP 68Papers, 1968.

SCAN 69 A Model for the Local Area of a Data Communication Network: Objectives and Hardware Organization. R.A. SCANTLEBURY. ACM Symposium on Problems in the Optimization of Data Communication Systems, Pine Mountain, Georgia. pp. 179-193. October, 1969.

SCAN 71 The Design of a Switching System to Allow Remote Access to Computer Services by other Computers and Terminal Devices. R.A. SCANTLEBURY & P. T. WILKINSON. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communications Systems. October, 1971.

SHAP 66 Random Store and Forward Communication Networks. S.D. SHAPIRO. Proceedings of the Polytechnic.Inst. of Brooklyn Symposium on Generalized Networks. Polytechnic Press, Brooklyn, New York. pp. 721-733. 1966.

SHER 70 The Simulation of a Multi-Computer System. J.F. SHERLOCK. IEEE Trans. Computers, Vol. C-19, pp. 1114-1117. November, 1970.

SHOS 70 Sequencing Tasks in Multi-Process, Multiple Resource Systems to Avoid Deadlocks. A. SHOSHANI & E.G. COFFMAN. Proc. 11th Annual Symposium on Switching abd Automata Theory. pp. 225-233. October, 1970.

SMIT 64 Determination of Path Lengths in a Distributed Network. J.W. SMITH. Rand Corp., Memorandum, RM-3578-PR. August, 1964.

SPRA 71 Analysis of Loop Transmission Systems. J.D. SPRAGINS. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communications Systems. Oct. 1971.

TEIT 69 A Network Simulation and Display Program. W. TEITELMAN & R.E. KAHN. Proc. of the 3rd Annual Princeton Conf. on Information Sciences and Systems. March, 1969.

THOM 72 McRoss - A Multi-Computer Programming System. R.H. THOMAS & D.A. HENDERSON. Spring Joint Computer Conference, 1972.

TRAF 71 Data Transmission Network Computer-to-Computer Study. P.J. TRAFTON, H.A. BLANK, & N.F. MCALLISTER. Proc. ACM/IEEE 2nd Symposium on Problems in the Optimization of Data Communications Systems. October 1971.

WALD 72 A System for Interprocess Communication in a Resource Sharing Computer Network. WALDEN. Communications of the ACM. Vol.15, No.4, 1972.

WEIS 71 Distributed Network Activity at IBM. A.H. WEIS. IBM Research Report RC3392. June, 1971.

WHIT 72 Comparison of Network Topology Optimization Algorithms. V. KEVIN MOORE WHITNEY. 1st Int. Conf. on Computer Communications, 1972.

WILK 68 The Control Functions in a Local Data Network. WILKINSON & SCANTLEBURY. IFIP 68 Papers, 1968

WILK 69 A Model for the Local Area of a Data Communication Network, Software Organization. P.T. WILKINSON. ACM Symposium on Problems in the Optimization of Data Communication Systems, Pine Mountain, Georgia. pp. 152-172. October, 1969.

ZEIG 71 Modal Blocking in Large Networks. J.F. ZEIGLER. Ph.D. thesis. U.C.L.A. October, 1971.

APPENDIX II

SIMULA 67

SIMULA 67 is a general purpose programming language which may be regarded as an extension of ALGOL 60. The language was defined by O. J. Dahl and K. Nygaard of the Norwegian Computing Centre, Oslo. Its syntax is particularly suited to the definition and manipulation of classes, which can be data structures, execution rules, or a combination of both. The language provides for very easy definition of list-processing and simulation procedures. The treatment of simulation is based on the languages SIMULA I and SOL. Below we give a brief summary of the features of SIMULA 67, which we will refer to as SIMULA.

In the course of this research some corrections and additions were made to the SIMULA compiler. The main addition was the provision of interactive execution of a SIMULA program. As well as data input and program at a terminal, file linkage prior to program execution was possible by entering DATASET cards from the keyboard.

Corrections to the compiler were made by normal software maintenance methods, namely fault isolation, fix writing and insertion. Where possible fixes provided by Control Data Corporation were used, including those for faults reported by users at other installations. The majority of faults required small amounts of corrective code, rather than major changes or extensions. The compiler was maintained by the author for the duration of the research described in this thesis.

There are two main additions to the concepts presented in ALGOL 60. The first is that of program entities called objects; and the second is a new type of variable called a reference variable, which may point to objects.

A class definition is quite similar to a procedure definition. It consists of a class name, a number of formal parameters, and a class body or execution rule. A simple example is:

```
class rectangle (a,b);
real


begin
real  area;


area : = a * b;
end;
```

A reference variable may point to objects of the class given when the variable is declared. For example:

ref (rectangle) p;

Here p is declared to be a pointer which may only point to objects of the class rectangle. An object is an instance of its class declaration, and we can generate one using the SIMULA symbol new.

new rectangle (5,6);

This statement will create an object of the class rectangle with parameters equal to 5 and 6. If we write:

p :- new rectangle (5,6);

then p will point to this object (:- is the symbol which means 'points to'). The null object is a member of all classes so that we may always write:

p :- none;

irrespective of the class for which p has been declared a reference variable. We may pass the value of one reference variable to another by writing:

```
p :- q;
```

after this statement p will point to the same object as q does. The main difference between a class and a procedure are that a class body may not alter the values of the actual parameters which correspond to its formal parameters, and that an object exists as long as some reference variable points to it. That is to say we may not write:

```
class a (x); name x; real ;
```

and further, an object will not disappear when its execution rule completes, unless there is no reference variable pointing to it. a, b and area are called the attributes of an object of the class rectangle and we may access them via the reference variable p by writing p . attribute, for example:

```
p :- rectangle (5,6);
.
.
.
.
x := p.a ;
y := p.b ;
z := p.area ;
```

A shorthand for this is the _inspect_ statement:

```
inspect p do begin
            x := a ;
            y := b ;
            z := area ;
            end;
```

Remote referencing can be concatenated indefinitely; i.e. if the class rectangle has a locally defined reference variable q which points to objects of class triangle, we may write:

```
x := p.q.side ;
```

where side is a local variable of an object of class triangle.

Classes may be concatenated to form subclasses. For example the statements:

```
class A (a,b); real a,b ;;
A class B (x,y); real x,y ;;
```

define a class B which is a subclass of A and having four attributes a, b, x, y and no execution rule. A reference variable declared for class A may also point to any of its subclasses. For example:

```
ref (A) p;
p :- new B(1,2,3,4);
```

If A and B have execution rules, then the body of class B may be inserted anywhere in that of class A using the symbol inner; for example:

```
class A (a,b);
real a,b;

begin
real c,d;

C := a * b;
inner;
c := c + d;
end;
```

```
A class B (x,y);
real x,y;

begin
d := x * y;
end;
```

The statemtn p :- new B(1,2,3,4); will produce an object with execution rule:

```
real a,b,x,y;
begin
real c,d;
c := a * b;
d := x * y;
c := c + d;
end;
```

A class name, with or without an actual parameter list, may prefix an ordinary block. This makes the attributes and capabilities of the class available to the block. For example:

```
class A;
begin
real procedure sqrt (z)
real z; begin sqrt := z   0.5; end;

A begin
  real x,y
  .
  .
  .
  .
  .
  .
  .
  x := sqrt (y);
```

.
.
.
.

<u>end</u>;

The above is a simplified explanation of the way in which classes and reference variables may be used. For more information see the SIMULA 67 Common Base Definition. The main incompatibilities of SIMULA with ALGOL 60 are:

1. The <u>own</u> symbol is not permitted.
2. The <u>string</u> symbol is replaced by a similar concept, text.
3. Procedure parameter transmission by name must be specified using the symbol <u>name</u>.
4. The input/output system is developed in terms of objects of <u>class</u> file and its subclasses. Users should refer to the Reference Manual for details.

SIMULA 67 provides two predefined classes, SIMSET and SIMULATION. These have no formal parameters, but define a number of procedures which allow a programmer to write a list-processing or simulation program more easily within a block prefixed by these class names. In fact SIMULATION is a subclass of SIMSET. This is so for two reasons. Firstly it allows the class SIMULATION to use list-processing procedures from SIMSET when predefining simulation facilities; and secondly it allows the programmer to have access to those procedures inside a block prefixed by SIMULATION.

The prefix SIMSET provides for simple manipulation of two-way of two-way lists. The following actions are possible using predefined procedures.

1. Accessing the successor of a list member
2. Accessing the predecessor of a list member

3. Inserting an object in a list before or after another one, or at the end of the list.
4. Removing an object from a list.
5. Creation of a list.
6. Accessing the first and last objects in a list.
7. Determining whether a list is empty.
8. Determining the number of objects in a list.
9. Removing all objects from a list, making it empty.

The prefix SIMULATION allows the programmer to define processes (objects of class process) which are scheduled and executed within a predefined quasi-parallel system with its own time axis. This is done by maintaining a time-ordered event list (SQS) whose events are executed in sequence. Events are themselves objects which have two attributed (in addition to those required for list membership), namely:

```
real eventtime; comment the time at which this event is scheduled
                         to occur;
ref (process) proc;  comment a pointer to the process whose acti-
                             vations or reactivation this event
                             represents;
```

Events may be created implicitly or explicitly by processes, which themselves can be generated or destroyed by object generation or completion and detachment. Processes may interact via global variables, automatic statistic gathering, or by altering the attributes of other processes through inspect statements. The following actions are possible using predefined procedures.

1. Accessing the current system time.
2. Referencing the currently active process, i.e. the process

executing the current event.

3. Referencing the successor and predecessor of an event in the SQS.
4. Waiting for a specified time before continuing execution of the process execution rule.
5. Entering a queue.
6. Halting execution of a process until activated by some other process.
7. Directly scheduling an event, before or after another, or at a given time.
8. Cancelling or rescheduling a scheduled event.
9. Activating or reactivating a halted process.
10. Accumulating the system time integral of a variable.
11. Drawing of random numbers from various distributions.

To illustrate the flexibility of SIMULA 67, we shall suppose that we have a recursively defined problem of the form:

```
real procedure solution (data);
real data;
begin
real nextdata;
nextdata := function (data);
if nextdata = simpledata then solution := simplesolution
                          else solution := solution (nextdata);
end;
```

If the solution of the problem involves performing a simulation we can define:

```
real procedure function (data);
real data ;
```

```
simulation begin
              .
              .
              .
              .
            end;
procedure function (nextdata, partsolution);
name partsolution;
real nextdata, partsolution;
begin
ref (G) x;
if nextdata = simpledata then partsolution := simplesolution
                        else begin
                                 x :- new G(nextdata, 0);
                                 partsolution := G.solution;
                             end;
end;
```

A SIMULA 67 program may be executed on the CDC 6600 at ULCC suing the following control cards:

```
JØB(.............)
ATTACH(SIMULA, SIMULA)
SIMULA (L,X)
LIBRARY(L = SIMULALIB)
LGØ.
RETURN(SIMULA)
7/8/9
  :
  :
  SIMULA SOURCE PROGRAM
  :
  :
```

7/8/9

:

:

DATA

:

:

6/7/8/9

The last two cards of the SIMULA source deck should be 'EOP' and FINIS, both punched in columns 10-14.

Most programs can be compiled using 20,000 words (decimal) of store. However, for larger programs both compilation and execution speed can be increased by allowing more store.

Files names used in a SIMULA program are related to SCOPE files by DATASET cards, which are the first cards of the data record.

APPENDIX III

SHAPE LIMITATIONS

For various reasons the SHAPE program does not provide all the features described in Chapter III. In this appendix we state the limitations and discuss briefly the reasons for them. Implementation of the modelling system was originally divided into two stages. In the first of these a simplified system was created, with dataset activity represented by giving negative signs to the appropriate REP matrix elements, rather than the provision of the ACT matrix. This course required that datasets were treated as read-write only (the negative sign indication for read-only having been preempted).

Further consequences of the absence of the ACT matrix were a difference in the implementation of DOloops and the inability to perform simultaneous activation. The latter was caused by the absence of the top and bottom ACT column elements which we used to record a processor allocation (reservation in the case of simultaneous activation). This lack also precludes the implementation of the error handling methods described in Chapter III. Since read-only datasets were not available in stage one there was no necessity for deactivation SIarcs, which must also be provided when read-only datasets are implemented.

The addition of these features was defined as the second stage of implementing the modelling system described in Chapter III, but this was not carried out because of various problems of implementation. The most serious of these was a deficiency of the loader in the SCOPE operating system at that time. This deficiency was a limit to the number of certain loader tables which could be processed by the loader for a single program. Unfortunately the SIMULA 67 compiler produces code in which these tables occur very frequently.

Consequently a large SIMULA 67 program will not load after compilation and the loader either aborts with a machine stop or enters an infinite loop.

Since no diagnostic is issued identification of the fault took some time. It was found that the stage one implementation was slightly over the limit and steps were taken to reduce the number of the offending tables. This required the elimination of topological verification after graph input, and the use of only one set of statistics (hardware or software). In addition all calls to the run-time input-output system were changed to calls to an equivalent local procedure.

These measures reduced the program size sufficiently to allow successful loading. However any further insertion of SIMULA 67 statements had to be balanced by deletions elsewhere. Implementation of stage two could not have been accomodated within the program size required for successful loading.

A way out of this problem is the use of code procedures within a SIMULA 67 program. This is a call to a procedure which is separately compiled, and linked to the main program at load time. Various portions of the SHAPE program, notably the graph input procedures, could then be compiled separately, and linked to a much reduced main program by the loader, thus overcoming the limits imposed on any single program. Code procedures are part of the SIMULA 67 language and described in detail in the manual for the CDC 6600 version. However they have not been implemented in the compiler at the time of writing.

Other problems in the compiler, while not insurmountable, considerably slowed down SHAPE implementation.

One of these was a residue of program stop instructions which had been used by the compiler writers for trapping purposes. When these occurred during compilation or execution of SHAPE, elimination of the statement responsible strangely resembled a process of trial and error. The lack of interactive facilities in the compiler and run-time system increased the time spent in debugging SHAPE when this was already in short supply. Circumvention of the compiler bugs revealed in this process required extra statements in some cases, eating into the allowable program size. Examples are the use of a bad approximation for the generation of random Poisson numbers, incorrect comparison of positive and negative zero, and failure to recognize compressed card images.

The decision to forego stage two was also influenced by considerations of resource availability. Any SIMULA program requires a run-time acratch area for the storage of dynamically created and destroyed class objects. This area is provided by the storage between the end of the program and the field length limit. When all the free space has been used the run-time system calls a procedure named the garbage collector which eliminates all defunct objects and compacts the remainder, so providing a new free space area. The smaller the overall scratch area, the more frequently the garbage collector must be called to clean it up. Since the garbage collector processing is not negligible a trade-off developes between core storage available and CPU time required for any given program run. Furthermore as a program grows in size, in order to maintain the scratch area the field length must increase by the same amount. To provide an adêquate scratch space for the stage one implementation between 50 K and 60 K of core storage is required.

As the field length is dropped from 60 K, increasing quantities of CPU time are devoted to garbage collection. Jobs run at University of London Computing Centre are categorized by resource usage. A J9 category job may use up to 50 K of memory and up to 120 secs of CPU time, and a J12 job is allowed 60 k and 1200 secs. respectively.

Consequently a SHAPE run with medium sized graphs will almost certainly be a J12 job. This is the largest job which receives a regular turnround at the Centre. If more store or CPU time is required the job is categorized as J15 and run as and when there is spare capacity available. Production runs should therefore be kept within the J12 limits if at all possible, to ensure regular turnround. Within these limits it is doubtful if adequate scratch storage would remain after expansion of the SHAPE program to include stage two facilities. The full implementation would therefore have to be run as a J15 job except with the simplest models.

In some models the value of the results is related to the length of run. For such cases the runs must have adequate CPU time available, so that if requested memory is reduced from J12 to J9 limits, the garbage collection trade-off increases the CPU time required and returns the job to the J12 category. This resource availability situation provided a further reason to use the stage one implementation. Even in this case because J12 jobs are the largest to receive regular service, they also have the slowest turnround time.

Most SIMULA 67 programs will use the run-time system a good deal, especially if any list-processing or simulation is performed, which is the case with SHAPE.

Therefore the CPU time used is greatly affected by the efficiency of the run-time system in executing its various functions. Its areas of weakness are the input-output procedures, block entry and closure, and the processing of goto statements.

The code produced by the compiler is split into 512 word segments for no apparent reason. The segments do not correspond to the user program, and jumps across segment boundaries are very slow (requiring a call to a segment control routine in the run-time system). If a program loop crosses a segment boundary degredation can be severe. The loop control itself is slow and does not take advantage of the simple case where the step is one. For these reasons a good deal of CPU time is required by SIMULA 67 programs.

In the paragraphs above we have tried to summarize the reason for which the full graphical modelling system was not implemented, and also some of the factors which slowed down the development of the restricted system.

The validation model was a simple one and consisted of a small number of modelling elements. The observation of interest was message delay and the validation runs were in the J12 category of resource usage, which allowed the generation of approximately 2,000 messages.

The model of the ARPA network link was also run as a J12 category job, and typically this run-time would generate between 250 and 300 round trips (message and response cycles). When considering alternative models of the link we were at pains to keep the memory requirement to a minimum. The implementation has only two variables which travel with the cut, namely LAMBDA and BETA.

If a further variable of this type were available it could be used to provide a message routing indicator. If the variable held the node number of the destination node for the message, then an IFloop whose outcome was a function of the variable value would be equivalent to a routing algorithm. The existence of these two facilities would be of great benefit, since it would be possible for messages with differenct destinations to use the same REP matrix elements on common sections of their routes. While leaving the CPU time almost unchanged, this would greatly reduce the memory required by the model.

Without these features it was necessary to provide separate matrix elements for messages with different destinations. Within this constraint we reduced the memory requirement by providing separate nodes for each route, so keeping the number of REP matrix elements much smaller than would be the case if the separate route elements were placed in a single REP matrix.

The CPU time required by a model is approximately proportional to the number of SIarc executions, and so to the number of SIarcs. These were kept to the minimum compatible with retaining the structure of the link activity.

In general penalties in resource usage were incurred because of the absence of the second stage implementation, or because of inefficiencies in the SIMULA 67 run-time system. Where possible these were alleviated by judicious manipulation of model structure.

APPENDIX IV

SHAPE USER INFORMATION

CARD FORMATS FOR GRAPH INPUT

CARD COL	1	11	21	31	41	51	61	71
RUN	GNO	MODE	DEBUG	PICODE	SICODE	SIMLIM	MAXREAL	
GRAPH HEADER	GRAPHNAME	TYPE	FIRST NODE	LAST NODE	ARC WIDTH	GRAPH SIZE	G FACTOR	NUMBER OF PROCESSORS
SINODE	NODENUMBER		INARCS	OUTARCS	{DATANODE}			
			PINODE	OUTARC1	OUTARC2	OUTARC3	OUTARC4	OUTARC5
			INARC1	REP[1,1]	REP[1,2]	REP[1,3]	. .	. .
			INARC2	REP[2,1]	.	.	. .	. .
			. .	. .	. .	. .	. . . .	. . . .
PINODE	NODENUMBER		INARCS	OUTARCS	{DATANODE}			
			CAPACITY	BLOCK SIZE	LATENCY	SEED	BLOCKS/TRACK	COST

CARD FORMATS FOR GRAPH INPUT

CARD COL	1	11	21	31	41	51	61	71
SIARC	NODENUMBER	NODENUMBER	{FIRST NODE	LAST NODE	ARC WIDTH	GRAPH SIZE	GFACTOR	NUMBER OF PROCESSORS}
			SEQ FRAC.	{DATANODE1	DATANODE2	DATASEQF}		
			IFCODE1	IFCODE2	A1	A2	B1	B2
			PHI[1]					
			PHI[2]					
			⋮					
			⋮					

CARD FORMATS FOR GRAPH INPUT

CARD COL 1	11	21	31	41	51	61	71
NODENUMBER	NODENUMBER	{FIRSTNODE	LASTNODE	ARC WIDTH	GRAPH SIZE	GFACTOR	NUMBER OF PROCESSORS}
		SEQ FRAC.	{DATANODE1	DATANODE2	DATASEQF}		
		ID	COST				
		PSI[1,1]	PSI[1,2]	PSI[1,3]			
		PSI[2,1]	PSI[2,2]	.			
		PSI[3,1]	.	.			
		:	:	:			

Items in brackets { } are optional.

A node must have its outarcs immediately following its own data; nodes can appear in any order. If an arc has a subgraph its parameters follow the node numbers on the arc card. The subgraph then follows the data of the arc.

DATA REPLICATION

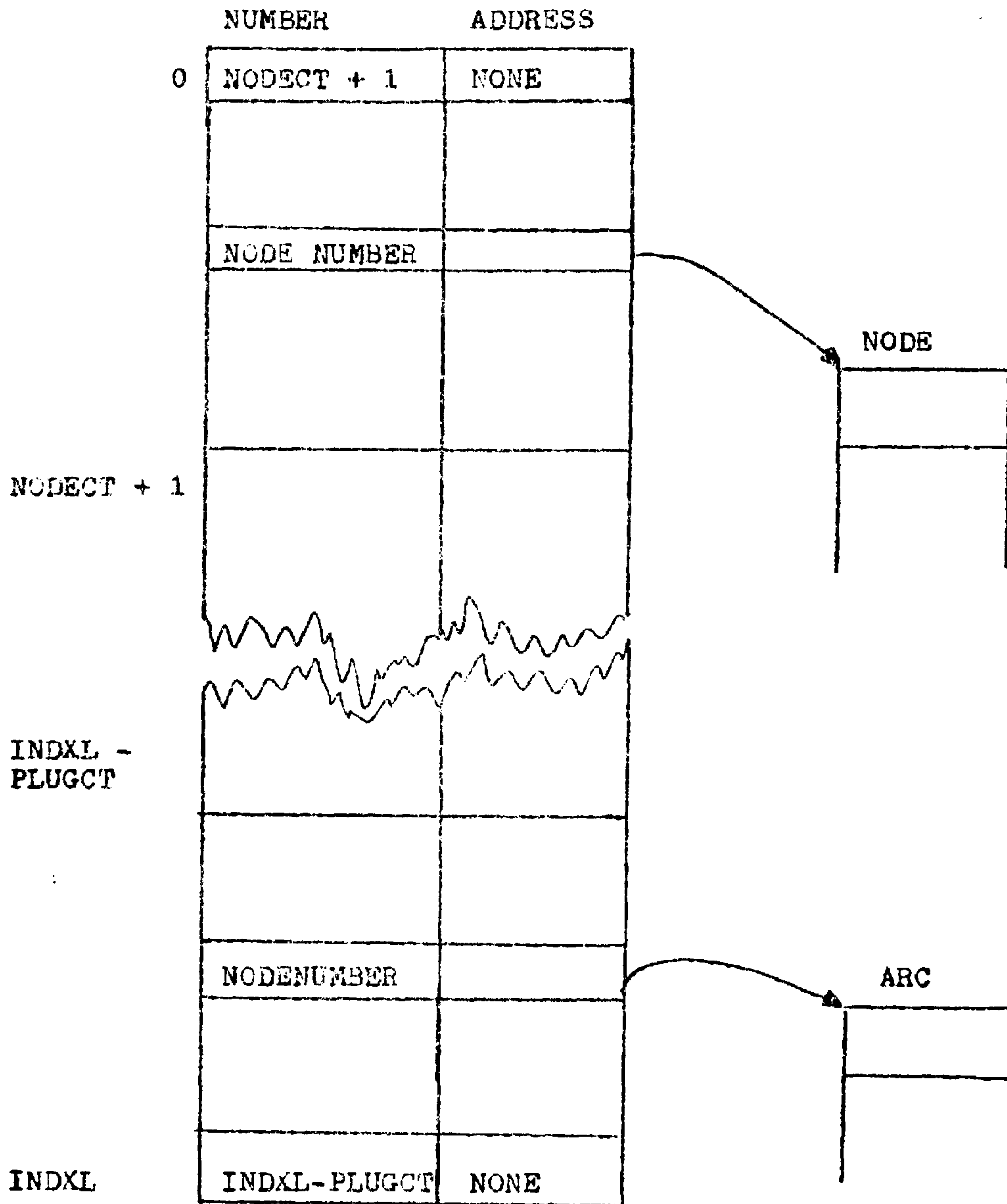
If a node (or arc) has identical data to some other then DATANODE (or DATANODE1, DATANODE2, DATASEQF) describe the other node (or arc) and the data is replicated automatically by graph input, so that there is no need to append the data again. The same occurs with a subgraph in which case replication is indicated by setting size = 0 and then FIRSTNODE, LASTNODE, GFAC give NODENUMBER1 and SEQF of the arc which heads the subgraph to be replicated. The following rules apply.

1) The DATANODE must precede a replica on the source file.
2) Data arc must precede a replica on the source file.
3) When data and replica subgraph arcs are OUTarcs of the same node, then the data subgraph must precede the replica.

MULTIPLE INARCS

For extra INarcs to a particular row of REP matrix set SEQF := REP[i,0] + final NODENUMBER - initial NODENUMBER. Arc specifier at initial node equals SEQF.

INDEX USAGE



If nodect + 1 = indxl - plugct <u>then</u> indexfull := <u>true</u> ;

<u>comment</u> nodes are held in ascending order by node number,

indxl is the length of the index,

on creation number [0] := 1 and number [indxl] := indxl,

on completion of [illegible]graph input number [indxl] := pno;

RUN CONTROL

The SHAPE program expects to find sequential input on two files named SIGRAPH and PIGRAPH. SHAPE prints logging information on file LOGGER and statistics on file STATS. Any SCOPE files may be used as long as their file names are equated to the expected ones using DATASET cards. These must be the first cards read on the standard input file INPUT. For example,

DATASET, SIGRAPH = HENRY

DATASET, PIGRAPH = XYAB

DATASET, LOGGER = P

DATASET, STATS = OUTPUT

DATASET, END

These cards are followed by a run control card whose format is shown above. The parameters can be set as follows.

GNO - Number of graphs for this run j may be 1 or 2

MODE - binding mode may be

1 = non-reentrant

2 = semi-reentrant

3 = completely reentrant

DEBUG - Debug parameter, if non-zero then extra logging information is output.

PICODE - code showing hardware statistics required, is an octal digit, i.e. 0 - 7, treated as three bits.

BIT 1 - low order bit set for memory statistics

BIT 2 - middle bit set for processor statistics

BIT 3 - high order bit set for processor state statistics

SICODE -code showing software statistics required, is an octal digit, i.e. 0 - 7, treated as three bits

BIT 1 - low order bit set for node statistics

BIT 2 - middle bit set for arc statistics

BIT 3 - high order bit set for cut statistics

SIMLIM - Default binding time limit

MAXREAL -value to which all program maxima are to be preset

GRAPH INPUT PROCEDURES

gin(g) g - pointer to object of class graph, null if graph input failure.

subgin (y, type, fstn, lstn,adl, size gfac)

y - pointer to object of class graph, provided by gin.

type - type of graph, usually 1 or 3.

fstn - node number of first node ingraph.

lstn - node number of last node in graph.

adl - length of arc data vectors.

size - number of nodes in graph.

gfac - factor to derive index capacity.

innode (x, type, adl, data, nodes)

x - pointer to current node.

type - as above.

adl - as above.

data - scratch array for arc data created by subgin.

nodes - pointer to index created by subgin.

inarc (x, type, adl, data, nodes, inn)

x - pointer to previous arc.

type - as above.

adl - as above.

data - as above.

nodes - as above.

inn - node number of node at head of chain.

find (n, e, ind) - address of node with number n.

ind - pointer to index to be searched

e - number of entry in index

plug (n, a, ind) - true if successfully completed.

n - node number of plug.

a - address of arc requiring node address.

ind - as above

putelem (n, a, ind) - true if element successfully entered in index.

n - node number

a - address of element

ind - as above

ERROR CODES

Code	Error
11	REP matrix of initial SInode contains no OUTarc specifier for SIarc of completed tie.
12	REP matrix of terminal SInode contains no INarc specifier for SIarc of completed tie.
14	Terminal node dataset found to be negative while activating terminal row in mode 1.
21	Memory inuse less than zero after memory change.
22	Memory inuse greater than capacity after memory change.
31	No SIarc found in OUTarc chain to match ready column in current REP matrix.
32	REP matrix of final node of ready SIarc contains no row for this SIarc - INarc specifier not found.
40	No PIarc found to execute ready SIarc.
41	Terminal store too small to hold max. requirement of terminal node.
42	Terminal store nodenumber not that required by terminal SInode of ready arc.
43	Terminal store not that to which terminal SInode of SIarc already tied.

44 Terminal store not the same as initial when SIarc is a loop.

45 Time for arc to execute less than zero.

46 Not enough storage free in terminal PInode.

47 Available process fraction is zero.

50 Arc time less than zero in PERT mode.

60 One or more terminal datasets of ready IFloop exceed current initial one.

70 No active initial dataset found for IFloop readied by completing tie.

80 No next event, system resources deadlocked.

81 Arc data rector lengths not matching in SI and PIgraph.

82 No next event but final node still active.

83 Binding time limit exceeded.

91 PSID[0] less than zero after release of allocated processor fraction.

92 PSID[0] greater than one after allocation of processor fraction.

ARRAY USAGE

PSI [0 : adl, 1: type]

	adl	id	arc cost
1	t_1	u_1	e_1
2	t_2	u_2	e_2
	.	.	.
	.	.	.
	.	.	.
adl	t_{adl}	u_{adl}	e_{adl}

PHI [0 : adl, 1: type]

	adl
1	ϕ_1
2	ϕ_2
	.
	.
	.
adl	ϕ_{adl}

MU [1: 6]

1	2	3	4	5	6
CAPACITY	BLOCK SIZE	LATENCY	SEED	BLOCKS/ TRACK	COST

ARRAY USAGE

PSID [0 : adl + 2]

0	pfrac
1	Φ_1
2	Φ_2
adl	Φ_{adl}
adl+1	id
adl+2	pcost

IFF [0: adl + 2]

ifcode1
ifcode2
vx
wx
vy
wy
:

PINODE RUN-TIME VARIABLES

inuse - current quantity of storage in use.

totuse - time integral of inuse.

mit - cumulative total of time inuse is non-zero.

mut - maximum observed value of inuse.

mef - sum over all periods in which inuse was non-zero of the product of period length and maxuse of that period.

The variables above are used to produce the following statistics for each PInode.

activity - mit/gt

utilization - mut/capacity

efficiency - totuse/(mut * gt)

To derive expressions for overall graph storage utilization and efficiency, we use the sums over all PInodes of cost, cost of PInodes with non-zero time used, products of cost and mef/capacity These are accumulated in totmem, gutmu, and gefmu respectively. Statistics output are then

gutmu = gutmu/totmem

gefmu = gefmu/(gutmu * gt)