Quaestiones Informaticae

Vol. 2 No. 3 September, 1983

Quaestiones Informaticae

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
Department of Computer Science and Information Systems
University of South Africa
P.O. Box 392, Pretoria 0001

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R6	$7	£3,0
Institutions	R12	$14	£6,0

Circulation manager
Mr E Anderssen
Department of Computer Science
Rand Afrikaanse Universiteit
P O Box 524
Johannesburg 2000
Tel.: (011) 726-5000

Quaestiones Informaticae is prepared for publication by Thomson Publications South Africa (Pty) Ltd for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Evaluating the Performance of Computer-Based Information Systems using a Restricted Linear Regression Model

P.J.S. Bruwer
Department of Computer Science, Potchefstroom University for CHE, Potchefstroom

Abstract
Performance evaluation techniques can contribute substantially to the successful management of the information services function in an organization. Unfortunately the design and implementation of computer-based information systems have for a long time been (and in some organizations are still being) accepted as being primarily technical activities. Some of the most serious constraints on the design, installation and use of computer-based systems are not technical in nature and from the literature it has emerged that many computer-based systems have failed, not necessarily as a result of poor technical quality, but because certain other important aspects which determine the success or the failure of a computer-based system have been ignored. The objectives of this study may be summed up in four main points, viz.:

(i) to investigate the use of computer-based information systems and their success as perceived by the user;
(ii) to determine which factors make the largest contribution to the success of a computerized system;
(iii) to determine, by means of optimizing methods, which factors are the most important to change in order to obtain optimal improvement in the performance of the information systems; and
(iv) to formulate a model which would enable the management of an organization to apply funds in such a way that the computerized systems may provide optimum performance.

1. Introduction

Various researchers most notably perhaps Lucas [10], [11], [13]) have investigated the use and application of computer-based information systems. Some others, who have also made contributions to this and related fields, include Enid Mumford [15], E. Burton Swanson [18], Russel R. Ackoff [1], R.D. Mason and I. Mitroff [14], M.S. Scott Morton [17], John T. Garrity [8] and J. Huysmans [9].

The main reason for this study was the fact that most computer-based information systems today fail not as a result of poor technical quality, but because certain other important factors are not considered [13]. If those factors are known, how can management use this knowledge to improve the information systems performance? How are they going to spend the available funds to improve the quality of the systems? Is there an optimal way to apply the funds to improve the success of the systems "optimally"? This study was undertaken to provide some answers to these questions.

In section 2 the field study is discussed. Included in this section are the various statistical methods used and the linear regression function obtained from the processed data. In section 3 the restricted linear regression model method used in the further analysis is described. Section 4 consists of a description of the cost models which were developed and in the fifth and last section a few conclusions are made.

2. The Field Study

From the literature it is clear that certain factors are of utmost importance in their contribution to the success of computer-based systems. These factors are used in a descriptive model (figure 1). In order to be able to determine the relationships as hypothesized in the model, data from users of computerized systems in a very large organization were collected. In this organization there are about 140 computerized systems used by 1 200 clerical staff and 114 managers.

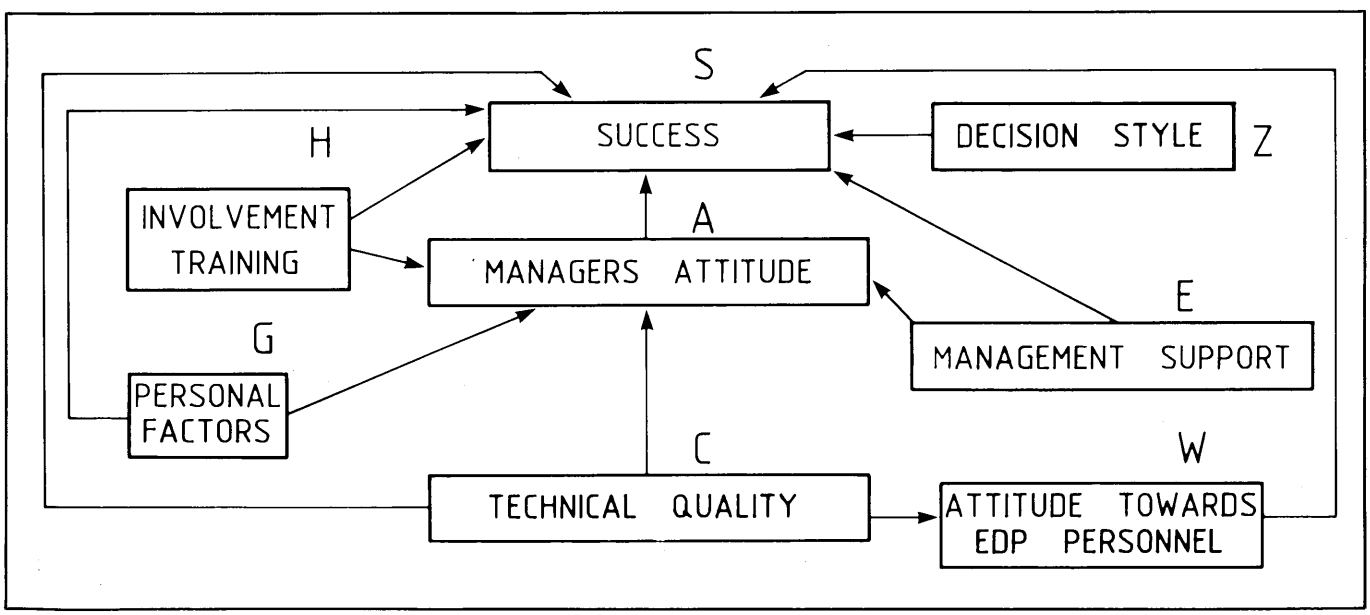


FIGURE 1: Hypothetical research model.

Questionnaires were designed for the managers and the full population of managers was involved in the investigation. Most of the questions in the questionnaire could be answered on a 7-point scale ([3], p. 282). With the aid of statistical computer programs [5], correlation and regression methods were used to investigate the existence of linear relationships between certain variables in the model and the success of computerized systems. How is success defined operationally? In past studies success has been measured by a number of indicators including actual use, intended use, attitudes, etc.

A cost/benefit study is one of the best ways to measure success. The benefits obtained from a computer-based system should exceed the costs of developing and running the system. Unfortunately, assessing the benefits of systems efforts has been difficult, particularly for systems supporting sophisticated decision making beyond the routine processing of transactions. What are the benefits of strategic planning to the firm? How can the results of strategic planning be evaluated against performance with no planning at all?

Because of the extreme difficulty of measuring success through cost/benefit studies, some other indicator of success is needed. The most appealing indicator from a measurement standpoint is system use. However there are instances where a high level of system use is not a sign of success of the system. When the use of a computer-based system is involuntary, this criterion can not be used. The third criterion which was used in this study, is user satisfaction with the system. Use and satisfaction are relative, that is, they are measured in general on a continuous scale as opposed to a binary scale. Less and more satisfaction are then defined operationally by where individuals fall on the continuous scale.

The data collected from the managers were used to obtain a linear regression model in terms of which success could be explained. To be able to do this, the following method was used: By using factor analysis [6], the variables used in the questionnaire to determine the success of the computerized systems were combined to form a single factor. Success could now be predicted by means of a linear relationship of a number of variables. With the aid of a stepwise linear regression computer program ([6], p.455), the number of independent variables in terms of which success could be explained was reduced to 18 variables (the number at the outset had been 39). These variables explained 72% of the variation of the success variable. However, because there was a need for as simple a model as possible for the rest of the study, a multiple linear regression program ([6], p.459) was used to select the "best" subset of independent variables in terms of which success could be explained by the following 12 variables:

- A — the attitude of management towards computerization
- E — management support
- C_1 — degree of detail in output reports (too much)
- C_2 — degree of detail in output reports (too little)
- C_3 — accuracy of information
- H — involvement of management
- G — term of service of managers
- T — availability of resources
- W_1 — technical quality of computer personnel
- W_2 — quality of the manager's managerial skills
- W_3 — training provided for users
- W_4 — extent to which users were involved in computerizing projects

The linear regression model which evolved is given below:

$$S = -3{,}74 - {,}08A, + {,}14C_1 - {,}14C_2 + {,}23C_3 + {,}05H + {,}08T - {,}07G + {,}34W_1 + {,}11W_2 + {,}10W_3 - {,}26W_4 \quad (1)$$

This model can be used for prediction purposes and the influence of a specific independent variable on the dependent variable is now of interest. Should there be no interdependence between the decision variables, one could merely note the regression coefficient of the specific variable. In such a situation it would be theoretically possible to determine the maximum of the dependent variable by making those values of decision variables which have positive regression coefficients as high as possible, and those which have negative coefficients, as low as possible.

The problem with this approach is the fact that the combination of levels of the variables could be of such a nature that it can not be physically implemented. For example in this project no data points were observed where the variable T (availability of resources) was at a low level while the variable W_1 (technical quality of computer personnel) was at a high level and vice versa. It should thus be accepted that one can only suggest levels of variables which would fall within the area of experience, that is, combinations of levels of variables that were observed.

3. The Restricted Linear Regression Model Method

The field of experience, that is, the space covered by the available data points, is of significance here. For this reason the convex hull of available data points was considered. In this project data were collected from 114 managers so that 114 data points were available. To determine the convex hull all convex combinations of data points observed must be considered. For simplicity we let X_j denote the appropriate variable in (1) above and $X_{i,j}$ the i-th observation on variable X_j. Should we consider the points

$$V_i\,(X_{i1}, \ldots, X_{ik}) \text{ for } i = 1, 2, \ldots, 114 \text{ and } k = 12 \quad (2)$$

the convex hull could be represented by the following set:

$$C = \{z = (X_1, X_2, \ldots, X_k)/z \epsilon E^k \text{ and } z = \sum_{i=1}^{N} \lambda_i V_i \text{ with } \lambda_i \geq 0 \text{ and } \sum_{i=1}^{N} \lambda_i = 1\} \quad (3)$$

where E^k represents the k-dimensional Euclidean space.

The behaviour of the regression function in the area C is now of importance because the area represents all combinations of data points observed.

Suppose the influence of a specific decision variable, for example X_p, on the dependent variable is studied. In the first place it is necessary to know the area of variation covered by X_p within the available data. In this project the data were derived from a 7-point scale so that the minimum values which may be attained for all the variables are 1 and the maximum 7.

If we let $X_p = q$ where $q \epsilon (1,7)$, it would be desirable to determine the values of the remaining decision variables so that S could be a maximum or a minimum. To be able to do this, the following linear programming problem was formulated:

$$\underset{(\min)}{\text{Max}}\ S = b_0 + b_1 X_1 + \ldots + b_k X_k \quad (4)$$

(where b_i = the regression coefficient for variable X_i) subject to the following constraints:

$$\sum_{i=1}^{N} \lambda_i X_{ij} = X_j \geq 0 \text{ for } j = 1, \ldots, k$$

$$\sum_{i=1}^{N} \lambda_i = 1, \quad \lambda_i \geq 0 \text{ for } i = 1, \ldots, N$$

$$X_p = q.$$

The solution of the linear programming problem then yields the maximum (minimum) of S as well as the levels of the decision variables $X_1, X_2, \ldots, X_k$, where this optimal level is reached. By solving this problem for various values of q in the interval (1,7), the maximum and the minimum values of S may be determined together with the optimal levels of the remaining decision variables.

The above solutions may be obtained by using parametric linear programming techniques (7). In this project, however, a linear program was developed in which q was incremented from 1 to 7 in steps of 1 for each of the "independent" variables.

Note that the difference between the maximum and minimum values of S at any level q of X_p is an indication of the influence of the remaining decision variables on S. This influence is relatively small (large) when the difference is small (large). In the above discussion only one decision variable X_p was restricted at a particular level and the linear program solved, but it is of course possible to restrict more than one decision variable at specific levels and then to solve the linear programming problem.

In this project this technique was applied and graphic

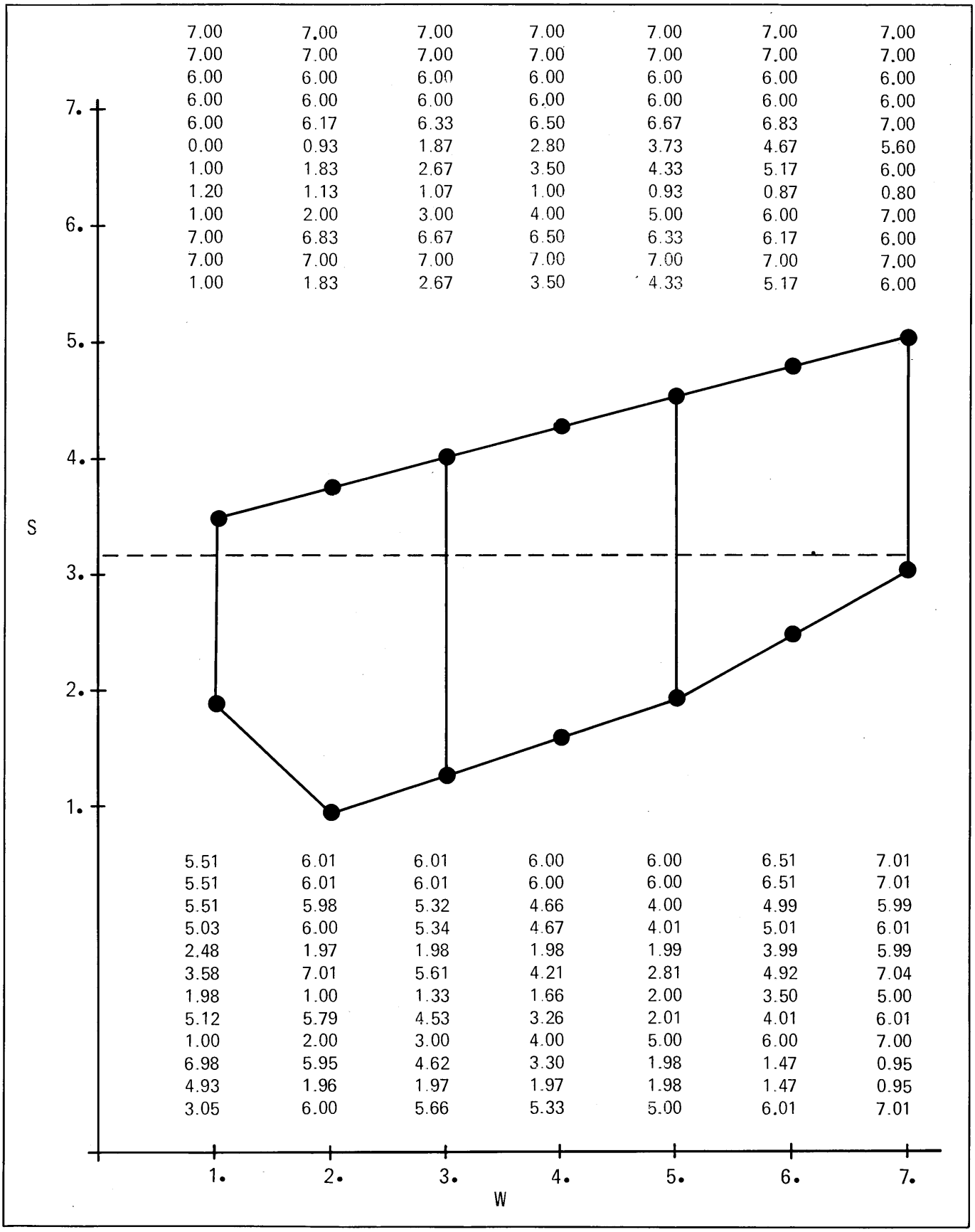


FIGURE 2: Maximum and minimum values of dependent variable S.

representations obtained for each of the 12 "independent" variables together with the optimal values of the decision variables when one variable at a time was restricted to the values 1,2,...,7. Figure 2 is an example of such a graphical representation. In this figure the numbers above represent the optimum values of the decision variables when the objective function is maximized while the numbers below represent the values of the decision variables when the same objective function is minimized.

With the aid of these solutions and graphical representations five factors which have significant influence on the success variable could be identified. These factors are the following:

— Accuracy of the information yielded by the systems.
— The availability of resources.
— The term of service of a manager within the organization.
— The technical quality of computer personnel.
— Training provided for users of the information systems.

One of the factors, viz. the term of service of a manager within the organization, can be regarded as a state variable (not totally under management control).

4. Cost Models

Although the cost involved in all eleven variables (excluding the state variable) is of significance when the success has to be improved, the four above are those which should preferably be given attention. For the management of an organization it is important to know how to apply available funds in order to maintain the success of the systems at a desired level of efficiency. The basic problem is of course the fact that the cost involved in increasing the levels of the various variables would differ in the different instances. The availability of capital is another limitation on the expansion of the variables. Suppose the total amount of capital which may be applied for computerizing purposes (that is, capital already spent plus capital available for further expansion) is R Rand. Suppose further the costs involved in raising variables $X_1, X_2,\ldots,X_k$ by one unit on the 7-point scale are $Q_1, Q_2,\ldots,Q_k$ Rand respectively. By using R and the Q's in a cost-restricting equation in a cost model in which one or more state variables are restricted to specific levels, the optimal values of the decision variables and the objective function values may be obtained by solving the linear programming problem for various values of R.

The cost model is the following:

Max $S = b_o + b_1X_1 + \ldots + b_kX_k$
subject to the following constraints: (5)

$$\sum_{i=1}^{N} \lambda_i X_{ij} = X_j = 1,2,\ldots,k$$

$$X_p = q$$

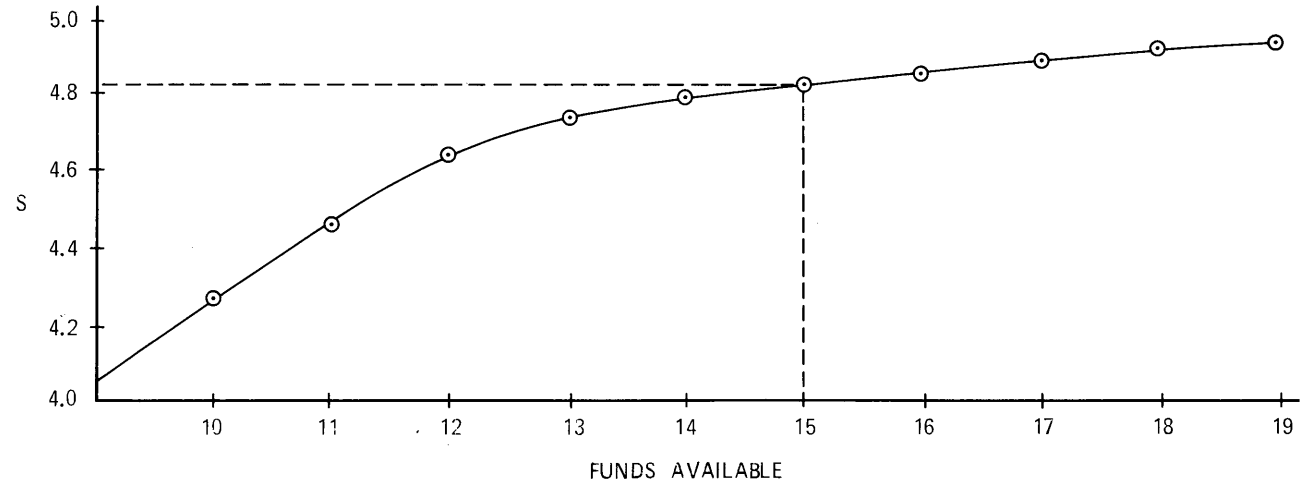
$$\sum_{i=1}^{N} \lambda_i = 1$$

$$\sum_{j=1}^{k} Q_j X_j \leq R,\ \lambda_i \geq 0 \text{ for } i = 1, 2,\ldots,N \qquad (6)$$

To demonstrate the model hypothetical costs were used and the linear programming problem solved varying R from 1 to 20 units in steps of 1. Figure 3 gives an indication how the success could be improved with more funds available. The table in this figure represents the values of the decision variables when the objective function (in this case the term of service of a manager) is restricted to a certain level. The difference between these values and the current values of the decision variables gives an indication how much the decision variables have to be raised to obtain the corresponding level of success of the system represented in the graph. When a single computerized system has to be improved, the simple cost model may be fruitfully applied. When more than one system is considered, this model may be applied in an iterative sense.

A GENERAL COST MODEL

Suppose the level of a variable X_j is a result of levels of its



OPTIMUM VALUES OF DECISION VARIABLES FOR MAXIMUM

A	=	2.7	3.3	4.0	4.7	5.2	5.5	5.8	6.2	6.5	6.8	7.0
E	=	5.1	5.1	5.1	5.0	5.2	5.5	5.8	6.2	6.5	6.8	7.0
C_1	=	3.3	4.0	4.8	5.7	6.0	6.0	6.0	6.0	6.0	6.0	6.3
C_2	=	3.3	3.2	3.1	3.0	3.3	3.8	4.3	4.7	5.2	5.7	6.3
C_3	=	5.9	6.0	6.0	6.0	6.1	6.3	6.4	6.6	6.7	6.9	7.0
H	=	.4	.3	.2	.1	.6	1.6	2.5	3.5	4.5	5.4	6.3
T	=	2.3	2.5	2.7	2.9	3.3	3.9	4.4	4.9	5.5	6.9	7.0
W_1	=	4.4	5.3	6.4	7.0	7.0	7.0	7.0	7.0	6.7	6.5	6.5
W_2	=	2.5	3.8	5.5	7.0	7.0	7.0	7.0	7.0	7.0	7.0	6.7
W_3	=	1.3	2.4	3.5	4.6	5.1	5.3	5.6	5.8	6.0	6.2	6.5

FIGURE 3: Improvement of success with more funds available.

components represented by the variables $Y_{j1}, Y_{j2},\ldots, Y_{j\ell}$. Suppose further the variable X_j represents the availability of resources. These resources consist of certain basic components such as power of the central processor, disc space, software, etc. The level of X_j may be raised by improving some of the components or perhaps all of them. The problem is complicated by the fact that the source components may be shared by different computerized systems.

A general cost model in which the contribution of the source components to the level of a specific variable is considered and in which provision is made for a number of systems to be improved simultaneously may be formulated as follows:

Let $X_i^j = \sum_{\ell=1}^{\ell_i} f_{i\ell j} Y_{i\ell}$ be the j-th variable influencing the success S_j of computerized system j, where $f_{i\ell j} \geq 0$, for all i,j and ℓ, with $\sum_{i=1}^{\ell_i} f_{i\ell j} = 1$ and ℓ_i the number of components Y_i of variable X_i^j. (7)

This linear relationship implies that X_i^j will be on its maximum level (level 7) when $Y_{i1},\ldots,Y_{i\ell_i}$ are all on the maximum level.

Suppose an organization wants to spend R Rand to improve the success of its computerized systems. The level of each variable X_i^j can be measured like it was done in [3]. The weights f_{inj} for each of the source component's contribution to the level of variable X_i^j could be estimated by the organization. The present level of the source components Y_{in} can be estimated by using the non-linear least squares method of Gauss-Newton ([5], p. 808) by minimizing the quadratic relationship.

$$\sum_{j=1}^{p} (X_i^j - f_{i1j}Y_{i1} - f_{i2j}Y_{i2} - \ldots - f_{i\ell_i j}Y_{i\ell_i})^2 \qquad (8)$$

for each variable X_i^j where i = 1, 2,...,k. This can be done by using a computer program ([5], p. 464). To be able to interpret the estimators it is desirable to restrict their values between 1 and 7. Suppose $V_1, V_2,\ldots, V_p$ are the levels of success we want to achieve for the p systems. Consider the following objective function to be minimized:

$$z = \sum_{i=1}^{p} a_i \left| V_i - b_0 - \sum_{t=1}^{k} b_t X_t^i \right| \qquad (9)$$

where k is the number of variables and a_i a weight associated with each system to be improved. This objective function is subject to the following constraints:

$$\sum_{p=1}^{N} \lambda_p^i X_{p\ell}, \; X_\ell^i, \; X_j^i = q_i,$$

$$\sum_{p=1}^{N} \lambda_p^i = 1, \text{ for } i = 1, 2,\ldots,p \text{ and } \ell = 1,\ldots,k. \qquad (10)$$

$$\sum_{s=1}^{\ell_i} f_{isj} Y_{is} = X_i^j \text{ for } i = 1,\ldots,k \text{ and } j = 1,\ldots,p.$$

We may assume the number of components of variable X_i^j is independent of j because a weight of zero is allowed.

Substitution of (10) in (9) yields

$$z = \sum_{i=1}^{p} a_i \left| V_i - b_0 - \sum_{t=1}^{k} b_t \sum_{n=1}^{\ell} f_{tni} Y_{tn} \right| \qquad (11)$$

In the same way the general cost model can be formalized to

$$\text{Min} \sum_{i=1}^{p} (C_{i1} + C_{i2})/a_i$$

subject to the following constraints: (12)

$$V_m - b_o - \sum_{t=1}^{k} b_t \sum_{n=1}^{\ell_i} ft_{nm} Y_{tn} + C_{m1} - C_{m2} = 0$$

$$\sum_{j=1}^{N} \lambda_j^m X_{li} - \sum_{n=1}^{\ell_i} f_{inm} Y_{in} = 0$$

$$\sum_{n=1}^{\ell_i} f_{inj} Y_{in} = q_i \text{ for all } j, \quad \sum_{r=1}^{N} \lambda_r^m = 1,$$

$$\sum_{i=1}^{k} \sum_{g=1}^{\ell_i} (k_{ig} Y_{ig}) \leq R \text{ where } i = 1,\ldots,k \text{ and } m = 1,\ldots,p.$$

REMARKS

1 In the objective function (12) C_{i1} and C_{i2} were used to allow positive or negative differences between the levels we want to achieve and the linear function determining the success of the system. C_{il} represents a negative difference and if it appears in the solution on a positive level it means that the success levels we want to achieve are exceeded. This means C_{il} may be deleted from the objective function because a solution better than those we want to achieve should not be penalised.

2 The level of variable X^j (for each system) is restricted since.

$$X_i^j = q_i \text{ or } \sum_{n=1}^{\ell} f_{inj} = q_i.$$

If more such variables should be restricted, additional constraints can be used.

3 In this model R can be varied. This will allow one to see exactly how R influences the improvement of success. A low value of R will typically imply that the success levels we want to achieve for the different systems cannot be reached. Higher values of R will allow the model to achieve higher levels of success.

5. Conclusions

Although a number of researchers have investigated success and failure patterns in computer-based information systems in the past, it is still not easy for the management of an organization to improve the quality of their systems by using the results obtained.

In this project a procedure was developed which could be quite useful for the management of an organization when the performance of the computer-based systems is poor. This procedure requires knowledge of statistical and linear programming techniques. Whenever the management of an organization follows the outlined procedures they will not only be able to pinpoint the most critical problems affecting the success of the computer-based systems in the organization, but the cost models developed will allow them to apply available funds in an optimal way to solve those problems.

ACKNOWLEDGEMENT

The author wants to thank Prof. J.M. Hattingh, Department of Statistics and Operational Research, Potchefstroom University and Prof. H. C. Lucas jr., Graduate School of Business Administration, New York University for their comments and useful hints.

REFERENCES

[1] R. L. Ackoff, (1967). *Management Misinformation Systems,* Management Science 14(4): 147-156.

[2] S. R. Barkin, and G. W. Dixon, (1977). *An Investigation of Information Systems Utilization,* Information and Management 35-45.

[3] P. J. S. Bruwer, (1982). *Bydraes tot modelontwiddeling ten einde gerekenariseerde stelsels se werkverrigting te evalueer.* Unpublished D. Sc. dissertation. PU for CHE, Potchefstroom.

[4] G. W. Dickson, and J. K. Simmons, (1970). *The behavioural side of MIS- some aspects of the "people problem".* Foundation for the School of Business, Indiana University 253-265.

[5] W. J. Dixon, and M. B. Brown, (1977). *Biomedical Computer Programs, P-series.* University of California Press, Berkeley, Los Angeles.

[6] J. W. Frane, and M. Hill, (1976). *Factor Analysis as a tool for Data Analysis.* Comm. Statist. Theory and Methods, VOl. A5, No. 6.

[7] S. I. Gass, (1958). *Linear Programming.* McGraw-Hill Book Company, Inc., New York.

[8] E. T. Garrity, (1963). *Top Management and Computer Profits.* Harvard Business Review. 6-12.

[9] J. Huysmans, (1970). *The Effectiveness of the Cognitive Style Constraint in Implementing Operations Research Proposals.* Management Science, Vol. 17, No. 1, 92-104.

[10] H. C. Lucas, jr. (1978). *Empirical Evidence for a Descriptive Model of Implementation.* MIS Quarterly, 27-42.

[11] H. C. Lucas, jr. (1974). *System Quality, User reactions and the use of Information Systems.* Management Informatics, Vol. 3, No. 4.

[12] H. C. Lucas, jr. (1975). *Performance and the use of an Information System* Management Science, Vol. 21, No. 8.

[13] H. C. Lucas jr. (1974). *Why Information Systems Fail.* Columbia University Press, New York.

[14] R. D. Mason, and I. Mitroff, (1973). *A Program for Research on Management Information Systems.* Management Science, 19(5), 475-487.

[15] E. Mumford, and O. Banks, (1967). *The Computer and the Clerk.* Routledge and Kegan Paul, London.

[16] P. Rabinowitz, (1968). *Applications of linear programming to numerical analysis.* SIAM Review, Vol. 10, No. 2, 121-159.

[17] M. S. Scott Morton, (1971). *Management Decision Systems.* Boston, Division of Research, Graduate School of Business Administration, Harvard University.

[18] B. E. SWANSON, (1974). *Management Information Systems — Appreciation and Involvement.* Management Science, Vol. 21, No. 2.

An Improved Implementation of Grimbleby's Algorithm

Stephen R. Schach
Computer Science Department, University of Cape Town

Abstract
Details are given of an implementation of Grimbleby's algorithm for the common spanning tree problem with running times up to 50% less than for the original implementation. An explanation is given as to why implementations with even lower running times are unlikely.

1. Introduction

The common spanning tree problem may be formulated as follows:
Given a set V of M vertices, and given two sets E_1, E_2 each of N (undirected) edges such that there exists a bijection $f : E_1 \rightarrow E_2$, find all spanning trees T_1 of $G_1 = (V, E_1)$ such that $T_2 = f(T_1)$ is a spanning tree of $G_2 = (V, E_2)$. (If T_1 denotes the tree $\{e_1, e_2, \ldots, e_{M-1} | e_i \text{ in } E_1\}$, then $f(T_1)$ denotes the graph $\{f(e_1), f(e_2), \ldots, f(e_{M-1})\}$).

The tree enumeration method of symbolic circuit analysis is applicable not only to passive circuits [1] and active circuits with active elements in the form of four-terminal current sources (VCCS) [2], but has recently been extended [3] to active circuits containing four-terminal infinite gain amplifiers (FTOA). Since any active device can be modelled by a combination of FTOA and passive components, any active circuit can be analysed in this way. The method consists of constructing voltage and current graphs as described in [3]; thereafter the problem reduces to finding the common spanning trees of the two graphs.

Grimbleby [4] has published an algorithm for enumerating these common spanning trees. Running times for his algorithm are non-trivial; for an implementation in Basic running on a microcomputer he cites execution times of 104 seconds (compiled) and 96 minutes (interpreted) for an active filter circuit of 20 nodes containing 24 passive components and 9 ideal operational amplifiers.

It has recently been pointed out [5] that the CPU time for Grimbleby's algorithm can be reduced by between a third and a half simply by using a doubly linked data structure for storing the trees. This improvement certainly constitutes a meaningful saving in computer time.

In this paper we account for the observed improvement by means of a detailed implementation as well as a complexity analysis. Finally, we explain why we do not consider it likely that implementations with significantly lower running times can be found.

In what follows, the term "original" will refer to Grimbleby's implementation, [4] while "modified" will be used to mean the implementation described in this paper.

2. Grimbleby's Algorithm

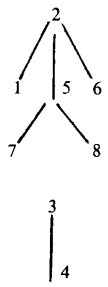
Grimbleby's original algorithm appears in reference [4]. In this paper we use the word "edge" in place of "branch", and "vertex" in place of "node", in order to conform with generally accepted graph theoretical nomenclature; otherwise, Grimbleby's notation is followed.

An inefficiency in the original implementation arises from Step 7 (Backtrack), which has complexity $O(N^2)$. The storage of the edges of the forests as an array of pairs of vertex pointers ($root_j$, $father_j$) requires a sweep through the edges to be repeated "until no further root modification is made" [4]. Consider now the case where the tree is a chain (N, N-1, . . ., 1), and edge (N, N-1) is to be removed. N-2 sweeps are required, and on each sweep O(N) operations are performed. Thus Step 7 has worst case complexity $O(N^2)$. In fact, the average complexity is also $O(N^2)$; this can be shown by a method similar to the complexity analysis of bubblesort [6].

The overall complexity is then at least $O(N^3)$, because a complete spanning tree has N-1 edges each of which has to be unstacked at Step 7 during the running of the algorithm.

3. The Modified Implementation

Instead of storing each edge as a pair ($root_j$, $father_j$) as in the original implementation, we now represent each edge by a quadruple of vertex pointers ($root_j$, $father_j$, son_j, $brother_j$). $Root_j$ is the root of the tree to which vertex j currently belongs. $Father_j$ is the next vertex on the path from vertex j to $root_j$. At any stage during the execution of the algorithm vertex j may have more than one son; son_j is defined to be the leftmost one. Finally, if vertex k = $father_j$ has more than one son vertex, then $brother_j$ denotes the son immediately to the right of vertex j. These definitions are illustrated in Figure 1.



j	$root_j$	$father_j$	son_j	$brother_j$
1	2	2	0	5
2	2	0	1	0
3	3	0	4	0
4	3	3	0	0
5	2	2	7	6
6	2	2	0	0
7	2	5	0	8
8	2	5	0	0

FIGURE 1. Example of nomenclature.

Consider vertex 5. We see that $root_5 = 2$, $father_5 = 2$, $son_5 = 7$, $brother_5 = 6$.

The three basic operations performed on a forest are 'addson' which makes vertex A a son of vertex B, 'removeson' which removes a vertex from its tree thus yielding two trees, and 'changerootpointer' which sets the root pointer of a vertex F and all its descendents in its tree to F. These procedures are implemented as follows:

```
PROCEDURE addson (A, B);
(* this procedure makes vertex A a son of vertex B *)
BEGIN addson
  IF B has no son
    THEN make B the son of A
    ELSE make B the brother of the rightmost son of B;
  father_A := B
END addson;

PROCEDURE removeson (D);
(* vertex D is a son of vertex C; this procedure inlinks D *)
BEGIN removeson
  C := father_D;
  IF D is the only son of C
    THEN set son_C := 0
    ELSE IF D is leftmost son
        THEN make brother_D the son of C
        ELSE scan the sons of C from left to right until a
             node E is found whose brother is D, and set
             brother_E := brother_D;
  father_D := 0
END removeson;

PROCEDURE changerootpointer (F);
(* this procedure sets root pointer of F and all its
  descendents in its tree to F *)
```

```
PROCEDURE recursivechange (G, rootnumber);
(* this procedure sets root pointer of vertex G, its
   son and its brother to rootnumber *)
BEGIN recursivechange
  IF G is non-null
    THEN BEGIN
         root_G := rootnumber;
         recursivechange (son_G, rootnumber);
         recursivechange (brother_G, rootnumber)
           END
END recursivehange;

BEGIN changerootpointer
  root_F := rootnumber;
  recursivechange (son_F, F)
END changerootpointer;
```

Procedures 'addson', 'removeson' and 'changerootpointer' have complexity O(≠S(A)), O(≠S(C)) and O(≠S(F)) respectively, where ≠S(v) denotes the number of sons of vertex v. Clearly 'addson (A,B)' cannot be performed unless A is the root of its tree; in this case procedure 'makeroot' must first be called.

```
  PROCEDURE makeroot (H);
  (* this procedure makes H the root of its tree *)
  PROCEDURE reverselink (J, K);
  (* this procedure makes J a son of K; before the call K is
     a son of J *)
  BEGIN reverselink
    IF J is non-null
    THEN BEGIN
           removeson (K); addson (J, K);
           reverselink (father_J, J)
           END
  END reverselink;
BEGIN makeroot
  reverselink (father_H, H);
  changerootpointer (H)
END makeroot;
```

The complexity of 'makeroot' is $\text{sum}_i[O(\neq S(i))] + O(N)$ where the sum is taken over all nodes i on the path from father_H to root_H in the original tree. The sum is clearly bounded by O(N) since the total number of sons of vertices along any path in a tree cannot exceed the number of vertices in the tree itself. Thus the complexity of 'makeroot' is also O(N).

Procedures 'linkedge' (Step 5 of the algorithm) and 'unlinkedge' (Step 7) can now be given.

```
PROCEDURE linkedge (e);
(* this procedure adds edge e = (P, Q) to the forest *)
BEGIN linkedge
  makeroot (P);
  addson (P, Q);
  changerootpointer (Q)
END linkedge;
```

```
PROCEDURE unlinkedge (e);
(* this procedure removes edge e = (P, Q) from the forest *)
BEGIN unlinkedge
  IF P is the son of Q
    THEN interchange P and Q;
  removeson (Q);
  root_Q := Q;
  changerootpointer (Q)
END unlinkedge;
```

From the complexity analysis of the routines called by 'linkedge' and 'unlinkedge' it is clear that both these procedures have complexity O(N); in the original implementation the corresponding procedures had complexity O(N) and $O(N^2)$ respectively.

The original and modified algorithms have been implemented in Pascal on a Sharp MZ-80B microcomputer. The modified algorithm is some 25% longer, and requires 2M more storage locations (for storing the edges). However, as predicted by the above analysis, the modified algorithm is more efficient. Comparative CPU times are given in Figure 2; savings from 33% to over 50% have been obtained.

Further Improvements

The question might well be asked: are further improvements in the running time of Grimbleby's algorithm possible? The answer is probably not, because in the worst case the main loop of the algorithm considers all possible combinations of M-1 edges from N edges. The algorithm as a whole therefore has worst case complexity $O(N^b)$ where b = min (M-1, N-M+1), and it is therefore unlikely that running times for Grimbleby's algorithm can be further significantly reduced.

M number of nodes	N number of branches	number of trees	CPU time original algorithm	CPU time modified algorithm	percent CPU time decrease
10	14	9	27 s	18 s	33%
10	20	93	255 s	147 s	42%
20	25	2	539 s	234 s	56%

FIGURE 2 : Comparative CPU times.

References

[1.] PERCIVAL, W.S.: The solution of passive electrical networks by means of mathematical trees, Proc. IEE, Vol. 100, Part III, Paper 1492R, 1953, p. 143.

[2.] PERCIVAL, W.S.: The graphs of active networks, Proc. IEE, 102C, Monograph 129R, 1955, pp. 470-471.

[3.] GRIMBLEBY, J.B.: Symbolic analysis of circuits containing active elements, Electron. Lett., Vol. 17, 1981, pp. 754-756.

[4.] GRIMBLEBY, J.B.: Algorithm for finding the common spanning trees of two graphs, Electron. Lett., Vol. 17, 1981, pp. 470-471.

[5.] SCHACH, S.R.: Comment on 'Algorithm for finding the common spanning trees of two graphs', Electron. Lett., Vol. 18, 1982, pp. 988-989.

[6.] KNUTH, D.E.: The Art of Computer Programming. Volume 3: Sorting and Searching, Addison-Wesley, Reading, MA, 1973.

Nebulas as Structural Data Models

H.O. van Rooyen and D.J. Weiermans
University of South Africa, South Africa

Abstract
Nebula modelling is a generalised structural modelling technique which applies to any finite set of objects (entities) and n-ary relationships ($n \geq 2$). A nebula model can be presented in either diagrammatic or tabular form, and combines the properties of graph (binary) data models with those of tabular (relational) data models. We give a brief introduction to nebula data modelling by means of a simple illustration.

1. Introduction

Modern binary ($n = 2$ only) structural modelling is based on the work of Harary, Norman and Cartwright [1]. Lendaris has defined a structural model as "a collection of elements and their relationships" and stated that "structural modeling holds the promise of converting a completely intuitive process of model building into a more systematic approach, and enhancing communication within a heterogeneous group" [2]. McLean and Shepherd defined the term "structure" as "the way in which the component parts of the complex whole are inter-related" and stated that "a structural model focuses on the task of selecting the components of a model and explicitly stating the interactions between them" [3]. To date, structural modelling has dealt only with binary relationships, but nebula modelling extends it to n-ary relationships with $n \geq 2$ [4,5,6,7,8,9 and 10]. Graph theory and network theory are special cases of nebula theory. Nebula based structural modelling provides an unquantified description of any real world situation in the following way. First the situation is reduced (simplified) to a description in terms of a set (countable or countably infinite) of object (entity/concept) names and (sentential) statements of relationships among the objects named. Each statement is then encoded as an n-tuple of object names, and these object names and n-tuples constitute the model.

2. Basic concepts, and an example

By an abstract *nebula* of type λ we mean a pair $\langle A,F \rangle$ where the underlying set A is denumerable, F is a function which correlates with every $i \in$ dom F a function

$$F(i): A^{\lambda(i)-2} \rightarrow P(AxA), \text{ and } \lambda: \text{dom } F \rightarrow (\omega-2)$$

where $\omega = \{0,1,2,...\}$ and $n = \{0,1,...,n-1\} \in \omega$.

In this section we use an example to illustrate some basic concepts of nebula theory, namely

- diagrammatic presentation of $\langle A,F \rangle$
- occurrence $x_{i;j}$, y of label i;j in a diagram of $\langle A,F \rangle$
- tabular presentation of $\langle A,F \rangle$.

Formal definitions of these concepts can be found in [5,6,7,8,9 and 10].

Example

The degree curricula in the Science Faculty at UNISA are composed of small course units called modules. Prerequisite and parallel requirements are defined for each module, specifying the conditions under which a student may register for that module. A *prerequisite* requirement for a module v is a set E_v of one or more modules for which a student shall have obtained credit before he may enrol for module v. A *parallel* requirement for a module v is defined to be a set L_v of one or more modules which is such that the student must enrol for every $u \in L_v$ for which he has not yet obtained credit at the same time as he enrols for v.

Consider the following selection of data, where COS denotes Computer Science and COS000 represents the entrance qualification.

Module	*Prerequisites*	*Parallel modules*
COS111	COS000	—
COS121	COS000	—
COS131	COS000	INF101
COS132	COS000	COS131
COS211	COS111,121,INF101	—
COS212	COS111,121,INF101	COS211,221
COS221	COS111,121,INF101	COS211
COS231	COS131,132	—
COS231	COS111,MAT101,102	—
COS311	COS211,221	—

Table 1

For every module b in the table an n-tuple [occurrence/datum]

$$\langle a, r_1, r_2, \cdots r_k, S_1, S_2, .., S_\ell, b \langle$$

can be constructed, representing a relationship between the prerequisites and the parallels for the relevant module, where

$E_b = \{a, r_1, r_2, ..., r_k\}$ represents the prerequisites, and
$L_b = \{s_1, s_2, ..., s_\ell\}$ represents the parallels, for the module b.

We choose a nebula model $\langle A,F \rangle$ of the data given in table 1. Let the members of A represent the modules. Each $i \in$ dom F represents a registration relationship, and has the form $i = \langle k,\ell \rangle$, $k, \ell \in \omega$, with k = (number of prerequisites) — 1, and ℓ = number of parallels, for some module. Then

$$F(i): A^{\lambda(i)-2} \rightarrow P(AxA), \text{ where } \lambda(i) = k + \ell + 2, \text{ and}$$

$$(F(i))(j) = \{\langle a,b \rangle \in AxA \mid \langle a, r_1, ..., r_k, s_1, ..., s_\ell, b \rangle \text{ belongs to the } \lambda(i)\text{-ary relation named by } i \in \text{dom } F \text{ and } j = \langle r_1, ..., r_k, s_1, ..., s_\ell \rangle\}; \; \phi \text{ otherwise.}$$

This nebula $\langle A,F \rangle$, is such that
A = {COS000, COS111, COS121, COS131, COS132, COS211, COS212, COS221, COS231, COS311, INF101, MAT101, MAT102}
dom F = $\{\langle 0,0 \rangle, \langle 0,1 \rangle, \langle 1,0 \rangle, \langle 2,0 \rangle, \langle 2,1 \rangle, \langle 2,2 \rangle\}$
and, for example,

$$F(\langle 2,0 \rangle)(\langle x_1,x_2 \rangle) = \begin{cases} \{\langle \text{COS111, COS211} \rangle\} & \text{if } x_1 = \text{COS121 and } x_2 = \text{INF101;} \\ \{\langle \text{COS111, COS231} \rangle\} & \text{if } x_1 = \text{MAT101 and } x_2 = \text{MAT102;} \\ \phi & \text{otherwise.} \end{cases}$$

A diagrammatic presentation of $\langle A,F \rangle$ is given in figure 1, and the tabular presentation in table 2.

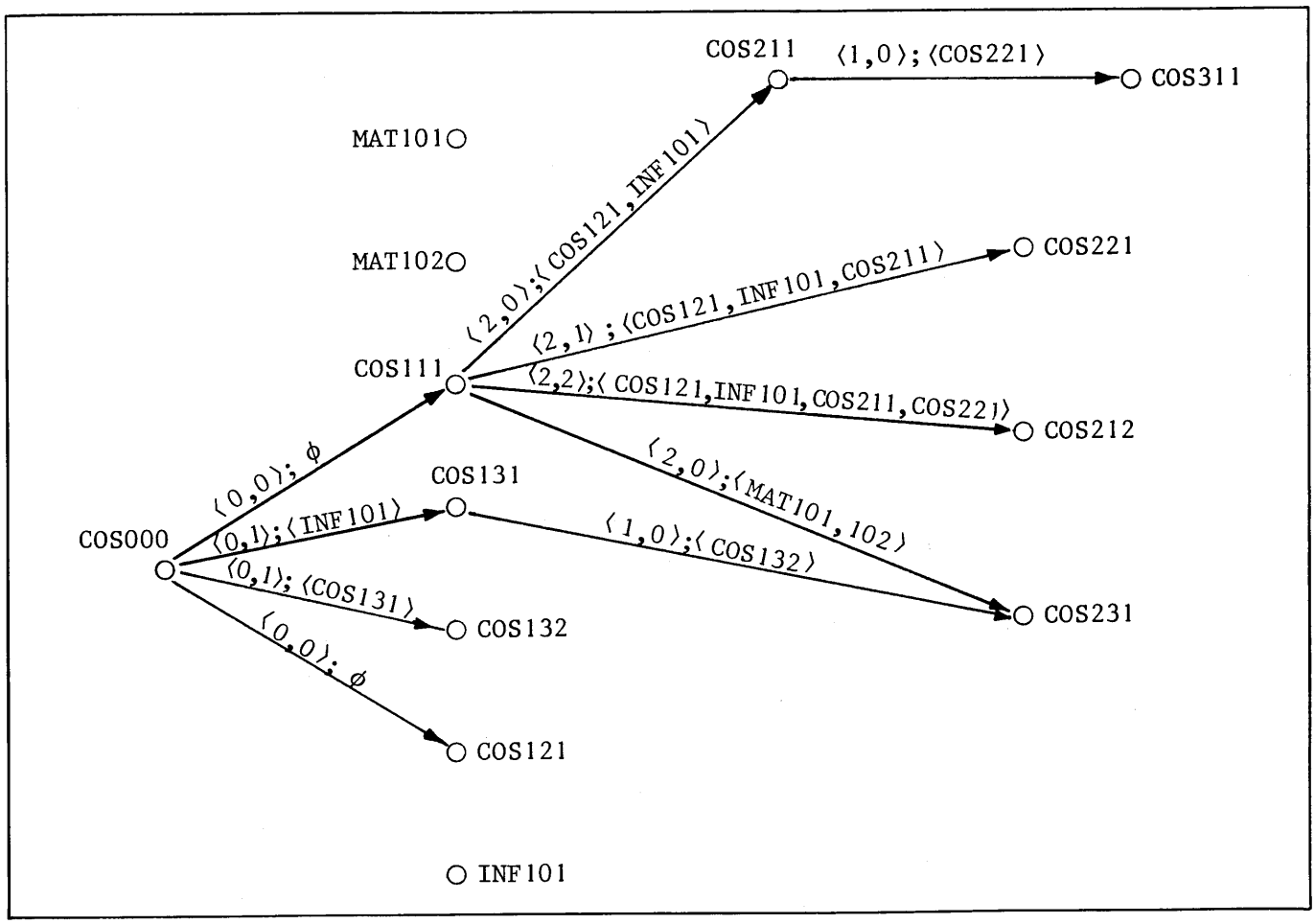


FIGURE 1 — Diagram of nebula ⟨A,F⟩

3. Substructures and Cascades*

Let ⟨A,F⟩ be a nebula of type λ and ⟨B,G⟩ be a nebula of type ρ. We say that ⟨B,G⟩ is a *subnebula* of ⟨A,F⟩, written ⟨B,G⟩ ≤ ⟨A,F⟩, iff

1. B ≤ A and
2. dom G ≤ dom F and
3. ρ = λ| dom G where | denotes restriction and
4. for each i ϵ dom G and for every j ϵ $B^{\lambda(i)-2}$we have (G(i))(j) ≤ (F(i))(j).

⟨B,G⟩ is called a *spanning subnebula* of ⟨A,F⟩ iff B = A. ⟨B,G⟩ is a *reduct* of ⟨A,F⟩ iff B = A and, for every i ϵ dom G, we have G(i) = F(i). Every reduct is a spanning subnebula, but the converse is not generally true.

Supremum (∨), infimum (∧), and complement of certain subnebulas of given nebula ⟨A,F⟩, can be formally defined, as can homomorphisms between nebulas [see 5, 6, 7, 8, 9, and 10].

The set X of all subnebulas of a given nebula ⟨A,F⟩, with partial order ≤ and corresponding operations ∧ and ∨, constitutes a distributive lattice with universal element ⟨A,F⟩ and null element ⟨ϕ,ϕ⟩. See [5].

In a nebula data model ⟨A,F⟩ *all* data manipulation is achieved by applying the operations ∨ and ∧ to subnebulas of ⟨A,F⟩, and data retrieval is attained by generating an appropriate subnebula using an operator scheme called "cascade".

4. Definition of a Cascade*

In general a *cascade* in ⟨A,F⟩ is a sequence {⟨B_k,G_k⟩|kϵω} of subnebulas of ⟨A,F⟩ such that if i≤j then ⟨B_i,G_i⟩ is a subnebula

RELATION TABLE			*ENTITY TABLE*
i	j	arrow	COS000
			COS111
⟨0,0⟩	ϕ	⟨COS000, COS111⟩	COS121
	ϕ	⟨COS000, COS121⟩	COS131
⟨0,1⟩	⟨INF101⟩	⟨COS000, COS131⟩	COS132
	⟨COS131⟩	⟨COS000, COS132⟩	COS211
⟨1,0⟩	⟨COS132⟩	⟨COS131, COS231⟩	COS212
	⟨COS221⟩	⟨COS211, COS311⟩	COS221
⟨2,0⟩	⟨COS121, INF101⟩	⟨COS111, COS211⟩	COS231
	⟨MAT101, MAT102⟩	⟨COS111, COS231⟩	COS311
⟨2,1⟩	⟨COS121, INF101, COS211⟩	⟨COS111, COS221⟩	INF101
⟨2,2⟩	⟨COS121, INF101, COS211, COS221⟩	⟨COS111, COS212⟩	MAT101
			MAT102

TABLE 2 — Nebula ⟨A,F⟩ in tabular form

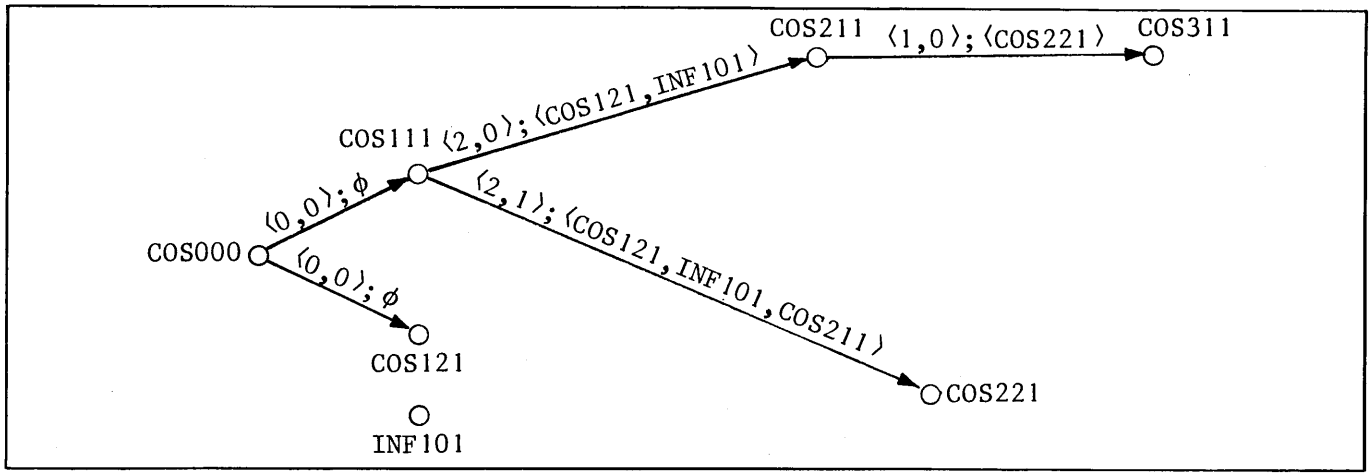


FIGURE 2 — Diagram of (B_3,G_3).

of $\langle B_j,G_j\rangle$. This definition can be limited in such a way that cascades are computer generable. An important example of this is the following.

Definition

The sequence $\{\langle B_k,G_k\rangle | k\epsilon\omega\}$ of subnebulas of $\langle A,F\rangle$ is called a *cascade generated from* $\langle B_0,G_0\rangle$ by Γ and τ iff

1. $B_o \leq A$ and G_0 is specified so that $\langle B_0,G_0\rangle \leq \langle A,F\rangle$ and
2. $B_{k+1} = B_k \cup (\Gamma_{R_k}(C_k) \cup \tau_{R_k}(C_k))$ where $C_0 = B_0$ and $C_k \leq B_k$ so that $\langle B_k,G_k\rangle \leq \langle B_{k+1},G_{k+1}\rangle$, and
3. dom $G_k \leq$ dom $G_{k+1} \leq$ dom F and for each i ϵ dom G_{k+1} we have $G_{k+1}(i) : (B_{k+1})^{\lambda(i)-2} \rightarrow P((B_{k+1})^2)$ where $(G_{k+1}(i))(j) = (G_k(i))(j) \cup ((F_{R_k}(i))(j) \cap (C_k \times B_{k+1}))$ and i ϵ dom G_{k+1} — dom G_k iff there is at least one $j \epsilon (B_{k+1})^{\lambda(i)-2}$ for which $(F_{R_k}(i))(j) \cap (C_k \times B_{k+1}) \neq \phi$, where the subscript R_k denotes the Γ and τ functions of some subnebula $\langle A,F_{R_k}\rangle$ of $\langle A,F\rangle$, with $R_k \leq \bigcup_{i,j}|(F(i))(j)|$, at each step k. Such a cascade is said to be *limited* iff, at each step k, R_k is chosen in such a way that $\tau_{R_k}(C_k) \leq C_k$, i.e. if $(F(i))(j) \epsilon R_k$ then $j \epsilon (C_k)^{\lambda(i)-2}$.

Strictly speaking it is not the cascade which is generated, but there is an $m\epsilon\omega$ such that for every $n \geq m$ we have $\langle B_n,G_n\rangle = \langle B_m,G_m\rangle$, and it is in fact $\langle B_m,G_m\rangle$ which is generated by Γ and τ from $\langle B_0,G_0\rangle$ in $\langle A,F\rangle$. Thus such a cascade generates a subnebula of $\langle A,F\rangle$, and with the appropriate choices of C_k and R_k at each step, every subnebula of $\langle A,F\rangle$ can be generated as the end product of a non-trivial cascade from some $\langle B_0,G_0\rangle$. Note that $\langle B_k,G_k\rangle \leq \langle B_{k+1},G_{k+1}\rangle$ is an overriding condition, so a cascade must stop when every choice of $C_k \leq B_k$ and $R_k \leq \bigcup_{i,j}|(F(i))(j)|$ yields $\langle B_{k+1},G_{k+1}\rangle \leq \langle B_k,G_k\rangle$. Cascades were originally defined to model thinking processes. Here every query defines a cascade, or sequence of cascades, which generates a subnebula containing the information relevant to that query.

To illustrate the use of "cascade" we return to our example.

Assume that a student specifies module COS311 as a goal for his degree. He has to know which modules lead to the specified module. This information can be retrieved by means of a cascade generated by Γ^{-1} and τ^{-1}, which is defined as follows:

i=0: $B_0 = \{COS311\}$ and $G_0 = \phi$ with dom $G_0 = \phi$.

i=1: $B_1 = B_0 \cup \Gamma_{R_0}^{-1}(C_0) \cup \tau_{R_0}^{-1}(C_0)$ where $C_0 = B_0$ and we take $R_k = \bigcup_{i,j}|(F(i))(j)|$ for each $k \epsilon \omega$ and drop the subscript R_k in this case.

Now $\Gamma_{R_0}^{-1}(C_0) = \{COS311,COS211\}$ and $\tau_{R_0}^{-1}(C_0) = \{COS221\}$,

therefore $B_1 = \{COS311,COS211\} \cup \{COS221\}$
$= \{COS311,COS211,COS221\}$.

$(G_1)_{10}(\langle x\rangle) = \{\langle COS211,COS311\rangle\}$ if x = COS221;
ϕ otherwise.

dom G_1 = dom $G_0 \cup \{\langle 1,0\rangle\} = \phi \cup \{\langle 1,0\rangle\} = \{\langle 1,0\rangle\}$.

Alternatively, we can take dom G_k = dom F for each k.

Proceeding in this way we find that $\langle B_3,G_3\rangle = \langle B_4,G_4\rangle$. A diagram of $\langle B_3,G_3\rangle$ is given in figure 2.

Several variations and combinations of cascades can be used, and appear to provide an adequate retrieval mechanism. Cascades are discussed in more detail in chapters 4 and 5 of [6].

5. Concluding Remarks

Nebula modelling is being used, with increasing success, in the field of education [7, 8 and 12], and has great potential for the modelling of knowledge structures, learning, and thought processes.

Many claims have been made as to the superiority of graphs and tables, one to another, for data modelling. Tsichritzis [13], for example, says the following: "Each approach has its strength and weaknesses. Neither is intrinsically simpler or more powerful. Relational data models . . . are elegant in their simplicity, but very limited semantically. Binary data models can represent complex relationships concisely, but are not very user oriented." About comparison of these data models he states: "The basis of comparison, therefore, should not be whether relations or graphs are simpler or more natural. Instead, one should consider the constructs that need to be added to tables and graphs to enable them to model adequately complex situations." Different data models provide us with different data modelling tools, and their usefulness depends on the problem to which they are applied. In this paper we have indicated, very briefly, that nebula theory provides a data model which has both a graph (binary) data model facet and a tabular (relational) data model facet in its representation of the structure of data. It retains the advantages, and some of the disadvantages, of both.

Tsichritizis [13] also states: "the ultimate data modeling tool which mixes data and knowledge about data is the predicate calculus . . . Strictly typed data models make some very ad hoc assumptions about what kinds of categories of objects will be allowed and what kinds of relationships between these categories can be specified. They force data to be homogenous. As a matter of fact, most data model research deals with finding good, general categories which can enforce this homogeneity requirement (e.g. the notion of a relation and a record type)." In the predicate calculus, "the emphasis is on allowing everything to be expressed in a uniform and formal environment without artificial restrictions on typing or categorizing objects". This last remark also applies to nebula modelling, because the members of the set A in an abstract nebula can be used to represent *any* "object", i.e. any "kinds of categories of objects" [13] are allowed. In nebula data modelling queries must be formulated in terms of subnebulas and paths, and thus the choice of an appropriate nebula model for a given problem may be difficult, or even impossible. Nevertheless, we feel that nebula data modelling deserves further investigation.

Experimental nebula models of a registration advice system

[6] and a syllabus (learning structure) design and analysis system [7,8] have been successfully implemented at UNISA.

References

[1] F. Harary, et al. *Structural Models: An Introduction to the Theory of Directed Graphs.* John Wiley & Sons. 1965.

[2] G. G. Lendaris, *Structural Modeling — A Tutorial Guide.* IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-10, No. 12, December 1980, pp. 807-840.

[3] J. M. McLean & P. Shepherd, *The importance of model structure. Futures.* Febr. 1976.

[4] H. O. van Rooyen, *Binary Networks in Graph Theory.* Ph.D. thesis, UNISA, 1976.

[5] H. O. van Rooyen, in collaboration with D. E. Wolvaardt and D. J. Weiermans, *Introduction to Nebula Theory.* Com. Sci. and Info. Systems Colloquium of the Univ. of S.A., Vol. 2, Nov. 1981.

[6] D. J. Weiermans, *Development of a Nebula Database for a Student Advice System.* Ph.D. thesis submitted for examination, UNISA, 1983.

[7] D. E. Wolvaardt, *Development of Nebula Theory with Application to Syllabus Databases.* Ph.D. thesis, UNISA, 1982.

[8] H. O. van Rooyen, D. E. Wolvaardt & H. Helm, *Formalized Concept Maps in a Science of Education.* Working Paper NEB≠4. UNISA and RHODES, 1982.

[9] W. A. Labuschagne & H. O. van Rooyen, *On a Generalisation of Cayley Diagrams.* UNISA, 1980. Accepted for publication in Q.I., 1983.

[10] H. O. van Rooyen, *Menger's Theorem in a Nebula.* Proc. Second South African Computer Symposium, 1981. Accepted for publication in Q.I., 1983.

[11] Calendar 1981 — University of South Africa.

[12] R. J. van den Heever, S. H. von Solms & D. E. Wolvaardt, *Programontwerp: Fundamentele Beginsels.* To be published, HAUM, 1983.

[13] D. C. Tsichritzis & F. H. Lochovsky, *Data Models.* Prentice Hall, Inc., 1982.

*In this paper ≤ is used to denote "less than or equal to", "subject of", or "subnebula of", depending on the context.

Data Structures for Generalized Network Algorithms

Desmond C. Currin
National Research Institute for Mathematical Sciences of the CSIR, Pretoria

Abstract
Generalized network problems involve the optimization of a flow through a network. In contrast to normal networks, generalized networks include multipliers which alter the flow as it passes through the arcs. This enables the modelling of changes in flow caused by factors such as interest rates, as well as by processes such as energy conversion.

Generalized network problems can be solved by using the simplex algorithm. However, if the simplex bases are represented as forests of quasi-trees, connected graphs with a single cycle, a far more efficient algorithm can be obtained. Matrix multiplications required by the simplex algorithm can be interpreted as traversals of the quasi-trees, and consequently many of the arithmetic operations can be replaced by logical operations. This increases the numerical stability of the algorithm.

In this paper we outline this approach and discuss some of the data structures that can be used in its implementation.

CR Categories and Subject Descriptors : G.1.6 Numerical Analysis: Optimization — Linear Programming; G.2.2 Discrete Mathematics : Graph Theory — Network Problems; E.1 Data Structures — Graphs and Trees
General Terms: Algorithms
Additional Key Words and Phrases: generalized networks, network optimization, quasi-trees

1. Introduction

Generalized network models include the assignment, transportation and transshipment problems. The major advantage over these models is that the introduction of multipliers permits the modelling of problems where material appreciates, depreciates or changes from one form to another. The utility of generalized networks is illustrated by the many applications [1,2,3].

A *generalized network* model consists of a set of n nodes N and a set of arcs A. The problem is to find a flow x_k ($k \in A$) along each of the arcs such that:
(a) it is of minimum cost (i.e. it minimizes $\Sigma c_k x_k$); (1)
(b) it satisfies the capacity constraints placed on the arcs (i.e. $\ell_k \le c_k \le w_k$): and (2)
(c) it is sufficient to meet certain demands d_i ($i \in N$) at each of the nodes. When d_i is negative, this is interpreted as a supply of $(-d_i)$ units available at node i.

For generalized networks this last requirement may be formulated as

$$\sum_{\text{Arcs leading to node i}} m_k x_k \quad - \quad \sum_{\text{Arcs leading from node i}} x_k = d_i \qquad (3)$$

The coefficients m_k are the *multipliers,* and they indicate by which factor the flow will increase (or decrease) as it passes along the arc. The above equation states that the total flow into node i minus the total flow from the node should equal the demand, or the negative of the supply, at that node.

The algorithm used to solve these models is derived from the simplex algorithm for linear programs, but, because of the special structure of generalized networks, the solution can be obtained thirty to fifty times faster than by means of a comparably sized linear program [4].

Past experience with this and a similar problem, namely the transshipment problem, has shown that the speed of the algorithm depends on the data structures used in its implementation.

2. Linear Programs and the Simplex Algorithm

The generalized network problem, as formulated above, corresponds to the capacitated linear program

$$\begin{aligned} &\text{Min} \quad c\,x \\ &\text{subject to} \\ &N\,x = d \qquad (4) \\ &\ell \le x \le w \end{aligned}$$

The coefficient matrix N has one column corresponding to each arc in the nework. Each column has at most two non-zero elements; if $k = (i,j)$ is an arc leading from node i to node j, then $n_{ik} = -1$ and $n_{jk} = m_k$; and if $k = (i,i)$ is a loop on node i, then $n_{ik} = m_k - 1$.

The *simplex algorithm,* which is used to solve the linear program, maintains a set of n *basic* variables. The non-basic variables are taken to be at either their lower limit ℓ_k or at their upper limit w_k. Consequently, the values of the n basic variables are determined by the n linear equations in [4].

At each iteration the simplex algorithm determines whether movement of one of the non-basic variables x_k from its lower or upper limit will reduce the total cost. If there is no such variable, the current solution is optimal. Otherwise, the algorithm determines how far that non-basic variable can move before it, or one of the basic variables x_r, reaches its upper or lower limit.

In the former case the non-basic variable x_k goes to its opposite limit, and in the latter case the basic variable x_r becomes non-basic while the non-basic variable x_k becomes basic. This is referred to as a *pivot.* The algorithm pivots until an optimal solution is obtained.

In brief the simplex algorithm employs three major steps:
(a) firstly it establishes an initial basis;
(b) in subsequent iterations it determines whether a non-basic variable is eligible for entry to the basis; and if so,
(c) it computes the effect of changing the value of that variable on the variables that are currently basic.

In this paper we will concentrate on step (b) and show how this can be efficiently implemented for generalized network models. The simplifications obtained for this step are comparable to simplifications that can be obtained in the other two steps.

3. The Structure of the Basis and the Effects of a Pivot

It can be shown that the arcs, associated with the basic variables in a generalized network, form a forest of graphs, each having a single *cycle* and a number of attached sub-trees. These graphs are called *quasi-trees*[5].

As an example consider the network in Figure 1. This shows the basic arcs as solid lines and the non-basic arcs as dotted lines. The basic arcs form two separate quasi-trees.

When the algorithm pivots, either of the following may happen:
(a) the non-basic arc selected may move from its current level

to its opposite level without any of the basic arcs becoming non-basic; or

(b) the non-basic arc replaces one of the basic arcs in the basis.

In the former case the structure of the basis is unaffected, but in the latter case a number of interesting occurrences may take place. When the basic arc leaves the basis, it either splits the cycle of a quasi-tree, transforming it into a tree as in Figure 2, or it disconnects a sub-tree from a quasi-tree as in Figure 3. In either case we are left with one tree and a number of quasi-trees.

The arc entering the basis changes this tree into a quasi-tree, either by forming a new cycle as in Figure 4, or by linking it onto an existing quasi-tree as in Figure 5.

4. The Predecessor Array

The basis can be represented by means of a *predecessor* array p(.). For a node on the cycle of quasi-tree this indicates the previous node encountered when circling the cycle in a clockwise direction. For a node in a tree attached to a cycle, the predecessor indicates the father of that node.

In Figures 1-5, the predecessors of each node is indicated by means of the arrows drawn on the basic arcs.

These figures also show that only those arcs on the path joining the incoming arc to the outgoing arc alter during a pivot and these changes merely involve a reversal of the direction of the arrows. Updating of the predecessor array during each pivot is thus very simple.

5. Finding a Variable Eligible for Entry to the Basis

In this section we consider the problem of finding a variable that will reduce the total cost when introduced into the basis.

Most books on linear programming contain an equation, which, apart from changes in notation, reads

$$\bar{c}_k = c_k - c_B B^{-1} N_k \qquad (5)$$

The quantity $\bar{c}_k$ is the *reduced cost* and indicates how the total cost will alter when the value of x_k is changed. If $\bar{c}_k$ is negative, then the total cost will decrease when x_k is increased. Similarly, if $\bar{c}_k$ is positive, we should decrease x_k in order to reduce the total cost. Consequently, the non-basic variables eligible for entry to the basis are those which are at their lower limit and have $\bar{c}_k$ negative, as well as those which are at their upper limit and have $\bar{c}_k$ positive.

Expression (5) involves

(a) the cost c_k associated with the variable x_k;

(b) the costs c_B associated with all basic variables;

(c) the matrix B which comprises the columns of N corresponding to the basic variables; and

(d) N_k which is the k'th column of N.

Efficient implementations of the simplex algorithm normally store B^{-1} as a product of elementary matrices $B^{-1} = E_1 E_2 \ldots E_\ell$ and then compute the quantity $c_B B^{-1} N_k$ by using matrix multiplications.

For generalized networks an approach without matrix multiplications is used. Firstly, we write $c_B B^{-1} N_k$ as uN_k by letting $u = c_B B^{-1}$. The values u are the *node potentials*. We then observe that for an arc $k = (i,j)$ leading from node i to j, N has at most two non-zero values, $n_{ik} = -1$ and $n_{jk} = m_k$. Consequently, the formula for the reduced cost simplifies to $\bar{c}_k = c_k + u_i - m_k u_j$. For a loop on node i, $k = (i,i)$, the formula reduces to $\bar{c}_k = c_k + (1 - m_k)u_i$.

It is thus simple to compute the reduced costs — provided that we know the values of the node potentials.

6. Computing the Node Potentials

Since it is known that the reduced costs of all basic arcs are zero, we can, in principle, determine the node potentials by setting the reduced costs of the n basic arcs to zero and solving the resulting n linear equations.

The problem can be simplified by observing that we can com-

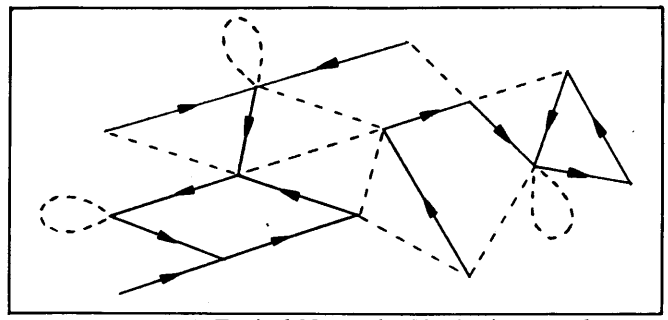


FIGURE 1. A Typical Network. The basic arcs, shown as solid lines, form two quasi-trees.

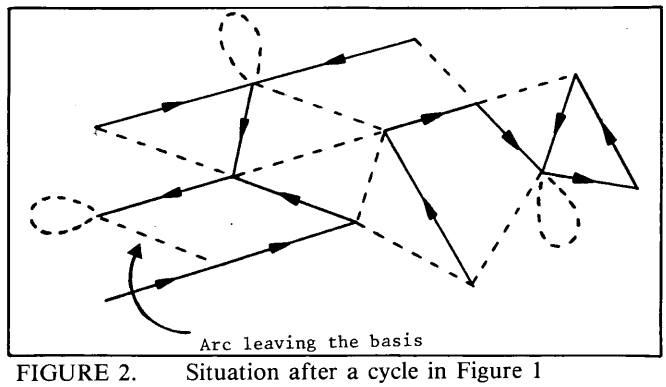


FIGURE 2. Situation after a cycle in Figure 1 has been split.

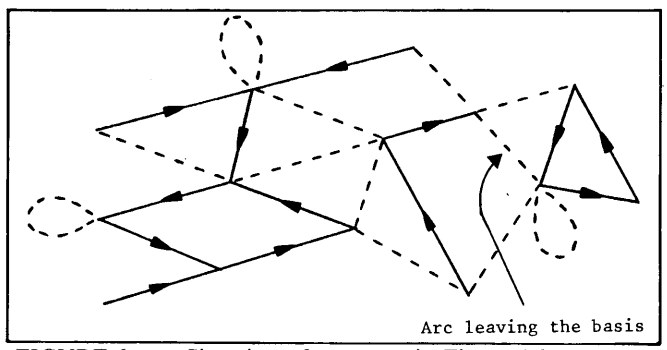


FIGURE 3. Situation after a tree in Figure 1 has been disconnected from a quasi-tree.

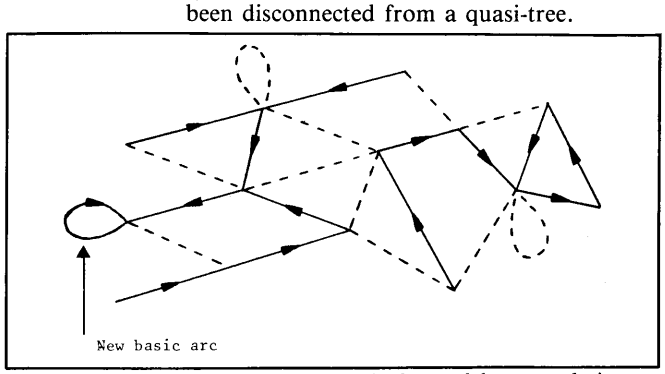


FIGURE 4. A new quasi-tree is formed by completing a cycle in Figure 2.

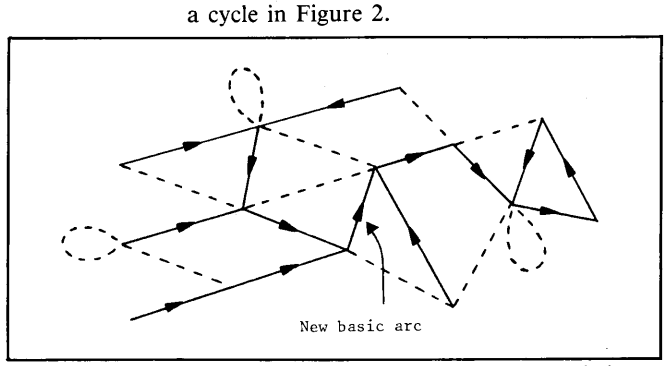


FIGURE 5. The tree has been connected to an existing quasi-tree in Figure 2.

pute the potential of a node in one of the sub-trees, if we know the potential of its predecessor (since the reduced cost of the intervening arc is zero). The potentials of the nodes in the sub-trees can thus be established by traversing the sub-trees in *pre-order* — we visit and compute the potential for a node after we have visited its predecessor.

The problem of determining the potentials of nodes lying on a cycle is slightly more complicated. If the potential of some node i is u_i we can circle the loop and compute the potentials of the other nodes j on that cycle *in terms of* u_i. These potentials will all be a linear function $a_j + b_j u_i$ of the initial potential u_i. During a first pass around the cycle we establish the values for a_j and b_j.

The last node ℓ encountered before returning to node i will relate $u_\ell = a_\ell + b_\ell u_i$ to u_i. This enables us to solve for u_i.

On a second pass around the cycle we compute the potentials for the other nodes j.

Not all the node potentials change at each pivot. When a tree is linked onto an existing quasi-tree, only the nodes on that tree need to be recomputed, and when a new cycle is formed all the potentials on the new quasi-tree need to be evaluated.

7. The Double-Linked Tree and the Thread Successor Array

In order to compute the node potentials we must be able to
(a) circle the loop of a quasi-tree; and
(b) traverse the sub-trees in pre-order.

The former can be done by using the predecessor array. While it is possible to implement the latter using only predecessor information, it is rather inefficient and for this reason a new data structure is normally introduced.

One method [6] that can be employed is storing for each node i the set S_i of its immediate neighbours in the quasi-tree. For each basic arc (i,j) we have $i \in S_j$ and $j \in S_i$. Consequently, this is referred to as a *double-linked tree.*

For a node in a sub-tree its sons are the nodes $S_i - \{p(i)\}$. By using this information and a stack-based procedure, we can traverse the sub-tree in pre-order.

During each pivot one arc becomes basic while another arc becomes non-basic. Consequently, two elements are removed from the sets while two are introduced. Minimal updating is thus required.

One disadvantage of this method is that 4n words of storage are required for its implementation. An alternative [7] is to store the *thread successor* array, which uses only n words of storage. The thread successor of a node is the next node to be visited when traversing the tree in pre-order.

The thread successor array is more difficult to implement, since the elements can change drastically from one iteration to the next. Details for updating this structure can be found in Currin [6]. Computational experience shows that, although the thread successor method is slower than the double-linked tree method, the difference is not large and the thread successor method can be employed profitably when storage is at a premium.

8. Conclusions

Solving a generalized network model by using a specialized computer code is far more efficient than using a standard linear programming code.

Because of the special structure, the basis can be stored as a graph (a forest of quasi-trees). In contrast, a linear programming code has to store the inverse basis B^{-1}. Similarly, calculations that are normally done using matrix multiplications can be replaced by simple scalar operations performed while traversing the quasi-trees.

The net result is that many arithmetic computations are omitted. Although there is an increase in the number of logical operations performed, these can be executed faster and more precisely than arithmetic operations. This results in the computer code being more efficient and numerically stable.

References

[1.] Mathematical Programming Study, 1981, vol 15.

[2.] Glover, F. & Klingman D. 1975. Real world applications of network related problems and breakthroughs in solving them efficiently. ACM Trans. Math. Software, vol. 1, no. 1.

[3.] Glover, F & Klingman, D. 1977. Network applications in industry and government. AIIE Trans., pp 363-376.

[4.] Glover, F., Hultz, J., Klingman, D. & Stutz, J. 1978. Generalized networds: A fundamental computer based planning tool. Man. Sci., vol. 24, no. 12.

[5.] Maurras, J. 1972. Optimization of the flow through networks with gains. Math. Prog., vol 15, pp 291-314.

[6.] Currin, D. 1983. A comparative evaluation of algorithms for generalized network problems. TWISK 289, CSIR, 141p.

[7.] Kennington, J. & Helgason, R. 1980. Algorithms for Network Programming, Wiley, 291p.

Modelling Blocking on Admission of Tasks to Computer Systems

S. Wulf
Comcon (Pty) Limited, Johannesburg

Abstract
The need for modelling techniques to allow blocking of tasks prior to admission to the computer system in queueing network models is described. A detailed algorithm which presents a technique to enforce blocking by priority of class for an arbitrary number of blocking constraints is presented. The resultant solution is exact and produces an analysis by class of admission queueing delays.

1. Introduction

Queueing network models [1] have been used extensively for modelling computer systems. Most of the modelling performed to date has been confined to construction of models to describe existing computer systems and extrapolate the growth of the component workloads so as to predict the effect of upgrading hardware components. In a few cases, attempts have been made to predict the effects of systems not yet implemented [2], [3] and [4]. The basic statistics required from queueing network models are response times per class together with the associated utilisations, waiting times and queue lengths at each component device of the system.

These statistics are consequent upon the accurate modelling of all system environmental components which contribute towards the overall response times. They include the data communications network, the user terminal characteristics and the behaviour of the computer system itself. In particular, the computer system component incorporates certain multiprogramming constraints by type of workload which affect overall response time. For example, a particular online system may only be capable of processing six tasks concurrently whereas the network volumes dictate the concurrent presence of ten tasks at the computer system. This means that a large proportion of the total number of tasks at the system must queue for admission into the system.

Unless the queueing network model used for analysing the system takes these constraints and resultant admission queueing into account, the resultant model output will not be realistic.

2. Approaches used to date

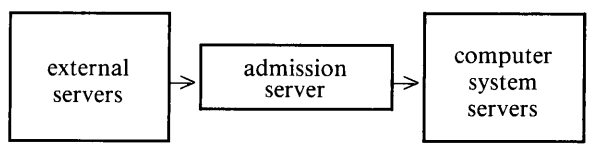
Decomposition [5] consists of analysing a submodel in isolation, (ie the computer system submodel component) and then replacing this subsystem in the original model by a single, composite load-dependent server which appears to the rest of the system to behave the same as the original subsystem. Such an approximation is accurate if the rate of interaction within the submodel is substantially higher than the rate of interactions between the submodel and the rest of the model. In the case under discussion, the submodel consists of computer system components with service times of the order of 10^{-2} seconds whereas the terminal network has service times of the order of 10 seconds. Consequently, decomposition has been widely used as a technique to model large terminal-based systems.

Although effective results can be derived using this approach, the multi-step nature of arriving at a model solution impedes the ease of formulation and solution of the model. Consequently, an alternative approach is needed to model blocking of tasks on admission from part of the queueing network (the external system) to the remaining part (the central computer system itself).

The approach to blocking presented here is based on a technique suggested in [6] and that presented in [4]. The latter reference contains a validation of the technique. We assume that the queueing network consists of a number of servers such that some are located external to the computer system and others are located within the computer system where the blocking constraints hold. We define an additional dummy server which serves as the system admission adjudicator. It initially has zero service time. The model is then solved. If the resultant solution is feasible in terms of the blocking constraints then the algorithm terminates. Otherwise, class-dependent service times are calculated so as to produce a feasible solution by delaying additional tasks at this admission adjudicator. The solution obtained at the end of this procedure is not only feasible — the extent of the delay prior to admission of tasks of each class can be measured.

3. Notation used in the Algorithm

We assume the the model topology is as follows:



where "external servers" refers to servers outside the computer system, "computer system servers" to those inside the computer system and "admission server" to the dummy server with initial class-dependent service times all set to zero.

Let K denote the number of classes and M be the number of queues (servers) in the network. We then define the notation used in the algorithm as follows:

N_k No. of class k jobs in the network for k = 1, . . ,K
S_{mk} Mean service time of class k job at queue m.
v_{mk} Visit ratio of class k jobs at queue m.
Q_{mk} Mean no. of jobs of class k at queue m.
W_{mk} Mean waiting time of class k jobs at queue m.
T_{mk} Throughput of class k jobs at queue m.

We define further the following set of multiprogramming blocking constraints:

$$\sum_{r \epsilon C_c} \quad \sum_{\substack{m = \text{computer} \\ \text{system queues}}} Q_{mk} \leq P_c \qquad (1)$$

where P_c is a constant and C_c is a set of classes such that $k \epsilon C_c$ implies $k \epsilon \{1, \ldots, K\}$

Define the admission server to be device f so that external devices are such that m = 1, . . .,f-1 and internal devices such that m = f+1 . . .,M

Each blocking constraint (1) is defined for a given set of classes C_c. Each class C_c has a given priority G_k.

The following algorithm calculates values S_{fk} (k = 1, . . . ,K) in such a way that if priority $G_r < G_s$ for r,s ϵ C_c then S_{fr} is increased before S_{fs}. (ie. lower priority classes are blocked on admission into the computer system before higher priority classes in order to satisfy given multiprogramming constraints).

4. The Blocking Algorithm

We assume the queue f is an infinite server device so that following [6]:

$$W_{fk} = S_{fk} \; v_{fk} \text{ for all } k = 1, \ldots, K \qquad (2)$$

Initially $S_{fk} = 0$ for all k = 1, . . . ,K. Then by global application of Little's Theorem [7] :

$$T_{mk} = N_k \, v_{mk} / (\sum_{i \neq f} W_{ik} \, v_{ik} + S_{2k} \, v_{2k}) \qquad (3)$$

By local application of Little's Theorem and (1) and (3) above:

$$\sum_{k \epsilon C_c} \sum_{m > f} Q_{mr} = \sum_{k \epsilon C_c} \sum_{m > f} (N_k v_{mk} W_{mk} / (\sum_{i \neq f} W_{ik} v_{ik} + S_{2k} v_{2k})) \qquad (4)$$

The following algorithm is then applied to evaluate the S_{fk}:

1 $S_{fk} = 0$ for all k = 1, . . . ,K
2 Compute W_{mk} for all m = 1, . . . ,M and k = 1, . . . K
3 Sort constraints so that if C_{c_1} is a subset of C_{c_2} then

constraint c_1 is processed before constraint c_2
4 For all constraints c
For all priorities Gq of classes $q \epsilon C_c$
until $H_c^* \leq P_c$

$S^1 = 0$
$H_c^{*1} = 0$
Do until $H_c^* \leq P_c$ or $H_c^* = H_c^{*1}$

$$H_r^c = \sum_{m > f} N_k v_{mk} W_{mk} / (\sum_{i \neq f} W_{ik} v_{ik} + S_{2k} v_{2k})$$

for all $k \epsilon C_c$
$H_c^* = \sum_{k \epsilon C_c} H_k^c$
If $S^1 = 0$
$S^1 = \sum_{r \epsilon C_c} H_k^c \, (\sum_{m > f} W_{mk} W_{,mk} v_{mk})$
$G_k = G_q$
else
$S^1 = S^1 (H_c^* / P_c)$
Do for all $k \epsilon C_c$ such that $G_k = G_q$
If $S_{2k} < S^1$
$S_{2k} = S^1$
Doend
Doend
Doend
Doend

5. Conclusions

Execution of the above algorithm during solution of the model is iterative. Initially, no admission delays by class are assumed. If a feasible solution does not result then delay service times of an infinite server device are calculated so as to keep tasks outside the system. This continues until a feasible result is obtained. The result is not only intuitively understandable — a delay queue prior to admission but also exact in queueing network solution terms. The technique has been widely used [4] and can be extended to model blocking and adaptive routing in data communications networks.

References

[1] ACM Computing Surveys. 10, No. 3, 1978.
[2] SMITH C., and BROWNE J.C., Performance Specifications and Analysis of Software Designs, Conf. on Simulation, Measurement and Modelling of Computer Systems, 1979.
[3] BUZEN J.P., et al., Predicting Software Performance with Crystal, Third Annual Int. Conf. on Computer Capacity Management, 1981.
[4] WULF S., Performance Prediction During Database System Development, Ph.D. Dissertation, UNISA, 1982.
[5] COURTOIS P.J., Decomposability, Instabilities and Saturation in Multiprogramming Systems, Comm. ACM 18, No. 7, 1975.
[6] REISER M., and LAVENBERG S.S., Mean Value Analysis of Closed Multichain Queueing Networks, IBM Research Report, RC7023, 1978.

An Analytical Model of a Mixed-Workload MVS Computer System

R.J. Mann
Comcon (Pty) Limited, Johannesburg

Abstract
A basic description of the facilities of the computer modelling package, AUTO-CONFIGURATER, is presented together with detailed procedures to calculate the required input parameters from monitor statistics. Techniques to calibrate the model are described together with an investigation into how the modelling package enables configuration and tuning alternatives to be evaluated.

1. The need for capacity planning

New computer applications are being developed in greater numbers than ever before. In addition, the workload growth for existing systems is constantly expanding.

In order to provide adequate hardware and software to accommodate demand, the capacity planner has to forecast performance bottlenecks and capacity shortfalls and evaluate alternative scenarios to address capacity problems by investigating various configuration and workload possibilities, using "what if" questions. In this way the planner can predict such performance indicators as response time, system throughput rates and device utilisations.

The AUTO-CONFIGURATER system (AUT83) described here provides a cheap and easy-to-use tool for these activities.

2. Auto-Configurater Theoretical background

Queueing network models have been successfully used for modelling computer systems and computer communications systems for some years, (eg. BUZ73).

The Mean Value Analysis (MVA) algorithm of Reiser and Lavenberg (REI78) avoids the problem of overflow of earlier techniques and presents an intuitively simple interpretation of flow through the system. It requires an excessive amount of main memory for all but the smallest models.

AUTO-CONFIGURATER uses A CONVERGENCE algorithm together with heuristics to handle a number of specific conditions found in computer systems including paging, blocking, class priorities, missed disk rotations and disk/channel overlap. (WUL82) This algorithm requires very little memory for solution.

3. Creating the MVS System Profile

3.1 Defining the time period

The analyst must inspect all periods of operation of the computer system to determine the appropriate interval. In macro terms he must evaluate the annual calendar to identify year-end, quarterly or monthly peaks. Thereafter, he must analyse each 24 hour period to determine basic modes of operation. For example, the night shift may be exclusively batch processing whereas the day shift consists of online processing. Based on this analysis of workload types and peaks, the analyst must select the appropriate time period which the model must cover which will generally be the peak period of the shift.

3.2 Sources of statistics

The basic starting lines used to build the model are derived from RMF. These are supplemented by SMF and IMS (or other DB/DC) statistics.

3.3 Defining the auto configurater model

The model should be defined with well in excess of the actual numbers of servers, classes, disk types and limits, (to allow further expansion of the basic system profile).

The following subsections define elements in building a model. The model topology is presented in FIGURE 1 and model dialogue is presented in FIGURE 2.

3.3.1 Model class definitions

The workload which runs in the machine can be classified into a number of different classes. The basic criteria which should be used for class definition are the use of common software or transactions of common duration. Examples are TSO, Batch, IMS short transactions and IMS long transactions.

It should always be borne in mind that the effect involved in model building increases with the number of classes defined. The analyst must evaluate the trade-off of increased accuracy versus increased effort in defining the model.

Each defined class is assigned a numeric priority value. (The higher the priority, the larger the priority value assigned). These class priorities are similar to operating system priorities assigned to jobs of these classes and we used *classes* in admitting tasks to the computer system or in scheduling the CPU.

3.3.2 Defining limits

The LIMIT command defines how tasks of the various classes are to be admitted into the computer system. Each limit command defines a limit on the number of tasks overall which may be admitted for the specified classes.

3.3.3 Defining disk types

The DISK command is entered for each disk type in the system (eg. 3350 or 3380). Default seek, rotation and transfer times are specified in each command. Unless these values are overridden for any particular disk server, they are used unchanged for that server. Supplier averages should be used.

3.3.4 Disk service times

The RMF report lists average service time (AST) for each disk drive.

Note that:

AST = seek + latency + missed rotations + transfer

latency = R/2 where R = rotation

$$\text{missed rotations} = \frac{UR}{1-U}$$

where U = utilisation of channel
and transfer = the default for the device type

3.3.5 Disk channels

Each logical channel is modelled as a channel using mean service time and utilisations of all physical channels which comprise the logical channel.

3.3.6 Calculation of class-dependent IOs and CPU service time

CPU service time and server IOs are specified by class. This means that total CPU seconds and total IOs at each device must be apportioned to each class. Ideally, this should be, as they are actually incurred during the processing of tasks of each class. In practice, this can never be done exactly. The degree of accuracy of these parameters is dependent on the amount of effort by the analyst to calculate them.

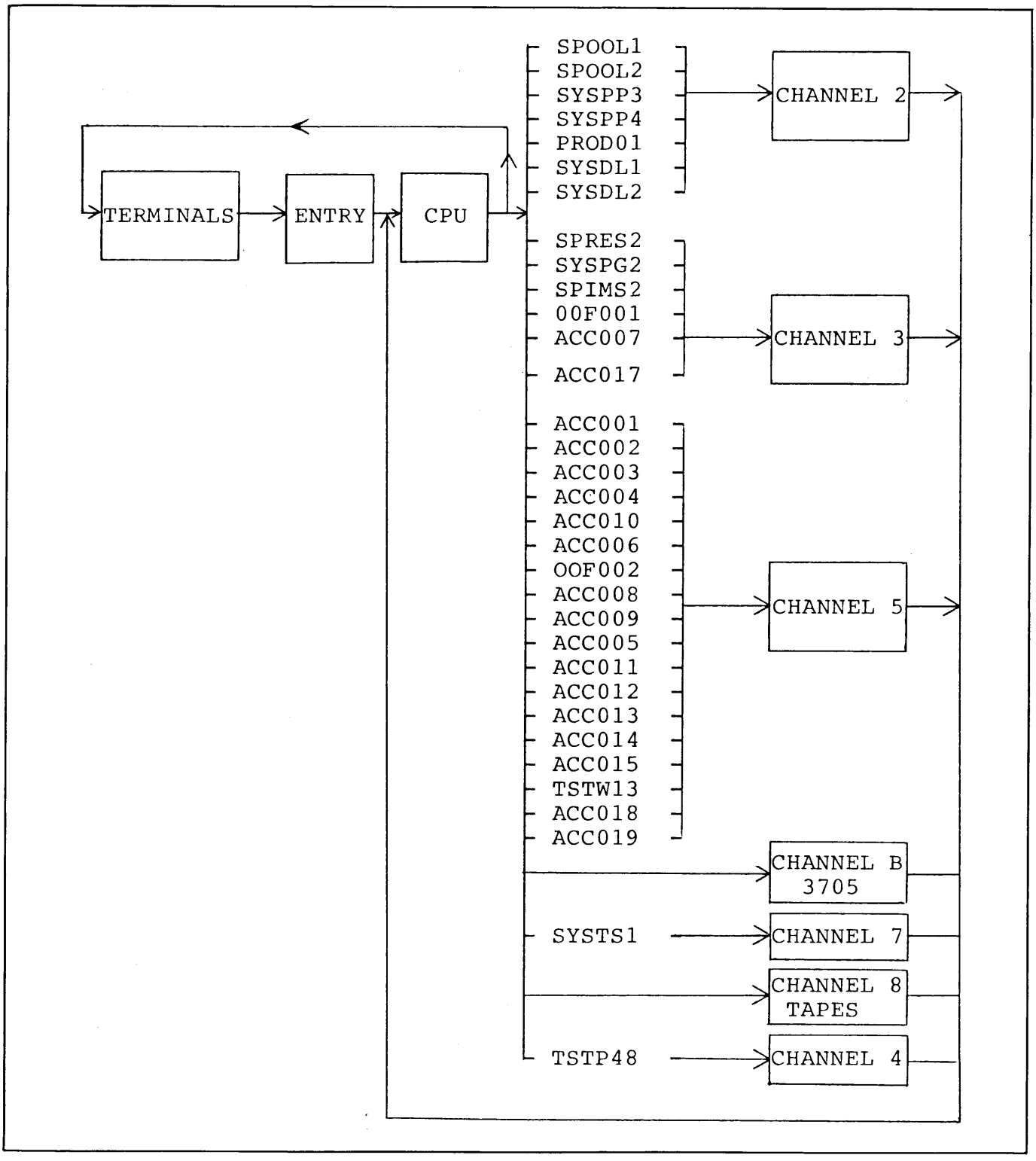


FIGURE 1 — QNA model topology.

A. The Detailed Method Using Accounting Statistics

Accounting statistics are machine statistics showing resource consumption by job or transaction. They include SMF statistics as well as DB/DC statistics. Usually, the SMF statistics (CPU seconds and IOs per device) are not complete since many operating system attributable components are not recorded by SMF.

The total CPU seconds and IOs recorded by SMF will be less than the consumption recorded by RMF. IBM has defined capture ratios for various workload types to apply to the SMF statistics in order to bring them into line with RMF data.

The SMF totals are used as the basis for allocating the more accurate RMF statistics.

RMF provides total CPU seconds used and IOs by device in the sampling period. The analyst must use the ratio of the usage for a particular class to the total usage as recorded by SMF as the basis for allocating RMF data.

B. An Approximate Technique Using RMF Only

The RMF Performance Group report contains summaries of IO, CPU and SRB usage by performance group. We assume using this technique that each class consists of one or more performance group and that no performance group is used by two or more classes.

Service definition coefficients for each of CPU, SRB and IOC (IOC refers to device IOs and CPU and SRB together constitute CPU service) are defined on each Performance Group report at the top of each page. These are used to calculate total CPU

```
NEW = BASE01,NOCL = 40,NSER = 100,NLIM = 10,NDSK = 3                    - (1)
MEMORY = 2862,TIME = 28800,A = 4.585,K = 2,MIPS = 4200000               - (2)
DISK = 3330,SEEK = 0,03,ROT = 0.0167,TFR = 0.004                        - (3)
DISK = 3350,SEEK = 0.025,ROT = 0,0167,TFR = 0.004
CLASS = 1,NAME = ONLINE,NO = 0.99,PR = 2                                - (4)
CLASS = 2,NAME = BATCH,NO = 0.07,PR = 1
CLASS = 3,NAME = ENVIRO,NO = 1.2,PR = 3
LIMIT = 1,VAL = 19                                                      - (5)
LIMIT = 2,VAL = 10,SI2 = NO,SI3 = NO
LIMIT = 3,VAL = 3,SI1 = NO,SI3 = NO
SERVER = TERMINAL,COST = 754,LOC = OUT,IO1 = 492000,IO2 = 1,IO3 = 1     - (6)
SERVER = CPU,COST = 253,ST1 = 0.0046,ST2 = 0.0042,ST3 = 0.0034
SERVER = CHAN2,COST = 0.4,S = 0.002
SERVER = CHAN3,COST = 0.4,S = 0.005
SERVER = CHAN4,COST = 0.4,S = 0.010
SERVER = CHAN5,COST = 0.4,S = 0.002
SERVER = CHAN7,COST = 0.4,S = 0.020
SERVER = CHAN8,COST = 0.4,S = 0.001,IO1 = 68134,IO2 = 5925,IO3 = 74059
SERVER = CHANB,COST = .4,S = .001,IO1 = 351216
SERVER = SPOOL1,COST = 2.2,CHAN = CHAN2,DISK = 3350,IO1 = 10457,IO2 = 742,
  IO3 = 11425,
SEEK = 0.005
SERVER = SPOOL2,COST = 2.2,CHAN = CHAN2,DISK = 3350,IO1 = 514,IO2 = 36,IO3 = 562,
SEEK = 0.0001
SERVER = SYSPP5,COST = 2.2,CHAN = CHAN2,DISK = 3350,IO1 = 618,IO2 = 44
  IO3 = 676,SEEK = 0.048
PAGE,DEV = SPRES2,DEV = SYSPG2,DEV = SPIMS2                             - (7)
SPLIT = SPRES2,FRAC = 0.531                                             - (8)
SPLIT = SYSPG2,FRAC = 0.227
SPLIT = SPIMS2,FRAC = 0.242
SOLVE                                                                   - (9)
STATISTICS = TEST,TYPE = ALL                                            - (10)
SERVER = PROD01,IO1 = 72616,IO2 = 5153,IO3 = 79340                      - (11)
SERVER = OOF001,IO1 = 74459,IO2 = 5284,IO3 = 81354
SOLVE                                                                   - (12)
STATISTICS = REMOVE DISKLOAD,TYPE = ALL                                 - (13)
SAVE = MODEL                                                            - (14)
END
```

- (1)	define new model	- (6)	define servers	- (11)	change server data
- (2)	define memory available	- (7)	define paging devices	- (12)	solve model
- (3)	define disk types	- (8)	define allocation of paging to devices	- (13)	print all statistics
- (4)	define classes	- (9)	solve model	- (14)	save model on library
- (5)	define blocking limits	- (10)	print all statistics		

FIGURE 2 — Sample Model Dialogue.

and IO usage ratios by class. The simplifying assumption is then made that all classes will use all devices in the same ratio as calculated here. These ratios are applied to the device IO totals and total CPU seconds presented by RMF.

3.3.7 Paging

The fraction of paging to any one page device is the total paging IOs to that device divided by the total paging IOs to all devices. The SPLIT command is used to define these calculated fractions.

3.3.8 Number of tasks per class

The basic principle used is that the number of tasks in each class must be adjusted until the total number of transactions of the model approximate those measured in the statistics.

Let M_i = no of tasks in class i
E = total elapsed time
R_i = response time of class i
N_i = total number of transactions for class i actually measured

Then for all classes:
$M_i = N_i R_i / E$

The model assumes a starting value for R_i and then automatically iterates until convergent values of M_i are found.

4. Calibration and Prediction Modelling

We have compared measured performance values with the performance values calculated by the model. Using the principle established by (BUZ78) that if reasonable agreement is observed under a series of conditions the model can be considered capable of predicting system performance, under related alternative conditions.

Our results are presented in FIGURE 3. As a presentation method we use the cost utilisation histogram (BOR77) together with three performance measures, F, B and P.

The F value is defined to be the resource utilisation indicator. The value B is defined as a cost-imbalance factor which ranges from 0 for a perfectly balanced system to 1 for a system totally out of balance and the P factor is a measure of system balance unweighted by cost and also varies between 0 and 1.

By examination of the device utilisations the bottleneck device was identified to be disk PROD01. We can eliminate this bottleneck by splitting a reasonable proportion of the disk I/O's to a low utilised device, in this case disk 00F001. We also modelled the effects of a CPU upgrade, moving from an IBM 3033 to a 3081XA. A summary of results is given in FIGURE 4 which shows both individual and cumulative effects on performance measures.

Utilisations

Device	RMF Actual	Model	% Error
CPU	.267	.272	1,87
CHAN2	.028	.027	3,57
CHAN3	.023	.024	4,35
CHAN4	.0	.0	—
CHAN5	.072	.076	5,56
CHAN7	.0	.0	—
CHAN8	.001	.006	*
CHANB	.0	.008	—
SPOOL1	.017	.016	5,88
SPOOL2	.001	.001	0
SYSPP3	.0	.001	—
SYSPP4	.0	.0	—
SYSPP5	.0	.003	—
PROD01	.349	.349	0
SYSDL1	.0	.0	—
SYSDL2	.0	.0	—
SPRES2	.001	.005	*
SYSPG2	.001	.003	*
SPIMS2	.0	.001	—
OOF001	.0	.002	—
ACC007	.082	.081	1,22
ACC017	.013	.011	15,39
TSTP48	.0	.0	—
ACC001	.047	.051	8,51
ACC002	.063	.082	30,16
ACC003	.0	.001	—
ACC004	.109	.122	11,93
ACC010	.036	.039	8,33
ACC006	.094	.104	10,64
OOF002	.0	.002	—
ACC008	.045	.049	8,89
ACC009	.038	.041	7,89
ACC005	.105	.118	12,38
ACC011	.024	.027	12,50
ACC012	.105	.116	10,48
ACC013	.105	.118	12,38
ACC014	.027	.029	7,41
ACC015	.011	.012	9,09
TSTW13	.0	.0	—
ACC018	.046	.051	10,87
ACC019	.103	.116	12,62
SYSTS1	.0	.0	—
PAGE RATE/ SEC	.28	.271	3,21

*High Relative Error in cases of low utilisation due to rounding error.

FIGURE 3 — Model calibration.

References

[1] AUT83 AUTO-CONFIGURATER Reference Manual, Comcon International, 1983.

[2] BOR77 I. Borovits and P. Ein-Dor, Cost/Utilisation: A Measure Of System Performance. Comm. ACM 20, No. 3, 1977.

[3] BUZ73 J. P. Buzen, Computational Algorithm for Closed Queueing Networks with Operational Servers. Comm. ACM 16, No. 9, 1973.

[4] BUZ78 J. P. Buzen, A Queueing Network Model of MVS. ACM Computing Surveys 10, No. 3, 1978.

[5] REI78 M. Reiser and S. S. Lavenberg, Mean Value Analysis of User Multichain Queueing Networks. IBM Research Report RC7023, 1978.

[6] WUL82 S. Wulf, Performance Prediction During Database System Development. Ph.D. Thesis, University of South Africa, 1982.

	Utilisations CPU	PROD01	00F001	System Throughput	% Improvement
Base Model	0.27	0.349	0.002	0.0000344	—
Scenario 1	0.29	0.185	0.099	0.0000369	7,27
Scenario 2	0.16	0.375	0.002	0.0000361	4,94
Scenario 1 and 2	0.17	0.201	0.108	0.0000383	11,34

FIGURE 4 — A summary of results

Add: The Automated Database Design Tool

S. Berman
Department of Computer Science, University of Cape Town, South Africa

Abstract
An automated database design tool called ADD has been developed at the University of Cape Town. This system obtains a requirements specification from a user and from this generates a relation scheme and a Codasyl schema. This is intended to be a prototype database. The specification is submitted using SDM, the Semantic Database Model. Semantic information in this model is included in the schema as integrity constraints.

1. Introduction

Database design is a difficult task which continues to present problems to theoreticians and practitioners alike. Currently no universally-accepted database design methodology exists, and the greatest part of database research is directed towards solving this problem. Some useful results have begun to emerge, such as normal form theory for relational database design. In the realm of network databases, there are three aids: data models, prototype databases and automated design tools. The ADD system is a database design tool which incorporates all these three facilities. The ADD design methodology is based on the SDM data model, and involves automated conversion of such a model to a prototype network database. After discussing the reasons for developing ADD, the SDM model is briefly described. Thereafter, the user interface, relation scheme design and network synthesis algorithms are outlined. In conclusion, some of the results of implementing this system are presented.

2. Motivation

The difficulties experienced in database design arise through inadequacy on the part of both users and designers. Frequently the user does not fully understand his data, cannot describe it accurately and completely, and cannot distinguish irrelevancies. Furthermore, there is generally a communication gap between user and database designer, and hence the latter can misinterpret the requirements. A data model facilitates recognising and understanding data relationships, and provides a fairly natural way of describing requirements in an unambiguous manner.

Even if the specifiation is correct and complete, database design is still a difficult task, because there are so many options facing the designer when deciding both logical and physical structure. Poor designs result if the structure is arrived at by examining processing requirements, rather than the fundamental properties of the data itself. The ADD system was designed to alleviate this by automatically generating a prototype database. Its logical structure should require minimal, if any, alteration when the final schema is decided. The synthesised network is based soley on a description of the data itself, and not on envisaged transactions. The system generates several listings showing the design decisions taken, which should assist the designer of the final operational database.

3. SDM — The Semantic Database Model

SDM is a data model developed by Hammer and McLeod, which first appeared in the literature in 1981 [1]. In such a model, the real world is described by means of classes and their attributes. A class represents an entity type; that is, an object of interest in the environment. Its attributes are the properties that such entities possess.

Every attribute has an associated valueclass. This specifies what type of value(s) it assumes, and can either be STRINGS, i.e. printable values, or some SDM class. For example, SHIPS attribute Type would have a valueclass STRINGS, e.g. 'merchant' or 'fishing', Captain would have valueclass OFFICERS, where OFFICERS is some class in the model. From this it can be seen that entities represent themselves, they are not represented by means of keys as in the relational model and others. This simplifies the model and makes it easier for non-DP persons to understand.

Relationships between entities can be represented either as classes, or as properties of the objects involved. For example, the relationship between a ship and an officer can be a class ASSIGNMENTS with attributes Ship-Assigned and Officer-Assigned. Alternatively, the class SHIP can have attribute Captain (valueclass OFFICERS) and/or the class OFFICERS can have attribute Ship-On (valueclass SHIPS). When a relationship is represented as a class, attributes of the relationship can be specified (example Assignment-Date).

The concept of generalisation [2] or subtyping is incorporated. A subclass of C defines a subtype or "special" type of C. As an example, TANKERS is a subclass of SHIPS. A subclass definition includes a condition which objects of the superclass must satisfy in order to belong to that subclass. Subclasses inherit all attributes of their superclass in addition to their own attributes. Subclasses of STRINGS, which is "system-defined", can be given to describe permissable values for attributes. This enables INTEGERS, DATES, PERSON-NAMES, etc to be distinguished from arbitrary character strings.

Grouping classes are used to describe classes comprising groups of like entities. CONVOYS would be a grouping of SHIPS. Each member of CONVOYS consists of not one but several SHIPS entities. The definition of a grouping class can stipulate how the groups are to be formed. Thus grouping class SHIPTYPE-GROUPS can be specified as a "Grouping of SHIPS on common value of Type". This causes all "fishing" vessels to constitute one grouping, etc. Classes which are neither subclasses nor groupings are called base. Each of these has one or more keys specified for it.

The semantic expressiveness of SDM can be seen when considering attribute descriptions. Each attribute must have a valueclass specified and must be designated either "member" or "class". A member attribute is applicable to individual entities in that class (e.g Type, Size, Hullnumber); the class attributes describe the class as a whole (e.g. Number-of-Ships, Average-Ships-Size). Where applicable, attributes can be described by properties such as "multivalued" (for repeating attributes), unchangeable (e.g. Birthdate) and cannot be null, amongst others.

Derived data can be included if their derivation can be explicitly defined in terms of other information in the model. Several attribute derivation primitives exist for this purpose. Some of these are: average, minimum sum, maximum, arithmetic expression, same as, subvalue, intersection, union and set difference. Subvalue enables a subset of a multivalued (repeating) attribute to be defined. For example, Last-Two-Inspections can be specified as subvalue of Ship-Inspections, where Order-for-Ship is less than 3. Intersection, union and set

difference can be applied to two multivalued attributes in a similar way.

SDM was chosen for the ADD system after studying a wide selection of existing data models. The main reason for this choice is that SDM is highly semantically expressive, providing many modelling tools, yet is sufficiently simple for a non-DP person to understand. In other models, a trade-of between expressiveness and simplicity is evident. Furthermore, SDM embodies only the meaning and nature of the data, and does not include any description of processing requirements. SDM could perhaps be criticised for not including functional or multivalued dependencies. This was not considered a disadvantage as these concepts are difficult for computer-naive people to understand and recognise in their environment. It was considered essential that the user should be able to specify the model himself.

4. The User Interface

The task of obtaining an SDM specification from a non-DP person is indeed an awesome one. The policy adopted was that of gradually introducing each SDM concept to the user in turn. Thus the novice is first asked for classes, then attributes. Subclasses and grouping classes are only introduced subsequently; and attribute derivations are considered in the final stages.

A menu-driven system was chosen as the safest approach, with large menus being split into several smaller ones. The top-level menu offers a choice between data entry, session termination, help, perusal and model editing. All data entry, i.e. of classes, attributes, etc, is handled in the following manner:

The user can submit a description of his organisation and its objects using natural language. However, only words in capital letters are recognised by the system. This is a slight modification of the data capture method used by Palme[3], with capitals instead of special symbols. They are easier to type, and a word that starts in lower case but ends in upper case is accepted. In this way a beginner can submit the following when asked for classes: "the univerSITY has DEPARTMENTS divided into courSES given by LECTURERS these are taken by STUDENTS for CREDITS" . . . The experienced user would simply give "UNIVERSITY DEPARTMENTS COURSES LECTURERS STUDENTS CREDITS".

When users have given all classes (or all attributes, etc), they respond "QUIT". They are then asked to supply details about the new items, so that a complete SDM model results. Since it can be dangerous to interrupt his train of thought[4], this process can be terminated prematurely by the user. It can be ommitted entirely, if he wishes, by submitting say "QUIT S", instead of "QUIT". This causes detail specification to be preempted and subclasses ("S") can immediately be entered instead. This mode of operation can be employed by experienced users to type-ahead, rather than be presented with the menu each time.

Perusal enables a user to review the current version of his model. This is helpful if he has forgotten his earlier work, or wishes to check that he is proceeding correctly. He can edit any item description by naming the item and then replacing errors by new values. An extremely simple method of editing is used, involving menu-selection. This is preferable to a special editing language, which is generally confusing to a novice.

Help can be requested at any stage by responding "?". This causes the program's most recent display to be expanded upon, giving greater detail on what is required. For this reason, all initial displays are brief, to avoid irritating the experienced user. If too many successive help requests are received, the user is referred to his User Manual.

When the model is completed, an optional component of the system may be executed. This allows functional dependencies to be specified and can be helpful if the user is capable of working with these, or is guided by the database designer.

5. Relation Scheme Design

The SDM model is converted into a relation scheme in the following manner: In general, all the attributes of a class constitute one relation describing that class. Exceptions occur in the following cases: Each candidate key for the class is removed to form a separate relation, and is replaced by an artificial key. Thus, for example, if Name is the key for OFFICERS, it is replaced by OFFICERS≠ (with domain INTEGERS), and another relation (OFFICERS≠, Name) is created. Not only can this save space, but it also ensures that the relation is in 2nd Normal Form, and that embedded dependencies between prime attributes are removed.

Multivalued (repeating) attributes also contribute separate relations, e.g. (SHIPS≠, Crew), so that the relation scheme is in 1st Normal Form, and data aggregates are replaced by their subfield components. A separate relation is used to store all class attributes, since these do not have multiplicity. If dependencies were specified in the model, the attributes involved form a new relation and the dependent ones are removed from the original. Similarly, any attribute derived from single-valued attribute(s) is also separated out. In this way, a model for which functional dependencies were specified is translated into a Boyce-Codd Normal Form scheme. Otherwise, a 2nd Normal Form schema results.

There are other types of relations in the relation scheme generated. An additional relation is created for each grouping class. As an example, relation (CONVOY≠,CONTENTS) enables several SHIPS to be stored "opposite" a CONVOY number. To handle subclasses, relations of the form C≠, CTYPE are generated, for all base classes C on which subclasses were defined. In this way a TANKER (subclass of SHIPS) would have its key in at least three relations: that for SHIPS, that for TANKERS and that giving (SHIPS≠,SHIPSTYPE). This is more efficient than duplicating SHIPS attributes in the TANKERS relation, and makes the relational database easier to use.

6. Network Schema Synthesis

The construction of a DMS 1100 Codasyl schema[5] is implemented as a 3-phase process. The first translates the relation scheme into a network structure; the next uses the SDM model to refine this and introduce integrity constraints. In the final phase, physical parameters are chosen and a complete schema created.

The first phase converts each relation in tern into records, sets and items. A record type is formed for each class, comprising the attributes in the relation describing that class. Repeating attributes constitutes a separate record type, connected to the class they characterise as member of a set. A functional dependency also forms a new record type, and is made the owner of a set linking it to the class it describes.

Relationships translate into sets, with their one:one/one:many/many:many nature distinguishing the owner from the member. A confluent hierarchy is used for the many:many case. The relationships in the network are derived from those attributes whose valueclass (domain) is not STRINGS, but indicates another class in the model. Thus throughout the network generation module, before an attribute is placed in a record, its domain is inspected. If this implies a class, a set is created instead of a field. (A class has "classname" as column name and "integer" as domain; while an attribute is represented by "attributename" and valueclass, respectively.) Thus when handling attributes, where there is no "≠" in the column name, the first step is to examine the domain for a "≠". If none exists, the attribute becomes an item in some record. If one is found, the "≠" is removed and what remains identifies the valueclass of the attribute. In such a case set(s) are required. For example, if "CAPTAIN, OFFICERS≠" is encountered in the SHIPS relation, then a set OFFICER→SHIPS is created. "Loops" are detected and replaced by two sets, in the manner advocated by Date[6]. This is necessary because the same record cannot be both owner and member record type in a DMS 1100 set.

Grouping classes are generally implemented as a set with the grouping class as owner and grouped class as member (example SHIPTYPE-GROUPS→SHIPS). Thus each such set occurrence represents one group. Subclasses are implemented in the following way.

If TANKERS is a subclass of SHIPS, then in addition to their own attributes, TANKERS have all the attributes of ordinary SHIPS. As most SHIPS will not be TANKERS, it is inefficient to declare a SHIPS record to contain both SHIP and TANKER attributes, as the latter will be null in most instances. It is also dangerous to treat SHIP and TANKERS as completely separate entities, because then if user asks for all SHIPS in the database, he must know to list all TANKERS too; also, all relationships in which SHIPS participate would have to be duplicated for the TANKERS record type. Thus separate record types are used, but they are connected by a set, such as SHIPS→TANKERS.

By adding flags to the superclass record, the following can be verified by means of appropriate CHECK and RESULT clauses:

1. A TANKER can only exist in the database if it is linked to 1 and only 1 SHIPS occurrence.
2. A SHIPS record cannot have more than on TANKER record linked to it.
3. A TANKER can only be linked to a SHIP if that SHIP meets the criteria set down in the TANKERS definition.

Verifying the latter, as well as attribute derivations, provided the greatest complexity in ADD. The problem lies essentially in the fact that in SDM one can specify relationships between two semantically distant attributes. This is beyond the capabilities of the DMS 1100 integrity facilities, since CHECK and RESULT clauses can only handle items in the same record or set.

In ADD, integrity constraints are incorporated wherever possible, to verify subclasses and derived attributes. Certain derivations are also examined in order to simplify the database structure. As an example, if attribute A is the union of B and C, this can mean replacing six sets by two.

In the final phase of schema generation the simplest physical structure is chosen for the database. This should be adequate for the prototype, but would have to be altered when the final schema is chosen. The only complexity that can arise occurs with multi-level paths in a SET SELECTION clause. ADD determines whether these are necessary, and chooses USING and ALIAS items along the path.

7. Conclusion

The ADD system automatically produces a schema which can be used as a prototype database. Its strength lies in its choice of a logical structure, which should require minimal, if any, alterations. The wealth of information in the SDM model is used to define records, sets and items in the best possible way. The characteristics of attributes, interclass connections and attribute derivations which enhance one's understanding of the organisation, are utilised to improve the network. As the logical structure is based solely on a description of the data, and not on functional requirements, it should not need to be re-organised when processing requirements change. Integrity constraints are included where permitted by DMS 1100. Add also relieves the human designer of much of the tedium associated with the specification of physical criteria. For example, sort keys are designated where appropriate, ALIAS items are declared and singular sets chosen as VIA sets wherever these exist.

The ADD system has shown clearly that considerable advantages exist when a data model forms the basis of design. Its implementation has high-lighted the fact that a data model should not provide features beyond the scope of conventional Database management Systems. A modified version of SDM has accordingly been designed.

The other major contribution of ADD lies in the fact that it enables a computer user to create an SDM model. Experimentation with novices showed that after some inital difficulty, a user can indeed construct a model with little, if any, assistance. Problems encountered in the beginning arise largely through misinterpretation of ADD terminology. Hence if this is discussed with the database designer before commencing, a user of average inteligence should be able to create his model relatively easily and quickly.

Thus ADD has shown that it is possible to develop a database design methodology, which is partially automated and directly involves the user in requirements specification. The entire design process cannot be automated; the analytical powers, intuition and creativity of the human mind are essential. All that can be done is to automate as much of the design process as is reasonable, to alleviate the task of human(s) involved. The ADD system is a step in this direction.

References

[1.] Hammer, M and McLeod, D. "Database Description with SDM; a Semantic Database Model". ACM Transactions on Database Systems, Vol. 6, No. 3 1981, pp 351-386.

[2.] Smith, J M and Smith, D C P. "Database Abstraction: Aggregation and Generalization". ACM Transactions on Database Systems, Vol. 20, No. 6, 1977, pp 105-133.

[3.] Palme, J. "A Human-Computer Interface for Non-Computer Specialists". Software Prac. and Exper., Vol. 9, 1979, pp 9-19.

[4.] Martin, J. "Design of Man-Computer Dialogue". Prentice Hall, Englewood Cliffs, NJ, 1973.

[5.] Sperry Univac. "Data Management System (DMS 1100)". UP7909, Rev 3A, St Paul, Minn., 1972.

[6.] Date, C J. "An Introduction to Database Systems". 3rd Ed., Addison Wesley, 1981.

A Disk Space Management System

A.J. Cuthbertson
University of the Witwatersrand Computer Centre

I.J. van Niekerk
University of the Witwatersrand Computer Centre

T. Turton
IBM South Africa

Abstract
This paper describes the research that was undertaken in designing a disk space management system. The system has strong similarities to the management of real storage in a virtual storage environment. These similarities were exploited in the design and evaluation of algorithms for implementation of the system. Research was based on modelling the system with historical data during the design phase. Research was conducted in three phases. Firstly, in determining actual historical disk space requirements quantitatively. Secondly, in modelling the proposed system with historical data to verify an observation that most users use the computer intermittently. Thirdly, in modelling the proposed system with historical data to evaluate alternative disk management policies. The system was implemented and actual performance monitored against research results to validate conclusions.

INTRODUCTION

The IBM VM/SP operating system was installed at the Computer Centre of the University of the Witwatersrand in late 1981 as the main vehicle for educational and research computing. The Centre provides a computing service to the campus and some outside users. It supports batch and online processing. All staff members and students may potentially make use of the Centre's facilities. At present, some 3 000 users have been registered to do so.

A characteristic of VM/SP is that there is a fixed minimum amount of online disk space that can be allocated to an individual user. With the standard system each user has a 'minidisk', which is a subdivision of a real disk, that is assigned in increments of 450k bytes. Providing such space to all users was regarded as uneconomical as most are under-graduate students and would require less than the minimum of 450k bytes of permanent storage.

A further consideration was that while a user is logged onto the system, there will be a requirement for an amount of workspace on disk to cater for compiler work areas, program listings, and similar purposes.

Initially, a disk management system was set up where groups of users shared a single large permanent minidisk. A group would typically consist of all the users in a particular department. In addition, when logging on, a user was allocated 450k bytes of disk for tempory work space. This was automatically reclaimed when the session was completed and it was the users responsibility to transfer any files to be retained to the permanent disk before logging off. This system did not work successfully as it impacted the functional capabilities of the operating system. Owing to this an alternative approach was sought.

During the first year of operation of the VM/SP system the Computer Centre staff investigated the disk space usage patterns and requirements of its users. It was observed that users tend to make intermittent use of the computer. There would be bursts of activity lasting days or weeks followed by weeks or months of inactivity.

Based on these findings, a proposal for disk space management was made. This was that users should be allocated a minidisk by the computer, only on demand. It would be large enough to contain both temporary and permanent files, so no distinction need be drawn between them. The minidisk would remain allocated to a user only as long as it was used. If the user became inactive for a period of time, the computer system would automatically archive all those disk files to magnetic tape and 'reclaim' the disk space. Reclaimed mini-disks would be returned to a pool, to be allocated to the next user requiring space. A catalog of files archived to tape would be maintained for each user, and made available by the computer on request. The user would then be able to migrate selected files back from tape to a newly allocated online minidisk.

RESEARCH AND DESIGN CRITERIA

User Space Requirements

The first phase of research was to examine the total amount of space used by each user under the original shared mini-disk system. Each departmental minidisk was examined and the space occupied by individual users was determined. The figures were used to calculate a distribution of users against amount of disk space used in 100k byte increments. The results of this analysis are indicated Figure 1.

Over 90 percent of the registered users were utilising less than 200k bytes of permanent disk space. This would mean that in the proposed system a minimum sized minidisk would satisfy almost all the users' space requirements.

Proposed Objectives for Disk Management

The concept of reclaiming inactive user's minidisk and making it available to a more active user has certain similarities to the principles of virtual storage management. Reclaiming the disk of the user who had been inactive the longest, would constitute a Least Recently Used or LRU reclaim policy, which Belady has shown to be a good strategy 1. On these grounds the following objectives were established.

— To minimise the number of minidisks in the pool.
— To provide an acceptable time that a user could be absent without the minidisk being reclaimed. That is, to achieve an acceptable 'Retention Period'.
— To minimise inconvenience to the user.
— To avoid a reduction in the functional capabilities of the standard VM/SP operating system.

Modelling a Least Recently Used Policy

The VM/SP operating system keeps accounting records of user

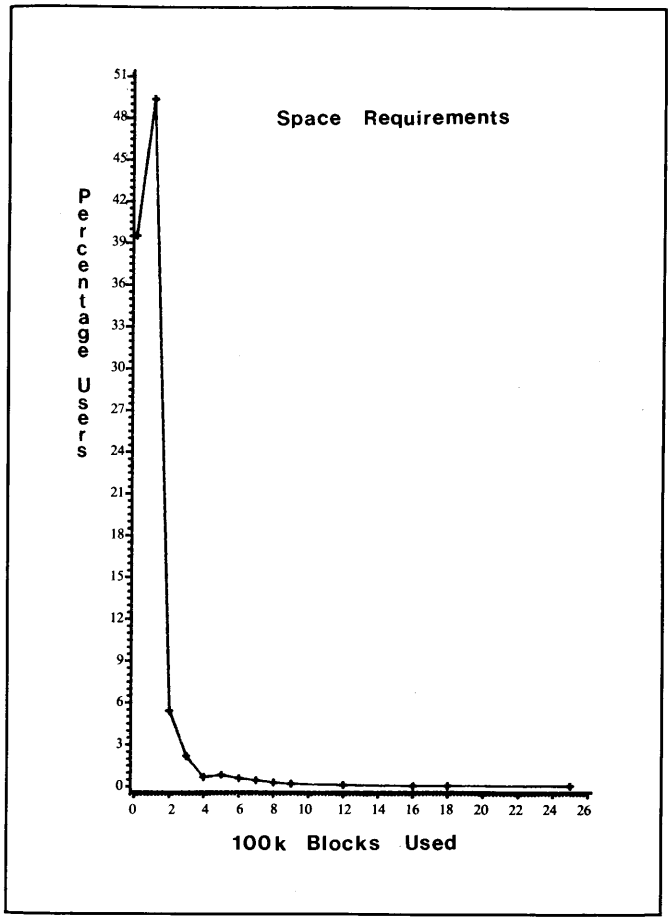


FIGURE 1

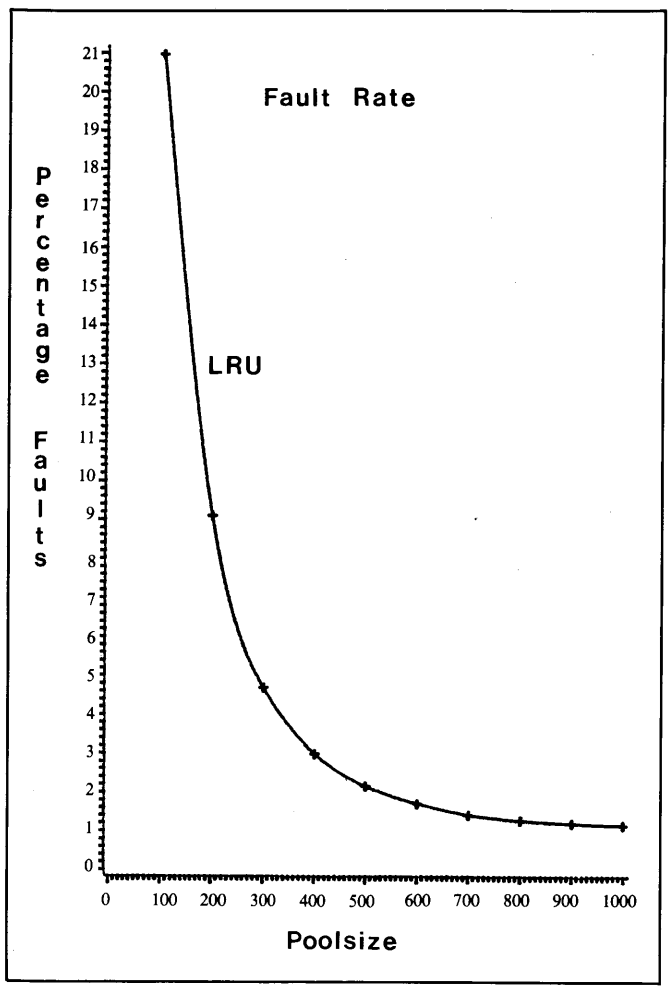


FIGURE 2

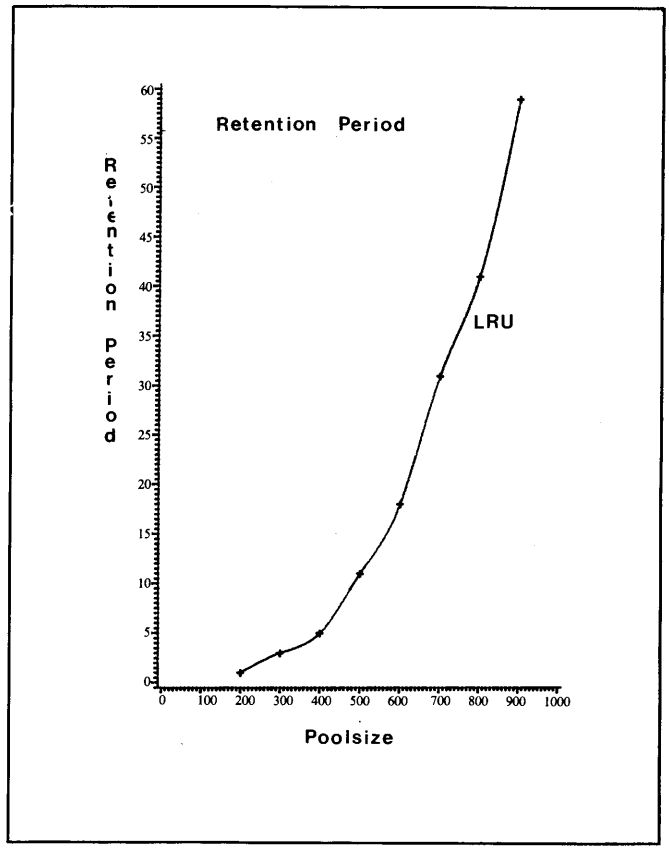


FIGURE 3

activity. A record is generated each time a user logs on to the system recording the time and date. A computer program was written to analyse logon records that had been accumulated over a nine month period covering most of the 1982 academic year.

This program was designed to model an LRU reclaim policy. It processed each logon record and assigned an imaginary minidisk to a user from a pool of available or 'free' disks. The total number of disks in the pool was termed the 'poolsize'. When a minidisk was assigned to a user, it was deemed to be 'active'.

If there were no free disks to be allocated, a 'fault' was deemed to have occurred and the least recently used active disk was reclaimed from its current user and assigned to the new user. The retention period for each reclaim was checked and the minimum for the run was recorded.

During the nine month period about 1500 different users had used the system. The model was run with poolsizes ranging from 100 to 800. Two graphs were produced from the results. The first indicates the percentage of logons which resulted in a fault occurring. The second indicates the minimum retention periods reported. The graphs are shown in Figures 2 and 3.

These results were more promising than expected. The curves indicated that for 750 disks in the pool, there would have been a minimum retention period of 35 days. Only 1,3 percent of the logons would have been faulted. The intermittent use of the computer meant that if this disk management policy could be implemented, a 50 percent saving in disk space would be achieved.

For practical reasons, the LRU policy could not be implemented as modelled. If a disk had to be reclaimed at logon time, the user would have to wait for all of the previous owner's files to be archived onto magnetic tape. This would cause an unacceptable delay.

To solve this problem, it was proposed that disks be preemptively reclaimed at night so that there would be sufficient free disks to meet the number of faults that might occur on the next day. The policy would be based on reclaiming all disks which had not been used for more than a fixed number of days.

This was termed the 'Fixed Retention Period', or FRP policy.

Modelling a Fixed Retention Period Policy

This algorithm was built onto the modelling program. It was run with various poolsizes and a fixed retention period of 21 days. This was considered to be a reasonable threshold between computing activity and inactivity.

In addition to the number of faults, the program reported on 'Pool Empty' situations, which occured if the faults exceeded the number of free disks. The results are presented in Figure 4.

The Fixed Retention Period policy does not utilize the disk space as efficiently as the LRU policy. For a poolsize of 750 disks the percentage of logons faulted is 1,8 compared to 1,3. A higher fault rate was however, considered to be a smaller user inconvenience factor than having to wait during logon for a disk to be reclaimed.

A Fixed Number Policy

At this point, an alternative pre-emptive policy was considered. This would be to reclaim a fixed number of the least recently used disks nightly, so that there would be enough free disks to service the next days faults. This was termed the 'Fixed Number', or FN policy. This Fixed Number policy was also built into the modelling program and it was run on the same basis as before. Percentage faulted logons are shown for all three policies in Figure 5.

For 750 disks the fault factor is only 0,05 percent worse than for the LRU policy. The characteristics of this policy are similar to the LRU policy in that the retention period varies for each poolsize.

Conclusions Drawn from Model

Both the Fixed Retention Period and Fixed Number policies would have the advantage of being pre-emptive. Although the fault rate is higher for the former policy, the retention period is fixed and therefore predictable.

It was decided that the Fixed Retention Period policy would be the most desirable. Being pre-emptive, the logon times would be acceptable and the disadvantage of a higher fault factor would be outweighed by offering a completely predictable service to the user.

The Fixed Retention Period policy has another favourable property in a University environment. During the long vacation at year-end there is little usage of the system. The FRP policy reclaims a large number of disks during this time. This means that the system is well equipped to accommodate the large number of new users who appear at the start of the next academic year.

SYSTEM IMPLEMENTATION

The historical data used for the model reflected computing activity during 1982 which was the first year that the VM/SP operating system was available. During that year, users were migrating from an older computer system and it was expected that there would be a substantial growth in 1983 when all users had migrated. For this reason, a poolsize of 1 100 minidisks and a retention period of 21 days were chosen to cater for some 3 000 registered users.

The system was implemented by having a service program maintain a directory of all the disks in the pool. At logon time, a message is automatically sent to this program on behalf of the user, requesting access to a disk. If the user is already reflected in the directory, the associated disk is simply attached and the date of last access is updated. If not, the first free disk in the pool is assigned and the users' identity and the date are recorded in the directory.

At night, a reclaim program scans the directory for all entries that are older than 21 days. The files on the associated disks are copied to magnetic tape and an entry added to an Archive Catalog for each file. The program also archives selected files at the request of the user without reclaiming the disk.

A dearchive program is run at hourly intervals during the day to enable users to selectively retrieve files from the Archive tapes.

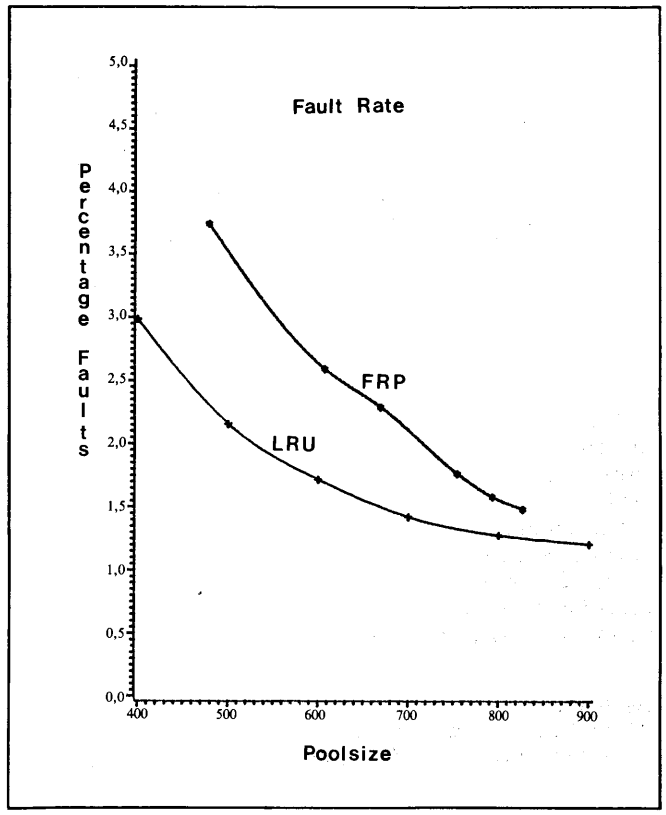


FIGURE 4

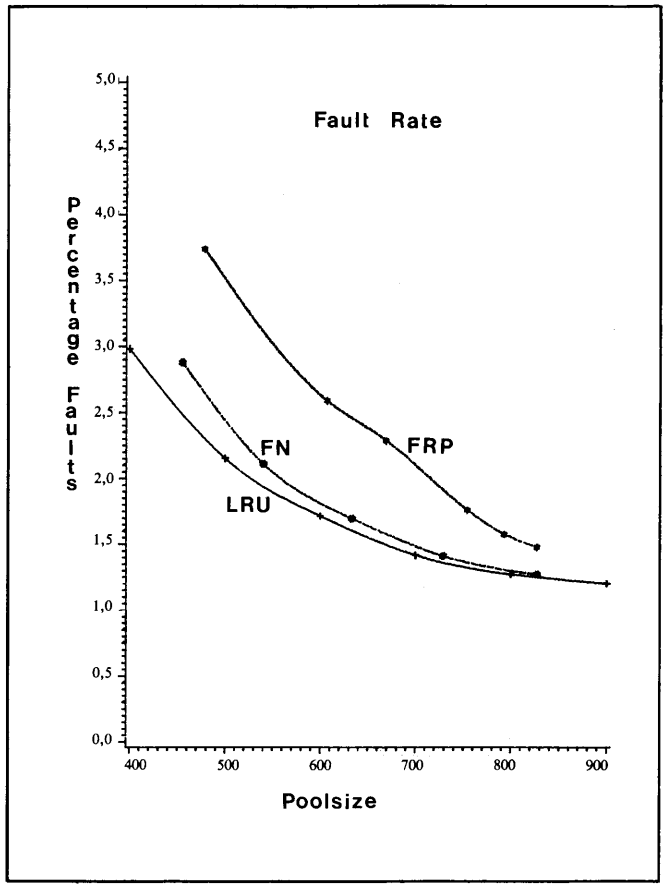


FIGURE 5

The Fixed Retention Period is parameterised in the reclaim program. If the pool of free disks becomes empty, the system operator is requested to run the reclaim program immediately, with a reduced Retention Period. It is hoped that this measure will cater for unexpected fluctuations in user demand on computing.

Based on the analysis of user space requirements, all pool minidisks were set to the minimum size of 450k bytes. If a user required more, a larger disk was added to the pool and marked as 'permanent'. This disk would not be eligible for normal reclaiming. Most of the permanent disks would belong to academic staff and post-graduates and would be managed manually.

Implemented in this way, the disk management system lengthens the logon time slightly, but otherwise has no effect on the functional capabilities of the standard VM/SP operating system.

ACTUAL RESULTS AND FUTURE RESEARCH

The system was implemented in January 1983, and the actual number of faults and active disks were monitored during the first five months. The model has been rerun with the latest historical data to validate the Fixed Retention Period algorithm used.

A modification to the Fixed Retention Period policy has been proposed and the authors believe that this would result in an improved service to the users. The modification would apply the policy as described, but instead of erasing the files when a disk is reclaimed, leave the disk active in the pool, but marked as free. If that disk then had to be assigned to a different user, the files would be erased at time of assignment. This operation would be almost instantaneous and would not delay the logon time appreciably. If, however, the previous owner of that disk returned before it had been re-assigned, the files would still be online.

This modification would guarantee the fixed retention period, but the files may very well remain online for much longer. Further research is being undertaken to evaluate this proposal.

Other research would relate to varying the frequency of disk reclaim operations, but current efforts have been confined to the case of reclaims on a once per day basis.

CONCLUSIONS

By adapting the concepts of virtual storage management to the management of disk space, it is possible to economise on the amount of online disk space required under conditions of intermittent computer access in a multi-user environment.

ACKNOWLEDGEMENTS

The authors would like to thank Mr L Clarke for producing the graphs presented in this paper.

References

[1] Belady, L.A. A study of Replacement Algorithms for a Virtual Storage Computer. In: IBM Systems Journal Vol. 5 No. 2 (1966).

Notes for Contributors

The purpose of this Journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to: Prof. G. Wiechers at:

Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by a summary of less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings, etc., should be submitted and should include all relevant details. Drawings, etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters between the letter O and zero; between the letter l, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT, E. and MANNA, Z. (1972). The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71,* North-Holland, Amsterdam, 250-255.
2. BÖHM, C. and JACOPINI, G. (1966). Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm. ACM,* **9**, 366-371.
3. GINSBURG, S. (1966). *Mathematical Theory of context-free Languages,* McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged against the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.

Quaestiones Informaticae

Contents/Inhoud

Evaluating the Performance of Computer-Based Information Systems using a Restricted Linear Regression Model.....1
P J S Bruwer

An improved Implementation of Grimbleby's Algorithm*.....7
S R Schach

Nebulas as Structural Data Models*.....11
H O van Rooyen and D J Weiermans

Data Structures for Generalized Network Algorithms*.....15
D C Currin

Modelling Blocking on Admission of Tasks to Computer Systems*.....19
S Wulf

Analytical Model of a Mixed-Workload MVS Computer System*.....21
R J Mann

Add: The Automated Database Design Tool*.....25
S Berman

A Disk Space Management System*.....29
A J Cuthbertson, I J van Nierkerk, T Turton

*Presented at the second South African Computer Symposium held on 28th and 29th October, 1981.

MILITHO