Improving the performance of an intelligent data management system

J. G. Kollias, P. M. Stocker and P. A. Dearnley

School of Computing Studies, University of East Anglia, Norwich NR4 7TJ, Norfolk

The paper solves two of the problems concerned with improving the performance of a particular class of intelligent data management systems: the self organising system. A common characteristic of the problems solved is the introduction of solution spaces which can be searched by linear programming techniques.

(Received December 1976)

1. Preliminaries

For the purposes of this paper, a Data Management System (DMS) is considered to be a software package designed to facilitate file creation, storage, retrieval and maintenance. DMS can be classified with respect to various features (CODASYL, 1974); the particular classification of interest in this paper (Darnley, 1975) distinguishes between conventional and intelligent DMS. Conventional DMS have been designed to serve known or anticipated data processing needs while intelligent DMS aim to support users' data processing requirements which change over time in a manner which may be unknown in advance.

The Self Organising Data Management System (SODMS) developed by Stocker and Dearnley (Stocker and Dearnley, 1973; 1974; Dearnley 1974) is an intelligent DMS which has the ability to automatically select its data structure and access methods to meet new patterns of usage. The operation of the system is monitored over each time period. At the end of each period the experience gained from the past system operation may be used to perform some database modification.

In particular, the pilot model developed for testing the applicability of the SODMS concepts assumes that users are primarily concerned with low operational cost as opposed to fast response time. The objective that the model is trying to achieve is the minimisation of the total operational cost. It is assumed that this cost is passed to the community of users as a whole.

Users of the model provide their information as a normalised set of records (Codd, 1970). During the system operation the model allows the duplication of the same data in other files called *versions* in an attempt to balance improved access costs against extra maintenance and storage costs. Another option of the model is that it allows its users to declare the costs they are prepared to pay for getting a reply to a query. If the model finds it impossible to satisfy the query at such a cost it places the query in a *stack* in the hope that batching or file organisation changes will eventually reduce the cost of satisfying the stacked queries.

Since it is usually uneconomic for the system to be organised for a cheap reply on every type of user query, some queries have to be satisfied from versions not ideally structured for the queries. The model, before placing such queries on the stack, utilises the following feature: the entire set of versions, called a *folio*, is viewed by the model as a directed graph. This may enable the model to satisfy a query by cheaper means (than a serial search) even when an ideally structured version does not exist in the folio. This is achieved by following the cheapest path among the required versions provided that this leads to a cost acceptable to the user.

Operational experience with the pilot model (Dearnley, 1974b) indicates that a viable DMS can be built around the SODMS concepts. A successful implementation of a SODMS is expected to encourage the use of DMS which will serve the interests of a variety of remotely located and unco-ordinated users who will share common interests in the same data. (One of the most important ideas behind the SODMS concept is that of having a DMS which can benefit from its past performance statistics by automatically reconfigurating itself. Senko states that: 'the SODMS efforts seem to indicate that the field now has a solid base and is beginning to move forward' (Senko, 1975).)

This paper addresses two open questions related (*a*) to the optimal scheduling of stacked queries to different versions, and (*b*) deciding how versions should be distributed among online and offline disc packs. Both problems have as a common characteristic the introduction of solution spaces which can be searched by linear programming techniques.

2. Variables

To formalise our problems we consider a file consisting of a number of normalised records, NREC, each record comprising values for each of r attributes $a_1, a_2, \ldots, a_r$. By convention the attribute, a_1, is taken to be the primary key for purposes of storage and retrieval. Each attribute, a_t, is assumed to have a length of l_t bytes and may take d_t distinct values, $t = 1, 2, \ldots, r$. Thus the record length is LREC, where $\text{LREC} = \sum_{t=1}^{r} l_t$. The SODMS recognises each attribute by a mnemonic name which is also used in the present paper each time we refer to that particular attribute.

Example 2.1

Throughout this paper an original version which contains forestry data on specimen trees is to be considered. Each record contains nine attributes having the names, meaning, length and number of distinct values listed in **Table 1.**

Attribute		*Meaning*	l_t	d_t
Number	*Name*			
1	NUMBER	A unique number assigned to tree	6	NREC
2	ADDRESS	Address of plantation (code)	3	500
3	TYPE	Tree description	14	120
4	AREA	Geographical area code	2	55
5	HEIGHT	Tree height in feet	3	250
6	WIDTH	Tree width in inches	3	400
7	DATE-1	Last time height and width measured	6	1,000
8	DATE-2	Last time sprayed	6	500
9	REFERENCE	Dossier reference	3	250

Table 1 Record description and related information

From Table 1 certain assertions can be made. The record length, LREC, is 46 bytes long. The attribute with the name NUMBER uniquely identifies the record. The reason that the d_1 value is deliberately unspecified is to keep the size of the data base as a variable in our problem. It is assumed that the other d_t values are independent of the value assigned to d_1.

We consider that the disc is physically partitioned into fixed size blocks of magnitude BS bytes called buckets. The SODMS performance is measured by the required number of bucket accesses necessary to perform the overall user transactions, while all the processing time is taken as free. Such a method of measurement is justified by the fact that the cost of an initial access to a word in a bucket of information in disc is typically three orders of magnitude more than the cost of accessing a word in the same bucket when the bucket is in main memory. In the event that records may not be divided between buckets (a restriction posed by the majority of operating systems) then the size of the first version in buckets, VS_1, can be evaluated by the formula $\lceil \text{NREC}/\lfloor \text{BS/LREC} \rfloor \rceil$, where $\lceil x \rceil$ ($\lfloor x \rfloor$) denote the least (greatest) integer greater (less) than or equal to the real number x. If we take NREC to be 20,000 records then VS_1 in our example is 1,819 buckets. Thus, 1819 is taken as the cost of satisfying a query for the first version.

To circumvent the high costs resulting from the use of the first version when satisfying queries the SODMS creates, if it finds them profitable, additional versions. The versions are dictated by the recent pattern of user queries and are files which are ordered and indexed on the values of the attribute appearing in the qualification part of a query and also containing the attributes which are included in the output part of the query. It can be seen that the concept of a version is similar to Codd's projection (Codd, 1970) which involves the selection of certain attributes from the original record and then the removal of duplicated records.

The only further quantitative measurement to be used in this study is the version sizes, VS_j, $j = 1, 2, \ldots, n$. To evaluate them we assume that independence exists between the attribute values which appear in a version and we introduce the sets V_j having as elements the numbers of the attributes which appear in the jth version. The number of distinct records in the jth version is taken to be $\min\left(\prod_{i \in V_t} d_i, \text{NREC}\right)$. The version sizes are then evaluated by the formula used above to estimate the quantity VS_1.

3. Servicing queries in the system's stack

This section deals with the problem of finding the most economic means of satisfying queries in the system's stack. During its operation the SODMS may place a number of queries in its stack with the intention of satisfying them at a later time with a lower cost per query. Since the concept of stacking the queries is similar to that of batching we clarify the concepts and explain the reasons why, although batch data base maintenance is a common operational practice, the majority of DMS do not support batching of queries.

In general both concepts (i.e. batching and stacking) refer to a means of scheduling the queries to achieve some economies of scale when satisfying them. These economies result from the fact that the larger the batch size the higher the probability that more than one query will be satisfied from the same bucket. In spite of such economies most DMS abandon altogether the idea of batching on the grounds that they operate on strict deadlines with respect to the answering of queries. This implies that it is taken for granted that no user can afford more than, say, five minutes waiting at the terminal before getting a reply to a query but during this period one cannot expect the accumulation of queries to result in batch sizes which offer significant possibilities of savings (Senko, 1972).

This study suggests a means of servicing stacked queries which can result in significant economies in SODMS operation even for a very small batch size. The philosophy of the approach is based on the concurrent service from a particular bucket of as many as possible of the queries among those in the system stack. Thus the problem considered here is, 'how should the SODMS automatically select those versions in the folio which satisfy all the stacked queries at a minimum possible number of bucket accesses?'

3.1. *Description of method*

We consider a folio with the data base characteristics presented in the previous section and assume that at a particular stage of the SODMS operation the folio contains the five versions shown in **Table 2** below.

Table 2 Folio content

Version No.	*Attributes involved*	VS_j	S_k
1	All	1,819	S_1
10	*AREA*, DATE-1, HEIGHT, WIDTH	556	S_2
12	*DATE*-1, HEIGHT, WIDTH	477	S_3
14	*AREA*, HEIGHT, DATE-1, DATE-2	667	S_4
16	*REFERENCE*, AREA, HEIGHT, WIDTH	435	S_5

In Table 2 we see (*a*) the version numbers which identify each version, (*b*) the names of the attributes appearing in each version with the underlined attribute names indicating that the versions are sorted and indexed on the values of that attribute, (*c*) the version sizes, i.e. the costs in bucket accesses of searching serially the versions, and (*d*) the variables S_k, $k = 1, 2, \ldots, 5$ which denote possible searching strategies. For example, the strategy S_3 refers to a serial search of the version no. 12 at a cost 477.

We also assume that the SODMS has stacked the six queries Q_j, $j = 1, 2, \ldots, 6$ displayed in **Table 3.**

Table 3 Stacked query types

Query type	*Description*
Q_1	Given HEIGHT find WIDTH, AREA
Q_2	Given AREA find HEIGHT
Q_3	Given HEIGHT find DATE-1, AREA
Q_4	Given WIDTH find DATE-1
Q_5	Given WIDTH find HEIGHT, REFERENCE
Q_6	Given DATE-1 find AREA, HEIGHT

From Tables 2 and 3 it can be seen that each query can be served individually by a number of searching strategies. For example, the Q_1 type of query can be satisfied by the S_1, S_2 and S_5 strategies. Assume at the moment that the strategy S_5 has been selected at a cost of 435. It is easy to verify that the same strategy can also satisfy the Q_2 and Q_5 types of query. With reference to this particular strategy (S_5) and queries (Q_1, Q_2 and Q_5) the required modification to the SODMS is: 'Each time a bucket of the version No. 15 is fetched into main memory make the SODMS prepare concurrently the replies to the Q_1, Q_2 and Q_5 type of queries'. By this method the cost of satisfying the queries drops from $3 \times 435 = 1{,}305$ (this cost would occur if the queries were served on arrival) to 435 at an average cost per query of 145. In general the method is based on the ground that the cost of applying a strategy remains constant regardless of the number of queries concurrently served by the strategy.

3.2. *Formalisation*

Let types of queries, $Q_j, j = 1, 2, \ldots, m$ be in the system stack and let $S_1, S_2, \ldots, S_s$ be the s possible strategies determined by the current state of the folio. We associate with each strategy the s-component column vector $x = (x_k)$ of binary variables. The kth variable of x takes the value 1 or 0 depending on whether or not the kth strategy is to be included at the optimal searching. The cost of applying the S_k strategy will be denoted by the s-component row vector $c = (c_k)$. Finally let an $m \times s$ matrix $A = (a_{jk})$ having elements with values 1 or 0 depending on whether or not the jth query can be satisfied by the kth strategy ($j = 1, 2, \ldots, m$; $k = 1, 2, \ldots, s$).

The optimal strategy to be applied depends on the values of the decision variables x_k obtained by the solution of the following zero-one integer programming problem:

$$\begin{array}{lll} \text{Minimise} & cx & \\ \text{subject to} & Ax \geq e & (LP_1) \\ & x_k = 0 \text{ or } 1 & \end{array}$$

The structure of (LP_1) is known in the literature as the 'set-covering' problem (Seppala, 1967). Lemke *et al.* (1970) report in detail computational experience of (LP_1) with their proposed algorithm. They indicate that a 15 × 32 problem (e.g. 15 queries and 32 strategies) has been solved in 0·044 minutes on an IBM 360/50. The viability of the proposed approach is indicated by the facts that (*a*) the above cost is approximately 30 bucket accesses (one bucket access is taken as 90 ms), (*b*) the method can result in the saving of hundreds or even thousands of bucket accesses, (*c*) it is most unlikely that the SODMS will face, in any practical instance, a problem larger than 15 × 32, and (*d*) the efficiency of any algorithm improves significantly as the size of the problem is reduced. In the next subsection we outline some means for reducing the complexity of (LP_1).

3.3. *Example of application*

The searching strategies can be formalised into the following (LP_1) type of linear program:

$$\begin{array}{lrcrcrcrcl} \text{Minimise} & 1{,}819x_1 & + & 556x_2 & + & 477x_3 & + & 667x_4 & + & 435x_5 & \\ \text{subject to} & x_1 & + & x_2 & & & & & + & x_5 & \geq 1 \\ & x_1 & + & x_2 & & & + & x_4 & + & x_5 & \geq 1 \\ & x_1 & + & x_2 & & & + & x_4 & & & \geq 1 \\ & x_1 & + & x_2 & + & x_3 & & & & & \geq 1 \\ & x_1 & & & & & & & + & x_5 & \geq 1 \\ & x_1 & + & x_2 & & & + & x_4 & & & \geq 1 \end{array}$$

From the above program it can be seen that the coefficients of the objective function are the search costs when applying a strategy (Table 2) while the constraints give the means by which every stacked query can be satisfied. For example the third constraint indicates that the third query (Given HEIGHT find DATE-1, AREA) can be satisfied by one of the S_1, S_2 and S_4 strategies.

The program is always feasible. This happens because the S_1 strategy provides a solution since it can satisfy all the stacked queries. Secondly, the cost 1,819 of applying this solution (i.e. (1, 0, 0, 0, 0)) forms an initial upper bound of any subsequent cheaper solutions. This observation can be used in accelerating the performance of the linear programming algorithms employed. For example, the solution (0, 1, 1, 1, 1, 1) need not be considered because it cannot be the optimal one since its cost 2,135 (556 + 477 + 667 + 435) exceeds the value 1,819.

Referring again to Lemke (1970), we notice the existence of a number of theorems relating to (LP_1). We mention only one of them which results in solving our example case by inspection. They state*: 'Suppose for some column S_k of (LP_1) there exists a set of other columns of (LP_1) whose sum covers S_k, and the cost of column $k \geqq$ the sum of the costs of the other columns. Then S_k may be deleted.'

The above theorem applied to our example case (prior to the utilisation of any algorithm for solving (LP_1)) results in the elimination of the S_1 strategy from further consideration. This is because the strategies S_2 and S_5 cover the S_1 and can be applied at a cost 991 (556 + 435) which is better than the c_1 value (i.e. 1,819). Having eliminated the S_1 strategy it is not hard to verify that the optimal solution is $x = (0, 1, 0, 0, 1)$ at a cost 991.

*We have modified their symbols to ours.

3.4. *Implementation notes*

The implementation of the method poses several subsidiary problems which we think are of importance to the design and operation of the SODMS. This subsection contains certain remarks with the hope that they will be helpful to the future SODMS designers.

The first problem is related to the implementation of the derived optimal strategy. For example, the solution derived in the previous subsection indicates that the SODMS will apply the S_2 strategy for satisfying the Q_3, Q_4 and Q_6 type of queries while the Q_5 query will be served by S_5. The queries Q_1 and Q_2 can be satisfied by either S_2 or S_5.

The first question to ask is 'what strategy should be employed for satisfying the Q_1 and Q_2 queries?'. There are two answers each of which provides some justification for their application. The first answer suggests that the S_5 strategy should be applied since it results in a balanced distribution of the queries to strategies (three queries to each S_2 and S_5). This avoids the over-utilisation of the CPU for a specific strategy (S_2). The second answer (which we support) states that the S_2 strategy should be used for satisfying all the Q_1, Q_2, Q_3, Q_4 and Q_6 type of queries while the Q_5 will remain on the stack. This approach drops the cost from 991 to 556.

Another question arises, namely, 'Assuming that a number of strategies belong to the optimal solution in what order should they be performed?'. An efficient answer, assuming that the SODMS uses private disc packs, is that the SODMS should issue commands to the operating system to serve the strategies in some monotonic (ascending or descending) sequence of the cylinder numbers utilised by the strategy. The advantages of such an implementation is that it optimises the necessary seek time for satisfying the optimal strategy (Teorey, 1972).

A second problem which we have not yet mentioned is the stimulus for initiating (LP_1). Possible stimuli might be either the lapse of a time period (say, every six hours) or the number of queries in the system stack (say, every 10 queries). The computer speeds, the performance of the application programs, the load of the SODMS and the efficiency of the exact or heuristic algorithm employed for solving (LP_1) are among the factors to be considered in solving this problem.

4. Allocation of versions to online and offline disc packs

The SODMS creates new versions with the objective of optimising the cost of predicted usage. When the amount of space required for versions exceeds the amount which can be kept online the system places the less frequently used version(s) on an offline disc pack. The aim of this section is to investigate the possibility of applying a more effective disc space allocation policy.

Despite the great number of papers published on DMS little has been written on the problem of allocating data bases to discs. We present four reasons which may explain this.

The primary reason is that the problem is difficult to handle by analytical techniques since it relates to scheduling problems. Scheduling problems can be conveniently expressed in terms of linear programming. Secondly, when DMS are studied there is often an implicit assumption that the data base will reside only

on online disc packs and that this accommodation is possible. Of course such an assumption does not always apply and is not made for the SODMS.

The essence of our approach lies in the third reason. In practice it is unlikely that any 'clever' allocation scheme can be derived unless a differentiation exists between either the file and index sizes or the frequencies by which parts of files or indices are accessed. However, such information is rarely considered when studying DMS. Our approach utilises exactly this knowledge concerning the frequencies of accessing various parts of the data base.

The fourth and final reason lies in the fact that even if a suitable mathematical formulation is derived the overheads involved in calculating solutions, and especially in rearranging the files into their optimal places on disc might well be so large as to outweigh any gains made by applying the obtained solution (Morgan, 1974). This is catered for in this section and we avoid factors which might prohibit the use of any derived solution in practice.

In order to alter the replacement rule logic of the SODMS we note that despite the fact that the system operates on the assumption that the user requirements change dynamically over time, it is not unreasonable to expect that for some time period the system will operate under some steady state situation. Galler (in Buxton and Randel, 1969), reinforces this observation. In such a case we propose useful measurements to be taken during the SODMS operation which can be used for an automatic tuning of the system in a way that warrants the expense of such tuning.

4.1. *Basic considerations*

We start with the specifications of an example case. We consider that the SODMS is called to serve the data processing requirements of a casual user like a research centre, social institute, etc. The folio usage conforms to the following pattern:

1. One can determine the type and load of user queries over a particular time period (say six months) and unit time of operation (say one day). The source of such knowledge (i.e. prediction or advance information) is immaterial.
2. The versions contain a rather large number of records which are to be requested at any time during the, say, eight working hours of the institute's operation.
3. The number of records retrieved per day is relatively small in comparison with the total number of records in the versions.
4. The response time is not such an important factor as cost provided that queries are served within some specified bounds, say, five to 20 minutes.
5. The system does not support daytime online maintenance. The users can afford, a, say, 24 hours wait until updates are performed.

Table 4 lists the types of query, Q_j, and their expected daily frequencies (loads) q_j, $j = 1, 2, 3$.

Table 4 Query types and loads

Query type	*Description*	*Frequency*
Q_1	Given ADDRESS find HEIGHT, WIDTH, DATE-1, DATE-2	24
Q_2	Given DATE-1 find HEIGHT, WIDTH DATE-2	37
Q_3	Given HEIGHT find ADDRESS, WIDTH, REFERENCE	18

To assess the effectiveness of different allocation policies a standard performance measurement must be established; in this section we use the 'number of disc mount requests issued to the operator'.

The dominant cost is that associated with the channel service time unless (*a*) the disc space to be kept online is large and it has a very low usage, or (*b*) the number of disc mount requests is excessively high. If one or both of the above cases occur one expects that the installation will impose high values (*a*) for the cost of every bucket occupied online, and (*b*) for mounting charges, M. It is worth noting that in a recent paper (Borowits, 1974) a value for the mounting charge of \$0·37 has been proposed and a charge of \$11·21 per hour for keeping an entire disc pack online regardless of its usage.

The usage displayed in Table 4 causes the SODMS to create three versions, each sorted and indexed on the values of ADDRESS, DATE-1 and HEIGHT respectively and containing also the attributes which appear on the output part of the corresponding queries. Naturally such a decision will pass through some justification process but the fact that we consider low maintenance and storage cost made us take for granted the fact that SODMS will find it profitable to create the three versions protrayed in **Table 5.** The information in Table 4 considers that NREC = 100,000 and the version sizes VS_j, $j = 1, 2, 3$ have been rounded so the presentation of the calculations involved in the rest of the section are simplified.

Table 5 Version size and disc space required

Version No.	*Attributes involved*	VS_j
1	*ADDRESS*, HEIGHT, WIDTH, DATE-1, DATE-2	4,000
2	*DATE*-1, HEIGHT, WIDTH, DATE-2	3,500
3	*HEIGHT*, ADDRESS, WIDTH, REFERENCE	2,500
	Disc space required	10,000

Our previous discussion on charging disc usages suggests that the SODMS approach, although guaranteeing low channel cost, can lead to the following practical limitations. Firstly, the quantity of the disc storage required might be so high that (*a*) it cannot be accommodated online, and (*b*) the storage cost becomes a significant factor. Secondly, when versions are kept offline the operator is asked to mount disc packs resulting in (*a*) increased operational cost, (*b*) degradation of the computer system performance, and (*c*) an increase in the response time to the average query.

The above undesirable situation led us to assume that 8,000 buckets are available (out of the 10,000 buckets required) and to investigate the SODMS performance when it is constrained to operate with 20% less space. One can suggest meaningful interpretations of our arbitrary choice of 8,000 buckets, e.g. it could be the space agreed with the installation management or it could be the storage capacity of a disc. One can also deduce from our previous discussion that the operation of the SODMS under limited disc space improves the disc space utilisation at the risk of an increased number of mount requests issued to the operator. This made us select the number of disc pack load requests as a performance indicator for alternative allocation policies.

Finally, we mention that in any future trading-off analysis we associate with any evaluated number of disc pack mount requests the effect on the expected response time of a query appearing in Table 4. To evaluate this we assume that it takes (*a*) one second to satisfy a query from an online disc pack, and

Table 6 The correspondence between the knapsack and the SODMS terminology

Item	Version
Value	Frequency
Weight	Version size
Allowable weight	Online disc space

(*b*) three minutes from an offline one. The latter time estimates both query satisfaction and the time taken for the operator to perform the disc mount. From these assumptions it is clear that the expected response time, E_Q, for a query is estimated by $E_Q = 1 \times p_{ON} + 180 \times p_{OFF}$ seconds, where p_{ON} (p_{OFF}) denotes the probability that versions are online (offline) when requested.

4.2. *The indivisible case*

We treat the problem of minimising the number of disc mount requests analogously to the 'knapsack problem' in OR (Cabot, 1970). The knapsack problem is to choose among a set of candidate items (the set contains only one item of each type) that subset which maximises the value of the subset without violating a weight constraint. The items differ in weight and value and the maximum allowable weight for the loaded knapsack is specified. **Table 6** lists the correspondence of the terms introduced above with the terminology used by the SODMS.

If we assume that items are indivisible, i.e. selecting a fraction of an item is forbidden, the knapsack problem is an *integer* linear program. If the items are divisible the problem is a linear program. Both instances of the knapsack problem can find their equivalent in the SODMS. The indivisible case relates to the situation in which we only allow a version to be either online or offline while the divisible case arises if we do not pose any restriction as to where the parts of the versions reside.

The mathematical formulation of the knapsack problem applied to our example case (Tables 4 and 5) is as follows:

Find x_1, x_2, x_3 maximising

$$24x_1 + 37x_2 + 18x_3 \quad (1)$$

subject to $4{,}000x_1 + 3{,}500x_2 + 2{,}500x_3 \leq 8{,}000$ (2) (LP_2)

$$x_1, x_2, x_3 = 0 \text{ or } 1 \quad (3)$$

(LP_2) corresponds to the indivisible case. By replacing (3) by the inequality $0 \leq x_j \leq 1$ (3′) the problem (LP'_2) consisting of (1), (2) and (3′) corresponds to the divisible case. In (LP_2) and (LP'_2) the objective function (1) reflects the desire to maximise the chances that a requested version will be online; the constraint (2) gives the disc space required for storing all the versions and of the amount of the oneline disc space available; (3) (or (3′)) introduces the decision variables x_j which in the optimal solution signify the versions (or part of them) which are to be kept online or offline.

If we take the current SODMS disc allocation policy which does not allow versions to be divided between online and offline disc packs, such a restriction is imposed by many operating systems, e.g. OS/360, then the optimal solution to (LP_2) is $x_1 = x_2 = 1$ and $x_3 = 0$. This implies the placement of the first two versions in the online disc pack and of the third version in the offline one. Such a policy results in 18 *disc pack mount requests* per day, i.e. the frequency that the third version is requested. The expected response time to a query is

$$E_Q = 1 \times \frac{24 + 37}{24 + 37 + 18} + 180 \times \frac{18}{24 + 37 + 18} \simeq 41{\cdot}7 \text{ secs.}$$

Since the SODMS can place various parts of a version in different disc packs (by introducing distinct version numbers to different parts) there are better allocation policies.

4.3. *The divisible case*

If versions are divisible the problem is expressed mathematically in terms of (LP'_2) which has a very simple solution. The SODMS is confronted with the problem: Load to an online disc pack those parts of versions (among the versions listed in Table 5) which will maximise the chances that a query will refer to an online disc pack but do not exceed the available space of 8,000 buckets. We describe below (by using the SODMS terminology) an algorithm which solves (LP'_2).

[K_1] Rank the versions in decreasing values of q_j/VS_j.

[K_2] Load to the available online disc space the version (*not* a part of it) with the highest value of q_j/VS_j.

[K_3] Repeat K_2 until a complete version cannot be accommodated to the online disc.

[K_4] Place in the online disc as much of the version as fits into it. The rest of this version plus any others left are placed offline.

In more descriptive terms the algorithm finds the optimal online disc loading by first allocating space for that version which contains buckets having the highest probability of being requested, followed by the version having buckets with the next highest probability, etc. until the entire available disc space has been exhausted. By applying the algorithm to our example case we find that the second version (3,500 buckets), the third (2,500 buckets) plus 2,000 buckets of the first version are to be placed online while the remaining 2,000 buckets of the first version are placed offline. For trading-off the obtained solution we assume that both parts of the first version, i.e. the online and the offline one are requested uniformly so each receive half of the requests. Since the first version supports 24 requests daily the solution obtained involves 12 *disc mount requests* and it offers an expected response time, E_Q, to a query equal to 28·1 seconds.

The above solution is better than the solution obtained for the indivisible case and was derived on the assumption that user queries are uniformly distributed over the version parts. However, Heising (1963) reports that in practice there is commonly observed the '20 to 80 rule' which states that 80% of the transactions fall within 20% of file records. Heising's observation motivated our investigations as to whether a further reduction on the number of disc mount requests can be achieved when version parts are not uniformly accessed.

4.4. *Non uniform access frequencies of version parts*

Consider a folio having versions which are physically partitioned into two equally sized parts and that the SODMS keeps statistics for the frequencies of requests for each individual part. Each part is identified as $V_{j,1}$ or $V_{j,2}$ where $j = 1, 2, 3$ and 1 and 2 refer to the first or the second physical part of the jth version. For example, $V_{3,2}$ refers to the second part of the third version in Table 5. The variables $VS_{j,1}$ and $VS_{j,2}$ denote the sizes of these parts. We assume that the frequencies, q_j, of accessing the entire versions are as shown in Table 4 but the frequencies $q_{j,1}$ and $q_{j,2}$ are distributed to the two parts of the versions as protrayed in **Table 7.** Table 7 also lists the ranking of the subversions.

Table 7 Subversions size, load and ranking

Version No.	$VS_{j,1}$	$q_{j,1}$	$q_{j,1}/VS_{j,1}$	*RANK*
1, 1	2,000	14	0·0070	4
1, 2	2,000	10	0·0050	6
2, 1	1,750	12	0·0068	5
2, 2	1,750	25	0·0142	1
3, 1	1,250	9	0·0072	2
3, 2	1,250	9	0·0072	3

Table 8 Subversion and versions size, load and ranking

Version No.	*VS*	*q*	*q/VS*	*RANK*
1, 1	800	19·2	0·024	1
1, 2	3,200	4·8	0·001	4
2	3,500	37	0·010	2
3	2,500	18	0·007	3

Referring to Tables 4 and 7 the observed variations among subversions accesses suggest the following folio usage. The users are slightly more interested in the trees in the Northern part of the country (compare $q_{1,1} = 14$ and $q_{1,2} = 10$ values); they concentrate on questions referring to the most recently taken measurements of trees (compare $q_{2,1} = 12$ with $q_{2,2} = 25$); their requests for the tree heights are completely symmetrical with respect to the two version parts ($q_{3,1} = 9$, $q_{3,2} = 9$).

By applying the algorithm for solving (LP'_2) to the data in Table 7 we find that the optimal solution suggests that the SODMS places all the subversions online apart from $V_{1,2}$. This solution results in 10 *disc mount requests* and an expected response time to a query of $E_Q = 23{\cdot}6$ seconds.

Let us assume that just the first of the versions follows the 20 to 80 rule. This implies that the first (or the second) part of the version which has a size of $0{\cdot}2 \times VS_1$ buckets satisfies the 80% of the Q_1 type of queries (Table 4) while the second part of size $0{\cdot}8 \times VS_1$ satisfies the rest of the queries concerned. **Table 8** lists all the necessary information for solving the problem.

The optimal strategy involves the placement of $V_{1,1}$, V_2, V_3 and of 1,200 buckets of the $V_{1,2}$ version to the online disc pack while the rest of 2,000 buckets of $V_{1,2}$ are to be kept offline. But since the 3,200 buckets of $V_{1,2}$ have an expected frequency of 4·8 requests per day it can be estimated that the 2,000 offline buckets will cause 3 *disc mount requests* during the SODMS operation. The resultant expected time, E_Q, for serving a query becomes 7·7 seconds. The results establish a dramatic improvement over the initial solution (i.e. 18 disc mounts and an average time per query of 41·7 seconds).

5. Conclusions

Despite the fact that an efficient design of a DMS requires the tailoring of its facilities in such a manner as to yield the minimum value of a selected objective (e.g. bucket accesses), rarely, if at all, does any DMS base its design decisions on some rigorous mathematical optimisation principle. The paper gives, in the context of the SODMS, two replies to the suggestions made by other researchers (Muller, 1969; Nunamaker, 1969; Severance, 1972; Teichrowe; 1971) that the design of DMS can be improved by formalising it and applying OR techniques. Additional material has been given by one of the authors (Kollias, 1976).

References

BOROWITS, I. (1974). The pricing of computer systems, *Data Processing*, May-June, pp. 160-163

BUXTON, J. N. and RANDEL, B. (Eds.) (1969). Software engineering techniques, Report on a conference sponsored by NATO Science Committee, Rome, p. 26

CABOT, V. A. (1970). An enumeration algorithm for the knapsack problems, *Operations Research*, Vol. 18, pp. 306-311

CODASYL SYSTEMS COMMITTEE (1974). Introduction to features analysis of Generalized Data Base Management Systems, *CACM*, Vol. 14, pp. 308-318

CODD, E. F. (1970). A relational model of data for large shared data banks, *CACM*, Vol. 13, pp. 337-347

DEARNLEY, P. A. (1974). A model of a Self Organising Data Management System, *The Computer Journal*, Vol. 17, pp. 13-16

DEARNLEY, P. A. (1974b). The operation of a Model Self Organising Data Management System, *The Computer Journal*, Vol. 17, No. 3

DEARNLEY, P. A. (1975). Intelligent data base management, *Management Datamatics*, Vol. 4, pp. 231-235

HEISING, W. P. (1963). A note on random addressing techniques, *IBM Syst. Journal*, Vol. 2, pp. 112-116

KOLLIAS, J. G. (1976). The design of Data Base Management Systems using linear programming techniques, Ph.D. Thesis, University of East Anglia

LEMKE, C. E., SALKIN, H. M. and SPIELBERG, K. (1970). Set covering by single branch enumeration with linear programming subproblems, *Operations Research*, Vol. 18, pp. 908-1022

MORGAN, H. L. (1974). Optimal space allocation on disk storage devices, *CACM*, Vol. 17, pp. 139-142

MULLER, M. E. (1969). Statistics and computers in relation to large data bases, *Statistical Computation*, pp. 87-176

NUNAMAKER, J. F. (1969). On methodology for the design and optimisation of Information Processing Systems, *Proc. AFIPS SJCC*, Vol. 38, pp. 283-294.

SENKO, M. E. (1972). Details of a scientific approach to information systems, Courant Symp. in *Data Base Systems*, Prentice Hall, pp. 144-174

SENKO, M. E. (1975). Information systems: Records, relations, sets, entities and things, *Information Systems*, Vol. 1, pp. 3-13

SEPPALA, Y. (1967). Definitions of extraction files and their optimization by zero-one programming, *BIT*, Vol. 7, pp. 206-215

SEVERANCE, D. G. (1972). Some generalized modelling structures for use in the design of file organizations, Ph.D. Thesis, University of Michigan

STOCKER, P. M. and DEARNLEY, P. A. (1973). Self Organising Data Management System, *The Computer Journal*, Vol. 16, pp. 100-105

STOCKER, P. M. and DEARNLEY, P. A. (1974). A model of a Self Organising Data Management System, *Proc. of IFIP Working Conf. on Data Base Management*, North Holland, Amsterdam, pp. 337-349

TEICHROWE, D. and SAHANI, H. (1971). Automation of information system building, *Datamation*, August 15, pp. 25-30

TEOREY, T. J. (1972). Properties of disk scheduling policies in multiprogrammed computer systems, *Proc. AFIPS SJCC*, Vol. 41, pp. 1-11