An adaptive on line data compression system

D. L. Dance* and U. W. Pooch†

A set of programs is described defining an interface between an online information system and the input/output control system of the computer system. These programs are grouped by the function they perform: buffering, item relocation, compression, and dynamic priority assignment. The interface is adaptive in nature by physically reorganising the data set based on usage statistics. Items are physically assigned to priority areas to reduce system I/O. The result of the reorganisation is to construct working set data sets, a subset of the original data set, having a substantial portion of all data set activity. This working set data set is maintained in core via buffering, thereby reducing I/O overhead. To implement the interface and reduce the cost of storing data sets, one of the many available compression routines was applied to the entire data base.

(Received March 1975)

Rapid, efficient user access to large masses of data is a requirement in a complex technological society since the ability to obtain and use the content of current, accurate data may dictate the success or failure of an operation. To improve the information dissemination process and cope with the information explosion, a computer-implemented process should be used to determine which data is of greatest user interest and thus make it available with minimal time delay.

An online data compression system (ODCS) must be an invisible interface between the information retrieval system and the input/output control system (IOCS) of the computer operating system. This paper describes a set of programs which is analogous to a demand-paged memory management system. These programs are classified into three main categories: compression/decompression, buffering, and relocation. The compression/decompression routines reduce the physical storage requirements to contain a particular set of data items. The buffering routines maintain several physical records in main memory to reduce the I/O traffic. The relocation routines provide the computer with the ability to 'recognise' those items which are requested most often, and to physically reorganise those items for easy, fast retrieval.

Compression techniques

Several authors have classified data compression techniques into the following categories; parametric extraction, adaptive sampling, redundancy reduction, and statistical encoding. Andrews (1967a, b) Kortman (1967), and Weber (1965) provide excellent survey descriptions of the four compression classes.

Parametric extraction is a technique that uses irreversible transformations on the original data, and as a result, the compressed information is an approximation of the original data. As long as this loss of information can be tolerated, parametric extraction offers good compression ratios. Posner (1964), Schueffer (1961), and Marron and de Maine (1967) all use transformations that lose information within varying tolerances.

Adaptive sampling allows adjustment of the sampling rate automatically or on command. Rather than actually compressing data, adaptive sampling treats the data as either redundant to previously detected patterns or not significant compared to previous data. Since no new information exists when data points are redundant, adaptive sampling requires adjustment of the sampling rate to match perfectly the information rate of the data source.

Redundancy reduction is a compression technique for recognising and removing those data values that are repetitive. These data values can be calculated from preceding or succeeding values or by comparison with reference patterns. This method does not always permit reconstruction from compressed data to the original data. Adaptive sampling varies the sampling rate with the information rate of the data source resulting in no redundant data. Using a sampling rate greater than the information rate of the data source, redundancy reduction removes the redundant data from the sampled data to obtain data values corresponding to the information rate of the data source. The two general classes of redundancy reduction algorithms are polynomial predictors based on finite differencing techniques and polynomial curve fitting algorithms. Davisson (1967, 1968a, b), Ehrman (1967), and Elias (1955) have done much work in redundancy techniques.

The final major class of compression techniques, statistical encoding, transforms a given message into one or more code words. Wozencraft (1961, 1965), Fano (1963), and Savage (1966) have developed a convolution encoder/tree decoder which may or may not regenerate the original source depending on the success of the tree decoder (Slagle and Dixon, 1969; 1970; Slagle and Lee, 1971). Huffmann coding (1952) develops binary codes based on the frequencies of the various data points.

Most of the data compression research to date has been for telemetry systems use, with several exceptions. The first was Marron's (1967) ANPAK system which tolerated no loss of information and yet achieved a 39 per cent savings in storage on a test sample. Frisch (1971) reduced the information content of a data base to binary responses (TRUE/FALSE) to selected attributes. Schwartz and Kleiboener (1967) developed word tokens to replace multiple English words occurring in air traffic control language.

Design philosophy

With the advent of time-sharing computer systems many of the functions and services provided by libraries can be automated. Storage devices such as discs contain information, maintenance programs keep the information current, and query languages provide a means of retrieval. If a system can perform these tasks, it is known as an information storage and retrieval system (ISRS). If in addition it can detect trends in the data, it is called a management information system (MIS).

An ODCS must be designed to relieve the ISRS/MIS of the burden of physical file structure (list, tree, etc.). Allowing the

*University of Arkansas, Little Rock.
†College of Engineering, Texas A. & M. University, College Station, Texas 77843, USA.

ISRS/MIS to maintain logical items in logical files simplifies the ISRS/MIS program complexity. Another interface function is reducing the volume of space required to contain the data file thereby freeing storage devices for either other use or lower rental rates for fewer devices. The compression, applied on a record-by-record basis rather than on the entire file, results in fewer I/O requests for a given amount of data because more data is transferred per request. The interface must also attempt to reduce I/O requests by physically grouping items based on their usage frequency. Thus one I/O access may retrieve several items having a high probability of being requested next.

Whenever a large number of data items must be maintained, some fast look-up or address generator mechanism must be used to locate the desired information. Hashing algorithms as illustrated in **Fig. 1** map logical keys into uniformly distributed equivalence classes. Unfortunately, these relations do not always provide unique physical addresses. Intra-record collisions occur when the mapping generates the same address more than once and the item can be stored at the location generated; a pointer locates the item within the record. Inter-record collisions occur when no more space is available at the target address and the item must be chained to another record.

Even though the creation of activity affinity groups can reduce the number of accesses, further reduction may be accomplished by performing buffering. This buffering is not to be confused with that performed by the input/output control system of the computer. Instead, this buffering should be designed to retain in memory for a 'reasonable' length of time those physical records most frequently and last requested (least recently used algorithm). This should allow additional logical items to be requested but with fewer physical accessions. The relationship and organisation of the new ISRS/MIS is shown in **Fig. 2.**

System implementation

FORTRAN language

The funding for this project was supplied by the Department of Agricultural Economics and Rural Sociology at Texas A. & M. University. Various departmental standards and decisions required the FORTRAN language to be used to implement the ODCS interface. Although FORTRAN is primarily for computations and not character and file manipulations, the results presented are indicative of the interface and not the language performance.

System tables

The implementation of the interface requires several tables to define interface parameters, to locate data sets in the data base, to define and describe data sets in the data base, to reduce excessive I/O activity, and to locate individual items in a physical record.

A single table, system map (SYSMAP), initially defines the interface parameters for system generation. These parameters include the current time or date, the initial and the incremental size of the collision relocation list (CRL), the sizes of n-tuples used to build other tables, threshold values for changing item priorities, and threshold values for data set reorganisation. Several of the parameters, discussed later, in SYSMAP can be modified during program execution to allow the interface to adapt to its environment.

The definition of the tables used to locate the data sets depends on both system parameters and user specified parameters. The tables used for data set location include the available space for information map list (AVSIML), the data set information map list (DSIML), and the data set information map (DSIM). Tables associated with each data set include the data set statistics table (DSST), the frequent collision list of items (FCLSTI), the most frequent item list (MFIL), the record information field (RIF), the CRL, the item information field (IIF), and the item value area (IVA). Each of these tables is described in some detail below and summarised in **Table 1.**

The AVSIML is a push-down stack pointing to those DSIML n-tuples not in use. The DSIML is a linearly linked list pointing to the DSIM for a given data base. The DSIM in turn describes and defines a data set by containing parameters associated with the data set. Some of the data set parameters are specified by

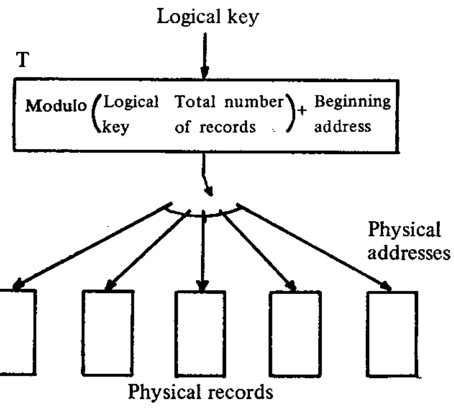

Fig. 1 Hashing transformation, T, uniformly mapping logical key to physical address.

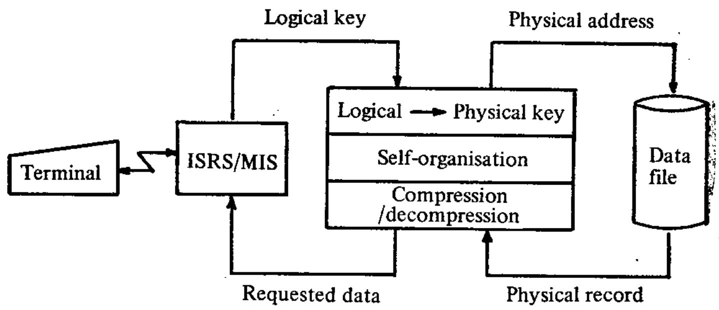

Fig. 2 ISRS/MIS interface system

Table 1 Table summary

Table name	*Contents*	*Function*
SYSMAP	system parameters; table sizes; threshold values	supplies parameters to DSIM; defines n-tuples
AVSIML	pointers	locates free space for DSIML n-tuples
DSIML	pointers	locates DSIM
DSIM	data set parameters; threshold values	defines record area boundaries; provides threshold values
DSST	statistics	provides synopsis of data set activity
MFIL	pointers	locates high activity items
FCLSTI	pointers	locates high activity relocated items
RIF	counters and pointers and reserve switch	records number and type of collisions; indicates if record is reserved; locates CRL, IIF, and IVA
CRL	pointers	locates relocated items
IIF	pointers and counters	describes and locates an item
IVA	data value	compressed storage area

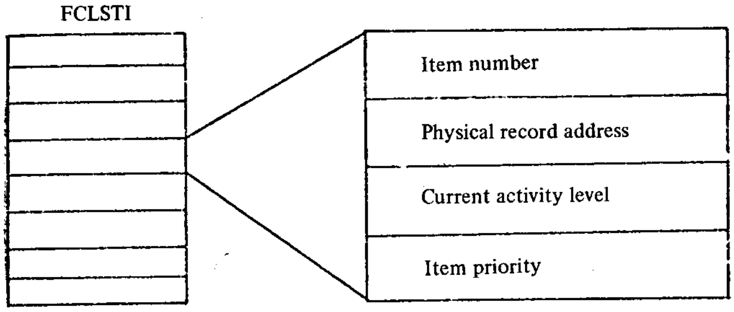

Fig. 3 FCLSTI table and four-tuple

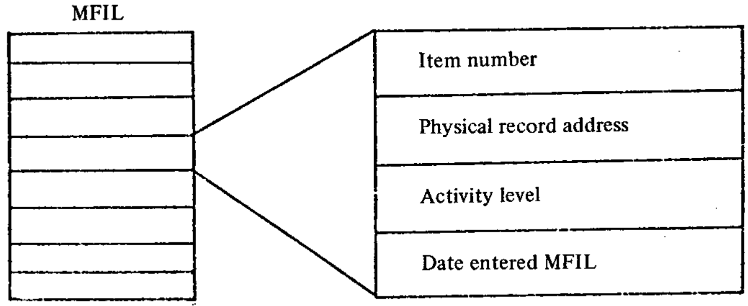

Fig. 4 MFIL table and four-tuple

the user while others are obtained from SYSMAP. User specified parameters include the acceptable time delay for updating, retrieving, and storing items in the data set; the beginning and ending physical record addresses for the initial storage area segregating the items by type; the beginning and ending physical address segregating the collision overflow area by item type; the beginning and ending physical record addresses for the reserved record area.

The FCLSTI, a set of four tuples, shown in **Fig. 3,** is used to indicate those relocated items that are requested most often. This is accomplished by placing only relocated items whose activity level is greater than the least active item in the list. The elements of the four-tuple are the item number, the number of the physical record containing the item, the current activity level of the item, and the item priority. Placing the record address in the four-tuple implies only one access is required to obtain the item. Consequently, the overall number of physical accessions is reduced because only one I/O request is required to retrieve those relocated items that have the greatest activity.

The MFIL, a set of four-tuples, illustrated in **Fig. 4,** is used to indicate those items in the data set having the greatest number of accessions. The entries in the four-tuple are the item number, physical address of the record containing the item, the item activity level, and the date the item entered the MFIL.

Placing the most active items in the MFIL permits relocating those items in a special area called the reserved record area. The objective is to physically group the most active items so that a single accession retrieves many items having a high probability of being requested next. This is the mechanism for implementing anticipatory I/O requests. While an item may be placed in the reserved area, it will not remain there unless it continues to remain one of the most active items in the data set. The reserved records can be constructed whenever a threshold for the number of calls to the system is exceeded. This threshold both allows a steady state to be reached and prevents excessive data set reorganisation. In addition to the call threshold, before an item can be placed in the MFIL, its activity must exceed a threshold percentage of a user specified activity level. Consequently, even though an item may be the most active item in the data set, it cannot enter the MFIL unless its activity exceeds this percentage level (implying that this threshold is crucial to successful and acceptable data set reorganisation). If the value is too high, few or no reserved records will be created. If the value is too low, excessive time will be spent creating reserved records. This situation is the same as that experienced with thrashing in a virtual memory management system using paging.

The final three tables, each contained in a record, comprise the physical records. The RIF is used to format that record, provide information about the number of inter- and intra-record collisions caused by hashing, provide information necessary to compress/decompress that record, and supply pointers locating the other tables in that record.

The CRL is the table used to process item relocation. Two types of relocation can occur. The first is inter-record collision while the second is reserved record creation. Item relocation results from either not being able to store the item on the record for lack of available space, compression time limit exceeded, inter-record collision, or temporarily grouping the most active items in the data set to facilitate anticipatory I/O accessions, i.e. reserved record creation. A CRL, a set of four-tuples shown in **Fig. 5,** exists, if at all, only on those records having relocated items.

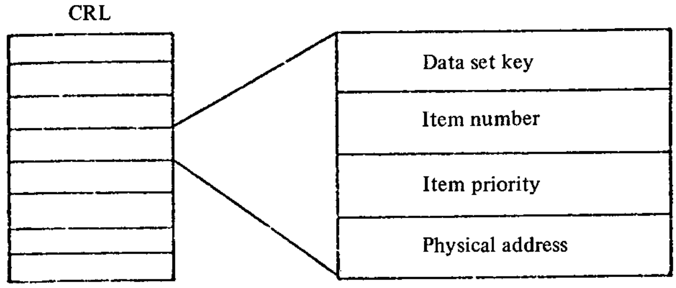

Fig. 5 CRL and CRL four-tuple

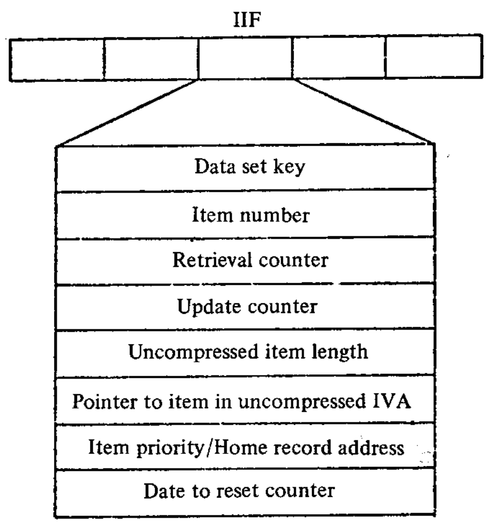

Fig. 6 IIF and IIF eight-tuple

The IIF, a set of eight-tuples, diagrammed in **Fig. 6,** handles intra-record collisions. That is, a one-to-one correspondence exists for each item in the physical record and the IIF eight-tuple. The data set key and item number uniquely identify the item. The next two entries are the retrieval and update counters to maintain the amount and type of activity associated with this term. The next value is the length of the item prior to compression. The pointer which follows the length is the location of the item after all the compressed data in the record has been decompressed. The next entry has two functions depending on whether or not the item has been relocated. If the item has not been relocated, i.e. at its home record, this entry is the priority.

If, on the other hand, the item has been relocated, this entry is the address of the home record. To distinguish between the two, the back pointer is stored as a negative number. The final element is the date after which the two activity counters are reset. By observing the value of the sum of the two counters, the activity level of the item can be tested to determine if the item priority should be increased, decreased, or should remain the same. The changing of item priority is discussed in detail later.

The last table to be described is the area containing the compressed data. The format, construction, and use of this table, IVA, is dependent on the compression/decompression algorithm. The particular algorithm used is discussed later.

Fig. 7 illustrates the logical table relationship for data base, as opposed to data set, maintenance. The dashed line indicates communication of parameters while the solid line indicates linking information. The two disc volumes shown in Fig. 7 are logical discs since they may physically be on one volume or require several volumes. Regardless of the physical size, the logical relationship remains the same. The SYSMAP supplies parameters to each DSIM which in turn contains parameters defining the boundary areas for the data records comprising a data set in the data base.

System areas

The system has parameters to define the size of the reserved and overflow areas. Lum (1971) presented tables indicating the expected percentage overflow for various load factors. The size of the reserved and overflow areas, used in the implemented system, was made arbitrarily to be equal to the size of the overflow area, as suggested by Lum.

All physical records were of a fixed size: 400 characters. While the system could handle variable size records, this size was selected simply to reduce program size, and to improve turn around times. To allow accurate statistics to be kept on disc accesses, the system did not put several logical records into a single physical record, although the system is capable of blocking logical records.

System operation

The system is first generated with a BLOCK DATA subroutine and data sets defined with both system- and user supplied parameters. Items which can be forced by type into partitioned areas may be entered, retrieved, or deleted. Items are relocated when one of the following occurs: insufficient space is available at the home record; the compression time was exceeded; the item activity forced a priority change; or the item enters the reserved area. The IIF handles intra-record collisions while the CRL controls inter-record collisions and item relocation into the reserved area. The system automatically changes item priorities and creates reserved records, both dependent on item activity. Excessive I/O activity is reduced by buffering and by using both the FCLSTI and reserved records.

To facilitate program development, maintenance, and modification, the system was written in modular subroutines. These program modules, with the exclusion of the main subroutine ODCS, can be divided into three main classes: buffering, compression/decompression, and relocation. The buffering routines are GETREC and PUTREC; the compression/decompression routines are TRYCMP and DECOMP; and the relocation routine is ANOTHR.

The purpose of the buffering routines, GETREC and PUTREC, is to maintain several physical records in memory to reduce the physical volume of I/O traffic. Records are read from disc when demanded by the system. Once in core, an activity counter is associated with each buffer in the buffer pool allowing frequently requested records to remain in core while other less-demanded records in the buffers can be replaced. Reading from disc into memory and passing the requested physical record to the other system routines is performed by GETREC. The routine PUTREC receives data from the processing routines and stores the data either in a buffer or on disc.

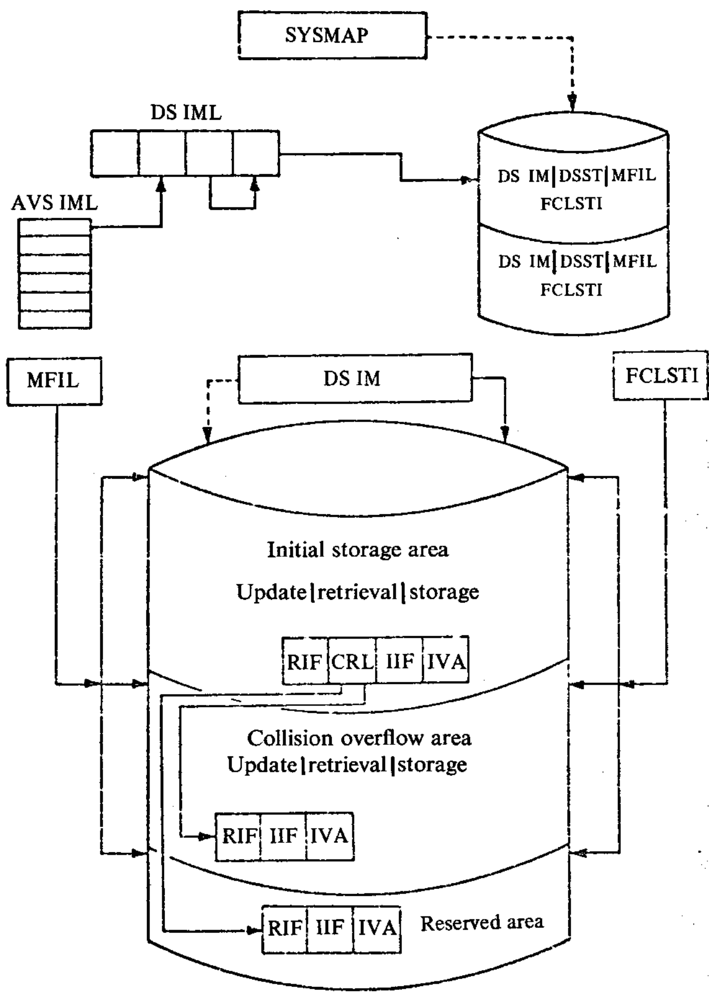

Fig. 7 Logical table relationship

uncompressed string
AAABCDEAAABBBB

repeat A	3 times
unique BCDE	4 words
34ABCDE	
repeat A	3 times
unique -	0 words
3A	
repeat B	4 times
unique -	0 words

compressed string
34ABCDE30A40B

Fig. 8 Compression algorithm to remove redundant words

The implemented compression algorithm consists of scanning a string of words (to be specific, on the IBM System/360, thirty-two bits per word), counting the number of consecutive equal words before the first unequal word, and counting the number of unique words preceding the next group of repeated words. This technique was used because of the constraints of FORTRAN. Since the entire program had to be written in FORTRAN only word (32 bits/360 word) manipulation was feasible. That is bit operations, while certainly possible, would be costly in program size, complexity, and execution time. Additionally the specific data base considered was such that the technique was acceptable though perhaps not optimal. **Fig. 8,** in which each letter represents a word, illustrates the technique. Two half-word counters are used to count the number of consecutive equal words and the number of consecutive unequal words respectively. The reason word manipulation is used is because FORTRAN, word-oriented, references words by variable names or array elements. Additionally, the large number of blanks in the Rural Sociology data allows word manipulation in FORTRAN to be an acceptable compression scheme. The decompression routine merely reverses the process using the counters to regenerate the data. The compression/decompression routines are TRYCMP and DECOMP respectively.

The relocation routine, ANOTHR, uses both the buffering and compression/decompression routines to relocate an item from its home record to another record. Depending on the parameters in the DSIM and the subroutine call arguments, the item is relocated in either the collision overflow area or the reserved area. Additionally, whenever an item is relocated, the two tables MFIL and FCLSTI must be examined to determine if these tables contain this item so that the entries can be updated. The relocation procedure is simply a sequential scan of the records in a predefined area. The scan stops whenever a record is located that can contain the item.

Changing item priority

The purpose of assigning priorities to items is to dictate the amount of time required to retrieve from the data set the desired information. That is, if *a priori* knowledge is available concerning the demand for the items comprising the data set, the items can be assigned priorities; the higher the priority the shorter the time acceptable to retrieve that item. For example, consider an airline company that might have its passenger reservations and personnel records in the same data set. Clearly, the needtoknow and the frequency of activity would be the greatest for the reservation rather than the personnel records. That is, the number of passengers on a particular flight would be requested much more frequently than the number of college graduates in the company. Consequently, in this situation priorities can be assigned to items prior to entering the information in the data set. A different example, in which less prior knowledge is available, might be a company selling stocks and bonds. The political events of the day might easily cause a set of stocks or bonds to be traded extensively. Consequently, a dynamic priority assignment would be needed to allow the retrieval time for stock quotations to be reduced as the volume of trade increases.

Dynamic priority modification is accomplished by maintaining an update counter and retrieval counter in the IIF n-tuple for each item in the data set. Also a date, in the IIF n-tuple, is used to trigger the process of determining whether the item's priority is to be increased, decreased, or remain the same. Based on SYSMAP parameters and threshold values, if item activity is sufficiently great, the item information will be entered into the MFIL and FCLSTI reducing future system overhead. This automatic, dynamic priority classification process allows items to be grouped physically by priority, reducing future I/O overhead.

Reserved records

The approach being followed by creating different organisational hierarchies is analogous to cache memory with the last area prior to main memory being the reserved area. The most active items in the data set regardless of priority reside in the reserved area. If the item activity drops below a threshold in the DSIM, the item is returned to its home record as determined by its priority.

The creation of reserved records is performed by initially entering into the MFIL only the most active items. Once the MFIL is 'sufficiently full', these items are then physically placed in the reserved area to prevent excessive overhead and to allow the system and data set to reach a steady state; the MFIL is not considered for dumping until after a threshold number of requests have been made of the system. Also, the MFIL cannot be dumped unless the number of items in the MFIL is above a certain threshold. The first threshold allows those items in the reserved area to remain for a period until a certain number of operations have been performed, possibly creating a new set of items in greatest demand. Consequently, after this threshold has been reached, if the number of items in the MFIL is above the second threshold, it is dumped. Clearly, these two threshold values, as well as the threshold values for priority assignment, determine to a large extent the amount of system overhead.

Item operation

After utilising a modular hashing function, dependent on item type, to determine the physical address of the item, the determination of the required operation is performed. The division technique (used as the hashing function) divides the key by the number of available addresses (an odd number), and uses the remainder as the key address. This technique was chosen based on a set of tables comparing the performance of several key transformation techniques (Lum, 1971).

If the operation is to enter an item, GETREC retrieves the desired record and DECOMP decompresses it. After searching the IIF table in the record to determine that no duplicate exists, TRYCMP attempts to put the item in the retrieved record. If successful, PUTREC restores the record. If the compression attempt is unsuccessful, either a CRL for that record is constructed, if one is not present, or the existing CRL is expanded, if full. If possible, the ANOTHR subroutine relocates the item resulting in updating the CRL and in restoring the record via PUTREC. If the item cannot be relocated or the CRL cannot be expanded, appropriate return codes are set and a return to the calling program is executed. After restoring the record following a successful item entry, the item is entered into FCLSTI if both relocated and more active than the least active entry in FCLSTI. Similarly, if the item is active enough, it is entered in the MFIL replacing the least active entry if necessary.

After examination of the appropriate thresholds, the MFIL is dumped to create reserved records. Using the address pointers in the MFIL n-tuples, GETREC retrieves and DECOMP decompresses the record containing the item to be placed in the reserved area. If the item is in its home record, it is deleted, is relocated by ANOTHR, the home record's CRL is updated, and the home record is restored via PUTREC. If the item, when retrieved by GETREC using the address supplied by the MFIL n-tuple, is determined to be a relocated item, the item is deleted from the relocation record, is relocated in the reserved area, the home record is retrieved, the CRL entry is updated, and the home record is restored. If at any time the creation of a reserved record is not successful (e.g. no available space), the procedure is halted and the item either is returned to the record from which it was obtained or is relocated in the nonreserved area.

If the operation is to retrieve an item, the FCLSTI is searched to obtain the physical address. If not located in the FCLSTI, the modular hashing function reveals the address. Regardless, GETREC retrieves the desired record, DECOMP decompresses it, and the IIF is searched for an item match. If found, the item is retrieved; if not found in the IIF, the CRL is searched for an item match to produce the relocation address. If the CRL points to the item, GETREC retrieves that record and the procedure is repeated. If the item is not locatable, return codes are set and a return to the calling program is executed. If retrieval is successful, priority change determination takes place depending on item activity, cutoff dates, and activity thresholds. If the priority is changed, the system deletes and re-enters the

Table 2 I/O accession summary

Data set	NIT-MFL	MFL-NTH	UPT-HRS	RTT-HRS	UDT-HRS	RDT-HRS	SDT-HRS	*Buffer records, Reserved records, Collisions* — *Result 1* — *Updates = retrievals* — *Accesses*	*% of 635*	*Buffer items, No reserved records, No collisions* — *Result 2* — *Updates = retrievals* — *Accesses*	*% of 600*	*Result 3* — *400 retrievals* — *Accesses*	*% of 400*
1	0·5	10	1	1	1	1	1	2072	326	480	80	80	20
2	0·5	10	1	1	1	1	100	1829	288	474	79	74	19
3	0·5	10	1	1	1	100	1	1920	302	472	79	72	18
4	0·5	10	1	1	100	100	1	1776	280	474	79	74	19
5	0·5	10	1	20	1	1	1	700	110	523	87	123	31
6	0·5	10	20	20	1	1	1	484	76	600	100	400	100
7	0·5	50	1	1	1	1	1	1196	188	603	101	203	51
8	0·5	50	1	1	1	1	100	1058	167	628	105	228	57
9	0·5	50	1	1	1	100	1	1158	182	643	107	243	61
10	0·5	50	1	1	100	100	1	1098	173	628	105	228	57
11	0·5	50	1	20	1	1	1	722	114	503	84	103	26
12	0·5	50	20	20	1	1	1	596	94	600	100	400	100
13	0·9	50	1	1	20	20	20	1006	158	633	106	233	58
14	0·9	50	1	1	20	20	40	960	151	608	101	208	52
15	0·9	50	1	1	30	30	30	960	151	683	114	283	71
16	0·9	50	1	1	50	50	50	1158	182	633	106	233	58
17	0·9	50	1	20	20	20	20	644	101	508	85	108	27
18	0·9	50	20	20	20	20	20	564	89	600	100	400	100
19	0·9	50	1	1	1	1	1	1028	162	643	107	243	61
20	0·9	50	1	1	1	1	100	1092	172	623	104	223	56
21	0·9	50	1	1	1	100	1	974	153	643	107	243	61
22	0·9	50	1	1	100	100	1	922	145	648	108	248	62
23	0·9	50	1	20	1	1	1	690	109	508	85	108	27
24	0·9	50	20	20	1	1	1	440	69	600	100	400	100
25								635	100	600	100	400	100

item, updating the priority status and frequency counters returned to the user. After completing the retrieval, the FCLSTI and MFIL checks are performed again.

If the operation is to delete an item, without searching the FCLSTI, the modular hashing function generates the address used by GETREC to retrieve the home record decompressed by DECOMP containing the item to be deleted. If the item is located via the IIF, it is deleted, the relocation record is retrieved, the item is deleted from that record, and any records modified are restored by PUTREC. Whenever an item is deleted, all the activity and priority information is returned with the item to facilitate re-entry if the item is to be modified. To complete deleting the item from the data set, any reference to the item in FCLSTI or MFIL is removed.

Deleting a single item from the data set begs the question, 'How do you delete a data set?' No restrictions are placed on the establishment of record area boundaries; moreover, no restrictions are set on data set boundaries. Consequently, multiple data sets could reside in the same physical set of records with a resulting mix of different data set items on a single record. If this mix of records is desirable, then the only way to delete an entire data set is to delete each item in that data set so as not to disturb other items from other data sets. Consequently, the system offers no mechanism for data set deletion en masse. Naturally, if the data sets are segregated, a separate program can be used to destroy all pointers in the system maps referencing that data set, followed by zeroing the information in the records. (The system expects the RIF field on the records used for the first time to be set to zero.)

Analysis and conclusions

System performance

Twenty-five data sets were generated for test purposes. Four hundred uniformly generated transformations, either retrieval or update, were performed because the transactions were uniformly generated. In all the data sets generated the initial two hundred entry operations were to generate the data set. The twenty-fifth data set was unique in that the results are indicative of an ISRS/MIS without an interface.

The summary table of I/O activity, appearing in **Table 2,** describes the accession perturbations, resulting from parametric variations, for record and item buffering. When records are buffered, the most active records are maintained in memory; whereas, when items are buffered, the most frequently requested items are kept in memory.

The ODCS was designed to improve overall performance. One of the expected improvements was a reduction in I/O traffic by maintaining a list, FCLSTI, of the most frequently referenced relocated items. The amount of reduction ranged from a low of twelve to a high of ninety-eight I/O accessions. The lower values were the result of changing the priority of an item on a collision chain and storing it on another record with no subsequent relocation. The higher values resulted from having a large number of items entering the reserved area and/or having the items on a collision chain. Regardless, the use of FCLSTI did reduce the number of I/O accessions.

Another expected improvement using the self-adaptive organisation is the automatic priority classification of items based on usage statistics. When records are buffered and no reserved records are created (e.g. data sets five, eleven, and seventeen), little or no advantage is apparent, because records are not kept dense with items as a result of the sequential scan used to locate free record space. If items were buffered, approximately a 20 per cent reduction in I/O activity (relative to data set twenty-five) could be produced. The reason for this reduction is because items that have a high request frequency would be stored in memory to create a working set data set. A trivial solution to determine the items to buffer in memory would be to set a low activity threshold so that high activity items would displace low activity items, resulting in a flurry of double accessions prior to reaching a steady state. An alternative to constructing these core resident records would be to require a sufficient number of disc accessions to each item to indicate its classification. Clearly, this is no waste of I/O requests since no reorganisation would yield the same results. The only additional I/O activity would be to retrieve items not active enough to be placed in the memory buffer area. Regardless of the priority determination procedure, using low threshold values presupposes *a priori* knowledge about the data set activity.

A major benefit expected of the interface is a reduction in the storage space required to store the data. Of the many compression techniques available, the FORTRAN algorithm was selected for economic reasons and Rural Sociology demands. This 'quick and dirty' word redundancy reduction procedure produced approximately 18 per cent savings in space because the data was generated with 90 per cent blanks. More advanced techniques (Marron and de Maine, 1967) should produce even greater compression. Applying this compression to the preceding discussion on priority determination means that by compressing the working set data set possibly additional items could be maintained in core.

The parameters modified during the testing procedure were the activity percentage for item entry into the MFIL (NITMFL); the number of operations occuring just prior to attempting reserved record creation (MFLNTH); the update (UPTHRS) and retrieval (RTTHRS) activity thresholds for changing item priority; and the cutoff time limits for update (UDTHRS), retrieval (RDTHRS), and storage (SDTHRS) items. Observing the statistical summary for data sets one, seven, and nineteen, the two most influential parameters were NITMFL and MFLNTH, with MFLNTH being the most important. Changing MFLNTH from ten to fifty decreased the number of accessions from 2072 to 1196 because requiring more transactions to transpire prior to attempting reserve record creation prevented unnecessary reorganisation. When the MFIL was sufficiently full and the required number of transactions had been performed, the MFIL was dumped regardless of whether the item in the MFIL was in the reserved area. Deleting an item from the reserved area and re-entering it is clearly wasted effort. Changing the value of NITMFL from 0.5 in data set seven to 0.9 in data set nineteen decreased the number of accessions from 1196 to 1028 because fewer reserved records were created as the threshold was increased even though the number of priority changes remained approximately constant. The remaining parameters 'fine tuned' the interface activity by determining the amount of effort allocated to modifying the item priorities. Increasing the value of UPTHRS and RTTHRS reduced I/O activity by forcing fewer priority changes. The I/O accessions eliminated were those required to perform the self-organisation of the data set. Increasing the parameters UDTHRS, RDTHRS, and SDTHRS reduced I/O traffic by extending the cutoff date after which an item is checked for item priority change. Once at a certain priority, the item remained in that classification for an extended period. Again the eliminated I/O was at the expense of self-organisation.

Attempting to establish these parameter settings requires some guidelines. If the data set is dynamic with the frequency of items varying rapidly over time, low values of UPTHRS, RTTHRS, UDTHRS, RDTHRS, and SDTHRS should be used to allow for quick reaction to the changing item priorities. If the transactions appear to reference the same set of items for an extended time period, then the UDTHRS, RDTHRS, and SDTHRS should be large to allow the items to remain at their respective priority level for longer periods. In addition, if information is known about those items to be requested and those less frequently requested, then the UPTHRS and RTTHRS values should be set to prevent the less frequently accessed items from being considered for reclassification. The setting of the UPTHRS and RTTHRS values should be based on the expected item activity level for those items referenced. The creation of reserved records is dependent on the values assigned to MFLNTH and NITMFL. Whenever an item is placed in the reserved area a minimum of two accessions is required; one to retrieve the item and one to restore the home record with an updated CRL, assuming the reserved record is in core. Constructing reserved records becomes an expensive process when low MFLNTH and NITMFL values are used. To emphasise, this extra activity is in addition to any activity required to change item prorities.

Modifications and extensions

The concept of reserved records as implemented was unsuccessful because items were both logically and physically relocated. The FCLSTI became full of items that were in the reserved area because of this relocation, thus defeating the purpose of that list. Also, reserved items were entered in the MFIL, causing wasted I/O to remove the item and re-enter it, requiring a minimum of three accessions to relocate an item in the reserved area to that area. The reserved area should be deleted entirely and telescoping item priorities should be used for reorganisation.

In addition to buffering, or possibly rather than buffering, the working set data set should be maintained in memory to substantially reduce I/O activity for those data sets that are primarily retrieval in nature and have requests directed over an extended period of item. A reduction in I/O would result for dynamic data sets as well, but the reduction would not be as great.

The first major modification would be to replace the sequential probe algorithm to locate free space for item relocation. One problem with this type of scan is demonstrated with the type of actions performed on the data set. A record may be full and subsequently have an item deleted, leaving free space. The sequential scan begins with the record last accessed by it to locate free space. If the record containing the deletion is 'behind' the scan, that space is not used until the end of the record area has been reached and the scan begins again. This is a problem when half of all transactions are deletions, thereby reducing the number of items per record and accordingly meaning an increase in I/O activity. If a list of free space were maintained, then better utilisation of these records could be achieved with a subsequent increase in the number of items per record and reduction in the number of I/O accessions.

Several major extensions are proposed to improve the interface. The first is to maintain statistics for determining the mean and standard deviation for the time to compress records. This could be used to aid in selection of a compression algorithm when several are available. For example, if in fact compression is indicated for a record then an algorithm can be selected by performing a test of hypothesis to determine if the algorithm is fast enough to apply to the record. Naturally the past performance of the algorithm would generate statistics for testing purposes. Similarly, accumulating means and standard deviations on the activity counters would allow the threshold values for item reclassification to be set and modified dynamically at execution time. For example, those items whose activity is within a certain distance of the mean should be put in a particular class. The use of these means and standard deviations for parameter re-evaluation at execution time leads immediately to self-optimising via linear or integer programming techniques. A set of equations could be developed such that the cost function would include the cost of storage, transmission speed, I/O accessions, delay time for the user, etc. and the constraints would put bounds on the compression time, number of accessions, delay time, etc. Since optimisation of the cost function should produce better system performance at reduced cost, tests of hypothesis could be applied to the statistics gathered during execution to determine if the system were not operating within established limits and thereby invoke the optimising routine.

Using multiple compression routines would allow selection of the 'best' techniques to compress a given record. By maintaining the history of previous applications of each compression algorithm to records in the data set, a test of hypothesis could be performed to determine if it should be considered as a candidate for compressing a record. Given a set of candidates, the routine selection could be performed by using game trees whose function could be based on the trade-off of the time spent to compress versus the amount of compression achieved. Consequently, even though extra effort would be spent selecting the appropriate algorithm the data set should be compressed more than if a single algorithm were applied to all the records.

Conclusions

The experimental results indicate that I/O traffic can be reduced by an internal assignment of items to various priority classes. Moreover, if the records containing these items can be kept dense (a function of the relocation routine), I/O traffic can be reduced further. If a memory area is set aside for storing the most active items, substantial I/O savings can be made. Naturally these results hold for data sets having a majority of the activity associated with a subset of that data set. The amount of reduction in I/O is a function of the number of items per record, which in turn is a function of the item length, track space, compression savings, and IIF space required.

The compression results are encouraging in that storage requirements can be reduced. However, compression systems in themselves are nothing new. The use of the compression algorithm with the self-adaptive capability to produce fewer I/O accessions is unique. Moreover, implementing both a self-optimising program to set parameters for acceptable compression limits and a tree searching program to select a particular algorithm would be unique.

Reserved records as used in this set of programs cannot be considered for any future implementation. Instead, a set of telescoping priority classes developing a working set data set should be used. The threshold levels, while presently manually set, could be set dynamically at execution time to improve system performance.

In summary, for data sets of thousands of items having a substantial activity on a subset of that data set, the application of an interface with suggested modifications would reduce I/O traffic, storage costs, and user delay time. Cache memory assignments can be easily and efficiently made for further reduction in cost and delay time. Finally, self-adaptive self-organising systems indicate performance improvement over strictly manually directed systems.

Bibliography

ALLEN, R. A. (1968). Omnibus: A Large Data Base Management System, AFIPS, *FJCC*, pp. 157-169.

AMIDON, E. L., and AKIN, G. S. (1971). Algorithmic Selection of the Best Method for Compressing Map Data Strings, *CACM*, Vol. 14, No. 12, pp. 769-774.

ANDREWS, C. A., and SCHWARZ, G. R. (1967a). Analysis and Simulation of Data Compression Techniques, *Proc. Nat. Telemetry Conf.*, pp. 57-64.

ANDREWS, C. A., DAVIS, J. M., and SCHWARZ, G. R. (1967b). Adaptive Data Compression, *Proc. IEEE*, Vol. 55, No. 3, pp. 267-277.

ANZELMO, F. D. (1971). A Data-Storage Format for Information System Files. *IEEE Trans. on Computers*, C-20, No. 1, pp. 39-43.

BACHMAN, C. W. (1969). Data Structure Diagrams, *Data Base*, Vol. 1, No. 2, Summer 1969, pp. 2-10.

BECHTOLD, W. R., and HOCHMAN, D. (1967). General Purpose Versus Special Purpose Computers for Data Compression, *WESCON* 11, pp. 1-11.

BLEIER, R. E. (1967). Treating Hierarchical Data Structures in the SDC Time-Shared Data Management System (TDMS), *Proc. ACM National Meeting*, pp. 41-49.

VAN BLERKOM, R., SCHWARZ, G. R., and WARD, R. J. (1968). An Adaptive Composite Data Compression Algorithm with Reduced Computation Requirements, *Proc. Nat. Telemetry Conf.* pp. 90-95.

BRADLEY, S. D. (1969). Optimizing a Scheme for Run Length Encoding, *Proc. IEEE*, Jan. 1969, pp. 108-109.

BREWER, S. (1968). Data Base or Data Maze? An Exploration of Entry Points, *Proc. ACM Nat. Conf.*, pp. 623-630.

CHAPIN, N. (1968). A Deeper Look at Data, *Proc. ACM Nat. Conf.*, pp. 631-638.

CODD, E. G. (1970). A Relational Model of Data for Large Shared Data Banks, *CACM*, Vol. 13, No. 6, pp. 377-387.

COFFMAN, E. G., and EVE, J. (1970). File Structures Using Hashing Functions, *CACM*, Vol. 13, No. 7, pp. 427-432.

CULHANE, J. L., and NETTLESHIP, R. (1968). Some Methods for the Compression of Binary Data in Spacecraft experiments, *IEEE Trans. Nucl. Sci.*, NS-15, pp. 3-13.

DAVISSON, L. D. (1967). An Approximate Theory of Prediction for Data Compression, *IEEE Trans.*, IT-13, No. 2, pp. 274-278.

DAVISSON, L. D. (1968a). The Theoretical Analysis of Data Compression Systems, *Proc. IEEE*, Vol. 56, No. 2, pp. 176-186.

DAVISSON, L. D. (1968b). Data Compression Using Straight Line Interpolation, *IEEE Trans.*, IT-14, No. 3, pp. 390-394.

DENNING, P. J., and SCHWARTZ, S. C. (1972). Properties of the Working-Set Model, *CACM*, Vol. 15, No. 3, pp. 191-198.

DODD, G. C. (1969). Elements of Data Management Systems, *Computing Surveys*, Vol. 1, No. 2, pp. 117-133.

DOWNING, J. J., SMITH, W. E., and STUBBLES, J. E. (1968). General Purpose Telemetry Data Compression, *WESCON Tech. Papers*, sess. 6, pp. 1-12.

EHRMAN, L. (1967). Analysis of Some Redundancy Removal Bandwidth Compression Techniques, *Proc. IEEE*, Vol. 55, No. 3, pp. 278-287.

ELIAS, P. (1955). Predictive Coding, *IRE Trans.*, IT-1, pp. 16-33.

FANO, R. M. (1963). A Heuristic Discussion of Probabilistic Decoding, *IEEE Trans.*, IT-9, pp. 64-84.

FRASER, A. G. (1967). Data Compression and Automatic Programming, *The Computer Journal*, Vol. 10, No. 2, pp. 165-167.

FRISCH, J. (1971). Bit Vectors Vitalize Data Retrieval, *Data Dynamics*, Aug. 1971, pp. 37-42.

FRY, J. P. (1971). Managing Data is the Key to MIS, *Computer Decision*, Vol. 3, No. 1, Jan. 1971, pp. 6-13.

GALLAGER, R. G. (1968). *Information Theory and Reliable Communication*, New York: John Wiley & Sons.

GARDENHIRE, L. W. (1964). Redundancy Reduction The Key to Adaptive Telemetry, *Nat. Telemetry Conf.*, June 1964, pp. 1-16.

GARNER, H. L. (1967). Information Theory and Codes, *Digital Computer Handbook*, New York, McGraw-Hill, pp. 29-62.

GATES, H. M., and BLIZARD, R. B. (1970). Coding/Decoding for Data Compression and Error Control on Data Links Using Digital Computers, *Proc. AFIPS*, FJCC, pp. 503-514.

GOLOMB, S. W. (1966). Run-Length Encodings, *IEEE Trans.*, IT-12, July 1966, pp. 399-401.

GOODMAN, L. M. (1967). A Binary Linear Transformation for Redundancy Reduction, *Proc. IEEE* (Lett.), Vol. 53, March 1967, pp. 467-468.

HUFFMAN, D. A. (1952). A Method for the Construction of Minimum Redundancy Codes, *Proc. IRE*, Vol. 40, pp. 1098-1101.

HUMPHREY, A. L., and MUNRO, W. G. (1970). Management Information Retrieval, *The Computer Journal*, Vol. 13, No. 2, pp. 127-130.

ISODA, S., GOTO, E., and KIMURA, I. (1971). An Efficient Bit Table Technique for Dynamic Storage Allocation of 2^n-word Blocks, *CACM*, Vol. 14, No. 9, pp. 589-592.

KLOTZ, L. A., and DUFFIN, P. M. (1968). Remote-Site Data Compression, *Proc. Nat. Telemetry Conf.*, pp. 96-102.

KORTMAN, C. M. (1967). Redundancy Reduction—A Practical Method of Data Compression, *Proc. IEEE*, Vol. 55, No. 3, pp. 253-263.

KREUTZES, P. J. (1971). *Data Compression in Large Business-Oriented Files*, AD 734394, Navy Fleet Material Support Office, Mechanicsburg, Pa.

LANG, C. A., and GRAY, J. C. (1968). ASP—A Ring Implemented Association Structure Package, *CACM*, Vol. 11, No. 8, pp. 550-555.

LANGEFORS, B. (1961). Information Retrieval in File Processing I, *BIT*, Vol. 1, pp. 103-112.

LANGEFORS, B. (1961). Information Retrieval in File Processing II, *BIT*, Vol. 1, pp. 103-112.

LANGEFORS, B. (1963). Some Approaches to the Theory of Information Systems, *BIT*, Vol. 3, pp. 229-254.

LANGEFORS, B. (1965). Information System Design Computation Using Generalized Matrix Algebra, *BIT*, Vol. 5, pp. 96-121.

LEFKOVITZ, D. (1969). *File Structures for Online Systems*, New York: Spartan Books.

LITTLE, J. L., and MOOERS, C. N. (1968). Standards for User Procedures and Data Formats in Automated Information Systems and Networks, AFIPS, *SJCC*, pp. 89-94.

LIU, H. (1968). A File Management System for a Large Corporate Information System Data Bank, AFIPS, *FJCC*, pp. 145-156.

LOH, G. M. (1967). Mechanization of a Digital Compressor for Biomedical Data, *IEEE WESCON*, pp. 1-12.

LOWE, T. C. (1968). The Influence of Data Base Characteristics and Usage on Direct Access File Organization, *JACM*, Vol. 15, No. 4, pp. 535-548.

LUCKY, R. W. (1968). Adaptive Redundancy Removal in Data Transmission, *Bell System Techn. J.*, April 1968, pp. 549-573.

LUM, V. Y., YUEN, P. S. T., and DODD, M. (1971). Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files, *CACM*, Vol. 14, No. 4, pp. 228-239.

LYNCH, T. J. (1966). Sequence Time Coding for Data Compression, *Proc. IEEE*, Oct. 1966, pp. 1490-1491.

MADNICK, S. E., and ALSOP, J. W. (1969). A Modular Approach to File System Design, AFIPS, *SJCC*, pp. 1-13.

MARRON, B. A., and DE MAINE, P. A. D. (1967). Automatic Data Compression, *CACM*, Vol. 10, No. 11, pp. 711-715.

MORRISON, W. L., HOGAN, W. P., and PENTZ, R. M. (1964). Application of Data Compression, *Proc. Nat. Telemetry Conf.*, sess. 1, Paper 3.

POSNER, E. C. (1964). The Use of Quantiles for Space Telemetry Data Compression, *Proc. Nat. Telemetry Conf.*, sess. 1, paper 3.

PRICE, C. E. (1971). Table Lookup Techniques, *Computing Surveys*, Vol. 3, No. 2, pp. 49-65.

RADUCHEL, W. J. (1970). Efficient Handling of Binary Data, *CACM*, Vol. 13, No. 12, pp. 758-759.

RANGEL, R. G. (1968). A Design Approach to User Customized Information System, AFIPS, *FJCC*, pp. 171-177.

RAY-CHAUDHURI, D. K. (1968). Combinatorial Information Retrieval Systems for Files, *SIAM J. Appl. Math.*, Vol. 16, No. 5, pp. 973-992.

SAVAGE, J. E. (1966). Sequential Decoding—The Computation Problem, *Bell System Tech. J.*, Jan. 1966, pp. 149-175.

SCHUEFFER, D. H. (1961). Logarithmic Compression of Binary Numbers, *Proc. IRE*, Vol. 49, pp. 1219.

SCHORR, H., and WAITE, W. M. (1967). An Efficient Machine Independent Procedure for Garbage Collection in Various List Structures, *CACM*, Vol. 10, No. 8, pp. 501-506.

SCHWARTZ, E. S., and KLEIBOENER, A. J. (1967). A Language Element for Compression Coding, *Information and Control*, Vol. 10, No. 8, pp. 315-333.

SCHWARTZ, E. S., and KALLICK, B. (1964). Generating a Canonical Prefix Encoding, *CACM*, Vol. 7, No. 3, pp. 166-169.

SCHWARTZ, M. H. (1970). MIS Planning, *Datamation*, Vol. 16, No. 10, pp. 28-31.

SHANNON, C. E. (1948). A Mathematical Theory of Communication, *Bell System Techn. J.*, Vol. 27, No. 3, pp. 379-423.

SLAGLE, J. R., and DIXON, J. K. (1969). Experiments with Some Program That Search Game Trees, *JACM*, Vol. 16, No. 2, pp. 189-207.

SLAGLE, J. R., and DIXON, J. K. (1970). Experiments with the M & N Tree-Searching Program *CACM*, Vol. 13, No. 3, pp. 147-154.

SLAGLE, J. R., and LEE, R. C. T. (1971). Application of Game Tree Searching Techniques to Sequential Pattern Recognition, *CACM*, Vol. 14, No. 2, pp. 103-110.

SNIVELY, J. W. (1965). A Bit Saving Encoding Scheme for a Set of Monotonic Numbers, *Proc. IEEE*, Vol. 53, May 1965, pp. 544-545.

STEEL, T. B. (1964). Beginning of a Theory of Information Handling, *CACM*, Vol. 7, No. 2, pp. 97-103.

TZANNES, N. S., SARANTAKIS, A. J., and FAN, E. H. (1970). Entropy and Data Compression, *IEEE Trans.*, IT-16, No. 1, pp. 74-75.

WEBER, D. R. (1965). A Synopsis on Data Compression, *Proc. Nat. Telemetry Conf.*, sess. 1, pp. 9-16.

WEIMMEISTES, C. J. (1971). The Science of Information Management, *Computers and Automation*, Vol. 20, No. 4, pp. 20-24.

WILKINS, L. C., and WINTZ, P. A. (1971). Bibliography on Data Compression, Picture Properties, and Picture Coding, *IEEE Trans.*, Vol. 1-17, No. 2, March 1971, pp. 180-196.

WILLIAMS, J. G. (1971). Storage Utilization is a Memory Hierarchy When Storage Assignment is Performed by a Hashing Algorithm, *CACM*, Vol. 14, No. 3, pp. 172-175.

WILLIAMS, R. (1971). A Survey of Data Structures for Computer Graphics Systems, *Computing Surveys*, Vol. 3, No. 1, pp. 1-21.

WODON, P. L. (1969). Data Structure and Storage Allocation, *BIT*, Vol. 9, pp. 270-282.

WOZENCRAFT, J. M., and REIFFEN, B. (1961). *Sequential Decoding*, New York: MIT Press and John Wiley & Sons.

WOZENCRAFT, J. M., and JACOBS, I. M. (1965). *Principles of Communication Engineering*, New York: John Wiley & Sons.