Analysis of self-indexing, disc files

S. J. Waters

LSE, Houghton Street, London WC2

This paper discusses the simplest, and often 'best', disc file organisation technique that supports all processing modes. Timing formulae are derived for each processing mode and are then compared to isolate significant parameters; once again, it is proven that generalisations can be misleading.
(Received July 1974)

This paper is further 'fall-out' from the CAM research project at the London School of Economics; this research is investigating computer-aided methods of developing computer-based, information processing systems. The project, outlined by Waters (1972a) is financed by the Science Research Council. The current area of research is computer systems design and attempts are being made to formalise manual methods as a necessary first step towards developing computer-aided methods.

Dodd (1969) surveys the extensive literature and experience of file organisation and processing techniques. Unfortunately, this literature is mainly qualitative and lacks such vital, quantitative aspects as estimating file sizes and processing times. Further, simple file organisation techniques are often ignored or summarily discarded as 'freak'.

Probably the simplest yet most underused technique is a SID (i.e. self-indexing, disc) file. This paper discusses the organisation and evaluation of a SID file and derives and compares timing formulae for the main processing modes. Once again, simple rules-of-thumb are shown to be so dangerous that they should be positively ignored. Instead, each particular case should be analysed on its own merits.

Organisation of a SID file

Waters (1974) discusses the various file organisation techniques that permit key retrieval of records (i.e. a record is located from its unique key number). Of these techniques, only three effectively support all of serial, skip-serial, sequential, skip-sequential and random processing modes; these are indexed sequential, indexed random and algorithmic sequential.

The simplest algorithmic sequential file organisation technique is undoubtedly self-indexing (sometimes termed direct-addressing or table look-up). In this case, the algorithmic function in

$$\text{Record position} = \text{Function (Record key number)}$$

is unity so that the record position equals its key number (e.g. the nth record in the file has key number n). The physical location of the record is given by

$$\text{Track number, } T = t_0 + \frac{n}{t} \text{ rounded down}$$

$$\text{Cylinder number, } C = \frac{T}{c} \text{ rounded down}$$

where

t_0 = absolute number of the first track in the file,

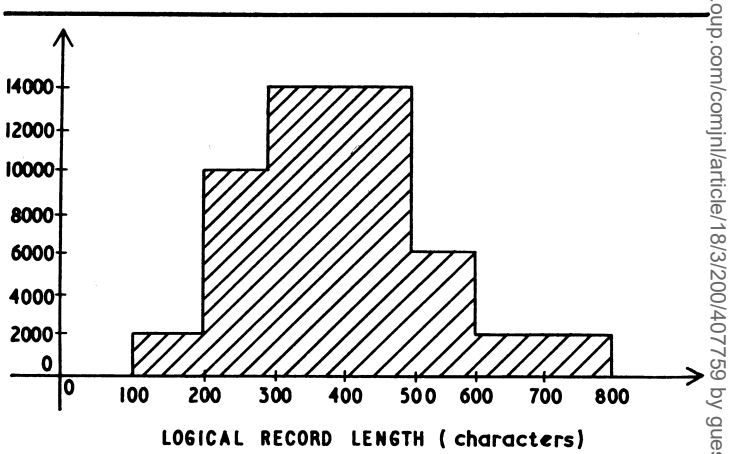
t = number of (fixed length) records per track, and

c = number of tracks per cylinder,

assuming disc numbering commences from zero.

Thus, a SID file is sequential and a record space is allocated on disc for every key number, irrespective of whether it is used or not. Old records are deleted by clearing their space and new records are inserted by checking that their space is clear and then setting up their new contents.

Harvey (1970) and others suggest that SID files only support fixed length records, probably because they are over-influenced by restrictive software. However, the record segmentation technique can be employed to split a variable length, logical record into a variable number of fixed length, physical records. Thus, a logical record consists of one header record and zero, one or more trailer records; the header record resides in the 'home' position of the key number and any trailer records can reside in file overflow areas (which may be embedded into the 'home' track and cylinder, by adjusting t and c above, and/or extended into cylinders at the end of the file); chaining can be used to retrieve trailer records. Clearly, the fixed physical record length is carefully chosen to reduce the number of trailer record accesses without unduly increasing the file size, due to unused space at the end of smaller records; **Fig. 1** illustrates the data for making such a choice of record length.



Physical record length (characters)	*Number of trailer records*	*Minimum* file size (characters)*	*Possible number of accesses per logical record*
100	176,000	22,600,000	4·52
200	76,000	25,200,000	2·52
300	42,000	27,600,000	1·84
400	24,000	29,600,000	1·48
500	10,000	30,000,000	1·12
600	4,000	32,400,000	1·08
700	2,000	36,400,000	1·04
800	0	40,000,000	1·00

*This must be increased by such overheads as unused overflow areas and unused physical record spaces.

Fig. 1 A typical 'space-time' conflict in choosing physical record length for a SID file with variable length, logical records

(*a*) Histogram indicating the distribution of logical record lengths for a file of 50,000 records

(*b*) Table approximating the corresponding file sizes and accesses for varying physical record lengths

Alternatively, any trailer records to a single header record can sometimes be combined into a single, variable length trailer record which is chained to an overflow area.

The following analysis is restricted to the more usual SID files having fixed length records.

Evaluation of a SID file

Waters (1974) suggest twelve objectives (and constraints) of computer systems design against which any design or technique should be evaluated. Using this framework, a SID file can be compared against its direct alternatives as follows.

Efficiency

A SID file is highly efficient with respect to disc time as any record can be key-retrieved by a single disc access; indexed files usually require extra accesses to search (at least part of) the index and to retrieve any overflow records; algorithmic random files usually require extra accesses to retrieve any synonym records. Further, periodic disc reorganisation is not necessary for a SID file.

A SID file is highly efficient with respect to CPU time as the simplest algorithm is used and index searching is not required.

However, a SID file is only efficient with respect to disc storage space if used key numbers are densely allocated (e.g. as with an insignificant code). If used key numbers are sparsely allocated (e.g. as with a significant code), then wasted space may be prohibitive; however, this space can sometimes be used to store transient (e.g. print) files.

Timeliness

If a SID file cannot meet the turnaround/response time constraints, then neither will an indexed nor algorithmic random file.

Security

The usual generation, dumping and duplication methods of file security, if used wisely, can guarantee a SID file.

Accuracy

A SID file is as accurate as any alternative, assuming appropriate controls are applied.

Compatibility

A SID file can be compatible with different subsystems, particularly as it supports all processing modes.

Implementability

A SID file is extremely simple to implement in most cases where software is not too restrictive.

Maintainability

Subsequently, this simplicity should not pose any additional maintenance problems.

Flexibility

Fig. 2 indicates that a SID file supports all processing modes and is therefore highly flexible; an algorithmic random file does not support sequential processing.

Robustness

A SID file is robust, provided the maximum key number has been accurately estimated, to provide room for file growth. Indexed and algorithmic random files are often sensitive to the volatility and overflow problems outlined in Waters (1972).

Portability

A SID file is usually portable from one hardware/software configuration to another. In particular, Fig. 2 indicates that a self-indexing file is relevant both to direct and random access devices whereas an indexed sequential file is not relevant to random access devices (because nothing is gained by searching indexes instead of the file itself, as access time is independent of data position).

Acceptability

A SID file should meet most sensible systems design standards.

Economy

Assuming dense key numbers, a SID file is cost-effective because it is highly efficient and simple to implement and maintain. Further, its additional benefits of flexibility, robustness and portability can cope with dynamic systems requirements.

Thus a SID file, unlike its usual alternatives, satisfies all design objectives when key numbers are dense. This condition implies insignificant key numbering which is often regarded as inconveniencing the user. However, this need not be true since:

1. Users of computer output can be supplied with both the insignificant code and its significant version or description, which is extracted from the SID file.
2. Users of 'predictable' computer input are often supplied with preprepared or computer turnaround documents which can include both significant and insignificant codes.
3. Users of 'unpredictable' computer input must supply the code themselves. If this involves looking up an on-line or off-line directory, even if only to establish a check-digit, then this directory can include the insignificant code.

Thus, it is probably true that insignificant codes, and therefore SID files, are under-used in practice. It is certainly true that indexed sequential files are widely over-used in practice and often fail to meet the above design objectives, sometimes with catastrophic results!

Parameters of a SID file

A SID file can be read and/or written either serially, skip-serially or randomly. Numerous variables are necessary to

File organisation		*Access device*		
Access method	*Sequence*	*Serial (e.g. tape)*	*Direct (e.g. disc)*	*Random (e.g. core)*
Search	Sequential	√	√	√
Search	Random	√	√	√
Index	Sequential	×	√	×
Index	Random	×	√	√
Algorithm	Sequential	×	√	√
Algorithm	Random	×	√	√

File organisation		*Processing mode*				
Access method	*Sequence*	*Serial*	*Skip-serial*	*Sequential*	*Skip-sequential*	*Random*
Search	Sequential	√	×	√	×	×
Search	Random	√	×	×	×	×
Index	Sequential	√	√	√	√	√
Index	Random	√	√	√	√	√
Algorithm	Sequential	√	√	√	√	√
Algorithm	Random	√	√	×	√	√

*The simplest algorithmic sequential file is a SID file.

Fig. 2 File organisation method versus supporting access devices versus supported file processing modes from Waters (1974)

derive timing formulae, usually as follows (asterisked parameters being independent variables).

1. *Parameters defining the self-indexing file*

*1.1 Fixed record length = R characters

*1.2 Number of records in block (integral) = B ($\geqslant 1$)

1.3 Fixed block size = BR characters, from 1.1 and 1.2

*1.4 Number of records in file = N

1.5 Number of blocks in file = $\frac{N}{B}$ from 1.2 and 1.4

1.6 File size = NR characters, from 1.1. and 1.4

*1.7 Number of used records in file = M ($\leqslant N$)

1.8 File packing density = $\frac{M}{N}$ ($\leqslant 1$), from 1.4. and 1.7.

2. *Parameters defining the file accessing during a run*

*2.1 Total number of accesses to file = T

*2.2 Number of (used) records hit = H ($\leqslant T$ and $\leqslant M$)

2.3 Record hit ratio = $\frac{H}{M}$ ($\leqslant 1$), from 1.7. and 2.2

2.4 Fan in/out ratio = $\frac{T}{H}$ ($\geqslant 1$), from 2.1 and 2.2

*2.5 Number of (consecutive) records in hit group = G ($< H$)

*2.6 Number of accesses to hit group records = S ($\leqslant T$)

2.7 Since a hit group has high activity, by definition, then $S/G > T/N$, from 1.4, 2.1, 2.5 and 2.6

2.8 Number of blocks hit = J
2.9 Number of cylinders hit = K
} discussed in Waters (1975)

3. *Parameters defining the (common) disc pack*

*3.1 Fixed track length = l characters ($\geqslant BR$), from 1.3

3.2 Number of blocks in track = $n = l/BR$ ($\geqslant 1$), from 1.3 and 3.1

*3.3 Number of tracks in cylinder (integral) = c

3.4 Number of cylinders in file = NR/cl, approximately, from 1.6, 3.1 and 3.3

*3.5 Seek time (i.e. arm movement time) and Random Seek time = f(cylinders traversed) ms = f_r ms.

*3.6 Revolution time = r ms.

*3.7 Transfer rate = t kc

3.8 If transfer of data is dominated by revolution speed, then $t = \frac{l}{r}$, approximately, from 3.1, 3.6 and 3.7

*3.9 Track searching method = s
(0 if a search commences from any point on the track, 1 if a search commences from the track start point)

3.10 Latency time (i.e. rotational delay time) = Lms
$= \frac{r}{2}\left(1 + \frac{n-1}{n}s\right)$ ms, from Appendix

*3.11 'Read-after-write check' indicator = v
(0 if no check, 1 otherwise)

*3.12 'Disc dedicated to file' indicator = d
(0 if file is the only one processed on the disc, 1 otherwise)

Thus, a large number of parameters contribute to the subsequent timing formulae and the above analysis is by no means exhaustive. This complexity offers some excuse for designers relying on simple rules-of-thumb but they do so at their peril! Any parameter can be critical to the effectiveness of file organisation and processing.

Timing formulae for a SID file

In a companion paper, Waters (1975) discusses techniques for estimating disc seek times for serial, skip-serial and random processing modes. The following SID file formulae supplement this by concentrating on other disc timing aspects.

Serial processing

The serial read time is given by

$\underline{SRT}$ = Seek time + Latency time + Read time

= [Number of cylinders in file × Single-cylinder seek time
or
Number of blocks in file × Random seek time, depending on whether disc is dedicated or not]
+ [Number of blocks in file × Latency time]
+ [Number of characters in file ÷ Transfer rate]

$$= \left[(1-d)\frac{NR}{cl}f(1) + d\frac{N}{B}f_r\right] + \left[\frac{N}{B}L\right] + \left[\frac{NR}{t}\right] ms$$

$$\simeq N\left[\frac{d}{B}f_r + \frac{L}{B} + \frac{R}{t}\right] ms$$

since from Waters (1975), dedicated serial seeks are usually insignificant for a single file pass.

The serial write time is given by

$\underline{SWT}$ = [Serial read time]
+ [Number of blocks in file × Read-after-write revolution time, if check is applied]

$$= [\underline{SRT}] + \left[v\frac{N}{B}r\right] ms$$

$$\simeq N\left[\frac{d}{B}f_r + \frac{L}{B} + \frac{R}{t} + \frac{vr}{B}\right] ms$$

The serial update time is given by

$$\underline{SUT} = \underline{SRT} + \underline{SWT} \simeq 2N\left[\frac{d}{B}f_r + \frac{L}{B} + \frac{R}{t} + \frac{vr}{2B}\right] ms$$

Skip-serial processing

The skip-serial read time is given by

$\underline{SSRT}$ = Seek time + Latency time + Read time

= [Number of cylinders accessed × Skipped-cylinders

seek time

or

Number of blocks accessed × Random seek time, depending on whether disc is dedicated or not]

+ [Number of blocks accessed × Latency time]

+ [Number of blocks accessed × Block transfer time]

$$= \left[(1-d)\,Kf\left(\frac{NR}{cl\ (K+1)}\right) + dJf_r\right]$$

$$+ [JL] + \left[J\frac{BR}{t}\right] ms$$

$$\simeq J\left[df_r + L + \frac{BR}{t}\right] ms$$

since, from Waters (1975), dedicated skip-serial seeks are usually insignificant for a single file pass.

The skip-serial write time is given by

$\underline{SSWT}$ = [Skip-serial read time]

+ [Number of blocks accessed × Read-after-write revolution time, if check is applied]

$$= [\underline{SSRT}] + [vJr]\,ms$$

$$\simeq J\left[df_r + L + \frac{BR}{t} + vr\right] ms$$

The skip-serial update time is given by

$$\underline{SSUT} = \underline{SSRT} + \underline{SSWT}$$

$$\simeq 2J\left[df_r + L + \frac{BR}{t} + \frac{vr}{2}\right] ms$$

Random processing

The random read time is given by

$\underline{RRT}$ = Seek time + Latency time + Read time

= Number of accesses to file × [Random seek time + Latency time + Block transfer time]

$$= T\left[f_r + L + \frac{BR}{t}\right] ms$$

Although it can be argued that the number of blocks accessed randomly is less than the number of accesses to the file, since consecutive accesses may hit the same block, the difference is usually negligible, being $\frac{TB}{N}$ approximately.

The random write time is given by

$\underline{RWT}$ = [Random read time]

+ [Number of accesses to file × Read-after-write revolution time, if check is applied]

$$= [\underline{RRT}] + [vTr]\,ms$$

$$= T\left[f_r + L + \frac{BR}{t} + vr\right] ms$$

The random update time is given by

$\underline{RUT}$ = $\underline{RRT}$ + $\underline{RWT}$

− [Number of accesses to file × Random seek time, if disc is dedicated]

$$= 2T\left[f_r + L + \frac{BR}{t} + \frac{vr}{2}\right]$$

$$- [(1-d)\,Tf_r]\,ms$$

$$= T\left[(1+d)\,f_r + 2L + \frac{2BR}{t} + vr\right] ms\cdot$$

Clearly, the complexity of the above timing formulae emphasise that each particular case should be considered on its own merits and that generalisations are dangerous.

Serial versus random updating for a SID file

Waters (1972) qualitatively denounced the 'record hit ratio' rules-of-thumb which are essentially based on serial versus random processing. The following disc time comparisons add quantitative evidence in a final attempt to convince the 'diehards'.

Probably the most common situations are given by

$$d = s = v = 0$$

whereby a dedicated disc searches from any point in a track and read-after-write check options are not taken. These conditions yield

$$\underline{SUT} = 2N\left[0 + \frac{r}{2B} + \frac{R}{t} + 0\right] ms$$

$$= N\left[\frac{r}{B} + \frac{2R}{t}\right] ms\,,$$

and

$$\underline{RUT} = T\left[f_r + 2\frac{r}{2} + \frac{2BR}{t} + 0\right] ms$$

$$= T\left[f_r + r + \frac{2BR}{t}\right] ms$$

If the SID file is pre-defined so that block size (BR) is fixed for all processing modes, then

$$\underline{RUT} = T\left[f_r + \frac{B}{N}\underline{SUT}\right] ms$$

$$> \underline{SUT} \text{ if } \frac{TB}{N} > 1$$

whatever the value of f_r (even if it is zero, as for a fixed-head drum or disc). Notice that in this condition favouring serial updating

$$\frac{TB}{N} = \frac{\text{Number of accesses to file}}{\text{Number of blocks in file}}$$

$$= \frac{T}{H}\frac{H}{M}\frac{M}{N}B$$

= Fan in/out ratio × Record hit ratio × File packing density × Number of records in block

Thus, record hit ratio and file packing density (both being less than or equal to unity) have similar effects on the condition whereas fan in/out ratio and number of records in block (both being equal to or greater than unity) have similar counter-effects. For example, even if the record hit ratio is very low (e.g. 1%) for a moderately packed SID file (e.g. 80% file packing density) with average block size (e.g. 10 records per block), then serial updating is faster than random updating for common fan in/out ratios (e.g. >12·5).

Further, $\underline{RUT} \gtrless \underline{SUT}$

if

$$T\left[f_r + r + \frac{2BR}{t}\right] \gtrless N\left[\frac{r}{B} + \frac{2R}{t}\right]$$

i.e. if

$$\frac{TB}{N} \gtrless \left[\frac{2BR + tr}{2BR + t(f_r + r)}\right]$$

which is a function, indicated by **Fig. 3**, of the block size and the disc. For example, let the SID file occupy a common disc pack typified by

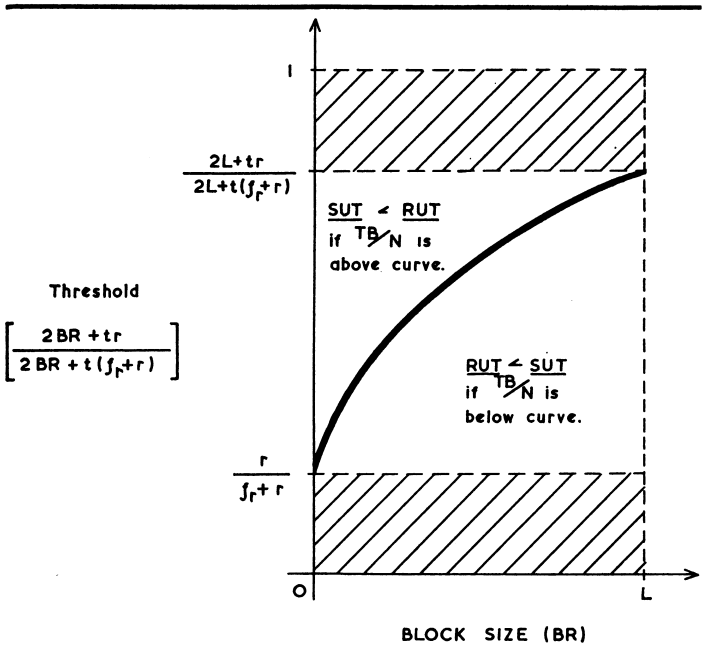


Fig. 3 Graph of threshold between serial and random updating for a pre-defined SID file

l = 4,000 characters per track
c = 10 tracks per cylinder
f_r = 75 ms random seek time
r = 20 ms per revolution, and
t = 200k characters per second

noticing that relation 3.8 holds, then

$$\underline{RUT} \gtrless \underline{SUT} \text{ if } \frac{TB}{N} \gtrless \frac{BR + 2000}{BR + 9500}$$

$$\therefore \underline{RUT} > \underline{SUT} \text{ whenever } \frac{TB}{N} > \frac{4}{9} \text{ since } BR \leqslant 4{,}000 \text{ from 3·1}$$

and $\underline{RUT} < \underline{SUT}$ whenever $\frac{TB}{N} < \frac{4}{19}$ since $BR > 0$ ·

Notice that TB/N is not merely record hit ratio, from above.

Alternatively, if the SID file is not predefined, then block-size (BR) is chosen to suit the processing mode. Serial updating time is minimised by maximising B to reduce latencies; thus, block size may be chosen by the Waters (1971) algorithm which often yields block size equal to track size (i.e. $BR \simeq l$). Random updating time is minimised by minimising B in order to reduce the transfer of spurious records; thus, single-blocking is chosen (i.e. $B = 1$). These choices yield

$$\underline{SUT} = N\left[\frac{Rr}{l} + \frac{2R}{t}\right] ms$$

and

$$\underline{RUT} = T\left[f_r + r + \frac{2R}{t}\right] ms$$

$$\therefore \quad \underline{RUT} \gtrless \underline{SUT} \text{ if } \frac{T}{N} \gtrless \left[\left((2R + \frac{Rtr}{l}\right)\Big/ 2R + t(f_r + r)\right] \cdot$$

For example, using the particular disc above,

$$\underline{RUT} \gtrless \underline{SUT} \text{ if } \frac{T}{N} \gtrless \frac{3R}{2R + 19000}$$

$$\therefore \underline{RUT} > \underline{SUT} \text{ whenever } \frac{T}{N} > \frac{4}{9} \text{ as before and } \underline{RUT} < \underline{SUT}$$

generally for only minimal values of $\frac{T}{N}$ · Once again, note that T/N is not merely record hit ratio.

These conditions for the serial versus random updating threshold yield numerous counter-examples where low record hit ratio favours serial updating and high record hit ratio favours random updating. (*Reductio ad absurdum*?) Further, this threshold varies with the record size and the disc.

Finally, skip-serial updating can be drawn simply into the comparisons by observing that

$$\underline{SSUT} = \frac{JB}{N}\underline{SUT} \cdot$$

Thus, serial update time is never less than skip-serial update time, since $J \leqslant \frac{N}{B}$, and the above conditions can be modified to compare random updating with skip-serial updating by replacing $\frac{N}{B}$ by J.

Note that the above comparisons are restricted to disc time and ignore other wider but relevant factors (e.g. sort and CPU times). Thus, since generalisations have been proven incorrect for these particular aspects, they are even more absurd in general.

Conclusion

This paper has discussed the organisation of a SID file, including the implications of variable length records. This file has been evaluated against twelve design objectives to demolish its usual alternatives when key numbers are dense; this condition could probably be applied more, in practice. Timing formulae have been developed for the main processing modes and compared to indicate significant parameters (which include the oft-forgotten fan in/out ratio and file packing density). Even this simple file organisation technique defies generalisations so that each variation should be considered on its own merits; therefore the same will be true for indexed and algorithmic files with their additional complications of indexes and/or overflow overheads.

Some practitioners might regard this paper and its companion 'seeking paper' as purely academic since they no longer estimate disc times (partly due to insufficient manufacturers' specifications). Instead, they often choose file organisation and processing techniques on the basis of the generalisations that these papers disprove! Timing is a vital aspect of systems evaluation and must be pursued unless users are prepared to place themselves at the mercy of computer salesmen!

Acknowledgement

The author wishes to acknowledge the assistance of his colleagues in the LSE Systems Research Group.

Appendix Estimating latency time

Whenever a block of disc information is read or written a latency (or spin or rotational delay) time may be suffered while the read/write head searches for the required block's starting point. This time may be estimated as follows:

Case 1:

$s = 0$ so that searching commences anywhere.

1.1 If the disc accessing is completely controlled by the user program (e.g. via channel command language) and the computer is single-programming and the disc is serially processed with double buffering and the program's CPU time is relatively small, then the latency may be the minimum of zero ms. These ideals are seldom met in practice, other than for fast file dumping.

1.2 In the worst case a complete revolution is necessary to

reach the block starting point, then latency is the maximum of r ms.

1.3 Thus, the average latency can be taken as $r/2$ *ms*, which is usual in practice.

Case 2:

$s = 1$ so that searching commences from the track starting point. Thus, from 1.3 above, an average delay of $r/2$ *ms* is suffered to reach this point. If there are n blocks per track then a further delay of $i - 1/n$ of a revolution is suffered to reach the starting point of block i ($i = 1, 2, \ldots, n$). Thus, the average latency can be taken as

$$\frac{r}{2} + \frac{1}{n}\left(0 + \frac{1}{n} + \frac{2}{n} + \ldots + \frac{n-1}{n}\right) r\ ms$$

$$= \frac{r}{2} + \frac{r}{n^2} \sum_{i=0}^{i=n-1} i\ ms$$

$$= \frac{r}{2} + \frac{r(n-1)}{2n}\ ms\ .$$

Thus, in general, latency can be taken as

$$L = \frac{r}{2}\left(1 + \frac{n-1}{n}\, s\right) ms\ .$$

References

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

HARVEY, T. A. (1970). Data Organisation for Very Large, Low Activity Files on Disc, Proceedings of BCS Conference on *Data Organisation*.

WATERS, S. J. (1971). Blocking Sequentially Processed Magnetic Files, *The Computer Journal*, Vol. 14, No. 2, pp. 109-112.

WATERS, S. J. (1972). File Design Fallacies, *The Computer Journal*, Vol. 15, No. 1, pp. 1-4.

WATERS, S. J. (1972a). A Survey of CAM and its Publications. Proceedings of NCC Conference on *Approaches to Systems Design*.

WATERS, S. J. (1974). *Introduction to Computer Systems Design*, National Computing Centre, Manchester.

WATERS, S. J. (1975). Estimating Magnetic Disc Seeks, *The Computer Journal*, Vol. 18, No. 1, pp. 12-17, February 1975.

Book reviews

Numerical solution of integral equations, edited by L. M. Delves and J. Walsh, 1974. (*Clarendon Press—Oxford Books*, £4·50)

In July 1973 numerical analysts of the Universities of Liverpool and Manchester arranged a Summer School on the numerical solution of integral equations. This book sees the publication of the lectures given at the School. It is mainly expository in character, but ranges widely over the subject. The school was evidently well prepared for there is very little overlapping in the presentations by the various speakers. The result is a goldmine of information on the numerical methodology for solving linear integral equations.

The first fifth of the book gives background theory required for the numerical analysis—with chapters covering an introduction to the theory of integral equations, theory of quadrature, numerical linear algebra, function spaces and linear operators and theory of approximation. Following this scene setting, the main body of the book amounting to three-fifths of the content systematically works through numerical techniques, amply illustrated by examples. First is a chapter on quadrature methods basically stemming from Fox and Goodwin's 1953 paper in *Phil. Trans. Roy. Soc.* There then follow chapters on methods using series expansions of the function, linear programming methods based on approximation theory, Rayleigh Ritz and Galerkin methods. Next follow chapters devoted to the particular cases of eigenvalue problems, Volterra equations of the first and second kind, Fredholm equations of the first kind and linear integro-differential equations. This part of the book is completed by three short chapters covering some 40 pages on non-linear integral equations. It is perhaps indicative of the state of the art that so relatively little can be said about this important branch of the subject.

The book concludes with a third part, again about a fifth of the whole, devoted to applications—all problems of mathematical physics.

The content of this book appears to be really up to date and must surely be regarded as providing a prime source book. If I have any criticism to make of it in this respect, it is a fault of omission rather than commission. The various eminent teachers who contributed to the work of the school have covered their allotted tasks well, but the numerical practitioner, coming to a problem involving integral equations, perhaps for the first time, will not find here any general discussion of the comparative advantages and disadvantages of the various methods; he will find excellent descriptions of the methods available, but little guidance on how to choose one for his problem. The book would have been more complete with some attempt at a critical comparison of methods—although I write this with an awareness of how difficult such a comparison is to make.

One other criticism concerns the editing. Each chapter has its own bibliography. In aggregate these add up to a very considerable asset, but they are somewhat repetitious, and in the absence of an author index they are less useful than they might have been.

However, Professor Delves and Dr. Joan Walsh have provided us with a valuable work which will long remain essential reading for numerical analysts, and they are to be congratulated for their achievement.

A. YOUNG (Coleraine)

Computers in Production Management Decisions, by T. A. J. Nicholson and R. D. Pullen, 1974. (*Pitman Publishing*, £4·00)

This book reports on a number of topics, drawn broadly from the production management area, of which the authors have personal knowledge. They make quite clear that they did not intend to write a detailed academic review but aimed to provide a bridge between the production manager and the computer specialist. Their aim has been reasonably true.

A dozen topics are covered of which some five or six are set squarely in the batch production environment. The emphasis being, quite rightly, on using a computer as a planning aid rather than a generator of shop paperwork. The rest of the book tackles assembly line balancing and production capability over the short and long-term. Each topic is treated in roughly the same fashion: a description of the problem, the appropriate modelling and a demonstration of a computer-based approach towards a solution, illustrated with specimen printouts and so on. In most cases the proposed approach is via a specially developed interactive program or an off-line general purpose package.

A minor criticism concerns the rather sketchy analysis of costs involved in developing a computer-based production control system. The development of such a system is as much an investment as new plant and should be appraised accordingly. This omission is perhaps surprising in view of the authors' chapter on investment decisions.

Nevertheless, the book should alert computer people to the fascination of production problems. And if production managers can cease contemplating their escalating work-in-progress long enough to read it also, then so much the better.

C. D. EASTEAL (Tunbridge Wells)