Flexibility of block-length for magnetic files

J. Inglis and E. G. Dee*

Department of Computer Science, Birkbeck College, University of London, Malet Street, London WC1E 7HX

A constraint imposed on systems designers by over-restrictive software is that the unit of transfer between secondary and primary storage is the *block* (physical record) or, at best, that every read operation must start at the beginning of a block. This paper demonstrates that many current computers provide channel programming facilities which may be used to overcome this constraint, enabling a distinction to be made between 'device block' and 'input block'. This distinction leads to greater efficiency of file handling (in terms of both transfer time and secondary storage utilisation) and to increased flexibility in file design. An algorithm is developed for selection of optimum input block length for a given file and a given input buffer length.

(Received April 1973)

Introduction

Waters (1971) presents a method for optimising the block lengths used for sequentially-processed files on magnetic storage devices; the basic algorithm was also developed independently by Walker (1970). Although this algorithm is probably satisfactory for most routine serial systems, it can become unwieldy when a file is read:

(i) by several different run-units which vary widely in the amount of store available for use as an input buffer area, or

(ii) on different occasions by the same run-unit under differing conditions of store availability.

It is generally accepted that run-units must be *block length compatible* with respect to a file, i.e. that all run-units which input a file must adhere to the block length used when the file was written. In practice, this implies that the run-unit which requires the smallest block length determines the block length used for the file. Thus

(*a*) a block length which is suitable for such a run-unit may result in increased file access time during the execution of the other run-units;

(*b*) the efficiency of usage of the file storage medium is constrained by main store availability during execution of the run-units which read the file;

(*c*) a given run-unit is unable (except by varying its buffering technique) to make use of space in main store which may be available during some executions but not during others.

Block length compatibility rests on the assumption (which is usually valid in the context of standard software) that the *input block length* (the amount of data read during one access to the storage device) must be equal to the *device block length* (the amount of data stored as one physical record on the storage device). The present authors have failed to find any textbook in the area of systems analysis and design which challenges this assumption.

This paper shows that, on most configurations involving machines such as the IBM System/360-370 and the ICL System 4, the foregoing assumption need not be made. A procedure is outlined for such machines which enables any run-unit which uses an input file stored on a direct-access device to determine its input length independently of the device block length. If such a procedure is used, the device block length for a file may be as large as is compatible with:

(*a*) the buffer space available in main store to the run-unit which writes the file;

(*b*) the characteristics of the storage device (which may result in selection of a value smaller than the buffer length).

The file is therefore stored as efficiently as possible, and the degradation in performance of one run-unit due to the storage limitations of another is considerably reduced. Every run-unit is able to use the amount of main store which is available on any occasion to improve the efficiency of its file-handling by reducing the number of device accesses. The procedure does, however, necessitate the use of more complex recovery programs to deal with hardware malfunctions. The current lack of awareness of this flexibility at hardware level can be attributed to its being masked from the system designer by standard software.

The principles described in this paper become of greater importance as the track capacities of direct access devices increase. Unless these principles are applied, full use of the capacity of the IBM 3330 disc storage device, for example, involves the use of some 13,000 bytes of main store as an input buffer.

The principle

The facility to be described is dependent on the existence of a hardware channel command language which includes data-chaining and a 'data-skip' facility. The concepts of channels and channel programming are described briefly by Padegs (1964) and Ricour and Mei (1967); a more detailed description is given by Flores (1970). Details of individual channel command languages are to be found in, for example, IBM (1970) and ICL (1969). The availability of data-chaining for a given configuration may be established by reference to the manufacturer's literature.

Briefly, on IBM System/360–370 and ICL System 4, input/output operations proceed as follows. The central processing unit passes to a channel information which includes the address of the device to be accessed, and the address in main storage of a *channel command word* (CCW). The channel then acts as an asynchronous processor and, under the control of the CCW, attempts to perform the required operation. It is possible for a CCW to cause the channel to fetch and process another CCW in just the way in which a normal processor fetches and executes instructions. When execution of the channel program (i.e. the chain of CCWs) terminates, either because of an error condition or on normal completion, the channel interrupts the central processing unit, passing back status information. It is possible to perform considerably complex sequences of operations involving a storage device (for example, to search through a complete disc cylinder for a record with a given key and to read the record, if found, into main store) quite independently of the central processor.

Two facilities of a channel command language which are relevant in the present context are:

*Now at Polytechnic of the South Bank, London SE1 0AA.

Downloaded from https://academic.oup.com/comjnl/article/16/4/303/416480 by guest on 19 April 2024

(*a*) reading of data from more than one block within a single activation of the channel;

(*b*) reading of only certain specified bytes of a block into main store.

It should now be clear that channel programming effectively makes the unit of transfer between the device and main store independent of the blocking arrangement used at the device. Consider as an example a sequentially organised file held on disc in fixed-length blocks of 3000 bytes each. This file may be read using an input block length of 2500 bytes so that successive accesses to the file retrieve:

1. bytes 0–2499 of block 0;
2. bytes 2500–2999 of block 0 and bytes 0–1999 of block 1;
3. bytes 2000–2999 of block 1 and bytes 0–1499 of block 2;
4. bytes 1500–2999 of block 2 and bytes 0–999 of block 3; etc.

The input block length can be determined by considerations of efficiency of execution and main store availability, and the device block length by efficient use of the storage medium. However, on the machines mentioned above, it is not possible to *write* to selected areas of a block on the device, and therefore a further constraint on the device block length is the amount of main store available during writing. Thus any optimising algorithm should ensure, in allocating buffer space for a run-unit, that the allocation is weighted in favour of output files.

The present treatment considers the use of the proposed facility in relation to sequentially organised files stored on direct access devices. The final section of the paper comments on the applicability of the principle to other file organisations, and to files stored on magnetic tape.

Implementation

The facility of varying the input block length has been used within PL/I programs, to read standard sequential files written by standard software. The implementation was carried out on an IBM System/360 in the following manner. Each file is represented by a File Control Block (FCB). The FCB contains the information normally stored in an OS Data Control Block (DCB), plus information relating to the buffering technique. For each input buffer used, there is a Buffer Control Block (BCB); for an active file, one or more BCBs are associated with the FCB.

A BCB contains information such as:

1. buffer id (identifier)
2. buffer length
3. buffer starting address
4. disc address (byte) of first byte transferred on last input to this buffer
5. number of bytes transferred on last input to this buffer.

An FCB contains programmer information about the last buffer used, and also fields such as the disc address (byte) of the next input block of the file to be read, the next buffer to be used, address within current buffer, etc.

At the lowest program level, there is an I/O routine which builds and executes a channel program to read a given number of bytes from a given file into a given main store area. This routine obtains its parameters from an FCB, and updates that FCB to reflect the change in the state of the file.

At the next program level, there is a buffer management routine which controls buffer usage. It updates the relevant fields of the FCB and calls the I/O routine to fill a buffer.

User interface

The user normally interfaces with the system by using a set of macros. To read a logical record, the traditional GET macro is used:

GET FCB-address [, area-address] .

A call of this macro results in the next logical record of the file being placed at the main store address specified by 'area-address' (both FCB-address and area-address are, of course, normally identified symbolically); if no 'area-address' is specified, then a pointer is returned to indicate the record's position in main store. The GET macro operates by calling the buffer management routine, which may directly return a pointer, and/or move data, and/or cause the I/O routine to be called.

To control the allocation of buffers, a BUFFER macro is used:

BUFFER FCB-address [, length [(number)]][, id [, DELETE]].

A call of this macro causes the allocation or deallocation of one or a number of buffers associated with a file.* It calls the buffer management routine to perform this action, as the loss of a buffer may require that the FCB be updated. When a buffer is allocated, a BCB is constructed for that buffer, and, when a buffer is deallocated, the BCB space is released.

It may be noted that the I/O routine may be used without the buffer management routine to give a more flexible but less convenient system. This allows the user to read parts of records (rather than complete records), by modifying the FCB between read operations.

Evaluation (*An example is given in Appendix* 2)

The effect of device block lengths on the *efficiency of usage of the storage medium* is well understood, and the computer manufacturers provide tables and formulae for calculating optimum block lengths from this point of view. However, evaluation of the effect on *timing* of selecting different input block lengths in relation to a given device block length merits some attention.

Notice first that, since we are considering sequentially organised files, the 'cylinder seek time' (i.e. the time to move from cylinder to cylinder) is constant for a given file, independent of the input block length used, and that head-switching time within a cylinder is negligible. The remaining elapsed time spent at the device (rotational delay and read time) will be referred to as *track time*. Evaluation of timings will therefore be in terms of track time alone, and a convenient unit of measure for track time is the time taken for one byte to pass the read heads. This unit will be referred to as a *byte time*.

Let d be the device block length and i be the input block length, each expressed as a number of bytes. (In practice, both d and i may vary during execution, since the file may be stored in variable-length blocks, and/or the number of bytes read may be different for successive input operations. However, for simplicity, the following timing estimates assume device blocks and input blocks of fixed length).

Regard the file as a string of bytes numbered serially from one. The pattern of mapping input blocks onto device blocks will clearly be repeated after each byte whose serial number n is such that:

$$n = 0 \bmod d \text{ and } n = 0 \bmod i ,$$

i.e. the pattern is repeated every $\frac{di}{\delta(d, i)}$ bytes, where $\delta(d, i)$ is the greatest common divisor of d and i.

Within one unit of the pattern, the number of device blocks is clearly the number of bytes divided by d, i.e. $\frac{i}{\delta(d, i)}$. By definition, there are therefore $\frac{i}{\delta(d, i)} - 1$ device block boundaries

*The BUFFER macro as specified above does not include 'buffer-address' as a parameter, since the operating system used in this implementation contains storage management software.

Downloaded from https://academic.oup.com/comjnl/article/16/4/303/416400 by guest on 19 April 2024

which are not also input block boundaries in each unit of the pattern; i.e. the number of occasions during sequential input on which an input operation 'straddles' device blocks is $\frac{i}{\delta(d, i)} - 1$ per pattern unit.

Now, since the number of input blocks per pattern unit is the number of bytes in the pattern unit divided by i, i.e. $\frac{d}{\delta(d, i)}$, the average number of 'straddles' per input operation is $s = \frac{i - \delta(d, i)}{d}$. (This average is obtained over one unit of the mapping pattern, but it can be shown to apply without significant loss of accuracy to any complete file of realistic length.) Notice in particular that:

(*a*) if $i = d$, $s = 0$;
(*b*) if d/i is an integer, $s = 0$;
(*c*) if i/d is an integer, $s = i/d - 1$.

By nature of the hardware, a complete device block is always read from the device, even if only part of it is transferred to main store. Therefore a reasonable measure of timing is obtained by considering how many times the operation of reading a device block occurs, and how many rotational delays occur, during input of a complete file. The average number of device blocks read per input operation is $s + 1$, and, if the file length is f bytes, then the number of input operations is $\left\lceil \frac{f}{i} \right\rceil$. Thus if a is the average rotational delay, and b is the time taken to read a device block of length d (including associated overheads), the time taken to read the complete file may be estimated as:

$$\left\lceil \frac{f}{i} \right\rceil \times \left(a + \left(\frac{i - \delta(d, i)}{d} + 1 \right) b \right) \qquad (1)$$

byte times.

The variable b is discussed in Appendix 1; its value is a function of the device and of the value of d. The value of a is constant for a given device, and the values of f and d are constant for an existing file; the only variable whose value may be determined by an input program is i. The working in the next section of this paper is based on the fact that all the variables in (1) take positive values only.

Selection of optimum input block length

(*An example is given in Appendix 3*)

In practical situations, if a program has I bytes of main store available as buffer space for a given input file, we require to find the value of i which gives optimum performance (i.e. which minimises the value of expression (1) above), subject to the constraint $0 < i \leqslant I$.

For files of any realistic length, the first term of expression (1) may be regarded as f/i, and the expression to be minimised may be simplified as follows:

(*a*) Since f is a positive integer whose value is fixed for a given file, we can divide by f, giving:

$$\frac{a + \left(\frac{i - \delta(d, i)}{d} + 1 \right) b}{i}$$

$$\equiv \frac{a}{i} + \left(\frac{1}{d} - \frac{\delta(d, i)}{di} + \frac{1}{i} \right) b .$$

(*b*) Since b and d are positive and have fixed values for a given file, we can multiply by d/b and subtract 1, giving:

$$\frac{\left(\frac{a}{b} + 1 \right) d - \delta(d, i)}{i}$$

$$\equiv \frac{x - \delta(d, i)}{i}, \text{ where } x = \left(\frac{a}{b} + 1 \right) d . \qquad (2)$$

Notice that, since max $[\delta(d, i)] = d$, this expression always takes a positive value.

Thus, for an input block length of $i\ (<I)$, $I - i$ fewer bytes of main store will be used, and the file accesses will be at least as efficient from a timing point of view when the following condition is satisfied:

$$\frac{x - \delta(d, i)}{i} \leqslant \frac{x - \delta(d, I)}{I} . \qquad (3)$$

Let $k = I - i$. Then expression (3) may be re-arranged as:

$$\frac{x - \delta(d, I)}{I} \leqslant \frac{\delta(d, i) - \delta(d, I)}{k} . \qquad (4)$$

The following points may now be noted:

1. In finding the optimum value for the input block length, we need consider only values of i such that $\delta(d, i) > \delta(d, I)$.

Since $\frac{x - \delta(d, I)}{I}$ is always positive, and k is always positive, expression (4) can be true only when $\delta(d, i) > \delta(d, I)$.

2. *If the input buffer length is a multiple of the device block length, the optimum input block length is equal to the input buffer length, i.e. no smaller input block length gives equal or better performance.*

This follows from *1*. If I is a multiple of d, $\delta(d, I) = d$; then, since max $[\delta(d, i)] = d$, $\delta(d, i)$ can never exceed $\delta(d, I)$.

3. If the input buffer length exceeds the device block length and is not a multiple of it, then the minimum value to be considered in finding the optimum input block length is that multiple of the device block length next below the input buffer length.

If I is not a multiple of d, consider $I = dn + m$, where n is a positive integer and m is an integer such that $0 < m < d$. From *2*, we know that, if $i = dn$ is considered, values of i in the range $0 < i < dn$ need not be considered. Thus any value of i which gives better performance than I satisfies the condition $dn \leqslant i < I$.

4. If the input buffer length is not a multiple of the device block length, and the multiple of the device block length next below it equals or exceeds a/b, then that multiple of the device block length is the optimum value of the input block length.

The condition under which the multiple of d next below I always gives better performance as an input block length than I may be stated by substituting $(nd + k)$ for I in expression (4), giving:

$$\frac{x - \delta(d, I)}{nd + k} \leqslant \frac{d - \delta(d, I)}{k} ,$$

where n is a positive integer and $0 < k < d$. Substituting for x and multiplying by the positive integer $k(nd + k)$, we obtain:

$$\frac{ak}{b} \leqslant n[d - \delta(d, I)]$$

$$\equiv \frac{ak}{b[d - \delta(d, I)]} \leqslant n . \qquad (5)$$

Now consider $\frac{k}{[d - \delta(d, I)]}$. Since both d and $(nd + k)$ are multiples of $\delta(d, I)$, then k also is a multiple of $\delta(d, I)$. Since k is smaller than d, k must be a smaller multiple of $\delta(d, I)$ than is d. Thus $\frac{k}{[d - \delta(d, I)]} \leqslant 1$, regardless of the values of I and k. Thus, if $n \geqslant \frac{a}{b}$ is true, condition (5) above is also true.

Downloaded from https://academic.oup.com/comjnl/article/16/4/303/416400 by guest on 19 April 2024

Notice that a practical consequence of this condition is that, when each device block occupies more than half a track (and consequently $\frac{a}{b} < 1$), and the input buffer length equals or exceeds the track length, then the optimum input block length is always a multiple of the device block length.

5. If a number of eligible values for the input block length have the same greatest common divisor with the device block length, only the highest of these values need be considered.
I.e. if n values $e_1, e_2, e_3, \ldots, e_n$ are eligible as a result of application of *1* to *4* above, and are such that $\delta(d, e_1) = \delta(d, e_2) = \delta(d, e_3) \ldots = \delta(d, e_n)$ and $e_1 < e_2 < e_3 \ldots < e_n$, then, of these values, only e_n need be considered as an eligible value of i and $e_1, e_2, e_3, \ldots, e_{n-1}$ may be discarded. This is self-evident from substitution for i in expression (2).

A simple algorithm for selection of input block length, based on the above observations *1* to *4*, is presented in the next section; for convenience, the algorithm is written in ALGOL 60 notation. Input timings are considered in greater detail in Appendix 1.

An algorithm for selection of input block length

```
integer procedure optimum (I, a, b, d); value I, a, b, d; integer
I, d; real a, b;
  comment This function returns the value of the optimum
  input block length for a file with fixed device block length
  = d, stored on a device having average rotational delay = a,
  and on which the time to read a device block = b. There are
  I bytes of main store available as an input buffer area. It is
  assumed that an integer function gcd (x, y) is defined to
  calculate the greatest common divisor of integers x and y.
  For practical efficiency, the function 'optimum' should be
  amended to avoid repeated calculation;
  begin integer j, k, n; real r;
    real procedure calc(y); value y; integer y;
      calc := ((a/b + 1)*d - gcd(d, y))/y;
    n := I ÷ d;
    k := I - n*d;
    if k = 0 ∨ n ⩾ a/b
      then optimum := n*d
      else begin optimum := I;
        r := calc(I);
        for j := I - 1 step - 1 until if n = 0 then 1 else n*d
        do if gcd(d,j) > gcd(d, I)
          then begin if calc(j) < r
            then begin r := calc(j); optimum := j;
              end;
          end;
      end;
  end;
```

Extensions

This paper has concentrated on the case of sequentially organised input files stored on direct-access devices. The same principle may be applied to index-sequential files and random files when these are used purely as input to a run-unit.

For sequential input of an index-sequential file, reading of the prime tracks may be speeded up by increasing the input block-length, or main store limitations may be overcome by decreasing the input block-length. For random access to a blocked index-sequential file, main store limitations may be overcome by reading the relevant block in smaller units. For blocked randomly organised files, each prime 'bucket' (i.e. a set of records whose keys hash to the same value and which are stored contiguously) may be treated in the same way as a sequentially organised file.

The principle can be applied in a limited fashion to magnetic tape input files, enabling several device blocks to be read on one access to the device. Note, however, that in this case the input block length should be a multiple of the device block length; if it is not, it will be necessary to reverse over the last block read. Consider also the case of a run-unit which requires to read a tape whose block length is too great for the main store space available. An alternative to using a special re-formatting run is to use an input block length of (say) one-third of the device block length, and to read the first and third input blocks from each tape block in the forward direction, and the second input block in reverse. This alternative may be preferred, for example, when there is a shortage of available tape or disc drives.

Undoubtedly, however, the main application of the principle is to sequentially organised files stored on direct-access devices, and, in this area, interesting work remains to be done on optimisation of the use of buffer space by run units which use common files.

Appendix 1 More detailed timing considerations

Expression (1) in the above paper makes use of a variable b, which is defined as 'the time taken to read a device block of length d (including associated overheads)'. This appendix considers how the value of b may be calculated for a given direct-access device and a given file.

The layout of a track on the devices under discussion is such that each device block of user data on the track generates two (or optionally three) physical 'areas' (blocks), separated by gaps. These are a *count area*, identifying the block and specifying its length, an optional *key area*, for use by search commands, and a *data area*, which contains the user's data. Further details of track layout can be found in the manufacturer's literature for a given device (e.g. IBM (1970), ICL (1969)).

Let c (expressed as a number of bytes) be the sum of:
the length of a count area and the gap which follows it (which are fixed for a given device); the length of the key area, if present, and the gap which follows it (which are fixed for a given file); and the total number of check bytes (for hardware checking; the number is fixed for a given device) for the count, key and data areas.

The value of c may therefore be calculated from the device specification and the file description.

The total time taken in reading a single block of length d from the device is then $(c + d)$ byte times. If, however, consecutive blocks are read in a single command chain, the time taken in reading each block after the first is $(c + d + g)$ byte times, where g is the average time taken to pass from a data area to the next count area. Notice that this average may have to take account of command chains which read from more than one track. Unless the data areas are small, g does not have a significant effect on the time taken to read a file.

Appendix 2 Calculation of file read time—example

Let the device block length (d) be 1000 bytes. If the input block length (i) is 4500 bytes, the number of bytes per pattern unit (n) is 9000. Hence, in each unit of the pattern, input operations straddle device blocks eight times, and the average number of straddles per input operation (s) is four.

For the purpose of illustration, consider a short file of 1,000,000 bytes (i.e. $f = 1{,}000{,}000$), giving $\left\lceil \frac{f}{i} \right\rceil = 223$. If the file is stored on a device whose byte time is 1 μs., and whose average rotational delay (a) is 7000 byte times (i.e. 7 ms), and if the time taken (including overheads) to read a block of 1000 bytes (b) is 1050 byte times (i.e. 1·05 ms), then the approximate total track time taken to read the complete file is 2,731,750 byte times (i.e. about 2·732 seconds).

Downloaded from https://academic.oup.com/comjnl/article/16/4/303/416400 by guest on 19 April 2024

Appendix 3 Optimisation of input block length —example

1. Consider again the short file of Appendix 2, where $a = 7000$, $b = 1050$ and $d = 1000$, and assume that an area of main store of length (I) 4700 bytes is available as a read-in area for this file. In this case, the constant $x = (a/b + 1)d = 7667$. To obtain an optimum input block length $i\ (<I)$, we may restrict the values considered as follows.

By *1*, $\delta(1000, i) > \delta(1000, 4700)$, i.e. $\delta(1000, i) > 100$. Applying *3*, the minimum value of i to be considered is 4000. Hence the possible values of i are 4000, 4125, 4200, 4250, 4375, 4400, 4500, 4600 and 4625.

Applying *5*, 4125, 4200, 4375 and 4400 may be discarded, leaving 4000, 4250, 4500, 4600 and 4625. The optimum value of i occurs when expression (2) is minimised, i.e. when $i = 4500$.

2. Consider the same file and device but this time with an available main store area of length (I) 7500 bytes.

In this case, the values of $i\ (<I)$ to be considered may be restricted as follows.

Applying *1*, $\delta(1000, i) > 500$

i.e. i is a multiple of 1000, and is less than 7500.

Applying *3* or *5*, $i = 7000$.

Alternatively, applying *4*, $7000 = 7d$, and $\frac{a}{b}\left(= \frac{7000}{1050}\right) < 7$.

$$\therefore i = 7000 .$$

References

Flores, I. (1970). *Data Structure and Management*, Englewood Cliffs, N.J.: Prentice Hall.

IBM (1970). *IBM System/360 Component Descriptions—2314 direct access storage facility and 2844 auxiliary storage control*, IBM order number GA26-3599-5.

ICL (1969). *Hardware reference—peripherals*, ICL Technical Publication 4505.

Padegs, A. (1964). The structure of System/360; part IV—channel design considerations, *IBM Systems Journal*, Vol. 3, No. 2, pp. 165-180.

Ricour, D. H. and Mei, V. (1967). Internal data management techniques for DOS/360, *IBM Systems Journal*, Vol. 6, No. 1, pp. 38-48.

Walker, E. S. (1970). Optimization of tape operations, *Software Age*, Vol. 4, No. 8/9, pp. 16-17.

Waters, S. J. (1971). Blocking sequentially processed magnetic files, *The Computer Journal*, Vol. 14, No. 2, pp. 109-112.

Book reviews

Pattern Recognition Techniques, by J. R. Ullmann, 1973; 412 pages. (*Butterworths*, £10·00)

In his book, *Pattern Recognition Techniques*, Dr. Ullmann surveys published material and investigates the theoretical background of the many and diverse techniques which have been attempted in the struggle to devise automatic reading machines. It is probably misleading for the book's title to suggest that the whole of pattern recognition is covered in the one volume. In fact large areas are passed over with little more than a mention; particularly conspicuous by its absence is any detailed consideration of the problems of handling grey level images which are, for many workers in this field, of prime importance when it comes to dealing with 'live' rather than simulated data. Throughout the book, optical character recognition is used to illustrate the pattern recognition methods described. This can perhaps be justified by the fact that OCR is the most fruitful area of application of pattern recognition technology, but it is disappointing that the whole of biological material classification, particle track analysis, finger print classification, automatic inspection and other promising fields where pattern recognition techniques could eventually be applied, are discussed in a few words.

Nevertheless, this book provides a useful compendium of the techniques which have been proposed or applied in OCR and the bibliography of 468 references is a valuable list for anyone wishing to read further in this area. Readers without a mathematical training (and perhaps some with, also!) will probably find some of the book rather hard-going. It could be argued that the notorious non-success of some of the methods described hardly merits the effort required to master the precise and formal descriptions used to discuss them. At the same time, however, one would not wish to discourage any attempt to force order and method into what has been a rather undisciplined subject over the past years. Unfortunately, the precise mathematics seems to have been applied most liberally in one particular aspect of the subject: the part concerned with deciding how to classify preprocessed data. Many workers would argue that it is the preprocessing which determines the success or failure of any system, and this is the area where formal theory and mathematics are least helpful at present.

Despite some omissions, Dr. Ullmann's book presents a formidable attack on a subject with few worthwhile general textbooks to serve it. Undoubtedly even the 'experts', for whom this book is not really intended, will wish to have a copy on their shelves.

M. J. B. Duff (London)

Pattern Classification and Scene Analysis, by Richard O. Duda and Peter E. Hart, 1973; 482 pages. (*John Wiley and Sons Ltd.*, £11·25)

The authors' purpose in writing this book 'has been to give a systematic account of major topics in pattern recognition, a field concerned with machine recognition of meaningful regularities in noisy or complex environments'. The book is in two parts. The first deals with classification theory, where the data, expressed as vectors, has to be classified into patterns. The main tool here is statistical decision theory and the approach is relatively formal. The second part is devoted to scene analysis: the attempt to describe a scene, or picture, in a vector form. Many tools are used and the dominant ideas are heuristic. (Your reviewer must declare his interest and say that the material here was strange to him, a statistician.)

The following is a list of the main topics considered. Bayesian decision theory, parameter estimation and supervised learning—still within the Bayesian framework—nonparametric techniques (e.g. the nearest-neighbour rule), classification by linear discriminants, clustering techniques. These are in Part I. In Part II the list includes representation (e.g. by templates), filtering, descriptions by line and shape, perspective transformations, projective invariants and descriptive methods. Each chapter has a lucid introduction and concludes with bibliographical and historical remarks and a set of problems.

Two things immediately strike me about this book. Firstly the great variety of techniques that are used, and secondly the authors' skill in explaining them. The result is a book that is much livelier than the usual run of texts. Part of this liveliness is due to the fact that the subject is still in an early stage of development before the ideas have set into a mould. It is good to have some of the techniques (like multidimensional scaling) explained and compared with other procedures. A dissappointment is the lack of good, genuine applications—particularly in Part I, where there are many of them. Fisher's Iris data seems to be the only exception. There are good examples in Part II, but they are mostly slight, presumably since the techniques are not yet very sophisticated. The only serious error I noted was on page 72 where a spurious argument is used that fails to take into account the change of the prior distribution with increasing numbers of factors. The authors seem to be unaware of the basic idea of coherence. But such slips can be forgiven in a text which covers so wide a field and uses so many tools. It should serve as a good introduction to the subject. I certainly enjoyed it.

D. V. Lindley (London)

Downloaded from https://academic.oup.com/comjnl/article/16/4/303/416400 by guest on 19 April 2024