Document retrieval using a substring index

P. W. Williams and M. T. Khallaghi

Computation Department, University of Manchester Institute of Science and Technology, PO Box 88, Sackville Street, Manchester M60 1QD

Recent research has suggested the indexing of documents by the substrings that are present in some content rich piece of text from the document; in particular, articles have reported on the analysis of titles. A method is suggested here, based on keywords, that reduces considerably the processing required for substring analysis, that generates some word truncation to group grammatical variants, that avoids the bias created by spaces and noncontent words, and that also maintains acceptable levels of dictionary size and retrieval precision.

(Received December 1975)

The problem considered in this paper is a familiar one, namely the selection of relevant books or articles from a large collection. For convenience, the term *document* will be used for the individual books, journal articles, etc. that make up the collection. In a large collection, a serial scan of the whole collection is not feasible so the documents are classified into groups, and an index is created for the documents.

A query is presented to a retrieval system by choosing terms which are compared with the entries in the index to select relevant documents. The retrieval system can be considered in four parts; a classification scheme is devised for the document collection, index terms are assigned to a document so that it can be entered into the classification, a query is formulated using terms from the classification scheme, and a search is made to find documents relevant to the query.

A variety of classification schemes are used. For example, an index number can be given showing the position of a document in an hierarchical system which progressively subdivides the subject matter, as in the Dewey Decimal system used in libraries. An alternative method is to assign *index terms* or *keywords* which indicate the subject matter of the document. For example, this particular article might have as index terms the phrases, 'Information retrieval', 'String processing', 'Free language indexing'. In a computer system this is often implemented by means of an *inverted file* system: a list of documents containing a particular index term is grouped under that index term. A query specified as a Boolean combination of index terms can then be carried out by searching only those lists of documents containing the index terms in the query.

In a computer system the entries which are used for searching, *the keys*, may be the original index terms or some mapping of these that gives a more convenient computer representation. For example, each index term may be given a numerical code, or only the first four letters of the term might be used to minimise the space requirement for the keys. The complete set of allowable keys will be called the *key dictionary*.

In some cases the index terms are rigorously controlled by establishing a dictionary of allowed terms to which the indexer and the searcher must conform, a *controlled language* system. Alternatively, both the indexer and the searcher may choose any term which they feel describes the subject matter, a *free language* indexing system.

A controlled language system has the advantage that the key dictionary is fixed, which simplifies computer implementation, but it suffers two disadvantages:

(*a*) the indexers and the users must be familiar with the authorised index term dictionary. This required considerable effort both when documents are indexed and when choosing search terms

(*b*) in rapidly developing subjects, new terminology is continually being created and it is difficult to maintain an up-to-date dictionary.

A system using a flexible controlled language dictionary, which grows by including new terms as they are identified, (such as the ASSASSIN retrieval system (Clough and Bramwell, 1971)) is not ideal for a large document collection since the dictionary soon becomes excessively large, although it has proved economically feasible on some large systems, such as the Lockheed Dialog and the SDC system to use large uncontrolled vocabularies.

Recent work on computerised text processing has suggested methods of index production that would allow free language indexing and also give a key dictionary of fixed size. Experience has shown that, within a particular subject area, a substring dictionary based on one year's articles does not change significantly in subsequent years so that a fixed dictionary is feasible (see Lynch, Petrie and Snell, 1973). The methods are based on a scan of the *string* of characters that form a complete piece of text in order to identify the *substrings* which are present (a substring is a group of consecutive characters of smaller length than the total string). Sections of text are scanned that are likely to contain important descriptive words, for example the titles in the work by Lynch and the free index terms in the method described in this paper.

Harrison (1971) suggests that a string of text be represented by a signature, i.e. a binary vector in which is recorded the presence or absence of every possible substring of some chosen fixed length. To maintain a sensible size for the signature, a *hashing* transformation is used to map several substrings into each position in the binary vector. An application of the Harrison method has recently been recorded by Goble (1975).

Lynch and co-workers (Clare, Cook, Lynch, 1972) have developed methods for selecting an equifrequent key dictionary of substrings from the total collection of substrings present in a typical sample of the text to be indexed. The Lynch method has been applied to full text, including spaces and delimiters such as punctuation marks, so that the results can be used either for text compression or to generate keys for retrieval. The criterion of equifrequency is justified by appeal to concepts from information theory, but the undesirability of variable frequency can be illustrated quite simply. The most common keys in a dictionary will generate the longest lists of documents and will also occur more often in the queries. Thus the most frequent queries will require searches of the longest lists, which reduces efficiency.

Similar ideas have been used by Schuegraf and Heaps (1973; 1974) to produce efficient compression for large data bases by using variable length equifrequent substrings.

As described below there are disadvantages in these methods. This paper presents a string processing approach based on the *index words* in the index terms. A set of descriptors is established for each document by making a complete list of the index words in the free language index terms, i.e. the grouping into index terms and the ordering within terms is not considered. An equifrequent key dictionary is generated from the descriptors from all the documents using a word based method which avoids the high overhead of the Lynch method and also generates some word-stemming which groups word variants. The keys are based on the substrings commencing with the first character of each descriptor and cannot be used for matching internal substrings.

The assignment of keys for each document and query also takes much less computation than other similar methods.

The transformation from index words to keys is a many to one mapping, followed by an exact matching process, and therefore all relevant documents are retrieved (*recall* is 100%). The method will therefore be evaluated by studying the percentage of relevant documents in the retrieved set, the *precision*.

The Harrison method

In the Harrison method a complete string of text of length L is represented by the set of its substrings of length k. (A substring is a set of k consecutive characters commencing at any of the first $L - k + 1$ positions; substrings can therefore overlap). Each substring of length k is subjected to a hashing transformation which generates an integer in the range l to m. Figures are given by Harrison for $k = 2$ and $m = 32$ showing the probability of obtaining two different substrings with the same hash value. A binary string of length m is set initially to zero, and then a 1 is entered in each position corresponding to the integers generated from the hashing process. Thus in a binary string, the 'signature' is generated from each string and used as an index entry.

If a string S_1, with signature G_1, has a substring S_2 with signature G_2, then G_1 will have 1's wherever G_2 has 1's. Comparison of G_2 with members of a set of signatures $\{G_r\}$ will give a necessary, but not sufficient condition for G_2 to be a substring of an element G_r, i.e. any position containing a 1 in G_2 must also contain a 1 in G_r.

The following properties of the method should be noted:

1. This method gives a key dictionary of controllable size by varying k and m.
2. The number of collisions (a 1 in a signature position resulting from different character strings) depends on the hashing algorithm and may vary considerably for different positions in the signature.
3. In document retrieval systems, because of the many to one nature of the hashing transformation, the signature match must be followed by a complete character by character matching of the search substring with the document substring.
4. The computation requires a scan along the string of L characters, forming the hash code from $L - k + 1$ substrings of length k.
5. The frequent occurrence of spaces dominates the frequency pattern.

The Lynch method

In the Lynch method, variable length equifrequent substrings are used as keys. The frequency of occurrence of substrings with a fixed length k in a document collection will show a hyperbolic curve when plotted against the rank order of the substring sets (see Zipf, 1949). Various methods can be used to reduce the variation in the frequencies. For example, the sets with high frequencies can be subdivided by substituting the sets of length $k + 1$ that are present. Any large sets of length $k + 1$ can again be subdivided by considering strings of length $k + 2$, etc. An alternative approach initially considers long substrings, accepting as index terms those strings with a frequency above a chosen threshold and inspecting shorter and shorter substrings to find sets which reach the required frequency level.

The method has the following properties:

1. The size of the dictionary can be controlled by varying the threshold frequency at which sets are accepted.
2. The substrings present are represented in full, without hashing. Ambiguity can still arise since the order of the substrings is not considered.
3. The number of substrings in a string of length L is $L \times (L + 1)/2$. Hence considerable processing is required to create the dictionary, if all these are considered.
4. Information theory shows that equifrequency of the keys gives efficient retrieval.
5. The frequent presence of spaces dominates the frequency pattern.

A word based method

Many of the difficulties of the Lynch method of substring analysis can be alleviated if the analysis is based on individual words rather than a complete string (see Fokker and Lynch, 1974, for application of the method to author names). This approach is particularly suited to analysis of index terms based on keyword phrases, which do not have the noncontent words found in titles.

The dictionary is created by taking a typical set of documents to be used in the retrieval system and listing all their index words and their frequency of occurrence. From these words, keys are chosen that are variable length substrings beginning with the initial letter of the word; the length of the substring is chosen to equalise the frequency of occurrence of the chosen keys. Two methods can be used to minimise the variation in frequency. In the $1{:}n$ process the analysis starts with a frequency count of the number of terms beginning with each character. A threshold is fixed and those substrings with a frequency count below the threshold are accepted as keys. Those sets which are bigger than the threshold are subdivided by considering frequency counts for the digram which start the word. A large set can be further subdivided by extending the frequency count to the trigram, and tetragram subsets, etc.

The second approach, the $n{:}1$, works in the reverse direction. An analysis, and frequency count, of the substrings of length ten is made and those substrings with a count above a chosen threshold are accepted as keys. Consecutive truncation of the substrings creates larger frequency counts and substrings are entered in the key set when the threshold is exceeded.

After the dictionary has been established, a list of keys is set up for each document. For each index word one key is selected by finding the longest entry in the dictionary generated from that word. A query composed of index terms connected by the Boolean 'and' also generates a list of keys, one for each word which must be present in a relevant document. The search phase then consists of finding documents in which the query key set is a subset of the document key set.

The word based method has the following properties:

1. The substring analysis always starts at the first character of an index word and has a maximum length of scan; in the work presented here, this maximum is ten. This means that the computation is considerably less than that for the Harrison or Lynch methods.
2. The method truncates words, which groups grammatical variations with the word stem. This is very useful for retrieval purposes.
3. The distortion of the frequency pattern caused by the spaces is avoided.

4. The size of the dictionary can be controlled by varying the size of the threshold.

5. The method gives a small key dictionary so that the key corresponding to an index word can be found economically.

Results

1. *The key dictionaries*

A series of computations was carried out on two files F100 and F1000 to establish key dictionaries and select the most suitable dictionaries for retrieval trials. A selection of these dictionaries showing different threshold levels, two different file sizes, and using the two alternative processing methods is presented in **Table 1.**

The files F100 and F1000 are derived from the INSPEC data base by extracting the index words from 100 and 1000 records respectively.

Table 1 The key dictionaries

		Threshold	*Average frequency*	*Size of key dictionary*
1:n method				
F100	A	35	16	92
	B	50	23	63
F1000	C	350	181	83
	D	500	225	67
n:1 method				
F1000	E	100	173	87
	F	40	75	200

The following points should be noted.

Threshold level

The figures for the 1:*n* method in **Table 1** show that if the file size is increased, then an increase in the threshold level in the same proportion maintains a dictionary of approximately the same size, as shown by the figures 92, 83; 63 and 67. 74 of the keys in dictionary C also appear in dictionary A.

Since the threshold level in the 1:*n* method is an upper bound, the average number of occurrences of any key is substantially less than the threshold; the threshold in the *n*:1 is a lower bound and the average is higher. This explains the marked difference in threshold level between the two similar size key dictionaries C and E for F1000.

Method of dictionary generation

The 1:*n* method for dictionary C and the *n*:1 method for dictionary E give the key distribution in **Table 2.**

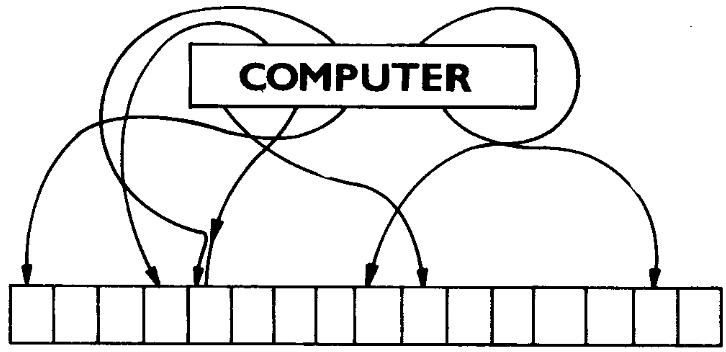
Table 2 Key distribution

	Dictionary C 1:*n*	*Dictionary E* *n*:1
No. of characters	36	36
No. of digrams	37	23
No. of trigrams	8	9
No. of tetragrams	2	19
Average length of keys No. of characters	1·71	2·13
Relative entropy	0·9372	0·9336

The entropy is a measure of equifrequency defined by

$$H = \sum_{i=1}^{N} - p_i \log_2 p_i$$

where N is the total number of different keys and p_i is the probability of occurrence of key i. $H_{max} = \log_2 N$ and relative entropy equals H/H_{max} (See Shannon, 1948). The difference in relative entropy is insignificant and therefore both methods were selected for the retrieval trials since the higher average key length is likely to be beneficial for retrieval work.



Document No.	*Signature*
1	0 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1
2	1 0 0 0 0 1 1 1 0 1 1 0 0 1 0 1
3	- - - - - - - - - - - - - - - -
- - -	- - - - - - - - - - - - - - - -

Fig. 1 A Harrison signature using Digram

Choice of dictionary size

The larger file requires a larger key dictionary to maintain the level of retrieval efficiency, (as confirmed by the precision figures given in **Tables 5** and **6**) so dictionary F was introduced to provide data for a key dictionary on F1000 at the higher levels of precision.

Choice of storage and search method

The second stage in the retrieval process is the allocation and storage of keys for the documents in the collection. The 1:*n* and *n*:1 methods of allocating keys are described in the papers by Lynch; attention here is directed to the storage and search methods. The Harrison method creates a Boolean vector for each document showing the presence or absence of the hashed value of each substring as shown in **Fig. 1.** The set of vectors is then scanned serially during searching. A similar approach can be implemented for the word based method by setting up for each document a Boolean vector showing the presence or absence of each key.

The cost of the serial search was too high even for these small key dictionaries (as shown by **Table 3**) so an inverted file was used. For each key a list of documents containing that substring is created. For any query there are generated several keys each of which has a corresponding list of documents. The documents relevant to the query are those in all the lists. Three search strategies are used:

I The first document number of the first list is checked to determine if it occurs in all lists. If so, a relevant document has been found.
The second entry in the first list is then checked against all the lists and the process repeated until all entries in the first list have been checked.

J. An array with a count for each document number is set up. The number of occurrences of each document number in the combined lists from the inverted index file is entered in the array.
The relevant documents are those with a count equal to the number of keys generated from the search term.

K. The lists are ordered, which suggests a third method. Pointers are used in each list to indicate the target elements for the next comparison.

In the case of a match in all lists the pointers all move to the next elements in the list. Otherwise the pointer for the smallest document number is moved. The process terminates when one list is exhausted.

The timings for these four methods, the serial search S and methods I, J, K, were evaluated for dictionaries A and B on File F100 for a set of 12 queries, consisting of 45 index words. The times in milliseconds for a CDC 7600 COBOL program are given in **Table 3**.

Table 3 Search times for F100 (times are in milliseconds)

	S	*I*	*J*	*K*	
Processing time	77	67	66	56	
Overhead	46	39	42	46	Dictionary *A* 92 keys
Total time	123	106	108	102	
Average search time/query	6·41	5·58	5·50	4·66	
Processing time	89	85	85	73	
Overhead	45	31	31	32	Dictionary *B* 63 keys
Total time	134	116	116	105	
Average search time/query	7·41	7·08	7·08	6·08	

Methods I, J and K were used on F1000 for the dictionaries C, D, F. Twentyfive queries were used for evaluating retrieval times which are shown in Table 4. These figures confirm the results of Table 3; Method K is the fastest method in each case.

Table 4 Search times for F1000 (times are in milliseconds)

		Dictionary C	*Dictionary D*	*Dictionary F*
No. of keys		83	87	200
Average Search	I	47·84	45·48	20·88
Time per Query	J	35·88	35·20	17·60
	K	35·64	34·52	17·32

It should be noted that both the precision and the timing can be improved by increasing the size of the key set, because the number of documents per key is the most significant parameter.

Table 5 Retrieval performance for F100

	Dictionary B	*Dictionary A*
No. of keys	63	92
No. documents retrieved	33	22
Precision, ratio of averages	55%	62%
Precision, average of ratios	60%	83%

No. of relevant documents = 18

Table 6 Retrieval performance for F1000

		No. of relevant documents			*No. of documents retrieved*		
Query	*No. of words in the query*	1 (*)	2 (*a*)	3 (*b*)	*Dictionary C* 83	*Dictionary D* 87	*Dictionary F* 200
1	4	1			1	1	1
2	4	1			3	3	1
3	4	1			1	1	1
4	4	1			1	2	1
5	4	1			28	13	1
6	3	—			8	21	1
7	3	1			12	15	2
8	3	1			10	6	7
9	3	2			13	6	3
10	3	1			1	19	2
11	3	4			15	14	4
12	3	7	14	16	16	16	16
13	3	7			12	12	8
14	3	5			14	12	7
15	3	3			8	7	3
16	3	23	28		38	30	28
17	3	1			2	2	2
18	3	1			4	1	1
19	3	1			3	2	1
20	3	2			6	5	4
21	3	1			6	2	1
22	2	1	6		45	29	9
23	2	1			7	4	4
24	2	10	16		17	45	16
25	2	2			9	6	5
	76	79	102	104	280	274	129

I Precision, ratio of averages (%)	(*)	24·60	28·84	61·24
II Precision, ratio of averages (%)	(*b*)	37·16	37·96	80·06
I Precision, average of ratios (%)	(*)	39·54	40·73	68·16
II Precision, average of ratios (%)	(*b*)	44·17	44·87	74·86

*Number of nondistributed hits

(*a*) Number of distributed hits

(*b*) Number of distributed relevant strikes

Neither the size of the key dictionary nor the search time increase in proportion to the increase in file size. A comparison of the 200 key set with the 92, which has a similar level of precision show that an increase in file size by a factor of ten increases dictionary size and processing time by factors of only 2·20 and 3·72 respectively. This indicates that the method could usefully be considered for larger files.

2. *Evaluation of precision*

A set of 12 queries, consisting of 45 key words, was used in retrieval tests for the dictionaries A and B on F100. The maximum and minimum number of words in the queries were six and two respectively. A search was considered to be successful if a match was obtained for each index word in the query. The results, presented in Table 5, show that the larger dictionary A is much more suitable for retrieval use.

A set of 25 queries, each composed of single index terms of between two and four index words was used for retrieval of F1000 for the dictionaries C, D and F.

In both Table 5 and Table 6 the precision for a batch of queries was evaluated in two ways. The precision for each query was calculated and then averaged (average of the ratios) and the overall ratio of the total relevant documents/total retrieved documents was used (ratio of averages).

A study of the results that are summarised in Table 6 revealed an interesting feature of the method. There were some cases in which the index terms of the query and the document did not match, but the keys derived from the individual words did match, owing to a different distribution of the index words in the query and the document. A careful discussion of relevance is given here which takes account of these distributed matches.

The term *strike* will be used for a match of two keys and *hit* for the case where the corresponding index words also match. When there is a strike but no hit, further inspection is necessary to determine if the retrieval is successful. A strike of words which are grammatical variations would mean that the retrieval was relevant, whereas a strike resulting from different index words (such as 'compression' and 'computer' with retrieval key 'comp') would give a nonrelevant retrieval.

Matches will be graded in three types:

(*a*) a hit

(*b*) a relevant strike

(*c*) a nonmatching key.

Further refinement is needed in the case of a *search* term composed of more than one index word. The matches may all be of type (*a*) or (*b*), which are adjudged successful, but the matches may not occur in the same index term in the document. The relevance score should therefore show how many document index terms contain the keys generated by the query term. The term *spread* will be used to indicate the number of document terms in which the strikes are distributed.

A match in which the spread is greater than one will be called a *distributed match*. The examples of distributed matches in **Table 7** show that the number of matches of type (*a*) or (*b*) is much more significant than the distribution. Indeed, one of the advantages of this method is that a match of two relevant documents can be obtained although the indexer and the searcher use different index terms. If this system were accepted, the frequent repetition in index terms for the same document of words such as 'system' or 'measurement' could be avoided, resulting in a saving of space. The two precision measures in Table 6 are for full matches of nondistributed hits and for full matches of distributed relevant strikes which may be type (*a*) or (*b*) matches.

The difference in the precision values I and II is a measure of the added power of the suggested method. It is substantial for dictionary F.

Table 7 Relevant distributed strikes

	Search descriptor	*Document description*
1.	Linear differential equations	*Linear* systems; *differential equations*; *Linear* plant model; homogeneous *differential equations*;
2.	Computer controlled telephone exchanges	*Telephone exchanges* in large city areas; *computer controlled*;
3.	Surface acoustic waves	*Acoustic* imaging; *surface waves*;
4.	Digital integrated circuits	*Digital* filters; *integrated circuits*;

Conclusions

1. The retrieval performance of around 80% precision with this method is good, which justifies the choice of word based analysis of free index terms. Most other work has been with titles which are unsuitable for a word based analysis owing to the high number of noncontent words such as 'the', 'and', 'of', etc.
2. Word based analysis introduces word stemming which is a useful feature for retrieval. Thus 'computing', 'computation' and 'compute' will all be retrieved under the key 'comput'.
3. A comparison of the figures for nondistribution hits and distributed relevant strikes shows that this word based method is on average higher in precision than a method based on string fragments. This results from changes in word order and distributed matches which other methods miss. This, coupled with the considerable economy in processing time, suggests that word based methods may make string processing competitive with conventional indexing, in view of the reduction of the effort needed by both the indexer and the user.
4. The search times are small resulting from the limited nature of the character scan. This could be further improved by more sophisticated storage methods. A comparison of full text substring fragment analysis with this word based fragment method indicates a substantial saving in computer time, both at the index production stage and at the matching stage. Consider a free language indexing field with 14 words of 8 characters, which were the averages for the material studied. This will generate $14 \times 8 = 112$ substrings to be counted in the frequency analysis. A full text field of this length, i.e. $8 \times 14 + 13 = 125$ will generate $(125 + 124 + \ldots 116) = 1205$ substrings if a maximum scan length of ten characters is used.

For the word fragment method there are 14 entries in the inverted file lists, whereas there are 125 for full text redundant analysis. In the matching stage a typical index term of three words generates 3 word fragments for matching compared with 26 text fragments from $3 \times 8 + 2$ characters.

Acknowledgement

We would like to thank the Institution of Electrical Engineers for the provision of a section of the INSPEC data base on which this work was carried out.

References

Clare, A. C., Cook, E. M., and Lynch, M. F. (1972). The Identification of Variable Length, Equifrequent Character Strings in a Natural Language Database, *The Computer Journal*, Vol. 15, pp. 259-262.

Clough, C. R., Bramwell, K. M. (1971). A Single Computer Based System for Both Current Awareness and Retrospective Search: Operating Experience with ASSASSIN, *J. Doc.*, Vol. 27, No. 4, pp. 243-253.

Goble, C. E. (1975). A Free-text Retrieval System Using Hash-codes, *The Computer Journal*, Vol. 18, No. 1, pp. 18-20.
Lynch, M. F., Petrie, J. H., and Snell, M. J. (1973). Analysis of the Microstructure of Titles in the INSPEC Database, *Inform. Stor. and Retr.*, Vol. 9, pp. 331-337.
Fokker, D. W. and Lynch, M. F. (1974). Application of Variety-Generator Approach to Searches of Personal Names in Bibliographic Data Bases, *J. Lib. Aut.*, Vol. 7, pp. 105-118.
Harrison, M. C. (1971). Implementation of the Substring Test by Hashing, *CACM*, Vol. 14, pp. 777-779.
Schuegraf, E. J. and Heaps, H. S. (1973). Selection of Equifrequent Word Fragments for Information Retrieval, *Inform. Stor. and Retr.*, Vol. 9, pp. 697-711.
Schuegraf, E. J. and Heaps, H. S. (1974). A Comparison of Algorithms for Data Base Compression by Use of Fragments as Language Elements, *Inform. Stor. and Retr.*, Vol. 10, pp. 309-319.
Shannon, C. E. (1948). A Mathematical Theory of Communication, *Bell System Tech. J.*, Vol. 27, pp. 379-423.
Zipf, G. K. (1949). *Human Behaviour and the Principle of Least Effort*, Addison-Wesley, Cambridge, Mass.

Book review

Logical Construction of Programs, by J-D. Warnier, translated by B. M. Flanagan, 1976; 221 pages. (*H. E. Stenfert Kroese bv*, Leiden, £7·15)

The program design method developed and advocated by J-D. Warnier and his colleagues has gained wide acceptance in France and in some other countries also. This book is an attempt to provide a reasonably succinct presentation of the method for readers who require a comprehensive view of its essentials rather than training in its use. Unfortunately, the translation is poor. Having read, in the introduction, that considerable improvements in productivity can be achieved, we encounter the sentences:

> 'The attempt to apply programming methods already yields appreciable improvements. Today this effort appears inadequate, since by using the logic of program construction and for the organisation of data that exist, the abovementioned results may be obtained. The use of this logic presupposes thorough training in theory and in practice.'

This kind of translation harms the book in two ways. First, it makes it rather uncomfortable to read. Second, and more important, it undermines the reader's confidence that he has understood exactly what is meant. The subject matter of the book, Warnier's design method, deserves better.

The foundation of Warnier's method is the formation of a program structure to correspond to the structure of the input data. The input data structure is represented in a hierarchy of repetitive and alternative structures; the program structure is based directly on this hierarchy and is represented first in hierarchical and then in flowchart form; lists of instructions (including branch instructions) are drawn up and allocated to the procedure boxes of the flowchart; the allocation of instructions is validated by reference to a hierarchical representation of the output data. The whole forms a basic design procedure that can be carried out step by step, with usable criteria for the correctness of what is done at each step.

Superimposed on this foundation is an elaborate treatment of truth tables and their optimisation, including the use of Karnaugh maps and some Boolean algebra. Also superimposed is the notion of 'processing phases'. A program consists of more than one processing phase if it contains any conditional branch instruction which uses an identification criterion not present in the input data but created by the program itself. For example, if an input record has to be tested for several possible error conditions, a switch may be set on to indicate whether or not the current record has been found to be in error; since there is no initial identification criterion indicating whether or not the record is in error, use of this switch in a conditional branch instruction involves a change from one phase of processing to another. Separate data diagrams (hierarchies) must be drawn for the data input to the second and subsequent phases.

To this reviewer, the concept of processing phases seems too weak to bear the weight that must fall upon it. Passage from one phase to another occurs only in the context of a higher level structure which is synchronised with—indeed, derived from—the input structure; it does not, therefore, seem possible to use the technique to handle programs whose data exhibits irreconcilably conflicting structures and which must therefore be decomposed into two or more synchronous processes. Warnier gives no hint in this book of the design process by which program specifications are obtained, nor of any basis for judging which specifications are acceptable for program design and which are not.

There are several detailed points which invite criticism. The use of flowcharts seems cumbersome both for design and for maintenance. The repetitive and alternative structures become awkward to handle without a sequence structure: what could have been simply a sequence of two alternatives becomes a 'complex alternative'; a sequence of repetition and alternative becomes a 'complex mixed' structure. The repetitive and alternative structures themselves present some difficulties: 'balance is either debit-balance or credit-balance' becomes 'balance is debit-balance (0 or 1 times) exclusive-or credit-balance (0 or 1 times)'; the repetitive structure contains one or more (rather than the preferable zero or more) of the repeated subset.

But these are minor criticisms of a serious and valuable attempt to fill what many now recognise to be a vital need: a coherent, rational, step-by-step approach to the design of computer programs. Everyone who is seriously interested in program design should read this book if he cannot read the French original (1974).

M. A. Jackson (London)

Reference

Precis de Logique Informatique: les Procedures de Traitement et leurs Données; par J-D. Warnier; Les Editions d'Organisation, Paris; 1974.