Some rules for introducing indexing paths in a primary file

M. Hatzopoulos* and J. G. Kollias†

A framework is developed which places in perspective with other approaches a model developed by Kollias (1976) for the selection of indexing paths in a primary file. The paper extends the model and proves a number of properties which characterise it. These properties not only simplify the solution of the model but they can also be used by file designers as rules when selecting indexing paths for their applications.

(Received January 1979)

1. Introduction

It is a common practice to have as a design objective for an efficient Information Processing System (IPS) the selection of the file organisations which perform the anticipated users' data processing requirements at an optimal number of disc references. File designers are faced with two main difficulties when trying to meet this design objective. The first difficulty is the result of observing that there is a large number of file organisations which makes it impossible to deal with every existing file organisation. The second difficulty stems from the fact that file organisations are always compared with respect to the type of user transactions they are asked to serve. This implies that one file organisation might be better than another in terms of retrievals but might be worse in terms of updates.

A number of authors have tried to describe (Dodd, 1969), formalise (Hsiao and Harary, 1970) and quantitatively to evaluate (Lefkovitz, 1969) possible file organisations to assist designers towards optimal decisions. Nevertheless these fail to produce workable rules for assisting the determination of the file organisations for new applications. Such a rule might state that: 'When updates are more than 10% of the total transactions and the file size is less than 2,000 buckets then apply the multilist file organisation, otherwise the inverted one'. The absence of any such break points (if such exist) among different file organisations explains why it is common for file designers to rely for the proper file organisation selection to intuition or experience.

File organisations can be classified as primary or secondary ones depending on whether or not they are used for queries based on primary or secondary key values respectively. In this study it is assumed that the primary file organisation problem has a satisfactory solution (Knuth, 1973). The most common approach to secondary file organisations is the introduction of indexing paths in the primary file, commonly called secondary indexes, which aim to improve the retrieval time because they make it possible to find all the records in the file which satisfy a query without accessing other records which do not. These indexing paths introduce extra costs to the IPS when new records are added, when old ones are deleted, or when updates are performed. This is so because it is not only the primary file that has to be modified (something that will be done in any case) but the indexing paths as well.

In Kollias (1976) an IPS is considered which serves, during a unit time period of operation, a number of transactions which may be classified into queries, updates, insertions and deletions. The IPS receives n types of query, Q_j, with expected respective loads, $q_j, j = 1, 2, \ldots, n$. Each query may contain in its qualification part one or more secondary attributes which may specify either a simple or a range of the values that they take. Update transactions are identified as U_k, with a known update load u_k, $k = 1, 2, \ldots, s$. Consideration is given to record insertion, I, and deletion, D; loads are I and D respectively. A model is developed which solves the problem of selecting indexing paths by treating it analogously to the 'plant location' problem in OR (Effroymson and Ray, 1966; Spielberg, 1969).

In this paper a number of properties are proved which characterise the model. These properties may be utilised in the following two contexts: (*a*) to introduce rules to be used by file designers when selecting indexing paths for their applications, and (*b*) to simplify the solution of the model. In the next section a framework is developed which places the work in perspective with other different approaches to the problem of organising indexing paths.

2. Different approaches to file organisation

Advances in the area of IPS technology imply a better physical structuring of the files in order to meet the users' data processing requirements at a lower cost than any other previous system. Related studies claim that such an improvement can be achieved by research in the following two directions:

1. To modify established techniques of organising files and/or to introduce new advanced techniques.
2. To improve the ways in which file organisations are selected and/or implemented.

With respect to the first approach it is thought difficult to develop some new technique for organising files which contributes much to the problem of improving the IPS performance. For supporting the argument a method of organising files developed by Lum (1970) is quoted. The idea of the method is that instead of inverting an attribute (simple inversion) two or more attributes may be combined to invert on all of them (compound inversion). This results in more efficient satisfaction of queries which contain more than one attribute in their qualification part. The method has been described by Knuth (1973) as the third one in the rank of existing methods for introducing indexing paths in a primary file (after the inverted and the multilist one). Cardenas (1975) reports that the method has never been used in actual practice, the reason being that its fast retrievals cannot outweigh the high cost of its updates, even for relatively static files (e.g. library indexes to documents, etc). Therefore Lum's method is superior to widely used simple inversion only under very restrictive IPS usage.

With respect to the second approach there are two trends. One by Cardenas (1973, 1975) and Lum and Ling (1971), and another by Senko (1968). Before presenting these two trends a formal concept of IPS is introduced.

The parameters affecting the performance of an IPS can be

*Unit of Applied Mathematics, University of Athens, Athens, Greece.
†Department of Computer Science, University of Patras, Patras, Greece.

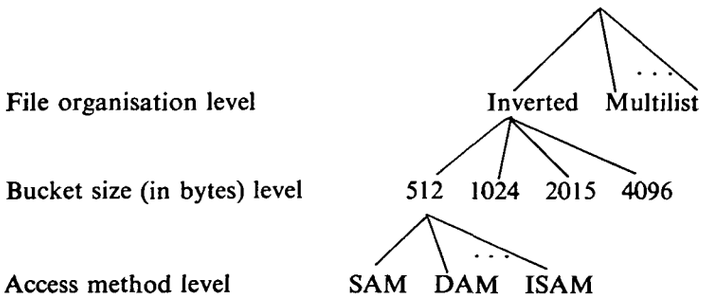


Fig. 1 Some of the parameters affecting the selection of file organisations.

hierarchically classified. In the higher level of this classification might be a description of the users' view of data quite independently of any implementation while the lower level might include design parameters which are located very close to the machine. A design parameter considered as an individual entity at one level can be considered as a composite object at the next lower level of greater detail.

Fig. 1 displays some parameters affecting the selection of file organisations. At level 1 of this figure are all the candidate file organisations to be considered. But within a particular file organisation one can employ different bucket sizes on disc (Level 2), where a bucket is taken to be the unit of transfer between disc storage and core memory. Depending on the level that the design possibilities are measured the solution space of the problem can have a number of points. For example consider a simple type of query to be satisfied either by the inverted or by the multilist file organisation when all the parameters affecting any decision are kept constant. The solution space of the problem has just 2 points (i.e. inverted, multilist) since all the design parameters to be found at the lower levels have shrunk into single points. For example, it is assumed that the bucket size is of constant length, say 512 bytes, and that the access method employed (among those offered by the data management part of the operating system) is, say, the Index Sequential Access Method (ISAM) (Fig. 1).

The followers of the first trend claim that an improvement can be achieved if instead of striving to optimise the design of IPS at the lowest possible level, concentration is given to the file organisation level. The research efforts are dictated either to more careful implementation and the analysis of trading off a limiting number of file organisations (Cardenas, 1973; 1975), or in undertaking a more comprehensive view of the impact of various file organisations within an IPS (Lum and Ling, 1971).

The creation of the FOREM model (Senko *et al*, 1968), developed for simulating IPS performances, is the result of the second trend. This trend holds that the inefficiencies of IPS are due to the fact that few design possibilities are examined prior to any implementation. (This case exists in general when optimisation is performed at the file organisation level). FOREM, for example, is capable of estimating the performance of a file organisation by varying lower level parameters like bucket sizes, access methods, etc. It is worth noting (*a*) that the reported utilisations of FOREM in the IPS design search the problem solution space by using educated guesses (Senko, 1972; Wang and Lum, 1971), and (*b*) no work is reported to define the size of the solution space or how to search it algorithmically.

It is now possible to describe precisely the approach to the problem of introducing indexing paths in a primary file. The same approach has been taken by Stonebraker (1974) and Sckholnick (1975). Firstly, the approach will use established techniques for organising indexing paths. Secondly, the source of improvements is a careful examination of the alternative possibilities at the file organisation level. All the lower level parameters are taken as constant. Thirdly the above view of the problem is applied to the operational environments presented in Section 1. This shows that thousands of different design possibilities arise even if the selection is made at the file organisation level and even if it is restricted to a single file organisation.

In Kollias (1976) it is shown that the problem solution space can be searched by integer programming techniques. In this study the properties of this integer programming model are generalised and explored. Similar analysis was undertaken by Grapa and Belford (1977) for the problem of placing copies of a file at different nodes of a computer network (Casey, 1972).

3. The extended model

When indexing paths in a primary file are being planned planners are faced with a number of policies which are called (*basic*) *policies*. The first policy, p_1 is termed *primary file policy* and serves every query with a time consuming serial scan but it does not introduce extra storage and maintenance costs in the IPS. To improve retrieval on secondary attribute values other types of policy should also be considered. The first type corresponds to the existence in the IPS of one indexing path for every attribute appearing at least once in the qualification part of the queries and is called *simple* type of policies.

Example

A primary file containing forestry data is to be considered (Kollias, 1976). The file requires 9,091 buckets of storage and consists of 100,000 records, each record corresponding to a specimen tree. The attribute with the name NUMBER is the primary key for purposes of storage and retrieval. It is assumed that the application supports the types of transaction and transaction loads described in **Table 1.** It is worth noting that the 8th transaction ('Given ADDRESS update DATE-2'), although of update type, has been identified as query (i.e. Q_6). This is because any update transaction based on secondary attribute values is facilitated by the use of indexing paths, in the same way as ordinary queries. Five additional candidate basic policies p_2 to p_6 result from the usage displayed in Table 1. These policies concern the existence in the IPS of an indexing path for each of the attributes DATE-1, ADDRESS, WIDTH, HEIGHT and REFERENCE respectively.

Assume now that the inverted type of file organisation has been employed for structuring the indexing paths. Then the cost of satisfying a query having two secondary attributes in its qualification part can be reduced if the following searching process is undertaken. Provided that both the indexing paths exist in the IPS then two lists are formed. Each of these lists has as elements the disc addresses of the records in the primary file which contain respectively the specified in the query values for the first and the second attribute. The buckets in the primary file to be retrieved can then be determined by taking the intersection of the two lists. To accommodate the savings resulting from this process, additional distinct candidate policies to those which allow more than one indexing path to be used in the searching process are incorporated. (Note: This type of policy was not considered in Kollias (1976)).

From Table 1 it can be seen that the Q_2, Q_4 and Q_5 types of query suggest that the next three policies p_7, p_8 and p_9 must include respectively the three indexing path pairs ADDRESS and DATE-1, HEIGHT and ADDRESS, and WIDTH and REFERENCE. These policies are called *multiple* policies.

To present the model it is assumed that the types of query dictate the consideration of m candidate policies, p_i ($i = 1, 2, \ldots, m$). Let c_{ij} = the cost of satisfying the j-th type of query under the i-th policy. When a policy is incapable of satisfying a query it is marked with a prohibitively large cost, $c_{ij} = L$;

c'_{ik} = The cost of satisfying the k-th type of update under the i-th policy. This cost is zero when the k-th update does not affect the indexing paths involved with the i-th policy; $C'_i(C''_i)$ = The cost of performing an insertion (deletion) to the i-th policy.

Two decision variables are required: $x_{ij} = 1$ if the policy p_i satisfies the type of query Q_j, $= 0$ otherwise; $y_i = 1$ if the policy p_i is employed, $= 0$ otherwise. To every basic policy there corresponds a fixed cost, f_i, associated with its maintenance. By taking the fixed costs to be

$$f_i = \sum_{k=1}^{s} u_k c'_{ik} + I C'_i + D C''_i$$

the problem of selecting indexing paths can be formulated as: Minimise

$$\sum_{i=1}^{m} \sum_{j=1}^{n} q_j c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i$$

subject to the constraints

$$\sum_{i=1}^{m} x_{ij} = 1$$

$$\sum_{j=1}^{n} x_{ij} \leqslant n y_i \qquad \text{(LP)}$$

$$y_i = 0 \text{ or } 1, \quad 0 \leqslant x_{ij} \leqslant 1$$

(LP) is known as the 'simple plant location' problem which has 2^m possible solutions, where m is the number of candidate policies. But 2^m grows exponentially, which implies that the efficiency of any algorithm (Effroymson and Ray, 1966; Spielberg, 1969) declines as the number of candidate policies increases. In the next section we introduce a number of rules which assist file designers to handle (LP). Before doing so the following observation is made. A closer look at the policies p_7 and p_8 shows (*a*) that these two policies satisfy the Q_2 and Q_4 types of query respectively at the optimal possible cost, and (*b*) that they contain a common indexing path. Their first characteristic suggests that it is likely to be included in the optimal strategy, while the second indicates that the extra maintenance cost needed is equal to the sum of the corresponding costs when maintaining the indexing paths for the ADDRESS, DATE and HEIGHT attributes. But any combination of policies considers that the maintenance required is the sum of the costs needed by each basic policy appearing in it. This implies that the policy combination p_7 and p_8 might be rejected as uneconomical on the assumption that it contains four indexing paths instead of three, which is indeed what the combination requires. For these reasons another candidate policy, p_{10}, has to be incorporated in the model, termed *combined multiple policy*. This policy involves indexing paths for the attributes ADDRESS, HEIGHT and DATE-1.

Table 1 Type and load of user transactions

Transaction

No.	*Type*	*Description*	*Load*
1	Q_1	Given *range* of DATE-1 retrieve records	q_1
2	Q_2	Given ADDRESS, DATE-1 retrieve records	q_2
3	Q_3	Given WIDTH retrieve records	q_3
4	Q_4	Given HEIGHT, *range* of ADDRESS retrieve records	q_4
5	Q_5	Given WIDTH, REFERENCE retrieve records	q_5
6	U_1	Given NUMBER update DATE-1, HEIGHT, WIDTH	u_1
7	U_2	Given NUMBER update ADDRESS	u_2
8	Q_6	Given ADDRESS update DATE-2	q_6
9	I	Insert a record	INS
10	D	Delete a record	DEL

Example

In **Table 2** are listed all the parameters related to the basic policies resulting from the usage displayed in Table 1 along with the cost of performing *one* user transaction under each candidate policy. These costs are expressed in terms of bucket accesses to disc and have been estimated by considering that the inverted type of file organisation has been employed to structure indexing paths. The formulae applied are reported in Kollias (1976). (Note: The range of values which appear in Q_1 and Q_4 (Table 1) is assumed to have the values 3 and 5 respectively).

4. Rules and properties

In the previous section it is seen that the complexity of (LP) can be expressed by the 2^m where m is the number of candidate policies. In practice it is likely that the queries supported by the IPS can be partitioned into k disjoint sets, each set requiring its own candidate indexing paths quite independently from the queries belonging to the other sets. If this is the case it can relate to every set of queries a distinct problem of the form (LP) instead of just one problem. For example the queries appearing in Table 1 suggest that two optimisation problems can be formed. The first one is dictated by the queries Q_1, Q_2, Q_4 and Q_6 while the second problem is instructed by the Q_3 and Q_5 types of query. This is because the attributes DATE-1, ADDRESS and HEIGHT which appear in the qualification parts of the first set of queries do not intersect with the attributes WIDTH and REFERENCE belonging to the second set.

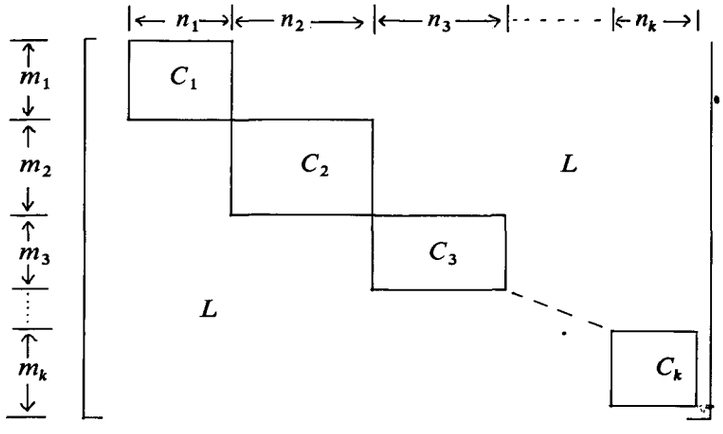
Table 2 Parameter and performance values related with basic policies

p_i	*Indexing paths**	c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{15}	c_{16}	c'_{i1}	c'_{i2}	C'_i	C''_i	f_i
p_1	None	9091	9091	9091	9091	9091	9091	0	0	0	0	0
p_2	D	311	111	L	L	L	L	24	0	12	12	48
p_3	A	L	211	L	1011	L	211	0	24	12	12	48
p_4	W	L	L	261	L	261	L	24	0	12	12	48
p_5	H	L	L	L	411	L	L	24	0	12	12	48
p_6	R	L	L	L	L	411	L	0	0	12	12	24
p_7	A, D	311	23	L	1011	L	211	24	24	24	24	96
p_8	A, H	L	211	L	26	L	211	24	24	24	24	96
p_9	W, R	L	L	261	L	23	L	24	0	24	24	72
p_{10}	A, H, D	311	23	L	26	L	211	48	24	36	36	144

*A, H, W, D and R within the column stands for the attribute names ADDRESS, HEIGHT, WIDTH, DATE-1 and REFERENCE respectively.

If it is assumed (*a*) that the original (LP) problem involves m candidate policies $m-1$ of which correspond to indexing paths and the other one is the primary file policy, and (*b*) each of the resultant k problems introduces $m_1+1, m_2+1, \ldots, m_k+1$ candidate policies, where $m_1, m_2, \ldots, m_k$ denote policies involving indexing paths (i.e. $m_1+m_2+\ldots+m_k=m-1$) and the 1's relate to the primary file policy then the computational effort to be extended after following the property is proportional to $2^{m_1+1}+2^{m_2+1}+\ldots+2^{m_k+1}$ as opposed to 2^m. With reference to (LP) the property can be translated to the following rule.

Rule 1
If for $i=2,\ldots,m$ there is a column/row permutation of the cost matrix c_{ij} that transforms the matrix to a block diagonal type of the form



where $m_1+m_2+\ldots+m_k=m-1$, $n_1+n_2+\ldots+n_k=n$ and all the elements which are not equal to L are on the diagonal blocks then (LP) can be transferred to k independent problems.

The above rule applied to Table 2 shows that the two problems defined involve respectively 7 (i.e. $p_1, p_2, p_3, p_5, p_7, p_8$ and p_{10}) and 4 (i.e. p_1, p_4, p_6 and p_9) candidate policies. This implies that the computational effort for deciding on the optimal selection of indexing paths to support the usage in Table 1 can be dropped from 2^{10} to 2^7+2^4.

Let I_m denote the set of all the m candidate policies and I is the set of policy indexes representing a given selection of indexing paths (i.e. $I=\{i \mid y_i=1\}$). (LP) can then be written as:

Choose the set I that minimises

$$C(I)=\sum_{j=1}^{n} q_j \operatorname*{minc}_{k\in I} c_{kj}+\sum_{k\in I} f_k \qquad (1)$$

Theorem 1
If for some query j

$$c_{ij}=\operatorname*{minc}_{k\in I_m} c_{kj} \qquad (2)$$

and

$$q_j c_{ij}+f_i<q_j \min_{k\in I_m-\{i\}} c_{kj} \qquad (3)$$

then the i-th policy appears in the optimal strategy.

Proof
Let I be the optimal solution of (1). Assume that the i-th policy satisfies (2) and (3) for some query j and that it is not included in I. Let $I'=I\cup\{i\}$.
From (1):

$C(I)=\sum_{j=1}^{n} q_j \operatorname*{minc}_{k\in I} c_{kj}+\sum_{k\in I} f_k$. By definition of I':

$$C(I')=\sum_{j=1}^{n} q_j \operatorname*{minc}_{k\in I} c_{kj}+\sum_{k\in I} f_k+f_i$$

$$=C(I)+f_i+\sum_{j=1}^{n} q_j\left(\operatorname*{minc}_{k\in I'} c_{kj}-\operatorname*{minc}_{k\in I} c_{kj}\right) \qquad (4)$$

Since $I\subset I'$ the relation $\operatorname*{minc}_{k\in I'} c_{kj}-\operatorname*{minc}_{k\in I} c_{kj}\leqslant 0$ is satisfied for every j.
So from (4):

$$C(I')\leqslant C(I)+f_i+q_j\left(\operatorname*{minc}_{k\in I'} c_{kj}-\operatorname*{minc}_{k\in I} c_{kj}\right)$$

$$=C(I)+f_i+q_j\left(c_{ij}-\operatorname*{minc}_{k\in I} c_{kj}\right) \qquad \text{(from (2))}$$

From (3):

$$q_j c_{ij}+f_i<q_j \operatorname*{minc}_{k\in I_m-\{i\}} c_{kj}\leqslant q_j \operatorname*{minc}_{k\in I} c_{kj}$$

i.e. $C(I')<C(I)$. This contradicts the assumed optimality of I. The above theorem can be translated to the following rule.

Rule 2
If a query is satisfied by the i-th policy and the cost of maintaining this policy is smaller than the smallest possible cost of satisfying the query by other policies then the i-th policy is included in the optimal strategy. With reference to Table 2 it can be seen that the indexing paths involved in the 9-th policy have to be included in the optimal solution. A theorem is proven which allows policies which cannot coexist in the optimal strategy to be located.

The following notation is used:

$$(a)_+=\begin{cases} a \text{ if } a\geqslant 0 \\ 0 \text{ if } a<0 \end{cases}$$

Theorem 2
If the i-th and l-th policies satisfy

$$f_i>\sum_{j=1}^{n} q_j(c_{lj}-c_{ij})_+ \qquad (5)$$

then these policies cannot both be included in the optimal strategy.

Proof
Let I be an arbitrary selection of policies that do not include the indexing paths involved in p_i and p_l.
Let $I'=I\cup\{l\}$ and $I''=I'\cup\{i\}$. In fact, it need only show that $C(I'')>C(I')$. Then:

$$C(I'')=C(I')+f_i+\sum_{j=1}^{n} q_j\left(\operatorname*{minc}_{k\in I''} c_{kj}-\operatorname*{minc}_{k\in I'} c_{kj}\right).$$

From (5) we get

$$C(I'')>C(I')+\sum_{j=1}^{n} q_j\left[(c_{lj}-c_{ij})_+ +\operatorname*{minc}_{k\in I''} c_{kj}-\operatorname*{minc}_{k\in I'} c_{kj}\right]$$

But every term

$$(c_{lj}-c_{ij})_+ +\operatorname*{minc}_{k\in I''} c_{kj}-\operatorname*{minc}_{k\in I'} c_{kj}\geqslant 0 \qquad (6)$$

Inequality (6) can be verified by checking cases, (*a*) if $\operatorname*{minc}_{k\in I''} c_{kj}\neq c_{ij}$ then the last two terms of (6) cancel and (6) is satisfied (because by definition $(c_{lj}-c_{ij})_+\geqslant 0$), and (*b*) if $\operatorname*{minc}_{k\in I''} c_{kj}=c_{ij}$ then $c_{lj}-c_{ij}+c_{ij}-\operatorname*{minc}_{k\in I'} c_{kj}\geqslant 0$ and (6) is again satisfied.
Hence $C(I'')>C(I')$ which proves the theorem.

By applying the theorem to the data of Table 2 it can be seen that the policies p_2 or p_3 (simple type) cannot coexist in the optimal solution of (LP) with p_7 (multiple type) and/or with p_{10} (combined multiple). A characteristic of these policies is that they contain either indexing path for the attribute DATE-1 (i.e. p_2, p_7, p_{10}) or a path for the attribute ADDRESS (i.e. p_3, p_7, p_{10}). This introduces the following rule:

Rule 3

A multiple or a combined multiple policy cannot coexist in the optimal strategy with none of the simple policies they contain. If it is assumed that all f_i's in Table 2 are equal to zero (i.e. there is no maintenance to IPS) then Rule 2 shows that p_9 and p_{10} have to be in the optimal solution. But from Rule 3 it follows that p_9 and p_{10} are the optimal strategy for the usage considered. If it is assumed now that all $q_j = 0$ (i.e. there is no querying to the IPS) then it is clear that no indexing path is profitable.

Rule 4

If only queries and no maintenance are performed then all the candidate indexing paths are included in the optimal strategy whereas if maintenance only is done, no indexing path appears in the IPS.

5. Conclusions

A file designer who cannot determine the effects of each alternative decision is bound to make subjective or intuitive design judgements instead of objective ones. The properties and rules stated (*a*) provide the means to improve the performance of IPS by expanding the current spectrum of alternative indexing paths examined prior to making any implementation decision, and (*b*) provide for increased confidence in the decision made. In Kollias (1979) (LP) is extended to cover the case where the user transactions follow periodic variations known in advance.

References

CARDENAS, A. F. (1973). Evaluation and selection of file organization—A model and system, *CACM*, Vol. 16 No. 9, pp. 540-548.

CARDENAS, A. F. (1975). Analysis and performance of inverted data base structures, *CACM*, Vol 18 No. 5, pp. 253-263.

CASEY, R. G. (1972). Allocation of copies of a file in an information network, *Proc. AFIPS*, SJCC, Vol. 40, pp. 617-625.

DODD, G. G. (1969). Elements of data management systems, *Computing Surveys*, Vol. 1, pp. 117-133.

EFFROYMSON, M. A. and RAY, T. L. (1966). A branch and bound algorithm for the plant location, *Oper. Research*, Vol. 13 No. 3, pp. 361-368.

GRAPA, E. and BELFORD, G. G. (1977). Some theorems to aid in solving the file allocation problem, *CACM*, Vol. 20 No. 11, pp. 878-882.

HSIAO, D. and HARARY, W. F. (1970). A formal system for information retrieval from files, *CACM*, Vol. 13 No. 2, pp. 67-73.

KNUTH, D. E. (1973). *The art of computer programming*, Vol. 3, Sorting and Searching, Addison-Wesley.

KOLLIAS, J. G. (1976). The selection of secondary file organizations, *Management Datamatics*, Vol. 5 No. 6, pp. 241-250.

KOLLIAS, J. G. (1979). File organizations and their reorganization, *Information Systems*, Vol. 4 No. 1, pp. 49-54.

LEFKOVITZ, D. (1969). *File structures for on-line systems*, Spartan Books, New York.

LUM, V. Y. (1970). Multiattribute retrieval with combined indexes, *CACM*, Vol. 13 No. 11, pp. 660-665.

LUM, V. Y. and LING, H. (1971). An optimization problem of the selection of secondary keys, *Proc. ACM Nat. Conf.*, Vol. 26, pp. 349-356.

SCKHOLNICK, M. (1975). The optimal selection of secondary indices for files, *Information Systems*, Vol. 1, pp. 141-146.

SENKO, M. E., LUM, V. Y. and OWENS, P. (1968). FOREM—A File Organization Evaluation Model, *Proc. IFIP*, North Holland, Amsterdam, pp. C19.C23.

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

SPIELBERG, K. (1969). Algorithms for the simple plant location problem with some side conditions, *Oper. Research*, Vol. 17, pp. 85-111.

STONEBRAKER, M. (1974). The choice of partial inversions and combined indices, *Int. J. Comp. Inform. Sci.*, Vol. 3 No. 2, pp. 167-188.

WANG, C. P. and LUM, V. Y. (1971). Quantitative evaluation and design trade-offs in file systems, *Proc. of the Symp. of Inf. Stor. and Retr.*, pp. 155-162.

Book review

A Programming Metholodology in Compiler Construction Part I: Concepts by J. Lewi, K. De Vlaminck, J. Huens and M. Huybrechts, 1979; 308 pages. (*North-Holland*, $41.50)

In the late 1950s the task of compiler construction was considered a major undertaking. The first FORTRAN compiler, for example, took 18 man-years to implement (Backus *et al*, 1957). Now, in the late 1970s, such a task is considered a reasonable computer science student project. The factors that have led to this over the last twenty years are (*a*) the comprehension of the organisation and modular design of the compilation process, (*b*) the development of systematic techniques for handling the majority of the important tasks that occur during compilation and (*c*) the construction of software tools that assist in the implementation of compilers and compiler components. Implicit in all these three developments is the closing of the gap between theory and practice. This book is the first part of a two-part description of an environment utilising a completely closed gap. Part I introduces the basic theoretical models whilst part 2 will consider the more practical aspects of the engineering of the environment (namely the language implementation laboratory [LILA]).

The book progresses in a formal manner from the theory for iterative language constructs (regular [translation] syntaxes) through nested language constructs ([extended] context free [translation] syntaxes) to attributed language constructs (attributed [translation] syntaxes). Within each language construct it develops, from an abstract theory model (acceptors [transducers]), an acceptor [transducer] program and then develops a generator program to produce systematically acceptor [transducer] programs from the associated syntax. As such, each section is the logical progression of the previous and the methodology used in each section is a reflection of the methodology of the previous section. Hence the book is structurally pleasing and easy to read.

In conclusion, the book is ideally suited to the software engineer who is actively involved in the application of language theory to compiler construction (or the construction of any systems software (Richards, 1979; Sassa, 1979)) and who seeks a well laid out methodology for doing so. It would also be useful to the theorist who is looking for a specific area of application. As an introduction to theory or compiler construction the reader would be better advised to do some introductory reading in other compiler construction literature (Gries, 1971; or Aho and Ullman, 1978) and the book gives a very satisfactory reference listing for this purpose.

S. K. ROBINSON (Uxbridge)

References

AHO, A. V. and ULLMAN, J. D. (1978). *Principles of Compiler Design*, Addison-Wesley.

GRIES, D. (1971). *Compiler Construction for Digital Computers*, Wiley, New York.

RICHARDS, M. (1979). A Compact Function for Regular Expression Pattern Matching, *Software—Practice and Experience*, Vol. 9, 527-534.

SASSA, M. (1979). A Pattern Matching Macro Processor, *Software-Practice and Experience*, Vol. 9, 439-456.