

There is a widespread feeling that the diversity of the students' background, especially their programming experience, causes problems. There is still no consensus about what to do about the problems but it does seem to be the case that the more radical the

approach to teaching programming the less the background of the students is an issue.

DAVID FURBER
Department of Computing,
King's College London,
Strand,
London WC2R 2LS

A Short Note on Doubly-Linked List Reorganizing Heuristics†

The class of memoryless heuristics for maintaining a Doubly-Linked List in an approximately optimal order is studied. Two mappings that relate Singly-Linked List and Doubly-Linked List heuristics are defined, and theorems involving these mappings are presented. A new heuristic referred to as the Swap heuristic for the Doubly-Linked List is introduced, and its asymptotic distribution has been derived.

Received January 1991

I. Introduction

In the development of self-organizing list structures, a problem that has been extensively studied involves the Singly-Linked List (SLL) or sequential list. McCabe⁹ was probably the first to consider the reorganization of data using self-organizing strategies, and in his pioneering work he introduced two schemes called the Move-To-Front and Transposition heuristics. These heuristic have been intensively analysed by many authors including Hendricks,⁵ Bitner,² Knuth⁷ and Rivest,¹² to name a few. Also, heuristics which use additional amounts of memory apart from the memory used by the list structure itself have been reported in the literature. We refer the reader to the following papers: McCabe,⁹ Kan *et al.*,⁶ Burville *et al.*,³ Oommen *et al.*^{10,11} The literature on adaptive SLL organization is extensive and a detailed survey on this topic is found in Ref. 4.

The Move-to-Front heuristic operates on the principle that when the accessed record is found it is moved to the front of the list. On the other hand, the Transposition heuristic exchanges the accessed record with the immediately preceding record; nothing is done if the accessed record is at the front of the list. Since these two schemes are fundamental to the study of self-organizing structures they will serve as benchmarks, in this paper.

Although the theory of self-organizing SLLs is well developed, very little work has been done to study the adaptive restructuring of a Doubly-Linked List (DLL). To our knowledge, the only other researchers who have worked in this area are Matthews *et al.*⁸

We shall first formalize the problem. Consider a set of N records $\{R_1, R_2, \dots, R_N\}$ which are held in a list in an arbitrary order π , so that R_i is in position $\pi(i)$ counting from the left to the right for $1 \leq i \leq N$. A search for R_i can begin at either end of the list, examining each position in turn until R_i is found. At each time instant, a key K_i is presented at one end of the list and the system is asked to retrieve the associated record R_i . Observe that if the key is

presented at the left, the number of comparisons made is $\pi(i)$, and if the key is presented at the right, the number of comparisons is $N - \pi(i) + 1$. The focus of this paper is to study heuristics which restructure the list so as to minimize the asymptotic expected number of comparisons.

We assume that the record R_i is requested at the left end and the right end of the list with a probability $s_{i,L}$ and $s_{i,R}$ respectively. Each access is stochastically independent and the access probabilities of the records are time invariant, with the constraint that,

$$\sum_{i=1}^N s_{i,L} + \sum_{i=1}^N s_{i,R} = 1 \quad (1)$$

The heuristics considered in this paper fall within the class of memoryless schemes. In other words, no extra memory variables are used beside the pointers used to maintain the list itself. Thus, any heuristic for the DLL can be defined as a set of permutations $\tau = \{\tau_{i,L}, \tau_{i,R}\}$, where the permutation $\tau_{i,L}$ is applied to the list when the element at the i th position is accessed from the left, and the permutation $\tau_{i,R}$ is applied to the list if this element is accessed from the right.

The application of DLLs are many and are presented by Tremblay and Sorenson.¹⁴ They are easy to implement when compared to more complex data structures, and are in general more flexible than SLLs. Of course, all applications which use a First-In-First-Out data structure can be implemented easily using a DLL. We hasten to add that the primary intention of this short paper is not to stress the applications of DLLs*, but to theoretically analyse a variety of scenarios involving self-organizing DLLs.

In this paper, we present two strategies by which an algorithm for maintaining a SLL can be extended to a DLL. These strategies involve, what we call, undirected and directed mappings. The characteristics of these two mappings are discussed in the body of the paper, and using them the results of Matthews *et al.*⁸ are generalized. Also, using a directed mapping a new heuristic called the Swap heuristic for the DLL is introduced and its asymptotic distribution is derived. This distribution is a generalization of that obtained for the Transposition heuristic for SLLs.

The access probabilities of the records can be represented in two distinct methods. The first representation is defined by (1). The representation used by Matthews *et al.*⁸ is the conditional probability representation in which $p_i = s_{i,L} + s_{i,R}$ is the probability that R_i is accessed and $p_{i,L} = 1 - p_{i,R} = s_{i,L} / (s_{i,L} + s_{i,R})$ is the probability that given R_i is accessed, it is accessed from the left. Both

* Apart from the computer science application of DLLs, one can easily conceive of library applications such as those discussed by Hendricks⁵ in which the search for a book can start at either end of the shelf.

Reference

1. R. Bornat, *Programming from First Principles*. Prentice-Hall, Englewood Cliffs, NJ, USA (1987).

these representations are equivalent and so we shall use them interchangeably.

The equivalence of these representations in terms of probability masses and the underlying operations was shown in Ref. 15. Also found in Ref. 15 are experimental results demonstrating the superiority of the Swap heuristic over the Move-to-End heuristic introduced in Ref. 8.

II. Doubly-Linked List Heuristics and their Analysis

Before discussing various heuristics, it is necessary to define a performance measure with which we can evaluate their efficiencies. Let $C_\tau(s)$ be the expected number of comparisons needed for an access when the underlying probability distribution is $s = (s_{1,L}, \dots, s_{N,L}, s_{1,R}, \dots, s_{N,R})$. We say that τ is more efficient than σ if $C_\tau(s) \leq C_\sigma(s)$ for all s . It is not necessarily the case that one heuristic is always more or less efficient than another, since in general, it can be the case that $C_\tau(s) \leq C_\sigma(s)$ and $C_\sigma(s') \leq C_\tau(s')$ for two distributions s and s' .

If the probability distribution s is known *a priori*, we can arrange the records in such a way that the cost is minimized. Observe that this is trivial because it is easily seen that the cost of a permutation π is

$$\begin{aligned} \text{cost}(\pi) &= \sum_{i=1}^N [s_{i,L} \cdot \pi(i) + s_{i,R} \cdot (N - \pi(i) + 1)] \\ &= \sum_{i=1}^N [(s_{i,L} - s_{i,R}) \cdot \pi(i) + s_{i,R} \cdot (N + 1)]. \end{aligned}$$

Since the second term of the summation is independent of $\pi(i)$, the optimal order is the permutation which arranges the list in the decreasing order of $(s_{i,L} - s_{i,R})$.

We now consider the transformation of SLL heuristics into DLL heuristics in terms of two types of mappings – undirected and directed mappings. In an undirected mapping, the operations done on a DLL depend only on the position of the accessed element, not on the direction from which the element is accessed. In contrast, in a directed mapping, the reorganization is based on the position of the accessed element and the direction of access. An example will clarify the point.

Consider the Move-To-Front heuristic for a SLL. This heuristic leads to the Move-To-Left rule if an undirected mapping is used to get a DLL scheme. This rule is quite simply described as follows: Whenever an element R_i is accessed, it is moved to the left end of the DLL independent of the end of access. Indeed, it is not obvious whether such a rule has any advantages at all. In contrast, a directed mapping transforms the SLL Move-To-Front heuristic into a DLL scheme called the Move-To-End heuristic (see Ref. 8), in which the accessed element is moved to the left end of the list if it is accessed from the left, and moved to the right end of the list if it is accessed from the right. We now prove the advantages of various undirected mappings.

† Partially supported by the Natural Science and Engineering Research Council of Canada.

II.1 Undirected Mappings of Singly-Linked List Heuristics

Let $\tau^u = \{\tau_i\}$ be a SLL heuristic. This means that if the record in position i is accessed, the data is recognized according to the rule τ_i . Similarly, let $\tau^p = \{\tau_{i,L}, \tau_{i,R}\}$ be a DLL heuristic. An undirected mapping exists between τ^u and τ^p if $\tau_{i,L} = \tau_i = \tau_{i,R}$.

Should such a mapping exist, $E_{\tau^p}[\pi(i)]$, the expected location of record R_i (counting from the left), no longer depends on the conditional probabilities $p_{i,L}$ and $p_{i,R}$, since the permutation performed will be the same regardless of the search direction. In other words, for a given environment s , $E_{\tau^p}[\pi(i)] \equiv E_{\tau^u}[\pi(i)]$.

Matthews *et al.*⁸ proved that if an optimal heuristic exists for the SLL problem, then the undirected mapping of this heuristic is also optimal for the DLL when $p_{i,L} = p > \frac{1}{2}$, $1 \leq i \leq N$. Although a counterexample due to Anderson *et al.*¹ shows that an optimal heuristic does not exist, Matthews *et al.*'s argument is still valid when comparing two SLL heuristics one of which is more efficient than the other. This is the first primary result of this paper.

Theorem I

Suppose τ^u and ρ^u are SLL heuristics for which τ^u is more efficient than ρ^u . Then if $p_{i,L} = p > 0.5$, $1 \leq i \leq N$, the corresponding pair of DLL heuristics, under the undirected mapping, are such that τ^p is more efficient than ρ^p .

Proof: For any doubly linked list heuristic σ with $p_{i,L} = p$ for $i = 1 \dots N$, the expected search time for any query is,

$$\begin{aligned} C_{\sigma^p}(s) &= \sum_{i=1}^N p_i p E_{\sigma^p}[\pi(i)] \\ &\quad + \sum_{i=1}^N p_i (1-p) (N - E_{\sigma^p}[\pi(i)] + 1) \\ &= (2p-1) \sum_{i=1}^N p_i E_{\sigma^p}[\pi(i)] + K, \\ &\quad \text{where } K = (N+1)(1-p). \end{aligned} \quad (2)$$

From the search cost of SLLs and given that τ^u is more efficient than ρ^u , we know:

$$\begin{aligned} C_{\tau^u}(s) &= 1 + \sum_{i=1}^N p_i E_{\tau^u}[\pi(i)] \\ &< 1 + \sum_{i=1}^N p_i E_{\rho^u}[\pi(i)] = C_{\rho^u}(s). \end{aligned} \quad (3)$$

This implies that,

$$\sum_{i=1}^N p_i \cdot E_{\tau^u}[\pi(i)] < \sum_{i=1}^N p_i \cdot E_{\rho^u}[\pi(i)]. \quad (4)$$

Combining (2)-(4) and using the facts that $E_{\tau^p}[\pi(i)] \equiv E_{\tau^u}[\pi(i)]$, we have,

$$\begin{aligned} C_{\tau^p}(s) &= (2p-1) \sum_{i=1}^N p_i E_{\tau^p}[\pi(i)] + K \\ &< (2p-1) \sum_{i=1}^N p_i E_{\rho^p}[\pi(i)] + K \\ &= C_{\rho^p}(s). \end{aligned}$$

The above inequality holds since $(2p-1) > 0$ for all $p > 0.5$. Hence the theorem.

A word about the consequences of Theorem I is not out of place. Matthews *et al.*⁸ proved that if an optimal SLL heuristic existed, this could be transformed to yield an optimal doubly linked heuristic for the case when $p_{i,L} = p > 0.5$ for all i . By this theorem, we have now been able to present a hierarchy on the set of DLL heuristics which are obtainable using undirected mappings. The theorem ensures us that for any environment, if τ^u is superior to ρ^u as a SLL heuristic, then the corresponding heuristics (τ^p and ρ^p respectively) obtained using an undirected mapping will satisfy the property that τ^p is superior to ρ^p whenever $p_{i,L} = p > 0.5$. Thus, although the result of Matthews *et al.* is only true for the special case if an optimal algorithm existed, it is interesting to note the result presented above is an abstract generalization, valid for all pairs of heuristics.

Observe too another implication of the above results. The result implies that if the user's access distribution satisfies the constraints of the theorem, then, the direction of access of the queries can be completely ignored. Indeed, by using only the information contained in the sequence of accessed records and ignoring the information about the direction from which the records have been accessed, a DLL *undirected* heuristic can be used to reorganize the data, and the optimality of this heuristic is completely dependent on the optimality of the corresponding SLL from which it has been derived.

Rivest¹² proved that the Transposition heuristic is more efficient than the MTF heuristic. Using this fact in conjunction with Theorem I, we have the following corollary:

Corollary I.1

For all DLLs, the Transposition heuristic under the undirected mapping is more efficient than the MTF heuristic under the undirected mapping whenever $p_{i,L} = p > \frac{1}{2}$.

II.2 Directed Mappings of Singly-Linked List Heuristics

Using the results of the previous sub-section we know that an undirected mapping is both powerful and useful whenever the user's query distribution satisfies the constraints of Theorem I. In all other cases, unfortunately, an undirected mapping is not a very realistic way to transform a SLL heuristic into a DLL heuristic. This is because the permutation is not sensitive to the direction of access, which contains crucial information and which ought not to be omitted from the reorganization process. Directed mappings utilize this information. A SLL heuristic $\tau^u = \{\tau_i\}$ is defined to possess a directed mapping on a DLL heuristic $\tau^p = \{\tau_{i,L}, \tau_{i,R}\}$, if

$$\tau_{i,L} = \tau_i = \tau'_{N-i+1,R} \quad \text{for } i = 1, \dots, N,$$

where τ' is the permutation identical to τ except that it is described from the right end of the list.

Suppose the MTE heuristic, described in the preamble of this section, is used and let $b(i, j)$ denote the asymptotic probability that R_i is to the left of R_j . Matthews *et al.* showed that:

$$b(i, j) = (s_{i,L} + s_{j,R}) / (s_{i,L} + s_{i,R} + s_{j,L} + s_{j,R}).$$

Also, the search cost of the MTE heuristic was

shown to be:

$$\begin{aligned} C_{\text{MTE}}(s) &= 1 + 2 \sum_{j=1}^N \\ &\quad \left\{ s_{j,L} \sum_{i < j} \frac{s_{j,R} + s_{j,L}}{s_{i,L} + s_{i,R} + s_{j,L} + s_{j,R}} \right. \\ &\quad \left. + s_{j,R} \sum_{i > j} \frac{s_{i,R} + s_{j,L}}{s_{i,L} + s_{i,R} + s_{j,L} + s_{j,R}} \right\} \end{aligned}$$

They also proved that for all distributions the MTE heuristic has an expected search time that is no more than twice that of the optimal policy. These results generalize the SLL results.

Another well studied heuristic for SLLs is the Transposition rule. We now extend this rule using a directed mapping to obtain a new heuristic for the DLLs. This heuristic will be called the Swap heuristic. With this scheme, whenever a record R_i is found in position $\pi(j)$, if the search originates from the left of the list, the list is arranged by swapping the positions of records R_i and R_{j+1} . This, of course, is except when j equals unity and R_i is at the left end of the list, for in this case no records are moved. Otherwise, if the search originates from the right of the list, the list is rearranged by swapping R_i and R_{j+1} , except when j equals N and R_i is at the right end of the list. Again, in this case, no records are moved.

The analysis of the Swap heuristic presented below is based on the properties of finite, time reversible Markov chains. In this case each state is one of the $N!$ possible orderings of the set of records $\{R_1, R_2, \dots, R_N\}$, and each ordering $(R_{i_1}, R_{i_2}, \dots, R_{i_N})$ has a unique stationary probability $P^*(R_{i_1}, R_{i_2}, \dots, R_{i_N})$ which is obtained as stated below.

Theorem II

Under the Swap heuristic, the stationary probabilities obey:

$$\frac{P^*(R_{i_1}, R_{i_2}, \dots, R_{i_j}, R_{i_{j+1}}, \dots, R_{i_N})}{P^*(R_{i_1}, R_{i_2}, \dots, R_{i_{j+1}}, R_{i_j}, \dots, R_{i_N})} = \frac{s_{i_j,L} + s_{i_{j+1},R}}{s_{i_{j+1},L} + s_{i_j,R}}$$

Proof: We are required to derive the stationary distribution for the Swap heuristic. We begin by showing that the Markov chain of the Swap heuristic is time reversible by proving that from any state $(R_{i_1}, R_{i_2}, \dots, R_{i_N})$ any path that returns to this state has the same probability as the reverse path (see Ref. 13). Thus, for example, when $N = 3$, a sequence of transitions that returns from the state (R_1, R_2, R_3) to itself are:

$$\begin{aligned} (R_1, R_2, R_3) &\rightarrow (R_2, R_1, R_3) \\ &\rightarrow (R_2, R_3, R_1) \rightarrow (R_3, R_2, R_1) \\ &\rightarrow R_3, R_1, R_2 \rightarrow (R_1, R_3, R_2) \\ &\rightarrow (R_1, R_2, R_3) \end{aligned}$$

The product of the transition probabilities in the forward direction is:

$$\frac{(s_{2,L} + s_{1,R})(s_{3,L} + s_{1,R})(s_{3,L} + s_{2,R})}{(s_{1,L} + s_{2,R})(s_{1,L} + s_{3,R})(s_{2,L} + s_{3,R})}.$$

This is the same as the product of the transition probabilities in the reverse direction:

$$\frac{(s_{3,L} + s_{2,R})(s_{3,L} + s_{1,R})(s_{2,L} + s_{1,R})}{(s_{2,L} + s_{3,R})(s_{1,L} + s_{3,R})(s_{1,L} + s_{2,R})}.$$

The general result proving the reversibility of the Markov chain follows in an analogous manner, and is omitted for the sake of brevity.

By using the properties of Markov chains¹³ we know that for any time reversible chain, if

$\phi(t)$ is the state occupied at time ' t ', and ϕ_m and ϕ_n are any two states with transition probabilities

$$\text{Prob}[\phi(t+1) = \phi_n | \phi(t) = \phi_m] = q_{m,n},$$

then, $P^*(\phi_m)$ and $P^*(\phi_n)$, the equilibrium probabilities of being in ϕ_m and ϕ_n respectively, obey:

$$P^*(\phi_m)/P^*(\phi_n) = q_{n,m}/q_{m,n} \quad \text{for all } m, n. \quad (5)$$

Now the transition from $(R_{i_1}, R_{i_2}, \dots, R_{i_j}, R_{i_{j+1}}, \dots, R_{i_N})$ to $(R_{i_1}, R_{i_2}, \dots, R_{i_{j+1}}, R_{i_j}, \dots, R_{i_N})$ occurs whenever $R_{i_{j+1}}$ is accessed from the left or R_{i_j} is accessed from the right. Similarly, the transition from $(R_{i_1}, R_{i_2}, \dots, R_{i_{j+1}}, R_{i_j}, \dots, R_{i_N})$ to $(R_{i_1}, R_{i_2}, \dots, R_{i_j}, R_{i_{j+1}}, \dots, R_{i_N})$ occurs whenever $R_{i_{j+1}}$ is accessed from the right or R_{i_j} is accessed from the left. Observing that these events are mutually exclusive and substituting them in (5), we have:

$$\frac{P^*(R_{i_1}, R_{i_2}, \dots, R_{i_j}, R_{i_{j+1}}, \dots, R_{i_N})}{P^*(R_{i_1}, R_{i_2}, \dots, R_{i_{j+1}}, R_{i_j}, \dots, R_{i_N})} = \frac{s_{i_j, L} + s_{i_{j+1}, R}}{s_{i_{j+1}, L} + s_{i_j, R}} \quad \text{for all } j. \quad (6)$$

Hence the theorem.

A similar result exists for the Transposition rule for the SLL.¹² Theorem II is a generalization of the latter result. Indeed, in the special case when the probability of access from the right is zero (that is $s_{i,R} = 0$ for all i), the Swap heuristic behaves identically to the Transposition heuristic and (6) becomes identical to the expression in Ref. 12:

$$\frac{P^*(R_{i_1} R_{i_2} \dots R_{i_j} R_{i_{j+1}} \dots R_{i_N})}{P^*(R_{i_1} R_{i_2} \dots R_{i_{j+1}} R_{i_j} \dots R_{i_N})} = \frac{s_{i_j, L}}{s_{i_{j+1}, L}}$$

Rivest¹² also proved that the Transposition heuristic is always more efficient than the Move-To-Front heuristic. We can now state an analogous claim concerning the Swap heuristic.

Conjecture I

The Swap heuristic is always more efficient than the Move-To-End heuristic.

Experimental results clearly support the conjecture that this result is true for all distributions.¹⁵ However, the proof of the claim is certainly non-trivial. It will involve obtaining expressions for the ratio of the asymptotic probabilities of the chain being in two states, say, ϕ_m and ϕ_n in which the list representation of ϕ_m can be derived by interchanging exactly two of the elements in ϕ_n and vice versa. Examples of two such states

are the orderings

$$\phi_m = (R_{i_1}, R_{i_2}, \dots, R_{i_j}, R_{i_{j+1}}, \dots, R_{i_N}),$$

and,

$$\phi_n = (R_{i_1}, R_{i_2}, \dots, R_{i_{j+1}}, R_{i_j}, \dots, R_{i_N}).$$

In the above, the representation of the states ϕ_m and ϕ_n are interchangeable by merely interchanging the position of the elements R_{i_j} and $R_{i_{j+1}}$. Computing such expressions for the transposition heuristic is straightforward because of the cancellations of the asymptotic probabilities of being in the intermediate states. But generalizing the latter for the case of the Swap heuristic is non-trivial because such cancellations do not occur. Furthermore, it is not certain whether such a generalization may even exist. If the latter is the case and such a generalization does not exist, the result will have to be proved using mathematical tools which have not been used in the literature to analyze adaptive data structures. These will involve not merely computing the asymptotic probabilities of the associated states but also deriving expressions for the asymptotic cost evaluated in terms of the asymptotic state occupancy probabilities.

To conclude this section, we also conjecture that if two SLL heuristics τ and ρ satisfy the property that τ is more efficient than ρ , then the corresponding pair of DLL heuristics, τ' and ρ' , obtained using the directed mapping, are such that τ' is more efficient than ρ' . This conjecture is a generalization of both Theorem I and the above conjecture for the set of directed mappings. The result seems intuitive since a more efficient SLL scheme tends to push the frequently accessed elements to the end, and thus plausibly do a better job to polarize the elements in the DLL.

III. Conclusion

In this paper, we have studied the problem of adaptively reorganizing a Doubly-Linked List (DLL) and have presented an interesting relationship between arbitrary pairs of heuristics for this problem and those used to reorganize a Singly-Linked List (SLL). This is achieved by introducing the concept of undirected and directed mappings between SLLs and DLLs. In particular, we have introduced a new heuristic called the Swap heuristic, and derived the asymptotic probabilities for the chain converging to the various possible orderings. Two conjectures have been proposed which suggest possible directions for future research in the area of adaptive DLLs.

Acknowledgements

The authors are grateful to various readers for their comments and in particular to Mr Valiveti for helpful discussions.

D. T. H. NG AND B. J. OOMMEN
School of Computer Science,
Carleton University,
Ottawa, Ontario K1S 5B6, Canada

References

1. E. J. Anderson, P. Nash and R. R. Weber, A counter-example to a conjecture on optimal list ordering, *J. Appl. Prob.* **19** (3) 730-732 (1985).
2. J. R. Bitner, Heuristics that dynamically organize data structures, *SIAM J. Comput.* **8** (1), 82-110 (1979).
3. P. J. Burville and J. F. C. Kingman, On a model for storage and search, *J. Appl. Probability*, **10**, 697-701 (1973).
4. J. H. Hester and D. S. Hirschberg, Self-organizing linear search, *Comp. Surveys*, **17** (3), 295-311 (1985).
5. W. J. Hendricks, The stationary distribution of an interesting Markov chain, *J. Appl. Prob.* **9** (1) 231-233 (1972).
6. Y. C. Kan and S. M. Ross, Optimal list order under partial memory constraints, *J. Appl. Prob.* **17** (4) 1004-1015 (1980).
7. D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, pp. 398-399. Addison-Wesley, Reading, Mass. (1973).
8. D. Matthews, D. Rotem and E. Bretholz, Self-organizing doubly-linked lists, *J. Comp. Maths.* **A**, **8** 99-106 (1980).
9. J. McCabe, On serial files with relocatable records, *Oper. Res.* **609-618** (1965).
10. B. J. Oommen and E. R. Hansen, List organizing strategies using stochastic move-to-front and stochastic move-to-rear operations, *SIAM J. of Comput.*, **16** (4) 705-716 (1987).
11. B. J. Oommen, E. R. Hansen and J. I. Munro, Deterministic optimal and expedient move-to-rear list organizing strategies, *Theoretical Computer Science*, **74**, pp. 183-197, (1990).
12. R. Rivest, On self-organizing sequential search heuristics, *C-ACM* **19** (2) 63-67 (1976).
13. S. M. Ross, *Introduction to Probability Models*. Academic Press, New York (1980).
14. J. P. Tremblay and P. G. Sorenson, *An Introduction to Data Structures with Applications*, McGraw-Hill, New York, 1976.
15. D. T. H. Ng and B. J. Oommen, Generalizing singly-linked list reorganizing heuristics for doubly-linked lists, *Proceedings of the 1989 Conference on the Mathematical Foundations of Computer Science, Rytro, Poland*, 380-389 (1989).

Indexing for multi-attribute retrieval

1. Introduction

In his book,⁴ Professor Wiederhold discusses the concept of multi-attribute indexing: this involves maintaining an index on the concatenation of several attributes, to provide a faster response to a query specifying values for several different attributes. Since in a given

query, an attribute may or may not be specified, it is necessary to maintain some index to serve every combination. (Here, an index is said to serve a subset of attributes, if it could be used to answer a query involving precisely that subset of attributes.)

An interesting feature of multi-attribute indexing is that one index can be used to answer queries for several different choices of subsets of specified attributes. For example,

the index on attributes (1,2,3,4) concatenated in that order would serve the subsets {1}, {1,2}, {1,2,3}, and {1,2,3,4}. In fact, a multi-attribute index on k attributes could serve k different subsets: one of cardinality i for every i with $(1 \leq i \leq k)$. The actual subsets served are determined by the order of concatenation of the attributes. Wiederhold noted that by a clever choice of order, the number of indexes required for 4 attributes could be reduced from 15 to 6.