Probability in the Engineering and Informational Sciences, 4, 1990, 19–27. Printed in the U.S.A.

THE MOVE-TO-FRONT RULE FOR MULTIPLE LISTS

COSTAS COURCOUBETIS
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, New Jersey 07974

RICHARD R. WEBER
Cambridge University Engineering Department
Management Studies Group
Mill Lane, Cambridge, CB2 1RX, United Kingdom

A number of data items $\{1,2,\ldots,n\}$ are to be maintained in a structure which consists of several linear lists. Successive requests to access items are independent random variables, and the probability that a particular request is for item i is p_i. The cost of accessing the jth item from the front of a list is j. For a single list, the move-to-front rule (MF) has been extensively studied and has been shown to provide good performance. In some actual circumstances, MF is the only physically realizable or convenient policy. We extend the study of move-to-front by examining the case where items are kept in several lists. Following its access, an item must be replaced at the front of one of the lists. In certain cases, assuming the p_i's are known, the policy which minimizes the average retrieval cost takes a particularly simple form: no item is ever moved from the list in which it is placed initially.

1. THE MOVE-TO-FRONT RULE FOR MAINTAINING A LIST

The so-called *library problem* has been studied by a number of authors. In this problem, n books are kept on a library shelf and are requested such that there is a probability p_i that any particular request is for book i, independent of other requests. The cost of retrieving a book is its position counting from the

 0269-9648/90 $5.00 + .00

left-most book on the shelf, and it is desired to minimize the long-run average cost of retrievals. When the p_i's are known, say $p_1 \geq \cdots \geq p_n$, the optimal policy is to shelve the books in order of *decreasing request probability* (DP), with book 1 left-most, followed by book 2, and so on. For the case in which the p_i's are unknown, a number of simple rules have been studied: (a) *Move-to-front* (MF), in which the accessed book is returned to the left of the shelf: (b) *transpose* (T), in which the accessed book changes places with the book previously on its immediate left; and (c) *frequency count* (FC), in which a frequency count is kept of the number of times each book has been requested and the books are maintained on the shelf in order of nonincreasing frequency count. Move-to-front and transpose are *no-memory rules*, in that the position in which book i is replaced depends only on the positions of the books at the time i was retrieved. Sleator and Tarjan [4] summarize a number of results in the study of the library problem and we briefly repeat them here. Denote by $E_A(p)$ the average cost associated with rule A, when $p = (p_1, \ldots, p_n)$. Then,

$$E_{DP}(p) = E_{FC}(p) \leq E_T(p) \leq E_{MF}(p) \leq 2\, E_{DP}(p),$$

where

$$E_{DP}(p) = \sum_{i=1}^{n} i p_i, \qquad \text{and} \qquad E_{MF}(p) = 1 + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} p_i p_j / (p_i + p_j).$$

The formula for $E_{MF}(p)$ can be understood as follows: the probability that i is the next item requested is p_i. When item i is requested, the probability that item i resides further from the front of the list than item j is $p_j/(p_i + p_j)$, which is simply the probability that j rather than i was requested on the last occasion that one of i or j was requested. Thus, $\left[1 + \sum_{j \neq i} p_j/(p_i + p_j)\right]$ is the expected position of item i from the front of the list.

Rivest [3] proved $E_T(p) \leq E_{MF}(p)$ and also conjectured that the transpose rule is optimal for all p in the class of no-memory rules. This was disproved by a counterexample of Anderson et al. [1], which effectively ruled out the possibility of a single simple rule being optimal for all p.

In this paper, we study a library-type problem which is extended to more than one shelf. In doing this, we are attempting to learn about reorganizing systems in which the data structure is more complex than a single linear list. One example of multiple linear lists occurs in a data base using hashing. Each hashing location is the start of a single linear list. In this paper, we focus attention on the move-to-front rule. Not only is this the simplest starting point, but there are theoretical and practical reasons why move-to-front is an important rule. Bentley and McGeoch [2] have reported results of empirical tests which indicate that on actual sequences, move-to-front is better than transpose and competitive with frequency count. This is not surprising if successive requests in real sequences are not independent. Results of Sleator and Tarjan show that for any

initial ordering of the items and any sequence s of m successive retrievals, $C_{MF}(s) \leq 2C_A(s) - m$, where $C_A(s)$ is the cost associated with using a rule A that replaces each item no further from the front than the position from which it was extracted, and otherwise does no rearrangement of items.

In some circumstances, move-to-front is the only practical rule. For example, the items might be crates stacked in piles in a warehouse. Or they might be cars, whose owners share for parking the use of several narrow driveways which run between their houses.

2. THE MOVE-TO-FRONT RULE AND MULTIPLE LISTS

We consider a model in which n items with known request probabilities $p_1 \geq \cdots \geq p_n$ are to be maintained in k lists with a move-to-front discipline. The cost of accessing an item which is j from the front of its list is j. Following each access, the item which is accessed must be replaced at the front of one of the k lists. This is the only operation allowed when an item is replaced; no other reordering is permitted. In the case $k = 2$, the number of stationary policies which are possible for the resulting Markov decision problem is $2^{(n+1)!n/2}$, which is 2^{15120} for just $n = 6$. This assumes that the choice of the list to which an item is replaced is a function of the items in each list and their orders in each list. However, as we shall show, the order of the items in the list is unimportant, so for $k = 2$ the true number of unique stationary policies is $2^{n2^{n-1}}$, which is 2^{192} for $n = 6$. The result that the order of the items in the lists is immaterial is contained in the following lemma.

LEMMA: *In specifying the list in which an item is to be replaced, the optimal policy need take no regard of the order in which the items appear in the lists.*

PROOF: The idea is very simple. A cost i is to be paid whenever an item is accessed at depth i in a list. When item i is replaced at the front of a list L, it increases by 1 the cost associated with the next retrieval of any item j in L, if the next retrieval of j occurs before the next retrieval of i. We can define a cost structure which, in the time-average sense, gives the same cost, for any stationary policy, as our original cost structure, but which clearly does not depend on the order in which the items are placed in the list. The new cost structure is one in which a cost of $[1 + p_j/(p_i + p_j)]$ is paid whenever i is replaced at the front of a list which includes j. Here $p_j/(p_i + p_j)$ is just the probability that the next request for j occurs before the next request for i. With the new cost structure, it is clear that information about the ordering of items in the lists is irrelevant. ■

The reader may also appreciate the following proof of this Lemma. For a given stationary strategy, let ϕ_{ij} denote the proportion of time for which items i and j are in the same list and item i is closer to the front. Let θ_{ij} be the proportion of all requests such that item i is requested and replaced at the front of a list which includes j. By considering the state of the lists before and after a re-

quest, one obtains the obvious balance equation $\phi_{ij} = (1 - p_i - p_j)\phi_{ij} + \theta_{ij}$. This follows from the observation that if i and j are in the same list, and i closer to the front than j, then either this circumstance held prior to the last request and neither i or j was requested, or the last request was for item i and it was replaced at the front of a list which included j. The time-average cost of retrieval is clearly $[1 + \Sigma_{ij}\, \phi_{ij} p_j]$. The time-average cost under a structure in which a cost $[1 + p_j/(p_i + p_j)]$ is paid whenever item i is replaced at the front of a list which includes j is $\Sigma_{ij}\, \theta_{ij}\{1 + p_j/(p_i + p_j)\}$. Using the relation $\theta_{ij} = (p_i + p_j)\phi_{ij}$ from the above balance equation, it is clear that the time-average costs are equal.

Lemma 1 provides a substantial simplification, but it is still very difficult to determine the optimal stationary policy for a problem of any size. Suppose we restrict our attention to a subclass of possible policies. We shall say that π is a *partition policy*, if it divides the items into k lists and then always replaces each item to the same list from which it was accessed. The class of partition policies will be denoted by Π. If the partition L is into sets $L_1, L_2, \ldots, L_k$, the time-average cost will be

$$c(L) = 1 + \sum_{\substack{i \neq j \\ i,j \in L_1}} p_i p_j/(p_i + p_j) + \cdots + \sum_{\substack{i \neq j \\ i,j \in L_k}} p_i p_j/(p_i + p_j).$$

The problem of minimizing $c(L)$ is an NP-hard combinatorial optimization problem, since for $k = 2$, $n = 2m$, $p_i = n^{-1} + \epsilon_i$, and small ϵ_i, with $\Sigma\epsilon_i = 0$, it is equivalent to the partition problem of finding a subset of exactly m ϵ_i's whose sum is as close as possible to 0. However, the number of possible policies which must be considered in an exhaustive search is much reduced. For $k = 2$, it is 2^{n-1} (which in the case $n = 6$ reduces the number of admissible policies from 2^{192} to 32).

In fact, numerical experimentation failed to turn up examples in which the optimal policy in the entire class of policies was not a partition policy. Only after looking for conditions guaranteeing that the optimal policy lies in the class Π did we gain the understanding to construct an example in which the optimal policy does not lie in Π. The following is our example.

Example: Suppose $k = 2$, $n = 4m$, $p_1 = \cdots = p_{2m} = a$, and $p_{2m+1} = \cdots = p_{4m} = b$, where $2m(a + b) = 1$. Essentially, there are two distinct types of item: type A for which $p_i = a$, and type B for which $p_i = b$. It is not difficult to show that an optimal policy in Π is one π, in which L_1 and L_2 contain exactly m items of each type. There is clearly a reduced state space in which items with the same p_i are undistinguished. In this state-space discription, we let x be that state in which there are m items of each type in each list. We let x^p denote the state obtained from x by moving one item, of the type having probability $p_i = p$, from list L_1 to L_2. We consider a policy σ which induces a stationary distri-

bution on the four states x, x^a, x^{ab}, and x^b. The policy σ is most easily described by its 4×4 transition matrix, in which $P(x,x^a) = ma$, $P(x^a,x^{ab}) = mb$, $P(x^{ab},x^b) = (m+1)a$, and $P(x^b,x) = (m+1)b$ are the only nonzero off-diagonal elements. Lengthy calculations show that for certain choices of a and b, and $m \geq 8$, the policy σ has a smaller cost than π. The most unfavorable comparison of π to σ occurs when $m = 8$, and a and b are approximately $\frac{1}{96}$ and $\frac{5}{96}$. For $a = \frac{1}{96}$ and $b = \frac{5}{96}$, the cost of σ is 6.707 per access and the cost of π is 6.722 per access (so σ costs only 0.9977 as much as π).

There are certain situations in which the optimal policy *must* lie in Π. Given a particular policy σ, we say an item i is *stable* in state x with respect to σ if when i is the next item requested it is replaced in the same list from which it was accessed. $S(x)$ denotes the set of stable items. *Unstable* items are those which are not stable and the set of these items is denoted by $U(x)$. Let $p_{S(x)} = \sum_{i \in S(x)} p_i$ be the probability a stable item is chosen in state x.

THEOREM 1: *Suppose we restrict attention to the class of policies such that for every recurrent state x,*

$$p_{S(x)} + (n - 2 - |S(x)|) \max_{i \in S(x)} \{p_i\} \leq 1. \tag{1}$$

Then the optimal policy lies in the class of partition policies Π.

Corollaries of this theorem are that the optimal policy lies in Π if either (a) $1/(n-2) \geq \max_{i \in S(x)} \{p_i\}$, that is, if the p_i are nearly equal, or (b) the number of unstable objects in any recurrent state never exceeds two, so that $|S(x)| \geq (n-2)$. If $n = 3$, then the optimal policy is always in Π. This follows from (b) since there must be a route into every recurrent state, so there are at most two unstable items in every recurrent state of a stationary policy. In fact, the partition $L_1 = (1)$ and $L_2 = (2,3)$ is optimal for $k = 2$. Similarly, the above, and other calculations, show that for $n = 4$, $L_1 = (1,4)$, $L_2 = (2,3)$ is optimal for $k = 2$, and $L_1 = (1)$, $L_2 = (2)$, and $L_3 = (3,4)$ is optimal for $k = 3$, for all values of $\frac{1}{2} \geq p_1 \geq \cdots \geq p_4$.

PROOF OF THEOREM 1: Suppose policy σ induces a recurrent Markov chain with stationary probabilities $\pi(x)$. Let

$$f(x) = 1 + \sum_{h=1}^{k} \sum_{i \in L_h} \sum_{\substack{j \in L_h \\ j \neq i}} p_i p_j / (p_i + p_j).$$

Here $f(x)$ is the expected cost associated with the next retrieval if whatever item is retrieved it is replaced in the same list from which it was retrieved. Let $\bar{x}$ be the state x for which $f(x)$ is minimal. That is, $f(\bar{x})$ is the average cost per retrieval of the optimal policy in Π. Denote by x^i the state which is obtained

from a state x if item $i \in U(x)$ is requested and replaced at the front of the list specified by σ. The average cost per retrieval can be written as

$$S = \sum_x \pi(x) f(x) + \sum_x \pi(x)(1/2) \sum_{i \in U(x)} \{f(x^i) - f(x)\}.$$

This follows by viewing costs as replacement costs (as in the Lemma at the start of this section) and noting that if in state x an item $i \in U(x)$ is requested then the cost of replacing that item differs from the cost that would be paid to replace it in the list from which it was requested by $(1/2)\{f(x^i) - f(x)\}/p_i$. Rewriting this,

$$S = \sum_x \pi(x) f(x) + (1/2) \sum_x \pi(x) f(x) \left\{ \sum_{i \in S(x)} \pi(x^i)/\pi(x) - |U(x)| \right\}.$$

Formally substituting $f(x) = f(\bar{x})$ in the above gives $S = f(\bar{x})$; hence,

$$S - f(\bar{x}) = \sum_x \pi(x) \{f(x) - f(\bar{x})\} + (1/2) \sum_x \pi(x) \{f(x) - f(\bar{x})\} \left\{ \sum_{i \in S(x)} \pi(x^i)/\pi(x) - |U(x)| \right\}.$$

From the above, we see that $S - f(\bar{x}) \geq 0$, implying that σ has an average cost of at least $f(\bar{x})$ is for all x,

$$\sum_{i \in S(x)} \pi(x^i)/\pi(x) - |U(x)| \geq -2. \tag{2}$$

Now the balance equation for state x is

$$\pi(x) = \sum_{i \in S(x)} \pi(x) p_i + \sum_{i \in S(x)} \pi(x^i) p_i,$$

or

$$\pi(x)(1 - p_{S(x)}) = \sum_{i \in S(x)} \pi(x^i) p_i \leq \max_{i \in S(x)} \{p_i\} \sum_{i \in S(x)} \pi(x^i). \tag{3}$$

The theorem follows by observing that Eqs. (1) and (3) imply Eq. (2). ■

Our experience with numerical examples indicates that it is difficult to find circumstances where a partition policy is not optimal, and if the optimal policy is not a partition policy it reduces costs only by a small amount. Our example for which a partition policy is not optimal required $n = 32$, and we would be surprised by an example in which the optimal policy is not in Π and n much less than this. It would be interesting to establish bounds on the suboptimality of partition policies, but we have had no real success in that direction. However, in the following section we discuss variations of the problem in which the optimal policy does indeed lie within the class of partition policies.

3. THE OPTIMALITY OF MOVE-TO-FRONT OF THE SAME LIST

In this section, we consider variations of the problem in which the cost structure is rather different and we can prove more. Consider the following two problems.

1. The spaces left by items when they are removed from the lists are not compacted. That is, although an item is moved to the front of some list after being requested, the place it vacates remains. (Think, for example, of a write-once, read-many data storage device, such as an optical disc.) The cost of retrieving an item is equal to the number of items and *vacated spaces* between it and the front of the list at its time of retrieval.
2. The items are boxes containing different numbers of some object. Box i contains n_i objects and has a weight proportional to n_i. The probability that box i is requested is proportional to its weight (think, for example, of stacks of heavy reference books containing differing numbers of pages). The cost of retrieving a box is the weight of boxes which must be temporarily lifted from above it.

Surprisingly perhaps, in both these cases, the optimal policy for operating a move-to-front-of-some-list discipline lies in the class of partition policies.

More generally, consider the following model. Items reside in k sets (the list structure is no longer the dominant feature). Whenever an item is requested it is removed from its set, at some cost. It is then replaced in another set (possibly the set it came from), at an additional cost. Suppose that an additive cost a_{ij} is incurred whenever item i is removed from a set which includes item j. Similarly, a cost b_{ij} is incurred whenever item i is replaced into a set which includes item j. Then we have the following theorem.

THEOREM 2: *For any k, p_i, a_{ij}, and b_{ij}, with $b_{ij} = b_{ji}$, the optimal dynamic policy for replacing items following their retrieval is one in which items are initially partitioned into k sets, and then items are always replaced into the same set from which they are retrieved.*

PROOF: Consider first the case in which there are no costs associated with replacing items into sets; that is, $b_{ij} = 0$ for all i,j. Costs are paid only when removing items. In this case, it is clear that the best scheme is to partition the items into k sets such that the expected cost of removing the next requested item is minimized. Then by always replacing each object in the set from which it was retrieved, we can enjoy the same minimal expected-retrieval cost each time an item is requested.

We now show that when replacement costs are not zero, the optimal policy can be determined by considering a related model in which replacement costs are zero. Recall that when i is replaced in a set which includes j, the replacement

cost b_{ij} is charged. Suppose we delay payment of this charge until the next time that either i or j is requested. To do this we must remember, at the time when either i or j is next requested, that there is still a charge of b_{ij} to pay. But under this scheme there will always be a charge of either b_{ij} or b_{ji} to pay when i or j is requested and they reside in the same list. This is because if i and j are in the same list then either i was last replaced in the list when it included j, or j was last replaced in the list when it included i. This is the key idea in the proof. Because $b_{ij} = b_{ji}$, it does not really matter whether the charge for replacement whose payment is pending is b_{ij} or b_{ji}. So it is just as if we are operating a system in which there are *no replacement costs* and the retrieval costs are $\bar{a}_{ij} = a_{ij} + b_{ij}$. Together with the remarks of the first paragraph, this completes the proof. ■

Model 1 is equivalent to the general model with $a_{ij} = 0$, $b_{ij} = 1$, since by replacing item i at the front of a list which includes j the cost of the next retrieval of j is increased by 1. This 1 is the incremental cost due to the fact that at the next retrieval of j there will be a cost of 1, due to the place which is, or was, filled by i. The resulting partition problem is to choose disjoint sets $L_1, \ldots, L_k$ to

$$\text{Minimize } |L_1| \sum_{i \in L_1} p_i + \cdots + |L_k| \sum_{i \in L_k} p_i.$$

The solution of this problem requires a calculation which is of polynomial time in n. Assuming $p_1 \geq \cdots \geq p_n$, the sets are $L_j = (i_j, i_j + 1, \ldots, i_{j+1} - 1)$ for some $1 = i_1 \leq i_2 \leq \cdots \leq i_k \leq i_{k+1} = n$. That is, all items in set L_i have either smaller or greater request probabilities than the items in set L_j.

In Model 2, suppose box i contains n_i objects, each of weight 1, and $N = \Sigma n_i$. Box i is requested with probability $p_i = n_i/N$. Then the appropriate charge to make when i is replaced on top of a pile which includes j is $n_i p_j/(p_i + p_j)$; namely, the cost n_i times the probability $p_j/(p_i + p_j)$ that box i is requested before box j. Here $b_{ij} = n_i n_j/(n_i + n_j)$ is symmetric. The resulting partition problem is

$$\text{Minimize } \sum_{\substack{i \neq j \\ i,j \in L_1}} n_i n_j + \cdots + \sum_{\substack{i \neq j \\ i,j \in L_k}} n_i n_j.$$

For $k = 2$, this is equivalent to a standard partition problem: for L a subset of $\{1,2,\ldots,n\}$, and $n_1, \ldots, n_n$ integers,

$$\text{Minimize} \left| \sum_{i \in L} n_i - \sum_{i \notin L} n_i \right|.$$

This problem is NP-hard. Not surprisingly, the weight of the boxes in the two stacks are to be made as nearly equal as possible.

In closing, we remark that all of the results of this paper continue to hold even if costs are discounted continuously in time with discount rate α. The proofs may seem to have relied on the fact that when costs are time-averaged one can shift the time at which costs are paid and make no difference to the average cost. However, with modification, one can repeat these ideas for a discounted-cost structure. For example, in the proof of Theorem 2, assuming times of successive requests to be a Poisson process of rate 1, we would need to write $\bar{a}_{ij} = a_{ij} + \{1 + \alpha/(p_i + p_j)\}b_{ij}$.

REFERENCES

1. Anderson, E.J., Nash, P., & Weber, R.R. (1982). A counterexample to a conjecture on optimal list ordering. *Journal of Applied Probability* 19: 730–732.
2. Bentley, L.J. & McGeoch, C. (1983). Worst-case analysis of self-organizing sequential search heuristics. In *Proceedings of the 20th Allerton Conference on Communications, Control, and Computing*. University of Illinois, Urbana Champaign, Illinois, Oct. 6–8, 1982, pp. 452–461.
3. Rivest, R. (1976). On self-organizing sequential search heuristics. *Communication of the ACM* 19: 63–67.
4. Sleator, D.D. & Tarjan, R.E. (1985). Amortized efficiency of list update and paging rules. *Communication of the ACM* 28: 202–208.