IDEAL LIST ORGANIZATION FOR STATIONARY ENVIRONMENTS*

B. John Oommen and David T. H. Ng

SCS-TR-154
March 1989

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

School of Computer Science, Carleton University
Ottawa, Canada K1S 5B6

IDEAL LIST ORGANIZATION FOR STATIONARY ENVIRONMENTS[†]

B. John Oommen and **David T.H. Ng**
School of Computer Science,
Carleton University,
Ottawa, K1J 5B6,
CANADA.

ABSTRACT

We consider the problem of adaptively organizing a list whose elements are accessed with a fixed but unknown probability distribution. We present a strategy which has constant additional space requirements and which achieves the reorganization by performing a data restructuring operation on an element **exactly once**. The scheme, which is stochastically absorbing, and is of a Move-to-Rear flavour is shown to be asymptotically optimal. In other words, by suitably performing the Move-to-Rear operation the probability of converging to the optimal arrangement can be made as close to unity as desired. Considering all of these features, this strategy is probably the most ideal list organization strategy reported in the literature. Simulation results demonstrating the power of the scheme have been included. The paper also includes a hybrid data reorganization scheme in which an absorbing Move-To-Rear rule and an ergodic rule are used in conjunction with each other to optimize the data retrieval process.

Keywords : Dynamic List Ordering, Self-Organizing Lists, Adaptive Data Structures.

[†] Partially supported by the Natural Sciences and Engineering Research Council of Canada.

I. INTRODUCTION

The most elementary and commonly used data structure in computer science is the linear list. The structure has a myriad of applications and is favoured primarily because of the ease in analysing its behaviour in a variety of applications and also because of the ease in implementing it. The latter is particularly true because each element of the list points only to the next element, and thus pointer manipulations are rendered extremely simple.

When a set of elements are to be organized as a list, one would like to arrange the list in such a way that the data retrieval process is optimal. Here, of cause, the data organization is dependent on the users' access probabilities because we would like the more frequently accessed elements to be towards the front of the list in comparison with the less frequently used ones. Indeed, if the access probabilities of the elements are known, it is a trivial exercise to show that the list is optimally arranged if the elements are arranged in the descending order of their access probabilities.

The problem is however far more complex when the access probabilities of the elements are unknown *a priori*. One can then, of cause, envision a scheme which estimates these probabilities and which arranges the elements in the descending order of these estimates. As opposed to this, a large amount of research has gone into the study of having the list organize itself. This paper deals with the problem of adaptively organizing the list **without** estimating the access probabilities of the elements.

To be more specific, let $\mathfrak{R} = \{R_1, ..., R_N\}$ be a list of elements where each R_i is accessed with a probability s_i. $\mathfrak{S} = \{s_1, ..., s_N\}$ is called the user's access distribution and in all the literature available this distribution is assumed to be stationary, and thus $\mathfrak{S}$ is time-invariant. We intend to arrange the elements of $\mathfrak{R}$ as a list so that the average access time is asymptotically minimized. Of course, to render the problem non-trivial, $\mathfrak{S}$ is assumed unknown.

Various adaptive strategies have been described in the literature. A few of them are listed below for the case when the accessed element is not at the front of the list.

(i) The Move-To-Front Rule : The accessed element is moved to the front of the list.

(ii) The Transposition Rule : The accessed element is moved towards the front of the list by interchanging it with its preceding element.

(iii) The Move-k-Ahead Rule : The accessed element is moved k positions forward toward the front of the list unless it is found in the first k positions, in which case, it is moved to the front of the list itself.

(iv) The POS-k Rule : The accessed element is moved to position k of the list if it is in positions k+1 through N. It is transposed with its preceding element if it is in positions 2 through k.

In all of the above cases the list is unchanged if the accessed element is at the front of the list. The properties of the various schemes are found in the respective references and the surveys [1, 3, 4, 6, 13, 15].

Observe that all of the above schemes have an ergodic Markovian representation. Thus, even in the limit, the list can be in any one of its N! configurations. Consequently, a list which was completely organized can be significantly disorganized by the Move-To-Front (or POS-k or Move-k-Ahead) rule by a **single** request of the most infrequently accessed element. Observe that after this unfortunate occurrence, it will take a long time for the list to be organized again - i.e., for this element to dribble its way to the tail of the list. In spite of this drawback, ergodic schemes are generally to be desired if the environment is non-stationary, (i.e., $\mathfrak{S}$ is time variant).

The literature reports the following **absorbing** list organizing methods :

(i) A stochastic Move-To-Front rule in which at time 'n' the accessed element is moved to the front of the list with a probability $f(n) = a^n$. The scheme is expedient [11] if $f(0) = 1$ and $0 < a < 1$.

(ii) A stochastic Move-To-Rear rule in which at time 'n', if the accessed element is R_j, the element is moved to the **rear** of the list with a probability $g(n) = a^{Z_j(n)}$, where $Z_j(n)$ is the number of times R_j has been accessed. The scheme is asymptotically optimal as $a \rightarrow 1$ [11]. The reason why this scheme is superior to the scheme which uses counters is given in [11].

(iii) A deterministic Move-To-Rear rule which moves the accessed element to the **rear** of the list if it has been accessed k times. The scheme is asymptotically optimal as $k \rightarrow \infty$ and performs exactly N data reorganization operations. The drawback of the scheme however is that it requires an extra linear amount of memory [12].

(iv) A deterministic Move-To-Rear rule which moves the accessed element to the **rear** of the list if it has been accessed k consecutive times [12]. This scheme was proven to be expedient but conjectured optimal as $k \rightarrow \infty$.

In this paper, we shall present a new Move-To-Rear rule. In one sense it is but a modified version of the one described in (iv) above. But in another sense, it is a rule in its own right and is distinct from all the other rules examined in the literature. The scheme requires a **constant** amount of additional memory locations and performs **exactly** N data reorganization operations. Finally, and most importantly, the scheme is absorbing and asymptotically optimal. It is thus the ideal list organizing strategy being advantageous in terms of the speed (i.e., the number of data movement operations is the minimum possible), in terms of accuracy and in terms of Markovian behaviour. But there is one fundamental difference between this algorithm and **all** of the other algorithms reported in the literature. In all the algorithms reported in the field of data reorganization, operations are executed based on the probability of the accessed element being requested by the user. However, in the case of the current algorithm (referred to as Algorithm Ideal_MTR), the data reorganization is based on a **conditional** access probability vector the details of which are described in the subsequent sections. The paper proves some of the theoretical properties of the scheme and also demonstrates the properties by simulation. A hybrid heuristic is also presented in this paper in which the MTR rule and an ergodic rule work simultaneously to enhance the data retrieval. Such a hybrid system is completely novel to the literature.

II. THE IDEAL MOVE-TO-REAR STRATEGY

To help present and explain the properties of our strategy, we shall introduce some additional notation.

Let $\mathfrak{V}$ be the set of elements that have not been moved from their initial positions. Note that $\mathfrak{V}$ is initially set to be $\mathfrak{R}$ and is repeatedly being shrunk until it is the null set. When an element R_i is accessed, if R_i is not in $\mathfrak{V}$ (i.e., if R_i has already been moved from its initial position) although the contents of the record R_i are presented to the user, the data reorganization strategy completely ignores the access and thus the list organizing strategy is by-passed. We shall merely now consider the case when $R_i \in \mathfrak{V}$.

Let M be any fixed integer and **Z** be a vector memory location having two integer fields, Z_1 and Z_2. Whenever the element $R_i \in \mathfrak{V}$ is accessed, if Z_1 is zero, it is set to i and Z_2 is set to unity. Furthermore, if Z_1 was previously set to i, Z_2 is incremented. Otherwise the values of both Z_1 and Z_2 are reset to zero. Whenever the value of Z_2 is exactly M, the element R_i (i.e. the element whose index is stored in Z_1) is moved to the **rear** of the list and $\mathfrak{V}$ is updated by deleting from it the element R_i. Subsequently, the element R_i is never moved. This describes the list organization strategy completely. Observe that essentially all that this scheme does is that it moves a record $R_i \in \mathfrak{V}$ to the rear of the list after it has been accessed M **consecutive** times (although in between these accesses, elements of $\mathfrak{R} - \mathfrak{V}$ may have been accessed). We define the condition of having $Z_1 = Z_2 = 0$ as the state of maximum uncertainty.

For the sake of completeness the algorithm is given formally below.

ALGORITHM Ideal_MTR

Input : A sequence of accesses on the list $\mathfrak{R}$. R_i is accessed with an unknown access probability s_i. This is read in using procedure ReadInput.

Output : A reorganized list.

Memory Requirements :

(i) An extra vector memory location **Z** with two integer fields Z_1 and Z_2. Z_1 stores the index of the last accessed record in $\mathfrak{V}$ and Z_2 records the number of times the record whose index is Z_1 has been consecutively accessed if accesses of elements in $\mathfrak{R} - \mathfrak{V}$ are ignored.

(ii) An integer constant M.

```
Method
    $\mathfrak{V} \leftarrow \mathfrak{R}$ ;
    $Z_1 \leftarrow 0$ ; $Z_2 \leftarrow 0$ /* Initialize the algorithm to the state of Maximum Uncertainty */
    Repeat
        ReadInput ($R_i$)
        If $R_i \in \mathfrak{V}$ Then
            If ($Z_1 = i$) Then
                $Z_2 \leftarrow Z_2 + 1$
            Else
                If $Z_1 \neq 0$ Then
                    $Z_1 \leftarrow 0$ ; $Z_2 \leftarrow 0$ /* Reset algorithm to Maximum Uncertainty */
                Else
                    $Z_1 \leftarrow i$ ; $Z_2 \leftarrow 1$
                Endif
            Endif
            If ($Z_2 = M$) Then              /* $R_i$ has been accessed M consecutive */
                Move-To-Rear ($R_i$)         /* times ignoring accesses in $\mathfrak{R}$ - $\mathfrak{V}$ */
                $\mathfrak{V} \leftarrow \mathfrak{V}$ - {$R_i$}
            Endif
        Endif
    Forever
End Algorithm Ideal_MTR.
```

We shall now prove the properties of the above strategy.

Theorem I

The Algorithm Ideal_MTR described above is absorbing. Furthermore, the total number of list reorganizing operations done is exactly N.

Proof :

The second assertion is obvious. The absorbing nature of the Markovian representation of the chain follows from the fact that after every element has been accessed M consecutive times (ignoring the elements which were previously accessed M consecutive times) the list remains in its final configuration, and no more list reorganizing is performed. • • •

Note that it is not just the **expected** number of move operations which is finite as in the case of the stochastic schemes described in [12]. In this case however, the number of moves performed is not a **random variable** and is exactly equal to the size of the list.

The following results concerning the probability of being absorbed in the optimal configuration is more powerful. We shall show that the scheme is asymptotically optimal, i.e., the probability of being absorbed in the optimal configuration can be made as close to unity as desired. To aid the proof, we now introduce the following notations. The access probability vector $\mathfrak{S}$ is an N by 1 probability vector satisfying :

$$\mathfrak{S} = [s_1, s_2, ..., s_N]^T, \text{ where,} \tag{2}$$

$$\sum_{i=1}^{N} s_i = 1. \tag{3}$$

Let $\mathfrak{V}$ be any subset of $\mathfrak{R}$. We define the conditional access probability vector $\mathfrak{S} \mid \mathfrak{V}$ as the vector of normalized probabilities in which only the quantities corresponding to the elements of the set $\mathfrak{V}$ have non-zero values. Thus, $\mathfrak{S} \mid \mathfrak{V}$ consists of the normalized probabilities $\{s_i'\}$, where,

$$s_i' = 0 \qquad \text{if } R_i \notin \mathfrak{V} \tag{4}$$

$$= \frac{s_i}{\sum_{R_i \in \mathfrak{V}} s_i} \qquad \text{otherwise.} \tag{5}$$

Observe that the vector $\mathfrak{S}$ defined by (2) and (3) is exactly equivalent to $\mathfrak{S} \mid \mathfrak{R}$, and thus these above normalized probabilities are indeed conditional probabilities appropriately conditioned.

The main result in Markov chains which we shall use to compute the probability of absorption into an absorbing barrier is stated below. The result is proved in [8].

Lemma 0

Let F be the transition matrix of an irreducible homogeneous Markov chain, where, $F_{i,j}$ is the transition probability associated with the transition from state i to state j. Let $x_{i,A}$ be the probability of being absorbed into an absorbing state A, where $i \in \tau$, the set of transient states. Then $x_{i,A}$ obeys the following recursive equation :

$$x_{i,A} = F_{i,A} + \sum_{j \in \tau} (F_{i,j} \cdot x_{j,A}) \qquad \bullet\bullet\bullet$$

Theorem II

Let $\mathfrak{S}$ be the vector of access probabilities. Then, the probability that R_u precedes R_v in the final list obtained by using Algorithm Ideal_MTR is exactly $s_u^M / (s_u^M + s_v^M)$.

Proof :

Let R_u and R_v be any two distinct elements with indices $u, v \in \{1, 2, ..., N\}$, and let their corresponding access probabilities be s_u and s_v respectively. Since the theorem is trivial for the case when either (or both) these probabilities are zero, with no loss of generality we assume that both s_u and s_v are positive. It is required to prove that in this case, the probability of being absorbed into a list in which R_u precedes R_v is $s_u^M / (s_u^M + s_v^M)$. We shall prove the theorem by performing a computation of the probability conditioned on the length of the list starting from the first element that is moved to the rear.

For the indices $u, v \in \{1, 2, ..., N\}$, let $\xi_{u,v}(\mathfrak{V})$ be the event that either R_u or R_v is the element which is selected from $\mathfrak{V}$ to be moved to the rear, where $\mathfrak{R} \supseteq \mathfrak{V}$. Also, let $\eta_{u,v}(i)$ be the event that the ith element moved contains the first appearance of R_u or R_v. Thus, by virtue of the scheme, R_u ultimately precedes R_v if it is selected to be moved **before** R_v, since if that is the case, R_u is the ith element moved, and R_v will be the jth element moved where $i < j$. Note that by the above definitions,

$$\Pr\left[R_u \text{ ultimately precedes} \mid \eta_{u,v}(N-|\mathfrak{V}|+1)\right] = \Pr\left[R_u \text{ ultimately precedes} \mid \xi_{u,v}(\mathfrak{V})\right] \quad (6)$$

Since $\mathfrak{V}$ is the set of elements which have not been moved to the rear of the list, by definition, the conditional access probabilities are defined by (5). Let

$$p = \frac{s_u}{\sum_{R_i \in \mathfrak{V}} s_i} \quad \text{, and,} \quad q = \frac{s_v}{\sum_{R_i \in \mathfrak{V}} s_i} .$$

Then p is the conditional access probability of accessing R_u, q is the conditional access probability of accessing R_v and 1-p-q is the conditional access probability of accessing any element in $\mathfrak{V} - \{R_u, R_v\}$.

We now compute the probability of R_u preceding R_v given $\xi_{u,v}(\mathfrak{V})$. To do this we describe the following homogeneous irreducible Markov chain. The chain has a set of states, and it is in state $\phi(n)$ defined as below :

$$\begin{aligned}
\phi(n) &= U_i && \text{if } Z_1 = u \text{ and } Z_2 = i && (7)\\
&= V_i && \text{if } Z_1 = v \text{ and } Z_2 = i && (8)\\
&= \theta_0 && \text{if } Z_1 = 0 \text{ and } Z_2 = 0 && (9)\\
&= \theta_t && \text{otherwise.} && (10)
\end{aligned}$$

Clearly, the Markov chain represents the contents of the memory locations Z_1 and Z_2 given $\xi_{u,v}(\mathfrak{V})$, where θ_0 is the state of maximum uncertainty, U_i is the state when R_u has been accessed i consecutive times if accesses of elements in $\mathfrak{R} - \mathfrak{V}$ are ignored, V_i is the state when R_v has been accessed i consecutive times if accesses of elements in $\mathfrak{R} - \mathfrak{V}$ are ignored and θ_t is the state when any element in $\mathfrak{V} - \{R_u, R_v\}$ has been accessed at least once (but not M consecutive times). Note that given $\xi_{u,v}(\mathfrak{V})$, the transition from θ_t to θ_0 has an associated probability of unity. Thus the transition probabilities of the Markov chain are given by the stochastic matrix F, where, for all $i \le i \le M-1$,

$$F_{U_i,U_{i+1}} = p \tag{11}$$
$$F_{U_i,\theta_0} = 1 - p \tag{12}$$
$$F_{V_i,V_{i+1}} = q \tag{13}$$
$$F_{V_i,\theta_0} = 1 - q \tag{14}$$
$$F_{\theta_0,U_1} = p \tag{15}$$
$$F_{\theta_0,V_1} = q \tag{16}$$
$$F_{\theta_0,\theta_t} = 1 - p - q \tag{17}$$
$$F_{\theta_t,\theta_0} = 1 \tag{18}$$

Observe that since U_M and V_M are absorbing states,

$$F_{U_M,U_M} = F_{V_M,V_M} = 1. \tag{19}$$

The transition diagram of the chain is given in Figure I.

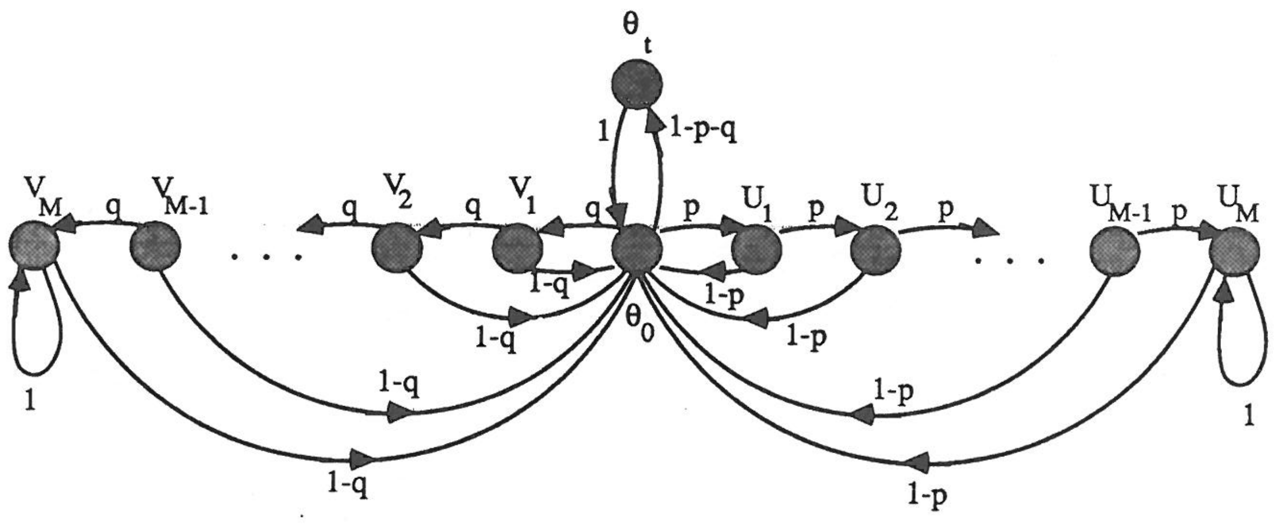

Figure 1 : The transition diagram for the Markov chain representation of the event that either R_u or R_v is moved to the rear given $\xi_{u,v}(\mathfrak{B})$. The conditional probability that R_u is accessed is represented by the transition p, and the conditional probability that R_v is accessed is represented by the transition q.

Let a_i be the probability of converging to U_M given $\xi_{u,v}(\mathfrak{B})$ and that the starting state is U_i. Similarly let b_i be the probability of converging to U_M given $\xi_{u,v}(\mathfrak{B})$ and that the starting state is V_i. Also let Δ_0 be the probability of converging to U_M given $\xi_{u,v}(\mathfrak{B})$ and that the starting state is θ_0 and let Δ_t be the probability of converging to U_M given $\xi_{u,v}(\mathfrak{B})$ and that the starting state is θ_t. Observe that Δ_0 is the quantity we aim to compute. Using Lemma 0, the quantities $\{a_i\}$, $\{b_i\}$, Δ_0 and Δ_t satisfy :

$$a_M = 1; \tag{20}$$
$$a_i = p\, a_{i+1} + (1 - p)\, \Delta_0, \qquad \text{for } 1 \le i \le M\text{-}1; \tag{21}$$
$$b_M = 0; \tag{22}$$
$$b_i = q\, b_{i+1} + (1 - q)\, \Delta_0, \qquad \text{for } 1 \le i \le M\text{-}1. \tag{23}$$

Finally, since θ_t is a reflecting barrier given $\xi_{u,v}(\mathfrak{B})$, $\Delta_t = \Delta_0$, and thus,

$$\Delta_0 = p\, a_1 + q\, b_1 + (1\text{-}p\text{-}q)\, \Delta_0. \tag{24}$$

Solving the difference equation (21) using (20) as a boundary condition yields the value of a_1 as :

$$a_1 = p^{M-1} + \left[\sum_{j=0}^{M-2} p^j \right] (1\text{-}p)\, \Delta_0. \tag{25}$$

Similarly, solving the difference equation (23) using (21) as a boundary condition yields the value of b_1 as :

$$b_1 = \left[\sum_{j=0}^{M-2} q^j \right] (1\text{-}q)\, \Delta_0. \tag{26}$$

Substituting for a_1 and b_1 in (24) yields the equation for Δ_0 as :

$$\Delta_0 = p^M + \left[\left\{ \sum_{j=1}^{M-1} p^j \right\} (1\text{-}p) + \left\{ \sum_{j=1}^{M-1} q^j \right\} (1\text{-}q) \right] \Delta_0 + (1 - p - q)\, \Delta_0,$$

whence, $\Delta_0 = \frac{p^M}{Den}$ (27)

where the Denominator Den is :

$$Den = (p + q) - \left[\sum_{j=1}^{M-1} p^j\right](1-p) - \left[\sum_{j=1}^{M-1} q^j\right](1-q). \qquad (28)$$

Summing the two geometric series in p^j and q^j after some simplifications yields,

$$Den = p^M + q^M.$$

But $p = s_u / \sum_{R_i \in \mathfrak{V}} s_i$ and $q = s_v / \sum_{R_i \in \mathfrak{V}} s_i$.

Since the denominators for p and q are the same, the common terms of the denominator of p and q cancel to yield :

$$\Delta_0 = \frac{s_u^M}{s_u^M + s_v^M} .$$

Since this is true for all $\mathfrak{V} \supseteq \{R_u, R_v\}$, we have proved that for all $\mathfrak{V}$ where $\mathfrak{R} \supseteq \mathfrak{V}$,

$$\Pr\left[R_u \text{ ultimately precedes} \mid \xi_{u,v}(\mathfrak{V})\right] = \frac{s_u^M}{s_u^M + s_v^M} .$$

Combining the above with (6) we have thus proved that for all $1 \leq i \leq N-1$,

$$\Pr\left[R_u \text{ ultimately precedes} \mid \eta_{u,v}(i)\right] = \frac{s_u^M}{s_u^M + s_v^M} .$$

To complete the proof we note that since $\eta_{u,v}(1), \ldots, \eta_{u,v}(M-1)$ are mutually exclusive and collectively exhaustive events, using the laws of total probability,

$$\Pr(R_u \text{ precedes } R_v) = \sum_i \Pr\left[R_u \text{ precedes } R_v \mid \eta_{u,v}(i)\right] . \Pr\left[\eta_{u,v}(i)\right]$$

Since $\Pr\left[R_u \text{ precedes } R_v \mid \eta_{u,v}(i)\right]$ is the same for all i,

$$\Pr(R_u \text{ precedes } R_v) = \Pr\left[R_u \text{ precedes } R_v \mid \eta_{u,v}(i)\right] \cdot \sum_i \Pr\left[\eta_{u,v}(i)\right]$$

$$= \frac{s_u^M}{s_u^M + s_v^M}$$

and the result follows. • • •

The final theorem regarding the asymptotic optimality of the scheme is now proved.

Theorem III

Let $\mathfrak{R} = \{R_1, R_2, ..., R_N\}$ be the list of elements with distinct access probabilities $\{s_1, ..., s_N\}$. Then, the probability of the list converging to the optimal arrangement converges to unity as $M \rightarrow \infty$.

Proof :

For every distinct pair u, v $\{1, ..., N\} \times \{1, ..., N\}$, we have shown that the probability of R_u ultimately preceding R_v is :

$$\Pr[R_u \text{ ultimately precedes } R_v] = \frac{s_u^M}{s_u^M + s_v^M}.$$

As M tends to ∞, we see that if $s_u > s_v$,

$$\lim_{M\to\infty} \Pr[R_u \text{ ultimately precedes } R_v] = \lim_{M\to\infty} \frac{s_u^M}{s_u^M + s_v^M}$$

$$= \lim_{M\to\infty} \frac{1}{1 + \left(\frac{s_v}{s_u}\right)^M} = 1.$$

Thus R_u ultimately precedes R_v w.p. 1 as $M \rightarrow \infty$.

Let P* be the probability that the list converges to optimal arrangement. Since the records are independently drawn according to the distribution $\{s_i\}$, the probability P* is greater than or equal to the probability that **every single pair** is in the decreasing order of the access probabilities. Thus,

$$P^* \geq \prod_{u \neq v} \Pr[R_u \text{ ultimately precedes } R_v \mid s_u > s_v]$$

The result that P* tends to unity follows since every quantity on the product term of the R.H.S. does so as $M \rightarrow \infty$. • • •

Apart from proving that the probability that R_u ultimately precedes R_v in the final list converges to unity as $M \to \infty$ (if $s_u > s_v$), the above results also give us a closed form expression for the probability of any final list arrangement and this probability can be derived for finite (and infinite) values of M. Let

$$\Lambda = [\lambda_1, \lambda_2, \ldots, \lambda_N]$$

be any permutation of the set {1, 2, ..., N}. Let $\delta(i, j) = s_{\lambda_i}^M / \left(s_{\lambda_i}^M + s_{\lambda_j}^M\right)$. Then, using the laws of total probability, the probability that the sequence of the indices of the final list is Λ is $\Pr(\Lambda)$, where, if

$$F(\Lambda) = \prod_{i=1\ ;\ j>i}^{M-1} \frac{s_{\lambda_i}^M}{\left(s_{\lambda_i}^M + s_{\lambda_j}^M\right)}$$

and if Λ^* is the set of all the permutations of the set {1, 2, ..., N}, then,

$$\Pr(\Lambda) = \frac{F(\Lambda)}{\sum_{\sigma \in \Lambda^*} F(\sigma)}.$$

Indeed, this value can be explicitly computed if the access probabilities are known.

At this juncture, it is not inappropriate to highlight the difference between the Algorithm Deterministic-MTR-CSpace [12] and the Algorithm Ideal_MTR described in this paper. The first difference is the fact that although in the former algorithm a MTR operation on R_i is done whenever it has been accessed M consecutive times, whenever a string of accesses for R_i was broken by an access for R_j, the value of Z_1 was set to j and Z_2 was set to unity. In this case, if the required sequence of R_i's was interrupted by an access of R_j, the algorithm goes into an intermediate state in which $Z_1 = Z_2 = 0$. This state is called the state of Maximum Uncertainty because while in this state, no record has an advantage over another.

But there is one fundamental difference between the Algorithm Ideal_MTR and **all** of the other algorithms reported in the literature. In all the algorithms reported in the field of data reorganization, operations are executed based on the probability of the accessed element being requested by the user. However, in the case of Algorithm Ideal_MTR, the data reorganization is based on the **conditional** access probability vector $\mathfrak{S} \mid \mathfrak{V}$. Thus, although accesses of elements in the set $\mathfrak{R} - \mathfrak{V}$ can occur and that, even with a high probability, the occurrence of such accesses **does not** place an impediment on the convergence of the scheme. Since elements in $\mathfrak{R} - \mathfrak{V}$ have been placed in their appropriate positions they are merely reported to the user on being accessed. But elements in $\mathfrak{V}$ (the ones which are still in their "unabsorbed states") are the only elements which are involved in any data reorganization computations.

The impact of this strategy is tremendous. Consider a case in which $\mathfrak{R} = \{R_1, R_2, R_3\}$ and $s_1 = 0.9$, $s_2 = 0.09$, and $s_3 = 0.01$. It is easy to see that since on the average R_1 will be accessed ten times more frequently than R_2 and ninety times more frequently than R_3, R_1 will quite quickly be absorbed to the tail of the list with a high probability. Observe that since only R_2 and R_3 will subsequently have to find their places, R_2 will probably follow R_1 with a high probability to the rear of the list. However, if the accesses on R_1 were to "reset" the learning process by which R_2 found its place, then the mean time for R_2 to converge to its place would be extremely high for this would require a string of M accesses of R_2 with **no interruptions** from R_1 and R_3. But since in Algorithm Ideal_MTR, we are now updating Z_1 and Z_2 based only on the conditional distribution of $\mathfrak{S} \mid \mathfrak{V}$, all the "interrupting" accesses on R_1 have no bearing on the reset operation of the data reorganization strategy. Thus the relative time taken for R_2 to converge to its place will **only** depend on the relative number of accesses on R_2 and R_3, and since R_2 is accessed on the average about nine times as frequently as R_3, it will find its place in a much shorter time duration. Indeed, our method is the only one reported in the literature which performs data reorganization based on a conditional access probability distribution.

Observe too the power of the method. It is absorbing. It requires only two additional memory locations (for a prespecified value of M), it performs exactly one data reorganization operation per record and finally, the probability of converging to the optimal ordering can be made as close to unity as desired by correspondingly increasing M.

III. IMPLEMENTATION DETAILS

The scheme that we have introduced, Ideal-MTR has been described in terms of certain set inclusion operations. Thus, we have chosen to move a record R_i if it is in a set $\mathfrak{V}$, which is a subset of $\mathfrak{R}$, the set of all records. Furthermore, the Move-To-Rear operation was performed to the tail of the list which requires that the most frequently accessed element will seem to linger at the tail of the list until all the elements found their places. Finally, the algorithm was also described in terms of transition to the state of maximum uncertainty and the transition depended critically on the question of whether R_i and R_j (where R_j is the record interrupting a string of R_i's) are elements of $\mathfrak{V}$. Clearly, implementing the algorithm can be clumbersome if the implementation involved only the set operations. However, we shall now show that the implementation is very straight forward if the list is maintained using pointers (as in usually the case) and an additional pointer is used to point of the element of the list which was first moved to the rear.

Let FrontOld be the pointer to the head of the list. This pointer always points to the front of the original list prior to any data reorganization. A second pointer, FrontNew, initially points to the tail of the list. (i.e., it is initialized to NIL). Subsequently as the elements of the list are moved to the rear, they are appended to the tail of the list pointed at by FrontNew. Thus, the elements which have found their place lie between FrontNew and the tail of the list and the elements which are still learning their place lie between FrontOld and FrontNew. Whenever the user request a record R_i, the search for R_i starts from FrontNew and continues towards the tail of the list. If R_i is found prior to encountering the null pointer, this indeed represent the case when R_i is contained in the set $\mathfrak{R} - \mathfrak{V}$. In such a case the counters Z_1 and Z_2 are unchanged.

However, if the accessed element R_i has not been found by the time the null pointer is encountered, the search for R_i continues progressing from FrontOld all the way down to FrontNew. This traversal could result in a data reorganization operation after the corresponding changes to Z_1 and Z_2 have been made.

The implementation of the scheme is shown in Figure II for a list of five elements namely the list $\mathfrak{R} = \{R_1, R_2, ... R_5\}$ in which $s_1 > s_2 > ... > s_5$. In Figure II, we have considered a case when the elements are reorganized (i.e. attain the value M) in the order (1, 2, 3, 4, 5). Initially, on R_1 being accessed M consecutive times, it is moved to the rear of the list. FrontNew now points to R_1. Subsequently, on R_2 being accessed M consecutive times (i.e. ignoring the accesses on R_1), it is moved to the rear of the list. As time proceeds whenever R_3 is accessed M consecutive times (i.e. ignoring the accesses of R_1 and R_2), R_3 is moved to the rear of the list. However, to catalyze the accessing process, the list pointers FrontOld and FrontNew point to the reorganized list.

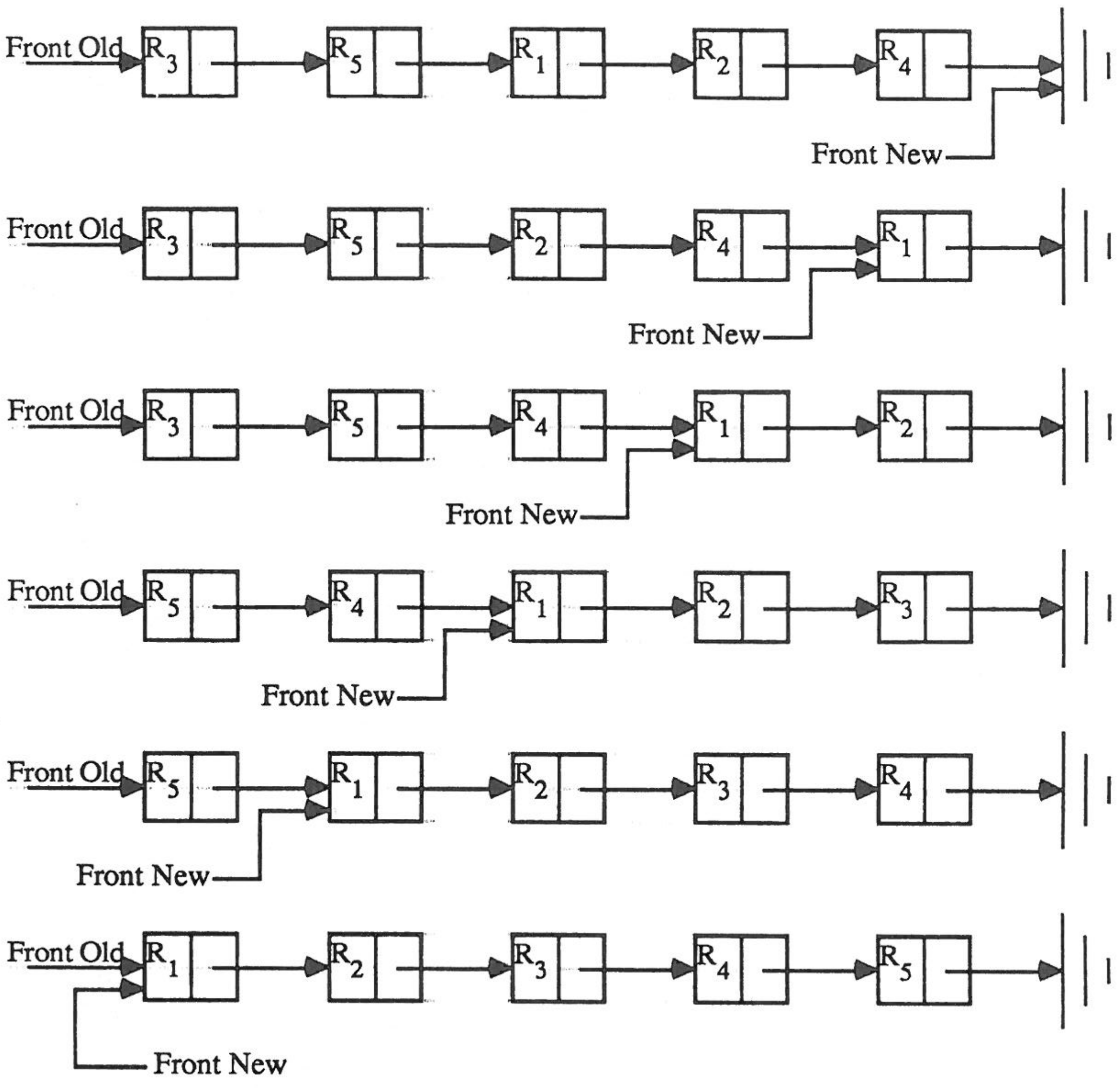

Figure II. The operations of MTR on a list of elements R_1, R_2, ..., R_5 for the case when R_1 is accessed M consecutive times, then R_2 is accessed M consecutive times (ignoring accesses on R_1), and so on for R_3, R_4 and R_5, until all 5 elements are reorganized.

Observe that whenever R_i lies between FrontOld and FrontNew, no data reorganization is performed. It is very easy to generalize this to yield a hybrid system in which the elements between FrontOld and FrontNew are reorganized based on an ergodic heuristic until they have finally found their place at the tail of the list. Thus, although R_i has not been accessed M consecutive times, (i.e. ignoring access of $\mathfrak{R}$ - $\mathfrak{V}$) it need not remain static in its original position. Whenever R_i is accessed, it can be moved forwards towards FrontOld by a heuristic such as the Transposition rule, and thus we are not only guarantied that the cost will be optimal when the list has converged to its final absorbing configuration but we are also guarantied that in between the time instances when a MTR operation is done, the elements which are still in their transient configuration will migrate towards FrontOld so as to minimize the retrieval cost. Such a hybrid system is completely novel to the field of data retrieval.

For the sake of completeness the pointer manipulating version of Algorithm_Ideal_MTR is presented below in the hybrid version described here. In this case, the MTR operations are done using pointer manipulations and the transposition rule is used to permit the elements in their "transient" states to migrate towards FrontOld.

Algorithm Ideal_MTR_With_Hybrid_Pointers

Input : A sequence of accesses on the list $\mathfrak{R}$, the head of the list is pointed to by FrontOld. R_i is accessed with an unknown access probability si. This is read in using procedure ReadInput.

Output : A reorganized list.

Memory Requirements :

(i) An extra vector memory location **Z** with two integer fields Z_1 and Z_2 and a pointer field FrontNew. Z_1 stores the index of the last accessed record in $\mathfrak{V}$ and Z_2 records the number of times the record whose index is Z_1 has been consecutively accessed if accesses of elements in $\mathfrak{R} - \mathfrak{V}$ are ignored. FrontNew points to a sublist in $\mathfrak{R}$ which represents $\mathfrak{R} - \mathfrak{V}$.

(ii) An integer constant M.

```
Method
    FrontNew ← NIL
    $Z_1 \leftarrow 0$ ; $Z_2 \leftarrow 0$        /* Initialize the algorithm to the state of Maximum Uncertainty */
    Repeat
        ReadInput ($R_i$)
        Temp_Ptr ← FrontNew
        While (Temp_Ptr <> NIL) and (Temp_Ptr^.Record <> $R_i$) Do
            Temp_Ptr ← Temp_Ptr^.next
        EndWhile
        If Temp_Ptr = NIL Then  /* An element in $\mathfrak{R} - \mathfrak{V}$ will not satisfy this If Condition */
            Temp_Ptr ← FrontOld
            While (Temp_ptr <> FrontNew) and (Temp_Ptr^.Record <> $R_i$) Do
                Temp_Ptr ← Temp_Ptr^.next
            EndWhile
            If ($Z_1 = i$) Then
                $Z_2 \leftarrow Z_2 + 1$
            Else
                If $Z_1 \neq 0$ Then
                    $Z_1 \leftarrow 0$ ; $Z_2 \leftarrow 0$     /* Reset to the state of Maximum Uncertainty */
                Else
                    $Z_1 \leftarrow i$ ; $Z_2 \leftarrow 1$  /* Move from the state of Maximum Uncertainty */
                Endif
            Endif
            If ($Z_2 = M$) Then                /* $R_i$ has been accessed M consecutive times */
                If (FrontNew = NIL) Then          /* This If Condition is satisfied */
                    FrontNew = Temp_Ptr           /* only for the first MTR*/

                Endif
                Move-To-Rear ($R_i$)
            Else
                Transposition_Rule ($R_i$)
            Endif
        Endif
    Forever
End Algorithm Ideal_MTR_With_Hybrid_Pointers
```

IV. EXPERIMENTAL RESULTS

The efficiency of the Move-To-Rear heuristic is tested for three types of environments which obey, the Zipf, the Geometric and the Wedge distributions respectively. The MTR parameter, M, was allowed to take values 2 and 3 so that the variation due this parameter could be observed. One hundred experiments were conducted for each case and each of the experiments was allowed to run until the final list arrangement was formed. The various distributions that were used as the environments in these tests can be represented by the following sets of equations.

Zipf's Distribution :

$s_1 = C/1,\ s_2 = C/2,\ \ldots,\ s_N = C/N.$

Geometric Distribution with parameter $0 \le \rho < 1$:

$s_1 = C,\ s_2 = C\rho,\ s_3 = C\rho^2,\ \ldots,\ s_N = C\rho^{N-1}.$

Wedge Distribution with parameter α :

$s_1 = C,\ s_2 = C-\alpha,\ s_3 = C-2\alpha,\ \ldots,\ s_N = C-(N-1)\alpha.$

In the above distributions, C is the normalizing constant to ensure the sum of the individual probabilities is unity. Table I shows the results of the simulations. The convergence rate is calculated for the average time required to form the final list arrangement and the asymptotic cost is derived from the sums of the average access costs of the final list arrangement. The asymptotic cost is averaged over the number of experiments to obtain the final ensemble average. Notice that the performance of the MTR scheme is rated against the Optimal list arrangement where the term in brackets indicates the percentage increase in the asymptotic cost relative to the Optimal list arrangement.

Notice that for the Geometric distribution with N=5 and M=2, the asymptotic cost is 2.535 which is 9.1% more than the Optimal arrangement. This percentage decreases to 6.4% when M=3. The change is more significant with larger value of N, as can be observed for the same distribution, when N=15. In this case, the percentage decreases from 11% to an incredible value of 4.3%. The decreasing asymptotic cost and the significant increase in convergence rate are clearly observed for Geometric distribution with N=5. In this case, the convergence rate changes from 81.36 to 182.8 for a change in the value of M from 2 to 3.

Environment	M / Measure / N	2		3	
		Convergence Rate	Asymptotic Cost	Convergence Rate	Asymptotic Cost
Zipf	5	90.47	2.372(8.3%)	211.1	2.299(5.0%)
	10	383.3	3.797(11.%)	1456.	3.609(5.7%)
	15	909.8	5.055(11.%)	4854.	4.803(6.3%)
Geometric (ρ =0.7)	5	81.36	2.535(9.1%)	182.8	2.471(6.4%)
	10	620.7	3.345(9.9%)	1702.	3.214(5.6%)
	15	4155.	3.629(11.%)	11030.	3.403(4.3%)
Wedge (α =0.009)	5	56.63	2.984(2.5%)	184.4	2.980(2.4%)
	10	249.9	5.189(9.1%)	1169.	5.059(6.3%)
	15	1560.	6.101(11.%)	4205.	5.844(6.6%)

Table 1 : The performance of the MTR scheme under various environments. The percentage increase in asymptotic cost with respect to the Optimal arrangement is shown in brackets.

V. CONCLUSIONS

In this paper, we have considered the problem of organizing the linear list of N elements in which the elements are accessed with a fixed but unknown probability distribution. We have presented a strategy which requires a constraint number of extra memory locations and which achieves the data reorganization by performing exactly N restructuring operations. The scheme is stochastically absorbing and of Move-To-Rear flavour, and we have shown that the probability of the scheme converging to the optimal arrangement can be made as close to unity as desired. The algorithm is fundamentally distinct from **all** of the other algorithms reported in the literature because, unlike the contemporary methods, operations are not executed based on the probability of the accessed element being requested by the user, but they are done based on a **conditional** access probability vector the details of which have been described in the paper. The paper also includes a hybrid data reorganization scheme in which an absorbing Move-To-Rear rule and an ergodic rule are used in conjunction with each other to optimize the data retrieval process, and numerous simulation results demonstrating the power of the absorbing Move-To-Rear strategy are presented.

REFERENCES

[1] Arnow, D.M. and Tenebaum, A.M., "An Investigation of the Move-Ahead-k Rules", Congressus Numerantium, Proc. of the Thirteenth Southeastern Conference on Combinatorics, Graph Theory and Computing, Florida, February 1982, pp. 47-65.

[2] Bitner, J.R., "Heuristics That Dynamically Organize Data Structures", SIAM J. Comput., Vol.8, 1979, pp. 82-110.

[3] Burville, P.J. and Kingman, J.F.C., "On a Model for Storage and Search", J. Appl. Probability, Vol.10, 1973, pp. 697-701.

[4] Gonnet, G.H., Munro, J.I. and Suwanda, H., "Exegesis of Self Organizing Linear Search", SIAM J. Comput., Vol.10, 1981, pp.613-637.

[5] Hendricks, W.J., "An Extension of a Theorem Concerning an Interesting Markov Chain", J. App. Probability, Vol.10, 1973, pp.231-233.

[6] Hester, J.H. and Hirschberg, D.S., "Self-Organizing Linear Search", ACM Computing Surveys, Vol. 17, 1985, pp.295-311.

[7] Kan, Y.C. and Ross, S.M., "Optimal List Order Under Partial Memory Constraints", J. App. Probability, Vol.17, 1980, pp. 1004-1015.

[8] Isaacson, D.L. and Madsen, R.W., "Markov Chains : Theory and Applications" New York, John Wiley & Son, 1976.

[9] Knuth, D.E., "The Art of Computer Programming, Vol.3, Sorting and Searching", Addison-Wesley, Reading, MA., 1973.

[10] McCabe, J., "On Serial Files With Relocatable Records", Operations Research, Vol.12, 1965, pp.609-618.

[11] Oommen, B.J. and Hansen, E.R., "List Organizing Strategies Using Stochastic Move-to-front and Stochastic Move-to-Rear Operations", to appear in SIAM Journal on Computing. Vol. 16, (1987), 705-716.

[12] Oommen, B.J., Hansen, E.R. and Munro, J.I., "Deterministic Optimal and Expedient Move-To-Rear List Organizing Strategies", to appear in Theoretical Computer Science (Preliminary abridged version was published in the Proc. of the 25th Annual Allerton Conference, Urbana, Illinois, Sept/Oct. 1987, pp.54-63).

[13] Rivest, R.L., "On Self-Organizing Sequential Search Heuristics", Comm. ACM, Vol.19, 1976, pp.63-67.

[14] Sleator, D. and Tarjan, R., "Amortized Efficiency of List Update Rules", Proc. of the Sixteenth Annual ACM Symposium on Theory of Computing, April 1984, pp. 488-492.

[15] Tenenbaum, A.M. and Nemes, R.M., "Two Spectra of Self-Organizing Sequential Search Algorithms", SIAM J. Comput., Vol.11, 1982, pp-557-566.