JOURNAL OF ALGORITHMS 14, 88–114 (1993)

Self-Organizing Doubly-Linked Lists

R. S. VALIVETI AND B. J. OOMMEN*

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

Received May 1990; revised September 1991

In this paper, we study the problem of maintaining a doubly-linked list (DLL) in approximately optimal order, with respect to the mean search time. We study two types of DLL reorganization strategies. Move-To-End (MTE) [12] and SWAP [14] are two memoryless DLL heuristics obtained from natural extensions of the well-known singly-linked-list (SLL) heuristics, move-to-front and transposition, respectively. We first derive a general sufficient condition which permits comparison of *any* two DLL heuristics. We use this condition as a guideline to identify families of access distributions for which SWAP yields a lower expected cost than the MTE. We have also presented an absorbing DLL heuristic. The strategy requires one additional memory location and is analogous to the scheme presented in [15]. The reorganization is achieved by moving each element *exactly* once to its final position in the reorganized list. The scheme is stochastically absorbing and it is shown to be optimal for a restricted family of distributions. Thus, for these distributions, the probability of the scheme converging to the optimal list order can be made as close to unity as desired. © 1993 Academic Press, Inc.

1. INTRODUCTION

In recent years considerable research effort has focused on the aspect of dynamically reorganizing a sequential list (also referred to as a singly-linked list (SLL)), so as to keep it in approximately optimal order. The literature on SLLs is exhaustive. For a detailed discussion of this topic, we refer the reader to [4, 6]. Among the heuristics that have been thoroughly investigated are the move-to-front (MTF) and the transposition rule (TR), and detailed analyses of these schemes can be found in [2, 11, 16].

Although SLLs have been very extensively studied and analyzed, very little work has been done to study the adaptive restructuring of doubly-

*Both authors partially supported by the Natural Sciences and Engineering Research Council of Canada. The work of the first author was also supported by a graduate award from Bell Northern Research. A preliminary version of this paper can be found in the "Proceedings of the Eleventh International Conference of the Chilean Computer Science Society, October 1991, Santiago, Chile.

0196-6774/93 $5.00
Copyright © 1993 by Academic Press, Inc.
All rights of reproduction in any form reserved.

linked lists (DLLs). To the best of our knowledge, the only papers in this regard are [12, 14]. The DLL can be defined as a linear list in which each item in the linked list has pointers to both its successor and its predecessor [10]. This structure facilitates searching the list commencing from either end. Various applications of DLLs are catalogued in [19]. DLLs are quite commonly employed in managing the requests for disk blocks. They can also be employed in system simulations [10] and memory management [7]. Apart from these applications consider the scenario in which two requestors are using the same list.[1] If the list is to be maintained so as to optimize the retrieval characteristics for both requestors we can envision two strategies. In the first we can maintain two conceptually distinct SLLs in which the data elements are not duplicated but in which two pointer fields are maintained to permit the reorganization of each list. Clearly, in such a situation, any SLL reorganization heuristic is applicable. The disadvantage of the scheme is that the concept of a unique successor/predecessor for each node must be forfeited. The alternate strategy is to maintain a distinct DLL in which the migration of a single element modifies both the successor and predecessor fields. Clearly in this case heuristics pertinent to DLLs should be sought.

Although we are at a loss in reporting various other applications of DLLs, we would like to emphasize that the fundamental contributions of this paper are the analytic results and the generalizations of the corresponding SLL heuristics. Indeed, as we shall show, although the conceptual generalizations of the heuristics are straightforward, the analytic generalizations are far from trivial and possibly not even true.

Formally speaking, a DLL, D, consists of the set of N records $\mathscr{R} = \{R_1, R_2, \ldots, R_N\}$ stored in some order. Accesses to the record R_i are made from the left and right with probabilities s_{iL} and s_{iR}, respectively. These probabilities obey the constraint that:

$$\sum_{i=1}^{N} (s_{iL} + s_{iR}) = 1. \tag{1}$$

We assume that the record accesses are statistically independent and identically distributed. We also assume that the starting position of the search for a record is independent of the record itself. In our case, we will always commence the search for an element from the end at which the element was requested.

Modeling the DLL using (1) indirectly implies that we are resorting to the expected case analysis for studying the efficiency of any heuristic used to reorganize DLLs. This is certainly not the only way by which reorganizing strategies can be studied. A whole domain of fascinating results are

[1] We are grateful to an anonymous referee for suggesting this potential application.

available for what is referred to as *amortized cost analysis* [1, 18]. In the latter type of analysis, probability distributions are not involved. Amortized analysis can be used to show that even though the worst case cost of performing an operation may be high, the average cost considered over a sequence of operations may not be high. The emphasis is to determine the *worst cost* over a *sequence* of operations, and then average this cost over the operations performed. This essentially yields the guaranteed *average cost* of each operation in the worst case. In the "aggregate method" of amortized analysis, an upper bound $T(n)$ on the total cost of performing n operations is determined. Thus the amortized cost per operation is $T(n)/n$. Another approach to bound the total cost of a sequence of operations is based on the use of potential functions. These methods assign an amortized cost to each operation. By a suitable choice of the potential function, the total actual cost of performing a sequence of operations is shown to be bounded from above by the total amortized cost [3]. The interesting thing about amortized analysis is that the results obtained are sometimes counterintuitive and apparently in contradiction to the expected case analysis. Thus, for example, whereas the TR rule for SLLs is always superior to the MTF rule in the expected sense, it is shown to be inferior to the MTF rule in the amortized sense. An amortized cost analysis for the DLL heuristics presented here is currently being attempted.

Throughout this paper we will consider various DLL heuristics and analyze them in the expected sense when the query patterns obey any arbitrary probability distribution satisfying (1). Our aim is to maintain D in an approximately optimal order with respect to the expected cost. Before we investigate any heuristics to reorganize the list, it is convenient to understand what constitutes an optimal arrangement.

1.1. *The Optimal Order in a DLL*

Let π be a given arrangement of the records. The position of the record R_i in π will be denoted by $\pi^{-1}(i)$. We note that the cost of locating the record R_i is $\pi^{-1}(i)$ if the search commences from the left and is $N - \pi^{-1}(i) + 1$ for searches from the right. Hence the conditional expected cost can be written

$$
\begin{aligned}
&E[\text{Cost} \mid \text{Configuration is } \pi] \\
&\quad = \sum_{i=1}^{N} s_{iL}\pi^{-1}(i) + \sum_{i=1}^{N} s_{iR}(N - \pi^{-1}(i) + 1) \\
&\quad = \sum_{i=1}^{N} (s_{iL} - s_{iR})\pi^{-1}(i) + \sum_{i=1}^{N} s_{iR}(N + 1). \qquad (2)
\end{aligned}
$$

Note that in (2), the second term is independent of the permutation π, and hence the optimal order is the permutation which arranges the list in the decreasing order of the difference $s_{iL} - s_{iR}$. Since we assume that the access distribution $\mathcal{S}$ is unknown *a priori*, we will consider the use of heuristics which endeavor to minimize the expected cost.

Two types of heuristics have been studied for SLLs and they can be categorized to be either ergodic or absorbing. These types of data reorganizations inherit their names from the underlying Markov chains used to analyze them. In the ergodic schemes, the list is dynamically reorganized according to the heuristic, and it has the property that even asymptotically, the list could be in any one of the $N!$ configurations. For stationary environments, the ideal heuristic would, of course, be one which gradually transforms the initial list into the optimal configuration and then never disturbs it. Absorbing schemes aim to attain this goal.

In this paper we study both ergodic and absorbing DLL heuristics. In connection with ergodic heuristics, we establish a sufficient condition for one DLL heuristic to be better than another, in the sense of lower expected cost. We make use of this condition to compare the DLL heuristics SWAP and MTE, which are obtained by extending the well-known SLL heuristics TR and MTF, respectively. Specifically, we identify sufficient conditions under which SWAP has a lower expected asymptotic cost than the MTE. The necessary conditions are still being investigated.

We subsequently turn our attention to absorbing rules for reorganizing DLLs. We present an absorbing scheme which is based on using one additional counter and is a generalization of the SLL absorbing heuristic proposed in [15]. For a restricted family of distributions, this heuristic is shown to be asymptotically optimal. Thus, for these distributions, the probability of converging to the optimal order can be made as close to unity as desired.

It is rather astonishing to discover that the families of distributions for which the above absorbing heuristic is asymptotically optimal happens to be a superset of the family of distributions under which SWAP outperforms the MTE heuristic. We begin our discussion by considering ergodic heuristics.

2. Heuristics for DLL

We begin with a few definitions.

Definition 1. A memoryless heuristic for a DLL is described by a set of permutations given by $\tau = \{\tau_i^L, \tau_i^R\}$. The permutation τ_i^L (τ_i^R) is applied

whenever the element located at position i in the list is assessed from the left (right) [12].

DEFINITION 2. Let $C_\tau(\mathscr{S})$ denote the expected cost resulting from the use of the heuristic τ when the access distribution is $\mathscr{S} = [s_{1L}, s_{1R}, s_{2L}, s_{2R}, \ldots, s_{NL}, s_{NR}]^T$. A heuristic τ is said to be *more efficient* than another heuristic σ if $C_\tau(\mathscr{S}) \leq C_\sigma(\mathscr{S})$ for all distributions $\mathscr{S}$.

The definition for memoryless heuristics for DLLs suggests a rather straightforward way in which SLL heuristics can be applied to DLLs. Let $\rho = \{\rho_i\}$ be a SLL heuristic. A very simple approach to mapping ρ to become a DLL heuristic is to set $\tau_i^L = \rho_i = \tau_i^R$. In other words, the data reorganization is insensitive to the direction of access. This mapping is termed as an *undirected mapping* [14]. In [14] Ng and Oommen show that for the distributions in which the conditional probability of accessing each of the records of a DLL from the left is the same quantity p, the ranking of heuristics in the SLL domain is preserved when they are mapped by an undirected mapping, and $p > 0.5$.

The other approach to synthesizing a DLL heuristic is by using a mapping called a *directed mapping*. In this approach, whenever a record at position i is accessed from the left, we apply the permutation $\tau_i^L = \rho_i$. On the other hand, if the access is from the right, we merely apply a "reflection" of the permutation ρ_i. In other words, we apply the same permutation with the assumption that the list starts at the right end and terminates at the left.

A simple application of this technique transforms the well-known MTF rule for SLLs to the MTE rule described in [12]. For fixing ideas, let us denote the order of elements in the DLL D, by the permutation $\langle R_{i_1}, R_{i_2}, \ldots, R_{i_N} \rangle$, where $\langle i_1, i_2, \ldots, i_N \rangle$ is a permutation of the integers $1 \cdots N$. The MTE operates in the following fashion: if the element R_{i_j} is accessed from the left ($j \neq 1$), the accessed element is moved to the left end of the list, resulting in the configuration $\langle R_{i_j}, R_{i_1}, R_{i_2}, \ldots, R_{i_{j-1}}, R_{i_{j+1}}, \ldots, R_{i_n} \rangle$. On the other hand, if the element R_{i_j} is accessed from the right (and $j \neq N$, i.e., the element is not already at the end), the accessed element is moved to the right end, yielding the list: $\langle R_{i_1}, R_{i_2}, \ldots, R_{i_{j-1}}, R_{i_{j+1}}, \ldots, R_{i_N}, R_{i_j} \rangle$.

In this paper, we study the effect of applying a directed mapping to the TR heuristic to generate the SWAP heuristic for DLLs. The SWAP heuristic was proposed by Ng and Oommen [14]. For the sake of clarify, a description of the heuristic is given below.

As before, let the order of elements in the DLL be $\langle R_{i_1}, R_{i_2}, \ldots, R_{i_N} \rangle$. In the SWAP heuristic, whenever a record R_{i_j} is accessed from the left,

the list is reorganized by swapping R_{i_j} with $R_{i_{j-1}}$. We note that no swap is performed if $j = 1$ and the element being accessed is at the left end of the list D. On the other hand, if the search for the element R_{i_j} commenced at the right end, we swap R_{i_j} with $R_{i_{j+1}}$, unless $j = N$. The special case identified corresponds to the case when the R_{i_j} happens to the last element in the list.

Rivest showed that the TR heuristic always has a lower expected cost when compared to the MTF heuristic [16]. Intuitively there is good reason to believe that SWAP should perform better than the MTE rule. While we conjecture that this is indeed true, a rigorous proof is by no means simple. The focus of this paper is, therefore, to establish sufficient conditions which will ensure that SWAP is better than MTE.

In the following derivations, we will assume, without loss of generality that $s_{iL} - s_{iR} \geq s_{jL} - s_{jR}$ $\forall i < j$. In other words, the optimal order of the records is $R_1, R_2, \ldots, R_N$. We shall make extensive use of the fact that the probability of a given record R_i preceding another record R_j under a heuristic τ (denoted by $b_\tau(i, j)$) determines the cost of using τ. We now state and prove a sufficient condition which allows us to compare the efficiency of any two arbitrary heuristics.

THEOREM 1.[2] *Let τ and σ be two DLL heuristics with the property that $b_\tau(i, j) \geq b_\sigma(i, j)$ $\forall i < j$. Then τ is more efficient than σ.*

Proof. We denote the expected cost produced by the heuristics τ and σ as $C_\tau(\mathcal{S})$ and $C_\sigma(\mathcal{S})$, respectively. We can write down the expected cost under the heuristic τ as

$$C_\tau(\mathcal{S}) = \sum_{i=1}^{N} s_{iL}\left\{1 + \sum_{j \neq i} b_\tau(j, i)\right\} + \sum_{i=1}^{N} s_{iR}\left\{1 + \sum_{j \neq i} b_\tau(i, j)\right\}.$$

After simplifying and representing the terms independent of i as K, we obtain

$$\begin{aligned} C_\tau(\mathcal{S}) &= K + \sum_{i<j} \left\{s_{iL} b_\tau(j, i) + s_{jL} b_\tau(i, j) - s_{iR} b_\tau(j, i) - s_{jR} b_\tau(i, j)\right\} \\ &= K + \sum_{i<j} \left\{(s_{jL} - s_{jR}) - (s_{iL} - s_{iR})\right\} b_\tau(i, j) + (s_{iL} - s_{iR}). \end{aligned}$$

[2]The following result is analogous, in spirit, to the result derived by Hendricks [5]. However, the latter is applicable only for SLLs and the generalization to DLLs is not a direct consequence of Hendricks's result. Indeed, as will be obvious, some limiting probability generalizations of SLLs to DLLs are not even applicable. We are grateful to the anonymous referee for highlighting the similarity. The same comment also applies to Lemma 0.

The expression for $C_\sigma(\mathscr{S})$ has an identical form, and hence we can write down the difference between $C_\tau(\mathscr{S})$ and $C_\sigma(\mathscr{S})$ as

$$C_\tau(\mathscr{S}) - C_\sigma(\mathscr{S}) = \sum_{i<j} \{(s_{jL} - s_{jR}) - (s_{iL} - s_{iR})\}(b_\tau(i,j) - b_\sigma(i,j)). \tag{3}$$

Since we have assumed that $b_\tau(i,j) \geq b_\sigma(i,j)$ and that $s_{iL} - s_{iR} \geq s_{jL} - s_{jR}$ $\forall i < j$, the RHS of (3) is never greater than zero and the result is proved. □

2.1. *Comparison of the SWAP and MTE Heuristics*

Because of Theorem 1, a sufficient condition for SWAP to perform better than MTE is that the probability of record R_i being before another record R_j (as viewed from the left) should be greater for the SWAP heuristic than for the MTE heuristic, whenever $i < j$. We wish to identify distributions for which $b_{\text{SWAP}}(i,j) \geq b_{\text{MTE}}(i,j)$ $\forall i < j$.

Ng and Oommen analyzed the Markov chain for the SWAP heuristic, making use of the fact that it is *time reversible* [14, 17]. The relation obeyed by the stationary distribution of the chain representing the SWAP heuristic is stated by Lemma 0 below [14]:

LEMMA 0. *Consider the two list configurations* $\langle R_{i_1}, R_{i_2}, \ldots, R_{i_j}, R_{i_{j+1}}, \ldots, R_{i_N}\rangle$ *and* $\langle R_{i_1}, R_{i_2}, \ldots, R_{i_{j+1}}, R_{i_j}, \ldots, R_{i_N}\rangle$. *The ratio of the asymptotic probabilities of these two configurations is given by*

$$\frac{\Pr\left(\langle R_{i_1}, R_{i_2}, \ldots, R_{i_j}, R_{i_{j+1}}, \ldots, R_{i_N}\rangle\right)}{\Pr\left(\langle R_{i_1}, R_{i_2}, \ldots, R_{i_{j+1}}, R_{i_j}, \ldots, R_{i_N}\rangle\right)} = \frac{s_{i_jL} + s_{i_{j+1}R}}{s_{i_jR} + s_{i_{j+1}L}}. \tag{4}$$

Proof. The proof is found in [14].

We now formally define the families of distributions examined in this paper.

DEFINITION 3. An access distribution $\mathscr{S} = [s_{1L}, s_{1R}, s_{2L}, s_{2R}, \ldots, s_{NL}, s_{NR}]^T$ satisfying

$$s_{iL} - s_{iR} > s_{jL} - s_{jR} \quad \forall i < j$$

is said to be *exclusive* if the following condition holds whenever $i < j$:

$$s_{kL}(s_{iL} - s_{jL}) + s_{kR}(s_{jR} - s_{iR}) + (s_{iL}s_{jR} - s_{jL}s_{iR}) > 0 \qquad \forall k \neq i, j. \tag{5}$$

Note. This family is termed exclusive, merely for want of a better characterizing property of the family of distributions. Indeed if the condition (5) is satisfied, it turns out that ratios of probabilities in which records R_i and R_j are separated by any number of elements has, as its lower bound, the analogous ratio in which records R_i and R_j are adjacent. Thus, in a sense, the influence of the intervening records can be ignored. The following theorem formally states this property.

THEOREM 2. *Let $\mathscr{S}$ be an exclusive distribution, and R_i and R_j be any two records of $\mathscr{R}$, where $i < j$. Also let $\langle k_1, k_2, \ldots, k_m \rangle$ denote some permutation of m of the integers in the set $\{1, 2, \ldots, N\} - \{i, j\}$. Then, the result*

$$\frac{\Pr(\langle L_1 R_i R_{k_1} R_{k_2} \cdots R_{k_m} R_j L_2 \rangle)}{\Pr(\langle L_1 R_j R_{k_1} R_{k_2} \cdots R_{k_m} R_i L_2 \rangle)} \geq \frac{s_{iL} + s_{jR}}{s_{jL} + s_{iR}}$$

holds, where L_1 and L_2 are arbitrary prefix and suffix lists respectively, made up of the elements of $\mathscr{R} - \{R_i, R_j\} - \{R_{k_1}, R_{k_2}, \ldots, R_{k_m}\}$.

Proof. For the sake of clarity, we denote

$$\xi_{ij}(\langle k_1, k_2, \ldots, k_m \rangle) = \frac{\Pr(\langle L_1 R_i R_{k_1} R_{k_2} \cdots R_{k_m} R_j L_2 \rangle)}{\Pr(\langle L_1 R_j R_{k_1} R_{k_2} \cdots R_{k_m} R_i L_2 \rangle)}.$$

The theorem trivially holds when $m = 0$ (by (4)) and thus the basis step of an inductive proof is established. We will prove the general result by induction, assuming in the inductive step that the theorem is true for $m = l - 1$. That is, we assume that

$$\xi_{ij}(\langle k_1, k_2, \ldots, k_{l-1} \rangle) \geq \frac{s_{iL} + s_{jR}}{s_{jL} + s_{iR}}.$$

Now, the ratio of the stationary probabilities of the two states in which records R_i and R_j are separated by l elements can be written using the

chain rule as

$$\begin{aligned}
&\xi_{ij}(\langle k_1, k_2, \ldots, k_l \rangle) \\
&\quad = \frac{\Pr(\langle L_1 R_i R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_j R_{k_l} L_2 \rangle)}{\Pr(\langle L_1 R_j R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_i R_{k_l} L_2 \rangle)} \\
&\qquad \times \frac{\Pr(\langle L_1 R_j R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_i R_{k_l} L_2 \rangle)}{\Pr(\langle L_1 R_i R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_j R_{k_l} L_2 \rangle)} \\
&\qquad \times \frac{\Pr(\langle L_1 R_i R_{k_1} R_{k_2} \cdots R_{k_l} R_j L_2 \rangle)}{\Pr(\langle L_1 R_j R_{k_1} R_{k_2} \cdots R_{k_l} R_i L_2 \rangle)} \\
&\quad = \xi_{ij}(\langle k_1, k_2, \ldots, k_{l-1} \rangle) \frac{\Pr(\langle L_1 R_i R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_{k_l} R_j L_2 \rangle)}{\Pr(\langle L_1 R_i R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_j R_{k_l} L_2 \rangle)} \\
&\qquad \times \frac{\Pr(\langle L_1 R_j R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_i R_{k_l} L_2 \rangle)}{\Pr(\langle L_1 R_j R_{k_1} R_{k_2} \cdots R_{k_{l-1}} R_{k_l} R_i L_2 \rangle)} \\
&\quad = \xi_{ij}(\langle k_1, k_2, \ldots, k_{l-1} \rangle) \cdot \left[\frac{s_{jR} + s_{k_l L}}{s_{jL} + s_{k_l R}} \cdot \frac{s_{iL} + s_{k_l R}}{s_{iR} + s_{k_l L}} \right] \qquad \text{(by (4))}.
\end{aligned} \tag{6}$$

Let us denote the term within the square brackets of (6) as f. The difference between the numerator and the denominator of f can be easily seen to be

$$s_{k_l L}(s_{iL} - s_{jL}) + s_{k_l R}(s_{jR} - s_{iR}) + (s_{iL} s_{jR} - s_{jL} s_{iR}).$$

Because of the assumption that $\mathcal{S}$ is an *exclusive* distribution, the quantity f is strictly greater than 1. Hence, (6) can be rewritten as:

$$\xi_{ij}(\langle k_1, k_2, \ldots, k_l \rangle) \geq \xi_{ij}(\langle k_1, k_2, \ldots, k_{l-1} \rangle). \tag{7}$$

The theorem follows because of the inductive hypothesis. □

THEOREM 3. *For all exclusive distributions* $\mathcal{S}$, $b_{\text{SWAP}}(i, j) \geq b_{\text{MTE}}(i, j)$ $\forall i < j$, *and hence, the SWAP heuristic is more efficient than the MTE.*

Proof. Our proof will be a generalization of the proof used by Rivest to prove that TR is better than MTF [16, 17]. We shall show that for all exclusive distributions, $b_{\text{SWAP}}(i, j) \geq b_{\text{MTE}}(i, j)$, for $1 \leq i < j \leq N$. We

then have

$$\frac{b_{\mathrm{SWAP}}(i,j)}{b_{\mathrm{SWAP}}(j,i)} = \frac{\sum_{l<m}\sum_{k's} \Pr(\langle R_{k_1},\ldots,R_{k_l},R_i,R_{k_{l+1}},\ldots,R_{k_m},R_j,R_{k_{m+1}},\ldots,R_{k_{N-2}}\rangle)}{\sum_{l<m}\sum_{k's} \Pr(\langle R_{k_1},\ldots,R_{k_l},R_j,R_{k_{l+1}},\ldots,R_{k_m},R_i,R_{k_{m+1}},\ldots,R_{k_{N-2}}\rangle)}. \tag{8}$$

Note that in (8), the summations are taken over all permutations $k_1, k_2, \ldots, k_{N-2}$ of the integers $1,2,\ldots,i-1,i+1,\ldots,j-1,j+1,\ldots,N$, for all $1 \le l < m \le N$. Thus from (8) a weak lower bound for the LHS of (8) can be immediately written as

$$\frac{b_{\mathrm{SWAP}}(i,j)}{b_{\mathrm{SWAP}}(j,i)} \ge \min_{1\le l<m\le N} \frac{\Pr(\langle R_{k_1},\ldots,R_{k_l},R_i,R_{k_{l+1}},\ldots,R_{k_m},R_j,R_{k_{m+1}},\ldots,R_{k_{N-2}}\rangle)}{\Pr(\langle R_{k_1},\ldots,R_{k_l},R_j,R_{k_{l+1}},\ldots,R_{k_m},R_i,R_{k_{m+1}},\ldots,R_{k_{N-2}}\rangle)}. \tag{9}$$

Because of Theorem 2, we immediately conclude that

$$\frac{b_{\mathrm{SWAP}}(i,j)}{b_{\mathrm{SWAP}}(j,i)} \ge \frac{s_{iL}+s_{jR}}{s_{jL}+s_{iR}}, \tag{10}$$

which implies that

$$b_{\mathrm{SWAP}}(i,j) \ge \frac{s_{iL}+s_{jR}}{s_{iL}+s_{jR}+s_{jL}+s_{iR}}. \tag{11}$$

Observe that the expression on the RHS of (11) is exactly $b_{\mathrm{MTE}}(i,j)$, as derived in [12]. Hence we have proved that

$$b_{\mathrm{SWAP}}(i,j) \ge b_{\mathrm{MTE}}(i,j) \qquad \forall i < j.$$

This information, together with Theorem 1, establishes the superiority of the SWAP heuristic, when compared to MTE, for all exclusive distributions. □

It only remains to show that entire families of exclusive distributions exist. Two such families are listed below.

COROLLARY 1. *Consider the families of distributions in which* $s_{iL} > s_{jL}$ *and* $s_{jR} > s_{iR}$ *for all* $i < j$. *A distribution of this type is said to be dually monotonic. All dually monotonic distributions are exclusive and hence, the SWAP heuristic is more efficient than the MTE heuristic for such query distributions.*

Proof. By virtue of the conditions satisfied by a dually monotonic distribution, we can easily deduce that $s_{iL} - s_{jL} \geq 0$ and $s_{jR} - s_{iR} \geq 0$ for all $i < j$. Moreover, whenever $i < j$, we trivially observe that $s_{iL}s_{jR} \geq s_{iR}s_{jL}$. In other words, this distribution satisfies the condition defined in (5), and hence all dually monotonic distributions are also exclusive. By Theorem 3, it follows that for this family of access distributions, SWAP performs better than MTE.

As an example of a dual monotonic distribution, consider the following distribution in which the access probabilities for the left and right directions are based on Zipf's law. Specifically, the probability of accessing a record R_i (for $i = 1, 2, \ldots, N$) from the left and right are given by

$$s_{iL} = c_1/i$$

$$s_{iR} = \frac{c_2}{N - i + 1}.$$

Note that the constants c_1 and c_2 must be chosen in such a manner that $\sum_{i=1}^{N}(s_{iL} + s_{iR}) = 1$.

COROLLARY 2. *Consider the family of distributions in which the total probability of accessing record* R_i $(1 \leq i \leq N)$ *is given by* p_i. *These probabilities obey the relation* $p_i > p_j$ *for all* $i < j$. *Furthermore, the conditional probability of accessing a record* R_i *from the left is* p *(where* $p > 0.5$*) independent of the index* i. *We call this family of distributions* directionally biased. *Directionally biased distributions are exclusive, and hence, the SWAP is more efficient than MTE.*

Proof. We note that as a result of conditions specified in the statement of the corollary, the probabilities s_{iL} and s_{iR} can be expressed as $p_i p$ and $p_i(1 - p)$, respectively, where p_i represents the total probability of accessing record R_i. For this family of distributions, the condition for their being exclusive, remains to be verified. Indeed, the LHS of condition (5) can be written as

$$\begin{aligned} &p_k(p_i - p_j)p^2 + p_k(p_j - p_i)(1 - p)^2 + p_i p_j\{p(1 - p) - p(1 - p)\} \\ &\quad = p_k(p_i - p_j)\{p^2 - (1 - p)^2\} \\ &\quad = p_k(p_i - p_j)(2p - 1). \end{aligned}$$

It is clear that because of the condition that $p > 0.5$, this family qualifies as an exclusive distribution. The claim immediately follows. □

Note that we have proved the superiority of SWAP relative to MTE, only for exclusive distributions. Since the TR heuristic is known to be better than the MTE for *all* access distributions, we expect SWAP to be always better than MTE. This result is by no means trivial to prove. Whenever $b_{\text{SWAP}}(i, j) \geq b_{\text{MTE}}(i, j)$ can be established for all $i < j$, SWAP has been shown to be the superior heuristic. However, it is an easy task to construct examples of distributions in which $b_{\text{SWAP}}(i, j) < b_{\text{MTE}}(i, j)$ at least for one pair of indices i, j, where $i < j$. Hence to compare SWAP and MTE, we apparently have to deal with the explicit expression for cost itself and not the asymptotic pairwise precedence probabilities. This problem is currently open and stated as a conjecture.

Conjecture 1. The SWAP heuristic has a lower expected cost when compared to the MTE heuristic, for **all** access distributions.

3. Absorbing Schemes for DLLs

In the preceding section, we have examined the memoryless heuristics SWAP and MTE for DLLs. These schemes are representative of the class of ergodic schemes because the Markov chain associated with the list configurations is ergodic (i.e., irreducible and aperiodic). In the equilibrium state, the probability of the list being in one of its $N!$ configurations can be computed as the stationary probability vector of the transition matrix describing the scheme. Thus, even in the limit, the list can be in one of the $N!$ permutations. These schemes are clearly to be preferred if the environment is non-stationary. Thus, even if the distribution of the queries changes with time, the data reorganization scheme is capable of following these changes.

If the environment is known to be stationary, it is better to use an absorbing scheme, which gradually transforms the list until an absorbing state is reached. After this, no further transformations in the list take place. If the probability of converging to the optimal order can be made exactly unity, all subsequent queries will benefit from the lowest possible cost. If this probability cannot be made exactly unity, we would like to use a heuristic which allows us to attain a value that is "sufficiently high." This would ensure that the expected cost is close to the optimal one.

We shall now present an absorbing strategy which moves each data element exactly once. In other words, once an element is moved, it is never moved again. After each of the elements in the list have been moved once, the list is considered to have converged to its "final" configuration.

In the following section, we explain this procedure in detail and derive some of its asymptotic properties.

3.1. *Absorbing-MTE*

We have seen earlier that the optimal configuration of records is one which arranges the records in the descending order of the difference $s_{iL} - s_{iR}$. Maintaining estimates of the difference in access probabilities clearly requires additional space proportional to the number of elements in the list. Instead of using $O(N)$ such counters, our scheme is based on the use of *one* additional counter and a parameter M which determines the accuracy of the scheme.

Let $\mathscr{V}$ be the set of elements that have not been moved from their initial positions. $\mathscr{V}$ is initially set to $\mathscr{R}$ (the set of all records in the list) and is progressively reduced until it is empty. At this point, the data reorganization algorithm is said to have converged, and further reorganizations will not take place with subsequent accesses.

Whenever an element R_i is accessed and $R_i \notin \mathscr{V}$ (i.e., the record R_i has already been moved from its initial position) although the contents of R_i are presented to the user, the data reorganization strategy completely ignores this access. We now consider the case when $R_i \in \mathscr{V}$.

To aid in understanding the scheme, let us imagine that a new "conceptual" list is being formed as we decide the final positions of the individual records of $\mathscr{R}$. For simplicity of exposition we shall consider that the new list being formed consists of two sublists, *Front* and *Tail* to be ultimately concatenated.

Initially *Front* and *Tail* are both empty. If a record R_i is accessed M consecutive times from the *left*, it is appended to the end of the list *Front*. As alluded to above, we note that these M consecutive accesses to R_i could be interleaved by any number of accesses to the elements of $\mathscr{R}$ that have been already moved. Analogously, if the record R_i is accessed M consecutive times from the *right*, we add R_i to the list *Tail* as its *first* element. The intuitive explanation is that elements being accessed very frequently from the left are moved to belong to the sublist *Front* and hence appear at beginning of the reorganized list. On the other hand, elements being accessed more frequently from the right are placed at the head of the list *Tail* and consequently appear near the end of the overall concatenated list. When all the elements of the original list have been moved once, the scheme has converged and we can replace the original list by the Newlist consisting of *Front* $\cup$ *Tail*, where $\cup$ is taken to represent the concatenation operator. Figure 1 illustrates the operation of this algorithm, for an example list consisting of five elements. The details of the sequence in which the elements are moved and the direction in which

Element Accessed M times	Direction	*Front*	*Tail*
R_1	Left	$\langle R_1 \rangle$	$\langle\rangle$
R_5	Right	$\langle R_1 \rangle$	$\langle R_5 \rangle$
R_4	Right	$\langle R_1 \rangle$	$\langle R_4, R_5 \rangle$
R_2	Left	$\langle R_1, R_2 \rangle$	$\langle R_4, R_5 \rangle$
R_3	Left	$\langle R_1, R_2, R_3 \rangle$	$\langle R_4, R_5 \rangle$

FIG. 1. Figure shows how the sublists *Front* and *Tail* are gradually built, as elements of the DLL are moved. The initial order of elements in the DLL is $\langle R_3, R_5, R_1, R_2, R_4 \rangle$. The final ordering is $\langle R_1, R_2, R_3, R_4, R_5 \rangle$.

they are accessed consecutively is also shown in Fig. 1. Observe that the initial ordering of the records is $\langle R_3, R_5, R_1, R_2, R_4 \rangle$ and the final ordering is $\langle R_1, R_2, R_3, R_4, R_5 \rangle$.

We now present the data reorganization strategy formally.

ALGORITHM Absorbing_DLL_1.
Input. A sequence of queries presented by the procedure GetQuery and an integer M
Output. The reorganized DLL.

Memory Requirements. (i) Two memory locations: Index, Count. Index stores the index of the last accessed record. Count stores the number of times the record R_{Index} has been consecutively accessed, if accesses to the elements of $\mathscr{R} - \mathscr{V}$ are ignored.

(ii) Two memory locations "Dir" and "Previous_dir". "Dir" takes on one of the values Left, Right.

Method:

```
𝒱 := ℛ;
Index := 0;          /* Maximum uncertainty */
Count := 0;
Repeat
    GetQuery (R_i, Dir);
    if (R_i ∈ 𝒱) then
        if ((Index = i) and (Dir = Previous_dir)) then
            if (Dir = Left) then
                Count := Count + 1;
            else
                Count := Count − 1;
            endif;
```

```
            else
                if (Index ≠ 0) then
                    Index := Count := 0;
                else
                    Index := i;
                    if (Dir = Left) then
                        Count := 1
                    else Count := −1;
                    endif;
                endif;
            endif;
            if (Count = M) then
                Front := Front ∪ {R_i};
                𝒴 := 𝒴 − {R_i};
            elseif (Count = −M) then
                Tail := {R_i} ∪ Tail;
                𝒴 := 𝒴 − {R_i};
            endif;
        endif;
        Previous_dir := Dir;
    until (𝒴 = { });
    NewList := Front ∪ Tail;
End Algorithm Absorbing_DLL_1
```

Program 1. Absorbing scheme for DLLs.

The explanation given above is only to aid in the conceptual understanding of the algorithm. To implement the scheme as presented in Algorithm Absorbing_DLL_1 would require additional $O(N)$ storage elements to build the "NewList." To avoid the overhead of $O(N)$ storage, we would like to update the list "in place." This modified version of the algorithm will be briefly presented in Section 3.2, after we have presented some of its theoretical properties.

LEMMA 1. *Algorithm Absorbing_DLL_1 is absorbing. Moreover, the total number of list reorganizing operations performed is exactly N.*

Proof. Obvious. □

We now investigate the conditions under which the above scheme converges to the optimal list order. We show that for a family of distributions $\{\mathscr{S}\}$, the above scheme is asymptotically optimal. To aid in the proof, we introduce some additional notation.

The access probability vector will be denoted by $\mathscr{S}$. Let $\mathscr{Y}$ be any subset of $\mathscr{R}$. We define the conditional access probability vector $\mathscr{S}|\mathscr{Y}$ as

the vector of normalized probabilities in which only the quantities corresponding to the elements of the set $\mathscr{Y}$ have non-zero values. That is,

$$\mathscr{S}|\mathscr{Y} = \{s'_{iL}, s'_{iR}\}.$$

where,

$$\begin{aligned} s'_{iL} = s'_{iR} &= 0 && \text{if } R_i \notin \mathscr{Y} \\ s'_{iL} &= \frac{s_{iL}}{\sum_{R_j \in \mathscr{Y}} (s_{jL} + s_{jR})} && \text{if } R_i \in \mathscr{Y} \\ s'_{iR} &= \frac{s_i R}{\sum_{R_j \in \mathscr{Y}} (s_{jL} + s_{jR})} && \text{if } R_i \in \mathscr{Y}. \end{aligned} \tag{12}$$

Observe that $\mathscr{S}$ is exactly equivalent to $\mathscr{S}|\mathscr{R}$. Also, we denote the event "Record R_u precedes R_v" as $R_u \prec R_v$.

We make use of a key result in Markov chains to compute the absorption probabilities. We state this result below.

LEMMA 2. *Let F be the transition matrix of a homogeneous finite Markov chain where F_{ij} is the probability of making the transition from state i to state j. Then, x_{iA}, the probability of being absorbing into an absorbing state A, starting from a transient state i is given by the recursive equation*

$$x_{iA} = F_{iA} + \sum_{j \in \tau} F_{ij} x_{jA}, \tag{13}$$

where τ represents the set of all transient states.

Proof. Can be found in [8]. □

We now state and prove our main result which computes the asymptotic probability of the Algorithm Absorbing_DLL_1 converging to the optimal order.

THEOREM 4. *Set $\mathscr{S}$ be the access probability vector. Then the* total *probability that a record R_u precedes another record R_v in the final list obtained by Algorithm Absorbing_DLL_1 is given by*

$$\Pr(R_u \prec R_v) = \frac{s^M_{uL} + s^M_{vR}}{s^M_{uL} + s^M_{uR} + s^M_{vL} + s^M_{vR}}.$$

Proof. Without loss of generality, we assume that $s_{uL}, s_{uR}, s_{vL}, s_{vR}$, are all non-zero. For the indices $u, v \in \{1, 2, \ldots, N\}$, let $\xi_{uv}(\mathscr{Y})$ be the event that either R_u or R_v is the element which is selected from the set $\mathscr{Y}$ to be moved, where $\mathscr{Y} \subseteq \mathscr{R}$. Also, let $\eta_{uv}(i)$ denote the event that the ith element moved consists either of R_u or R_v. As a result of the data reorganization scheme, we have the simple relation that

$$\Pr(R_u \prec R_v | \eta_{uv}(N - |\mathscr{Y}| + 1)) = \Pr(R_u \prec R_v | \xi_{uv}(\mathscr{Y})). \quad (14)$$

Since $\mathscr{Y}$ is the set of elements which have not been moved to their final positions in the list, the conditional access probabilities are defined by (12). We now introduce the probabilities:

$$p_L = \frac{s_{uL}}{\sum_{R_j \in \mathscr{Y}} (s_{jL} + s_{jR})}, \qquad p_R = \frac{s_{uR}}{\sum_{R_j \in \mathscr{Y}} (s_{jL} + s_{jR})}$$

$$q_L = \frac{s_{vL}}{\sum_{R_j \in \mathscr{Y}} (s_{jL} + s_{jR})}, \qquad q_R = \frac{s_{vR}}{\sum_{R_j \in \mathscr{Y}} (s_{jL} + s_{jR})}.$$

Note that p_L, p_R (q_L, q_R) are the conditional probabilities of accessing R_u (R_v) from the left and right, respectively. We now proceed to compute $\Pr(R_u \prec R_v | \xi_{uv}(\mathscr{Y}))$. In order to compute this quantity, we first define the Markov chain for the data reorganization strategy under the condition that R_u or R_v is the record to be selected for the move to its final position. The Markov chain is in state $\phi(n)$ at time instant n, where

$$\phi(n) = \begin{cases} U_i & \text{if } z_1 = u,\ z_2 = i \\ U_{-i} & \text{if } z_1 = u,\ z_2 = -i \\ V_i & \text{if } z_1 = v,\ z_2 = i \\ V_{-i} & \text{if } z_1 = v,\ z_2 = -i \\ \theta_0 & \text{if } z_1 = 0,\ z_2 = 0 \\ \theta_t & \text{otherwise.} \end{cases}$$

Clearly the Markov chain represents the contents of the memory locations z_1, z_2; θ_0 represents the state of maximum uncertainty in which there is insufficient information about which of the records R_u or R_v should be moved first. Also θ_t represents the state in which the elements of $\mathscr{Y} - \{R_u, R_v\}$ have been accessed at least once (but not M consecutive times).

The transition probabilities of the Markov chain are given by the stochastic matrix F given below ($1 \leq i \leq M-1$):

$$\begin{aligned}
F_{\theta_0 U_i} = F_{U_i U_{i+1}} &= p_L, & F_{U_i \theta_0} &= 1 - p_L \\
F_{\theta_0 U_{-i}} = F_{U_{-i} U_{-(i+1)}} &= p_R, & F_{U_{-i} \theta_0} &= 1 - p_R \\
F_{\theta_0 V_i} = F_{V_i V_{i+1}} &= q_L, & F_{V_i \theta_0} &= 1 - q_L \\
F_{\theta_0 V_{-i}} = F_{V_{-i} V_{-(i+1)}} &= q_R, & F_{V_{-i} \theta_0} &= 1 - q_R \\
F_{\theta_0 \theta_t} &= 1 - p_L - p_R - q_L - q_R \\
F_{\theta_t \theta_0} &= 1 \qquad (\theta_t \text{ is a reflecting barrier}) \\
F_{U_M U_M} &= 1, & F_{U_{-M} U_{-M}} &= 1 \\
F_{V_M V_M} &= 1, & F_{V_{-M} V_{-M}} &= 1.
\end{aligned}$$

The Markov chain is shown pictorially in Fig. 2. We define the following absorption probabilities, for values of i in the set $\{-M, -(M-1), \ldots, 1, 2, \ldots, M-1, M\}$:

$$\begin{aligned}
a_i &= \Pr(\phi(\infty) = U_M | \phi(0) = U_i; \xi_{uv}(\mathscr{Y})) \\
b_i &= \Pr(\phi(\infty) = V_{-M} | \phi(0) = U_i; \xi_{uv}(\mathscr{Y})) \\
c_i &= \Pr(\phi(\infty) = U_M | \phi(0) = V_i; \xi_{uv}(\mathscr{Y})) \\
d_i &= \Pr(\phi(\infty) = V_{-M} | \phi(0) = V_i; \xi_{uv}(\mathscr{Y})).
\end{aligned}$$

Note that a_i denotes the conditional probability of getting absorbed into the state U_M given that we start in the state U_i (and $\xi_{uv}(\mathscr{Y})$). The quantities b_i, c_i, and d_i can be given a similar interpretation. Convergence to either U_M or V_{-M} indicates that the record R_u will appear before R_v in the final list. Also note that the following boundary conditions are obeyed by a_i, b_i, c_i, d_i:

$$\begin{aligned}
a_M &= d_{-M} = 1 \\
a_{-M} &= b_M = b_{-M} = c_M = c_{-M} = d_M = 1.
\end{aligned} \tag{15}$$

Since the algorithm starts off in the state θ_0, we are interested in computing the probability of reaching one of the states U_M or V_{-M}. We define the following quantities:

$$\begin{aligned}
\Delta_{0u} &= \Pr(\phi(\infty) = U_M | \phi(0) = \theta_0; \xi_{uv}(\mathscr{Y})) \\
\Delta_{0v} &= \Pr(\phi(\infty) = V_{-M} | \phi(0) = \theta_0; \xi_{uv}(\mathscr{Y})).
\end{aligned}$$

Clearly, $\Delta_0 = \Delta_{0u} + \Delta_{0v}$ is the quantity we wish to compute. Now, using

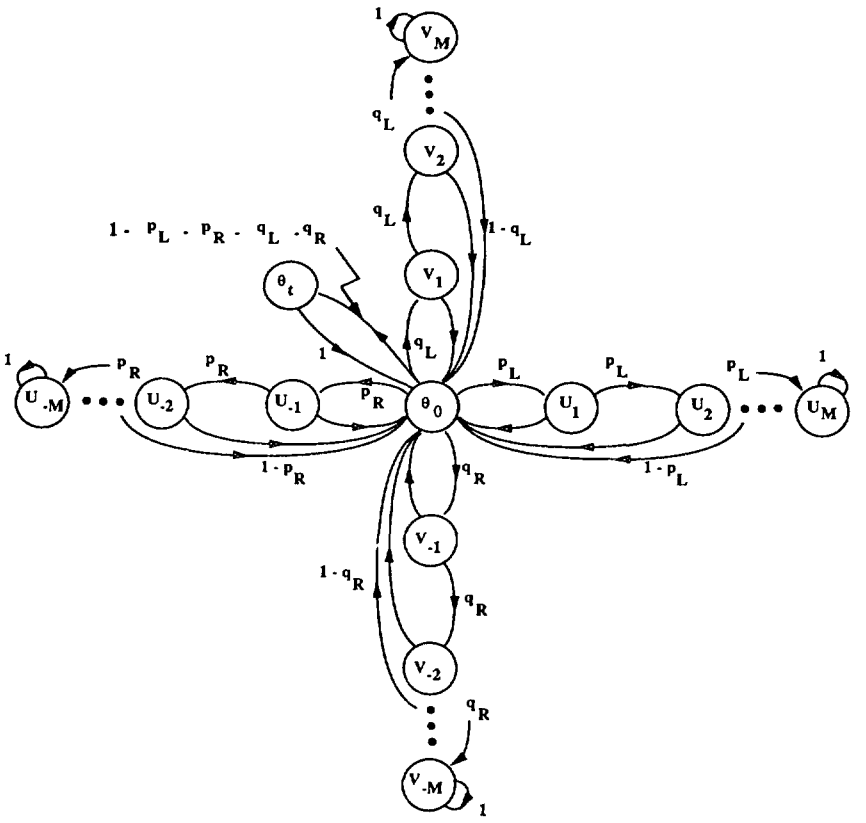

FIG. 2. The figure depicts the absorbing Markov chain used to model the operation of Algorithm Absorbing_DLL_1.

Lemma 2, the recurrence relations obeyed by a_i, b_i, c_i, and d_i can be written as ($1 \leq i \leq M-1$; $1 \leq -j \leq M-1$):

$$a_i = p_L a_{i+1} + (1 - p_L)\Delta_{0u}, \qquad a_j = p_R a_{j-1} + (1 - p_R)\Delta_{0u}$$

$$b_i = p_L b_{i+1} + (1 - p_L)\Delta_{0v}, \qquad b_j = p_R b_{j-1} + (1 - p_R)\Delta_{0v}$$

$$c_i = q_L c_{i+1} + (1 - q_L)\Delta_{0u}, \qquad c_j = q_R c_{j-1} + (1 - q_R)\Delta_{0u}$$

$$d_i = q_L d_{i+1} + (1 - q_L)\Delta_{0v}, \qquad d_j = q_R a_{j-1} + (1 - q_R)\Delta_{0v}. \quad (16)$$

Solving (16) subject to the boundary conditions given by (15), we obtain

the following expressions for a_i, b_i, c_i, d_i for $i = \pm 1$:

$$\begin{aligned} a_1 &= p_L^{M-1}(1 - \Delta_{0u}) + \Delta_{0u}, & a_{-1} &= \Delta_{0u}(1 - p_R^{M-1}) \\ b_1 &= \Delta_{0v}(1 - p_L^{M-1}), & b_{-1} &= \Delta_{0v}(1 - p_R^{M-1}) \\ c_1 &= \Delta_{0u}(1 - q_L^{M-1}), & c_{-1} &= \Delta_{0v}(1 - q_R^{M-1}) \\ d_1 &= \Delta_{0v}(1 - q_L^{M-1}), & d_{-1} &= q_R^{M-1}(1 - \Delta_{0v}) + \Delta_{0v}. \end{aligned} \quad (17)$$

Thus, Δ_0 can be written as

$$\begin{aligned} \Delta_0 = {} & p_L a_1 + p_R a_{-1} + q_L c_1 + q_R c_{-1} + p_L b_1 + p_R b_{-1} + q_L d_1 + q_R d_{-1} \\ & + (1 - p_L - p_R - q_L - q_R)\Delta_0. \end{aligned} \quad (18)$$

Substituting the expressions in (17) into (18) and simplifying, we obtain

$$\Delta_0 = \frac{p_L^M + q_R^M}{p_L^M + p_R^M + q_L^M + q_R^M}. \quad (19)$$

Since p_L, p_R, q_L, q_R have the same denominator, we can rewrite (19) as

$$\Pr(R_u \prec R_v | \xi_{uv}(\mathcal{Y})) = \frac{s_{uL}^M + s_{vR}^M}{s_{uL}^M + s_{uR}^M + s_{vL}^M + s_{vR}^M}. \quad (20)$$

Combining (20) with (14), we have proved that for all $1 \le i \le M - 1$,

$$\Pr(R_u \prec R_v | \eta_{uv}(i)) = \frac{s_{uL}^M + s_{vR}^M}{s_{uL}^M + s_{uR}^M + s_{vL}^M + s_{vR}^M}. \quad (21)$$

Because of the fact that the events $\eta_{uv}(i)$ for $1 \le i \le M - 1$ are mutually exclusive and collectively exhaustive, the unconditional (total) probability that the record R_u precedes R_v can be written using the laws of total probability as

$$\Pr(R_u \prec R_v) = \sum_i \Pr(R_u \prec R_v | \eta_{uv}(i)) \Pr(\eta_{uv}(i)).$$

Since $\Pr(R_u \prec R_v | \eta_{uv}(i))$ is independent of i (by (21)), and $\sum_i \Pr(\eta_{uv}(i)) = 1$, we obtain

$$\Pr(R_u \prec R_v) = \Pr(R_u \prec R_v | \eta_{uv}(i)) = \frac{s_{uL}^M + s_{vR}^M}{s_{uL}^M + s_{uR}^M + s_{vL}^M + s_{vR}^M}. \quad (22)$$

This completes the proof. □

We now investigate the conditions under which the probability of the above scheme converging to the optimal order can be made arbitrarily close to 1 (by increasing M, the parameter of the scheme). To study this, observe that it is sufficient to attempt to make $\Pr(R_i \prec R_j)$ tend to 1, as $M \to \infty$. Our next result examines the distributions which force this limit to unity.

THEOREM 5. *Let $\mathscr{S}$ be an access distribution. Then the necessary and sufficient conditions that Algorithm Absorbing_DLL_1 converges to the optimal order with probability* 1 *as* $M \to \infty$ *is*

$$\max(s_{iL}, s_{jR}) > \max(s_{jL}, s_{iR}) \qquad \forall i < j. \tag{23}$$

Proof. We first prove that the conditions stated in (23) are sufficient to ensure that Algorithm Absorbing_DLL_1 converges to the optimal order with a probability of unity. By Theorem 4, the probability of a record R_i preceding another record R_j is given by

$$\Pr(R_i \prec R_j) = \frac{s_{iL}^M + s_{jR}^M}{s_{iL}^M + s_{iR}^M + s_{jL}^M + s_{jR}^M}.$$

The expression for $\Pr(R_i \prec R_j)$ is of the form

$$\Pr(R_i \prec R_j) = \frac{1}{1+\delta}, \qquad \text{where} \quad \delta = \frac{s_{jL}^M + s_{iR}^M}{s_{iL}^M + s_{jR}^M}.$$

In the following proof, we assume for convenience that s_{iL} and s_{jR} are distinct, implying that there is a unique maximum. The same assumption will be made with regard to the access probabilities s_{jL} and s_{iR}. The case when $s_{iL} = s_{jR}$ and/or $s_{jL} = s_{iR}$ can be treated very similarly, without changing the conclusion. Now,

$$\delta = \frac{s_{jL}^M + s_{iR}^M}{s_{iL}^M + s_{jR}^M} = \frac{\max\left(s_{jL}^M, s_{iR}^M\right)}{\max\left(s_{iL}^M, s_{jR}^M\right)} \cdot \frac{1 + \min\left(s_{jL}^M, s_{iR}^M\right)/\max\left(s_{jL}^M, s_{iR}^M\right)}{1 + \min\left(s_{iL}^M, s_{jR}^M\right)/\max\left(s_{iL}^M, s_{jR}^M\right)}.$$

Clearly,

$$\lim_{M\to\infty} \frac{\min\left(s_{jL}^M, s_{iR}^M\right)}{\max\left(s_{jL}^M, s_{iR}^M\right)} = 0 \qquad \text{and} \qquad \lim_{M\to\infty} \frac{\min\left(s_{iL}^M, s_{jR}^M\right)}{\max\left(s_{iL}^M, s_{jR}^M\right)} = 0$$

and hence,

$$\lim_{M\to\infty} \delta = \lim_{M\to\infty} \frac{\max\left(s_{jL}^M, s_{iR}^M\right)}{\max\left(s_{iL}^M, s_{jR}^M\right)} = \lim_{M\to\infty} \left(\frac{\max(s_{jL}, s_{iR})}{\max(s_{iL}, s_{jR})}\right)^M = 0.$$

Hence δ tends to 0 as $M \to \infty$, and consequently, $\Pr(R_i \prec R_j)$ tends to 1 as $M \to \infty$. Note that because of the input conditions, this property holds for all $i < j$.

Let P^* be the probability of converging to the optimal order. Then,

$$P^* = \Pr\left(\bigcap_{i<j} (R_i \prec R_j) \right)$$
$$\geq \prod_{i<j} \Pr(R_i \prec R_j), \tag{24}$$

Since we have already established that each of the terms of the RHS of (24) tend to 1 as M tends to ∞, $P^* \to 1$ as $M \to \infty$. The fact that (23) is necessary, can be proved along similar lines. The proof is omitted for the sake of brevity. □

It is useful to point out that the necessary and sufficient condition specified in (23) is not merely of theoretical interest. Indeed, various families of distributions satisfy this condition. In fact, we shall show that the families of distributions presented in Corollary 1 and Corollary 2 indeed satisfy condition (23).

COROLLARY 3. *Let $\mathcal{S}$ be a dual monotonic access distribution obeying the conditions of Corollary* 1. *Then, the probability of Algorithm Absorbing_DLL_*1 *converging to the optimal order approaches* 1 *as* $M \to \infty$.

Proof. Because of the conditions specified in Corollary 1 we have the conditions $s_{iL} > s_{jL}$ and $s_{jR} > s_{iR}$, whenever $i < j$. Combining both these inequalities, we obtain $\max(s_{iL}, s_{jR}) > \max(s_{jR}, s_{iR})$, whenever $i < j$. This information, together with Theorem 5, establishes our claim. □

We now state our final result.

COROLLARY 4. *Let $\mathcal{S}$ be a directionally biased distribution, as defined in Corollary* 2. *Then for this distribution, Absorbing_DLL_*1 *is asymptotically optimal.*

Proof. The input conditions are: $p_i > p_j$ for all $i < j$ and that the conditional probability of accessing any record from the left is $p > 0.5$. Since $p > 0.5$, it is clear that $p > p_j/(p_i + p_j)$; as a result we conclude that

$$\max(s_{iL}, s_{jR}) = \max\big(p_i p, p_j(1-p)\big) = p_i p. \tag{25}$$

We note that $\max(s_{jL}, s_{iR})$ can be $p_j \cdot p$ or $p_i(1-p)$. In the first case, the term in (25) is greater, since $p_i > p_j$. In the second case, the difference between $p_i p$ and $p_i(1-p)$ is $p_i(2p-1)$ which is greater than 0, because

$p > 0.5$. Hence, in either case, the conditions of Theorem 5 are met and the required result follows. □

Remarks. 1. The conditions imposed by Corollary 4 are rather strict. In fact it is easy to show that Absorbing_DLL_1 is optimal for a family of distributions for which the following (weaker) conditions are satisfied: $s_{iL} > s_{jL}$ $\forall i < j$ and $s_{iL} > s_{iR}$, $1 \leq i \leq N$.

2. In employing the above absorbing DLL reorganizing heuristic in a "real" situation, the choice of the parameter M is crucial. If the chosen value of M is too small, the scheme may converge to a "bad" configuration very quickly and remain in that state forever. On the other hand, an unusually large value of the parameter would cause data reorganizations to become rather infrequent and the access costs to remain high for a very long time. This is identical to the situation encountered in the design of ε-optimal fixed structure learning automata [13]. In all such cases the automaton can be made as close to optimal as desired by increasing the memory capabilities of the machine. In this case, the designer is left with a speed–accuracy dilemma. Our recommendation is that, just as in the case of ε-optimal learning automata, the designer reconcile himself to a degree of accuracy $\Pr(R_i \prec R_j)$ (if $s_{iL} + s_{jR} > s_{jL} + s_{iR}$) greater than $1 - \varepsilon$. By using crude (bounded) estimates of the quantities $s_{iL}, s_{iR}, s_{jL}, s_{jR}$, Eq. (22) can be used to determine the *smallest* value of M. Clearly this will lead to the fastest convergence for a prespecified degree of accuracy. Note that it estimates of the above quantities are available, the knowledge of the transition probabilities in conjunction with M can be used to determine the mean time of convergence of *the* Markov chain introduced in the proof of Theorem 4.

3. A natural way to enhance the retrieval characteristic is to employ a combination of the above absorbing scheme and an ergodic schemes such as SWAP (or MTE).[3] In this hybrid scheme, the set of elements yet to be moved would be reorganized using the chosen ergodic heuristic. This has the effect of lowering the access costs in the interim period during which the scheme is waiting for its absorbing component to converge.

4. It is interesting to compare the performance of this absorbing scheme with ergodic schemes obtained as directed mappings of the k-in-a-row SLL heuristics suggested by Gonnet *et al.* [4]. Rather than compare the two directly we shall do so using the intermediate results of Kan and Ross [9]. In [9] Kan and Ross describe an elegant scheme to deal with the entire family of k-in-a-row heuristics. They show that the input distribution $\mathscr{P}$ of a k-in-a-row scheme can be suitably transformed to yield a new

[3]This was pointed out to us by an anonymous referee, but the basic approach has been earlier documented for absorbing and hybrid SLLs by Oommen and Ng [15].

distribution $\mathscr{P}'$ responding to the corresponding heuristic in which $k = 1$. The performance of the k-in-a-row heuristic can now be quite simply derived by using the hypothetical distribution $\mathscr{P}'$ in conjunction with the basic reorganization heuristic. A corollary of this result in the DLL domain tells us that the analysis of the k-in-a-row MTE can be accomplished by first obtaining the new distribution $\mathscr{P}'$ and then applying this distribution to the known results for the MTE heuristic. While the k-in-a-row MTF scheme for SLLs becomes asymptotically optimal as $k \to \infty$, it is not clear if a similar claim can be made about either the MTE or SWAP heuristics. The basic difference between these schemes and the one described above is that ours is an absorbing scheme and so the entire *modus operandus* of proving optimality has to be drastically different. We note in passing that absorbing schemes have been shown superior to ergodic schemes in far more general settings when the environment is stationary [13].

3.2. *Implementation Considerations*

In Section 3.1 we had presented Algorithm Absorbing_DLL_1 in a manner that simplified the exposition. The only drawback of implementing the algorithm verbatim from the pseudocode presented is that it necessitates the maintenance of two auxiliary lists *Front* and *Tail* whose total length gradually grows to N, the size of the original list. This was explicitly clarified in Section 3.1.

There are two reasons for not implementing the procedure as described. The first is that it requires $O(N)$ storage for maintaining the auxiliary lists. The second, and more important, reason is that for the period in which the ultimate positions of only a few records have been decided, subsequent queries do not benefit from the (possibly) improved placement of these records. Thus an "in place" update strategy is desirable, and the following strategy answers both these concerns.

The simple modification to Algorithm Absorbing_DLL_1 is effected by maintaining the sublist *Front* to be the initial part of the list (viewed from the left) and making the sublist *Tail* overlap with the last portion of the DLL (see Fig. 3). In Fig. 3, note that the pointer *EndOfFront* locates the end of the sublist *Front*; it actually points to the element which follows the last element in the sublist *Front*. The pointer *StartOfTail* points to the head of the sublist *Tail*. The elements of the DLL between the positions *EndOfFront* and *StartOfTail* are those elements whose final positions have not been determined.

Whenever Algorithm Absorbing_DLL_1 decides that an element is to be appended to the sublist *Front*, it is merely removed from its current position and inserted the position indicated by *EndOfFront*. Analogously,

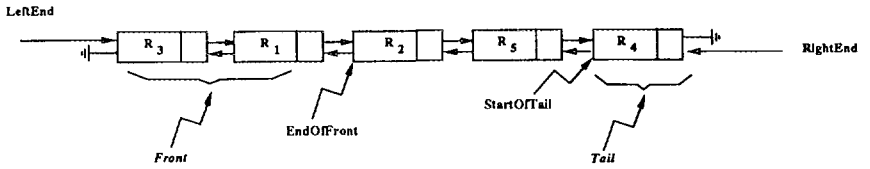

FIG. 3. A DLL being reorganized.

the goal of adding an element as the head of the sublist *Tail* is achieved by merely inserting the element just before the pointer *StartOfTail*. Note that in our original Algorithm Absorbing_DLL_1 we needed to determine if the record being accessed belongs to the set $\mathcal{Y}$ of already moved records. In the modified form of Algorithm Absorbing_DLL_1, it is clear that all the elements of the DLL between its head and *EndOfFront*, as well as the elements encountered between *StartOfTail* and the end of the DLL, belong to the set of already moved records. Thus the set $\mathcal{Y}$ is implicitly maintained.

A formal description of the above procedure and helpful schematics can be found in [20] and are omitted here in the interest of brevity. It is easy to modify the above procedure so as to implement a hybrid heuristic which combines the absorbing scheme presented above with an ergodic heuristic such as SWAP or MTE.

4. CONCLUSIONS

In this paper we have studied the problem of dynamically reorganizing the elements of a DLL. We have compared two DLL heuristics, namely the MTE and the SWAP. These heuristics (MTE and SWAP) can be derived as generalizations from the well-known SLL heuristics MTF and TR, by the application of directed mappings. Our comparison of these heuristics is based on expected cost analysis. A comparison of the same two heuristics in the amortized sense is currently under investigation.

Since the TR heuristic is superior to the MTF heuristic for SLLs, it is natural to believe that SWAP should be more efficient than MTE for all access distributions. A general proof of this conjecture remains open. In this paper we have identified a sufficient condition for one DLL heuristic to yield a lower expected cost than another. This criterion merely examines the pairwise record precedence probabilities for the two heuristics under consideration. Since SWAP and MTE are the heuristics being compared, we have identified families of distributions for which this sufficient condition is satisfied, and consequently for all these distribu-

tions, SWAP is asymptotically more efficient than MTE. We have demonstrated that it is relatively easy to construct examples of distributions for which SWAP is the superior rule. A general proof for all families of distributions would necessarily involve the explicit expression for the cost of the overall list.

We have also proposed an absorbing scheme for reorganizing the elements of a DLL. The proposed scheme utilizes a single parameter, denoted as M. In this scheme, each data element is moved into its final position in the reorganized list if it accessed for M consecutive times. Again, it has been shown that for two rather general families of distributions, the probability of our scheme converging to the optimal order can be made as close to unity as desired.

We note in passing that the family of distributions for which algorithm Absorbing_DLL_1 is shown to be asymptotically optimal, happens to be a superset of the family of distributions for which SWAP has been shown to be more efficient than MTE.

REFERENCES

1. J. L. BENTLEY AND C. MCGEOCH, Amortized analyses of self-organizing sequential search heuristics, *Comm. ACM* **28**, No. 4 (1985), 404–411.
2. J. R. BITNER, Heuristics that dynamically organize data structures, *SIAM J. Comput.* **8**, No. 1 (1979), 82–110.
3. T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, "Introduction to Algorithms," McGraw–Hill, New York, 1990.
4. G. H. GONNET, J. I. MUNRO, AND H. SUWANDA, The exegesis of self-organizing linear search, *SIAM J. Comput.* **10**, No. 1 (1981), 613–637.
5. W. J. HENDRICKS, An account of self-organizing systems, *SIAM J. Comput.* **5**, No. 4 (1976), 715–723.
6. J. H. HESTER AND D. S. HIRSCHBERG, Self-organizing linear search, *ACM Comput. Surveys* **17**, No. 3 (1985), 295–311.
7. E. HOROWITZ AND S. SAHNI, "Fundamentals of Data Structures," Computer Sci., Rockville, MD, 1980.
8. D. L. ISAACSON AND R. W. MADSEN, "Markov Chains: Theory and Applications," Wiley, New York, 1976.
9. Y. C. KAN AND S. M. ROSS, Optimal list order under partial memory constraints," *J. Appl. Probab.* **17**, No. 3 (1980), 1004–1015.
10. D. E. KNUTH, "The Art Of Computer Programming. Vol. 1. Fundamental Algorithms," Addison–Wesley, Reading, MA, 1973.
11. D. E. KNUTH, "The Art Of Computer Programming. Vol. 3. Sorting and Searching," Addison–Wesley, Reading, MA, 1973.
12. D. MATTHEWS, D. ROTEM, AND E. BRETHOLZ, Self-organizing doubly-linked lists, *J. Comput. Math. Sect. A* **8** (1980), 99–106.
13. K. S. NARENDRA AND M. A. L. THATHACHAR,, "Learning Automata: An Introduction," Prentice–Hall, Englewood Cliffs, NJ, 1989.

14. D. T. H. NG AND B. J. OOMMEN, Generalizing singly-linked reorganizing heuristics for doubly-linked lists, *in*, "Proceedings, Conf. on Mathematical Foundations of Computer Science," Rytro, Poland, 1989, pp. 380–389.
15. B. J. OOMMEN AND D. T. H. NG, "Ideal List Organization for Stationary Environments," Technical Report SCS-TR-154, School Of Computer Science, Carlteon University, Ottawa, Canada K1S 5B6.
16. R. L. RIVEST, On self-organizing sequential search heuristics, *Comm. ACM* **19**, No. 2 (1976), 63–67.
17. S. M. ROSS, "Introduction to Probability Models," 2nd ed., Academic Press, New York/London, 1972.
18. D. D. SLEATOR AND R. E. TARJAN, Amortized efficiency of list update and paging rules, *Comm. ACM* **28**, No. 2 (1985), 202–208.
19. J. P. TREMBLAY AND P. G. SORENSON, "An Introduction to Data Structures with Applications," McGraw-Hill, New York, 1976.
20. R. S. VALIVETI AND B. J. OOMMEN. "Self-Organizing Doubly-Linked Lists," Technical Report SCS-TR-173, School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6, May 1990.