ADAPTIVE LINEAR LIST REORGANIZATION UNDER A GENERALIZED QUERY SYSTEM

R.S. Valiveti, B.J. Oommen
and J.R. Zgierski

SCS-TR-181, OCTOBER 1990

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

Adaptive Linear List Reorganization Under A Generalized Query System

R. S. Valiveti, B. J. Oommen, and J. R. Zgierski *
School of Computer Science
Carleton University
Ottawa, Ont, K1S 5B6
Canada

Abstract

The problem of reorganizing a linear list, when the individual records are accessed independently, has been well studied. In this paper, self-organizing linear list heuristics are examined under a more general query system which allows accesses to any subset of the list of elements. We propose a pragmatic model for the query generator, characterized by a set of parameters of size equal to the number of elements in the list. We derive the distribution of accesses to the individual records of the list, and show that these records are statistically dependent. Throughout this paper, the set accesses are processed by serializing the set elements.

We then present extensions to the classical Move-To-Front (MTF) and Transposition (TR) rules under this generalized query generation mechanism. The resulting heuristics are termed MTF_TQS and TR_TQS respectively. Under this query generation model, the optimal (static) list is shown to be one in which the elements are ordered in the descending order of the total probability of accessing the records. The expected cost under the MTF_TQS heuristic is shown to be no more than $\pi/2$ times the mean access cost for the optimal list. We also prove that MTF_TQS and TR_TQS are superior to the simpler reorganization scheme in which the classical MTF or TR heuristics (respectively) is employed in conjunction with the (serial) stream consisting of individual record accesses. Experimental results for these heuristics are also reported.

*The work of the first author was supported by a postgraduate award from the Natural Sciences and Engineering Research Council (NSERC) of Canada, as well as a postgraduate award, from Bell-Northern Research. The work of the second author was supported by NSERC of Canada. The work of the third author was supported by an NSERC summer grant.

1 Introduction

The problem of dynamically reorganizing a linear list with a view to minimize the average access cost has been well studied. In all the studies done to date, the situation analyzed can be described as follows. Let $\mathcal{R} = \{R_1, R_2, \cdots, R_N\}$ be a set of N distinct elements from which a linear list L is constructed. The elements of $\mathcal{R}$ are independently accessed, with a time invariant access distribution governed by the access probability vector $\mathcal{S} = [s_1, s_2, \cdots, s_N]$. In this vector, the component s_i represents the probability of accessing the record R_i. Note that the components of $\mathcal{S}$ satisfy the condition $\sum_{i=1}^{N} s_i = 1$. If the access probability vector $\mathcal{S}$ is known *a priori*, the list can be trivially organized in the optimal order, which corresponds to arranging the elements in the descending order of the access probabilities $\{s_i\}$ [11, 16].

In the absence of this knowledge about the access probability vector, the intention motivating the research has been towards that of maintaining the list in a nearly optimal form. Of course, if we are allowed to estimate the access probabilities, it is a simple matter to attain the optimal order, because after a sufficiently large number of queries have been processed the estimates will converge to the true values. We therefore focus on methods that do not directly estimate the access probabilities and which consequently require no additional storage or a reasonable amount of additional memory. Stated in very general terms, these list reorganizing schemes are based on various heuristics which alter the ordering of elements of the list with each consecutive access. Two broad categories of reorganization strategies can be envisioned: ergodic, and absorbing, and their salient features will be clarified in a subsequent section.

The fundamental assumption made in all the research conducted to date is that the queries have been composed of **single** elements of the list. In this paper, we wish to relax this assumption and study the implications of a generalized query system. Specifically, we shall allow queries made against the system to consist of accesses to *any subset* of the elements of $\mathcal{R}$. These generalized queries are termed "Multiple Access Queries" (MAQ) and the underlying query generating mechanism will be referred to as the Multiple Access Query Generator (MAQG).

As a motivation to studying MAQs, consider the following scenario where such queries are encountered. We focus on a distributed environment with multiple users, where each user may initiate an access to the elements of a shared data structure stored on a centrally located data storage facility (typically the file server). The data structure being maintained at the file server, could be any one of the standard forms of organizing data, including a Linear list, the Doubly-Linked List, A Binary Search Tree etc., or any other structure suitable to

the applications. As a consequence of the fact that queries can originate at various sites, it is natural to expect that at any given instant, any subset of the data structure may be requested. In such a case, the file server would accumulate all the requests that occur in a "time unit", search for these elements, and subsequently report the relevant information to the respective sites. The problem investigated in this paper is one of determining the optimal way in which the data structure can be maintained when queries of this type are encountered.

The scenario of processing MAQs is not just encountered in distributed computing. Indeed, every database system uses a front-end which facilitates the data retrieval process and this serves as an interface between the user and the physical data. For most queries the response that is received is a table (relation) which indeed represents the sets of elements that satisfy the query. Thus, if the question was one of organizing the data structure which contains the actual physical data, the "systems manager" would have to arrange this physical data in such a manner that the average access time is minimized — the averaging being considered over all the possible sets which can be emitted as the response.

Having introduced the concept of MAQs and briefly motivated their occurrences and applications, it is not inappropriate to consider the aspect of specifying the distribution of queries. A straightforward method to model the Multiple Access Query Generator (MAQG) comes easily to mind. To fully describe the MAQG it is evident that, we can specify a vector akin to $\mathcal{S}$, whose components comprise of the probabilities of accessing each of the possible 2^N subsets of $\mathcal{R}$. More formally, if ξ denotes an MAQ, then the MAQG is fully defined by completely specifying the probability $Pr(\xi)$ for all $\xi \subseteq \mathcal{R}$, where $\sum_{\xi} Pr(\xi) = 1$. It is obvious that this model of query generation subsumes the previous model in which only single elements are accessed.

There is one potential difficulty with specifying the MAQG by resorting to enumerating the access probabilities for all possible subsets of $\mathcal{R}$. In order to specify the *total* query probabilities $Pr(\xi)$ one would need to specify a vector of size 2^N, where N is the number of elements in $\mathcal{R}$. Although this method of describing the MAQG is both complete and exact (i.e., it is not an approximation) this is obviously unrealistic in terms of both time and space consumption. Ideally, we would like to have a simpler (albeit approximated) characterization of the query generation process, based on fewer parameters. We now present our model called the P-model, for MAQGs.

1.1 The P-model for MAQs

The model we propose in this paper, will be called the P-model (the Parametrized model for the MAQG), and is characterized by exactly N parameters forming the vector $\mathcal{P} = [p_1, p_2, \cdots, p_N]$. In the P-model for the MAQG, we assume that the records comprising a single query are statistically independent. Moreover, the queries at different time instants are also statistically independent. Finally, in any query ξ, we assume that the probability of a specific record R_i being requested is simply p_i. Notice that unlike the components of the S-vector used in the traditional single-access query generating mechanisms, the components of $\mathcal{P}$ need not sum to unity.

Using the above model it is easy to see that any arbitrary query $\xi \subseteq \mathcal{R}$ can be fully described in terms of a bit-vector $\mathcal{B} = [b_1, b_2, \cdots, b_N]$, where *each* component $b_i \in \{0, 1\}$ is a binary random variable obeying a Bernoulli distribution. Thus each bit b_i assuming a value in $\{0, 1\}$ signifies whether the record element R_i is included or not included in the query ξ. Throughout this paper we shall make use of the obvious convention that $b_i = 1$ implies that R_i has been included in the query. Since the queries are assumed to be generated in an independent manner, this assertion reflects back to imply that the elements of $\mathcal{B} = [b_1, b_2, \cdots, b_N]$ are chosen independently based on the distribution $\mathcal{P}$, and hence the probability of any particular query ξ being processed can be written down as:

$$Pr(\xi) = \prod_{i=1}^{n} p_i^{b_i}(1 - p_i)^{1-b_i} \tag{1}$$

To clarify the P-model we consider an example for the case when $N = 3$, and when $\mathcal{R} = \{R_1, R_2, R_3\}$. Then the set of possible queries is:

$$\{\{\}, \{R_1\}, \{R_2\}, \{R_3\}, \{R_1, R_2\}, \{R_1, R_3\}, \{R_2, R_3\}, \{R_1, R_2, R_3\}\}.$$

Thus an application of (1) suggests that under the P-model the probability of query $\xi = \{R_1, R_3\}$ occurring is exactly $p_1(1-p_2)p_3$. Similar expressions can be written down for other MAQs.

Having completely defined the P-model for MAQGs, we now consider the aspect of adaptively reorganizing a structure, when set queries of the type described above are encountered.

1.2 Data Reorganization Under MAQs

To fully comprehend data reorganization in the presence of generalized queries permitted by MAQ, it is convenient to draw some parallels from the former (single element access) query generation model. Observe that in this case, for the case of the linear list, the optimal

order corresponds to the descending order of the access probabilities $\{s_i\}$. In the absence of this information, the most obvious way to approach the problem is to estimate the unknown parameters. After processing a sufficiently large number of queries, these estimates of s_i's will be very "close" to the *true* values and thus, with a high probability, the list can be arranged in the optimal order.

Coming now to the situation of MAQs, the question of obtaining an optimal ordering is unclear even if we are allowed to estimate a finite number of parameters from the "infinite" stream of queries. Furthermore, it is not even clear which parameters will serve as sufficient statistics in the estimation strategy. Indeed this brings us to the first result of our paper, for in Theorem 1 we establish that the optimal (static) list can be obtained by arranging the list in the descending order of the p_i's.

Since methods based on estimating the unknown parameters are generally discouraged in the literature, we next examine the possibility of employing data reorganizing heuristics in the context of MAQs. Let us first begin by briefly presenting some of the known ergodic heuristics operating under the simpler model of query generation. The general characteristic of ergodic heuristics is that they continuously transform the list with each record access, and even in the limit, the list can be in any one of the possible $N!$ possible configurations. Two well known memoryless (ergodic) heuristics are the Move-to-Front (MTF) and Transposition (TR). Under the MTF heuristic, the accessed element is moved to the front of the list. The TR heuristic, on the other hand, transposes the accessed element with the record that immediately precedes it. In both these heuristics, no reorganization is done if the accessed element happens to be at the front of the list. Many other ergodic heuristics are known. For a description, as well as detailed analyses, we refer the reader to [2, 3, 6, 9, 16, 17]. Similar strategies for the doubly-linked list [13, 19], and the binary search trees [1, 4] are also available, but in the interest of brevity we shall not discuss these heuristics. For examples of absorbing heuristics for singly-linked lists, we refer the reader to [14, 15].

In this paper, we shall propose two new heuristics for MAQs derived from the MTF and TR heuristics. These modified heuristics will be called MTF_TQS and TR_TQS respectively. To describe their *modus operandus* we begin by examining the query processing phase. This study naturally leads to an understanding of what constitutes the optimal order for the linear list. Furthermore, it is shown that the expected access cost under the MTF_TQS heuristic is no more than $\pi/2$ times the optimal cost. This is a natural generalization of the corresponding result under the simpler model of query generation. Additionally, we show that MTF_TQS and TR_TQS are superior to the heuristics which merely employ the classical

MTF and TR heuristics in conjunction with the stream consisting of requested records, and ignore the fact that the queries occur in sets. Thus, we claim that there considerable latent information in the set boundaries, and in this perspective, our results represent a significant improvement from the traditional schemes.

We shall now proceed to consider various possible responses to queries generated by the MAQG.

2 Processing MAQs

Once a query ξ is presented, the system must search for the elements in ξ and report them to the user. We propose that this be done by choosing an element from the query at random and performing a linear search on the list starting from the front. The access cost of the element is exactly equal to the position of the element in the list, and it is also equal to the number of comparisons done (between the value of the query element and that of the list element) in order to find the element. Thus it will take one time unit to find an element which is at the head of a list, as opposed to N time units for the element which is at the tail of the list. Once the element is found, the next (as yet unsearched for) element in ξ is chosen randomly from the query and the search begins anew from the head of the list. The access cost associated with the query ξ is thus the sum of the access costs of all the elements in ξ.

An alternate strategy for processing could involve scanning the list from head to tail and comparing each element in the list with elements in the query until all the elements in ξ are located. The access cost associated with such a processing mechanism would now be quite different from the type of processing described above. Indeed, this cost would depend on what operations are permitted on the *query itself* (i.e. if we are permitted to sort the query etc.), and since this is a far more general problem we shall not discuss this aspect in the current work. The problem of reorganizing both the list and the query currently remains open.

Under the former model of query processing and the previously described model (the P-model) of query generation we shall now derive a fundamental result involving the optimal ordering of a linear list.

Theorem 1 *Let $\mathcal{R} = \{R_1, R_2, ..., R_N\}$ be a set of elements, and $\mathcal{P}$ be the parameter for the P-model of the MAQG. Then if the query elements are processed sequentially, the optimal static list is obtained by maintaining the elements of $\mathcal{R}$ in the descending order of the components of $\mathcal{P}$.*

Proof:

In terms of notation let Π be any arbitrary permutation of elements in $\mathcal{R}$. Also let $\Pi(i)$ be the element in the i^{th} position in the list, and $\Pi^{-1}(i)$ represent the position of element R_i in Π. The expected cost of accessing this list can be computed as the expected value of the cost given that the query is ξ. Thus,

$$E[Cost|\Pi] = \sum_{\xi} E[Cost|\xi, \Pi] Pr(\xi|\Pi). \tag{2}$$

Rewriting the expected cost of a query as the sum of costs of individual accesses, and using the fact that the probability of generating a particular query is independent of the list ordering we obtain:

$$E[Cost|\Pi] = \sum_{\xi} \sum_{i=1}^{N} \Pi^{-1}(i) b_i Pr(\xi), \tag{3}$$

where $Pr(\xi)$ is defined in equation (1) and b_i is a bit representing the event that $R_i \in \xi$. Notice that in (3), the summations on the right hand side are both finite. Hence we can interchange their order and obtain:

$$E[Cost|\Pi] = \sum_{i=1}^{N} \Pi^{-1}(i) \sum_{\xi} b_i Pr(\xi). \tag{4}$$

We note that b_i is a binary random variable and hence,

$$\sum_{\xi} b_i Pr(\xi) = E[b_i] = Pr(b_i = 1) = p_i,$$

and therefore (4) simplifies to:

$$E[Cost|\Pi] = \sum_{i=1}^{N} \Pi^{-1}(i) p_i. \tag{5}$$

One can see that given a specific list ordering Π, this expected cost is similar to the one obtained in the traditional model if the S-vector was the vector of access probabilities (i.e., where the s_i's summed to one). Following Knuth's proof in [11], we can deduce that the optimal order corresponds to one in which the elements are arranged in the descending order of probabilities as given by $\mathcal{P}$. □

Corollary 1 *For an MAQG whose access probability vector is $\mathcal{P}$ the expected cost for the optimal list, denoted by the ordering Π_{opt}, is given by:*

$$C_{opt}(\mathcal{P}) = E[Cost|\Pi_{opt}] = \sum_{i=1}^{N} i p_i. \tag{6}$$

As a result of Theorem 1, we know that if the entire $\mathcal{P}$ vector is known a priori we can trivially compute the optimal list order that minimizes the expected access cost. On the other hand, if the vector $\mathcal{P}$ is not known (which is the case in real life) we will have to resort to heuristics that aim to maintain the data structure in states for which the expected cost is close to the optimal expected cost. We shall now study ergodic heuristics for singly-linked-lists under the general query generation model. It is our aim to develop heuristics which reorganize the list without attempting to estimate the P vector. We propose natural extensions to the well known MTF and TR heuristics which satisfy the above conditions. Indeed, both of these require only the space to maintain the list itself. We shall begin by first examining the MTF heuristic.

3 Avenues for Studying MAQ Heuristics

In the preceding section, we have determined that the optimal ordering of records corresponds to the descending order of the components of the vector $\mathcal{P}$ characterizing MAQs. In this paper, we are interested in pursuing the study of heuristics which do not estimate the vector $\mathcal{P}$ in order to minimize the expected access cost. Additionally we shall focus on memoryless ergodic heuristics [16], which generally reorganize the list, in response to each query.

Note that each of the queries emerging from MAQG consists of a *set* of elements. We shall refer to this stream of queries as the "True Query Stream" (TQS). At this point, two distinct directions of presenting the query information to the data reorganization heuristic exist. In one approach, the heuristic receives the queries as they occurred in the TQS, together with the information about the boundaries between the sets. For example, if the first few queries in the TQS are $\{\{R_1\}, \{R_2\}, \{R_2, R_3\}, \{R_1, R_2, R_3\}\}$, the queries seen by the heuristic are $\{R_1\}$, $\{R_2\}$, and so on.

In the second approach to studying heuristics, we reduce the amount of information made available to the data reorganizing heuristic. We merely present the list of individual records accessed, without the information about the occurrence of the set boundaries in the TQS. Thus, for example in the TQS presented above, the sequence of records seen by the heuristic could be $\{R_1, R_2, R_2, R_3, R_1, R_2, R_3\}$. Notice that the stream of records accessed is still available except that the set boundaries have been "erased". There is still one problem remaining to be resolved — namely, that of selecting the ordering of elements as presented in a *single set* to the heuristic. Since we do not have any *a priori* information about the record accesses, we should not discriminate between the records based on their "indices". A fair strategy must ensure that all permutations of a given set query are equally likely. This

stream consisting of accesses to individual records will be called the Serialized Query Stream (SQS).

In studying the behaviour of heuristics dealing with single record accesses, such as those observable in the SQS, we must determine two fundamental quantities. First of all we must determine the (asymptotic) probability of accessing the individual records of $\mathcal{R}$. Additionally, it is imperative that the statistical dependence (if any) between the successive record accesses be characterized. This information is definitely required in the analysis of *any* heuristic dealing with SQS. In order to obtain this we now proceed study these aspects of the SQS, and in doing so we shall not pay any attention to the heuristic which may be employed by the data reorganization strategy.

3.1 Analysis of the Serialized Query Stream

In the introductory section, we had introduced the idea of query generation under the P-model. It was shown that any query $\xi \subseteq \mathcal{R}$ can be specified alternately in terms of a bit vector $\mathcal{B} = [b_1, b_2, \ldots, b_N]$, where the bit b_i taking on the value 1 (0) indicates the presence (absence) of record R_i in the set under consideration. Given this bit-vector representation of the query ξ, the probability of generating this query is given by:

$$Pr(\xi) = \prod_{i=1}^{n} p_i^{b_i}(1-p_i)^{1-b_i}$$

Note that in this model of query generation, the possibility that none of the records of $\mathcal{R}$ are accessed in a particular set, exists. In fact the probability of the occurrence of this event can be written down as $\prod_{i=1}^{N}(1-p_i)$. The complementary event, namely that a non-null subset has been requested, occurs with a probability represented by the quantity η which is given by:

$$\eta = 1 - \prod_{i=1}^{N}(1-p_i).$$

We now turn our attention to a simple characteristic of SQS. We first determine the probability that a "set boundary" occurs at a randomly chosen point in SQS. After evaluating this quantity, we shall use its form in the determination of the asymptotic probability of obtaining the various records of $\mathcal{R}$ in the SQS.

Let z_k, for $k = 1, 2, \ldots, N$, denote the probability of encountering a set of cardinality k in the TQS. We can express z_k as follows:

$$z_k = \sum \prod_{j=1}^{k} p_{i_j} \left\{ \prod_{j=k+1}^{N} (1-p_{i_j}) \right\}, \tag{7}$$

where $\langle i_1, i_2, \ldots, i_N \rangle$ is a permutation of the integers in the set $\{1, 2, \ldots, N\}$. The outermost summation is to be understood to be spanning over the $\binom{N}{k}$ possible choices of subsets of size k. As an illustration of (7), we present an example for the case when $N = 4$. For this case, z_2 has the form:

$$\begin{aligned} z_2 &= p_1p_2(1-p_3)(1-p_4) + p_1p_3(1-p_2)(1-p_4) + p_1p_4(1-p_2)(1-p_3) \\ &\quad + p_2p_3(1-p_1)(1-p_2) + p_2p_4(1-p_1)(1-p_3) + p_3p_4(1-p_1)(1-p_2). \end{aligned}$$

Reverting to the general case, we observe that $\sum_{i=1}^{N} z_i = \eta$. Also we note that the occurrence of the null set is not observable in the SQS. In fact, as far as SQS is concerned, since the null query sets are neither observed nor processed, we can treat the events as occurring in the *conditional space* in which *only non-null query sets* are generated. We now define the quantity y_i as the *conditional* probability that a set with size i is generated, given that a non-null set has been generated in the TQS. By the laws of conditional probability,

$$y_i = \frac{z_i}{\eta}. \tag{8}$$

Note that $\sum_{i=1}^{N} y_i = 1$.

Our first concern now is to determine the probability of a set boundary having occurred at a random point in the SQS. To motivate the development of the mathematical model used to evaluate this quantity, let us suppose that the current point in time corresponds to the occurrence of a "set boundary". If a set with k elements is now generated, the next set boundary occurs only k time units later. With some insight, this general philosophy can be embedded in a Markovian setting. Indeed, we endeavour to evaluate the probability of a set boundary occurring, using the markov chain $\mathcal{M}$ described as follows:

$$\mathcal{M} = (\Phi, M)$$

where,

Φ is the set of states $\{\phi_i | 0 \leq i \leq N-1\}$. The state ϕ_i represents the event that i additional records have to be seen before the next set boundary is encountered. ϕ_0 is the state corresponding to the situation in which a set boundary has just been encountered. ϕ_1 is the event where the last record in a set has been processed and thus a set boundary will be encountered after one unit of time.

$M : \Phi \times \Phi \mapsto [0, 1]$ is the matrix of transition probabilities from ϕ_i to ϕ_j, for $0 \leq i, j \leq N-1$.

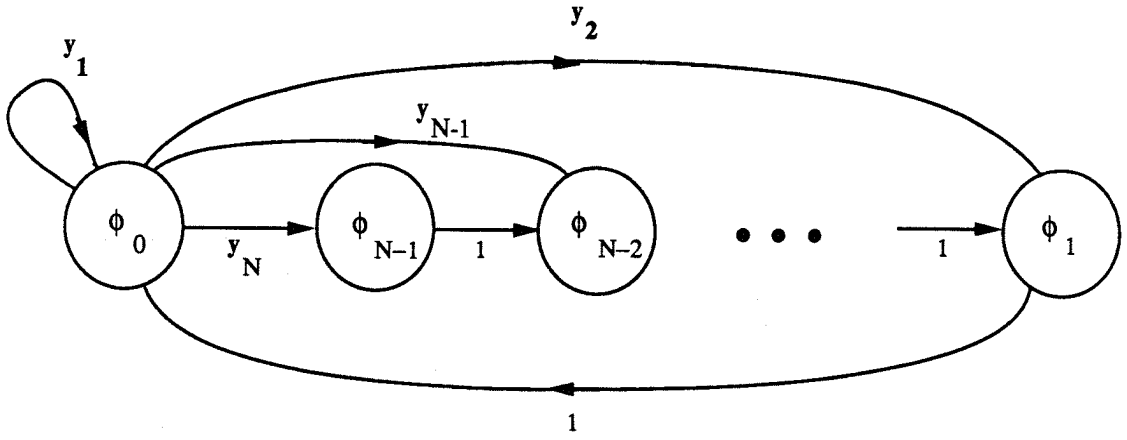

Figure 1: The Markov chain capturing the information about set boundaries in SQS.

The graphical representation of the Markov chain is shown in Figure 1. Note that the matrix M is sparse, and its non-null components can be written down as:

$$\begin{aligned} M_{0,i} &= Pr(\phi_i|\phi_0) = y_{i+1}, \quad i = 1, 2, \ldots, N-1 \\ M_{i,i-1} &= Pr(\phi_{i-1}|\phi_i) = 1, \quad i = 1, 2, \ldots, N-1 \end{aligned}$$

Using the above representation, we shall now state and prove our first result which involves the computation of the probability of encountering a set boundary in the SQS.

Theorem 2 *The asymptotic probability that a set boundary occurs at a random instant in the SQS is given by:*

$$\pi_0^* = \frac{\eta}{\sum_{i=1}^{N} p_i}. \tag{9}$$

Proof:

The Markov Chain $\mathcal{M}$ is homogeneous, aperiodic, and recurrent, and is therefore ergodic [10]. As a result, the asymptotic distribution exists and is unique. We denote the asymptotic probability vector as $\pi^* = [\pi_0^* \pi_1^* \ldots \pi_{N-1}^*]^T$, where π_i^* is the asymptotic (i.e. stationary, limiting, or equilibrium) probability of the chain being in state ϕ_i.

The stationary distribution of this Markov chain can be found as the solution of the set of linear equations given by $M^T \pi^* = \pi^*$. Of course, these equations need to be augmented with the obvious constraint that the components of π^* must sum to unity. Using the sparseness

of M, the components of π^* can be shown to obey the following scalar equations:

$$\begin{aligned} y_1\pi_0^* + \pi_1^* &= \pi_0^* \\ y_{j+1}\pi_0^* + \pi_{j+1}^* &= \pi_j^* \quad 2 \le j \le N-2 \\ y_N\pi_0^* &= \pi_{N-1}^* \end{aligned} \tag{10}$$

The set of equations in (10) can be rewritten to yield all the components of π^* in terms of the common quantity π_0^*. Thus,

$$\begin{aligned} \pi_1^* &= (1-y_1)\pi_0^* \\ \pi_{N-1}^* &= y_N\pi_0^* \\ \pi_{N-2}^* &= y_{N-1}\pi_0^* + \pi_{N-1}^* \\ &= (y_{N-1}+y_N)\pi_0^* \\ &\vdots \\ \pi_2^* &= (y_N + y_{N-1} + \cdots + y_3)\pi_0^* \end{aligned} \tag{11}$$

Combining the constraint $\sum_{i=0}^{N-1}\pi_i^* = 1$, with the set of equations in (11), we get:

$$\{1 + (1-y_1) + (N-2)y_N + (N-3)y_{N-1} + \cdots + y_3\}\,\pi_0^* = 1 \tag{12}$$

Since $\sum_{i=1}^{N} y_i = 1$, we can equivalently modify (12) to be:

$$\left\{\sum_{i=1}^{N} y_i + \sum_{i=2}^{N} y_i + \sum_{i=3}^{N}(i-2)y_i\right\}\pi_0^* = 1,$$

which simplifies to,

$$\left\{\sum_{i=1}^{N} iy_i\right\}\pi_0^* = 1,$$

whence we obtain,

$$\pi_0^* = \frac{1}{\sum_{i=1} iy_i} = \frac{\eta}{\sum_{i=0}^{N} iz_i}. \tag{13}$$

Consider the denominator of (13). Observe that it consists of a single term, $\sum_{i=0}^{N} iz_i$, which is merely the expected number of elements in a random set query. Considering that the elements of $\mathcal{R}$ are independently chosen, the number of elements in a query, U, can be written down as:

$$U = \sum_{i=1}^{N} U_i,$$

where $U_i = 1(0)$ indicates the presense (absence) of record R_i in a query. Note that U_i's are indicator random variables. Therefore, the expected number of elements in a query is given by $E[U] = \sum_{i=1}^{N} E[U_i] = \sum_{i=1}^{N} p_i$. Combining this result with (13) yields the required result in a straightforward fashion. □

To aid in the presentation of our next major result, we introduce a key result.

Lemma 1 *Let $\gamma_k(\xi)$ denote the* **conditional probability** *that a superset of ξ of cardinality k, where $k \geq |\xi|$, is generated in the TQS, when the event is conditioned on the occurrence of the non-null set. Then, the following results hold:*

$$1.\quad \gamma_k(\xi) = \frac{1}{\eta} \cdot \sum_{\zeta \supseteq \xi,\, |\zeta| = k} \left[\prod_{R_i \in \zeta} p_i \right] \left[\prod_{R_i \in \mathcal{R} - \zeta} (1 - p_i) \right] \tag{14}$$

$$2.\quad \sum_{i=1}^{N} \gamma_l(\{R_i\}) = l \cdot \gamma_l(\{\}). \tag{15}$$

$$3.\quad \sum_{j \neq i} \gamma_l(\{R_i, R_j\}) = (l-1)\gamma_l(\{R_i\}). \tag{16}$$

Proof:

We note in passing that if $k < |\xi|$, the probability $\gamma_k(\xi)$ will be 0.

The probability $\gamma_k(\xi)$ can be written down by enumerating the various ways of forming the various supersets ζ of ξ, such that $|\zeta| = k$. The result given in (14) is obvious since the terms within the first and second square brackets represent the joint probability corresponding to the elements which have been included in the set ζ, and the remaining elements of $\mathcal{R}$, i.e. $\mathcal{R} - \zeta$, respectively. This proves the first result of this Lemma.

Note that this definition for $\gamma_k(\xi)$ generalizes the definition for y_k originally defined in (8). In fact, y_k can be easily seen to be equivalent to $\gamma_k(\{\})$.

We now proceed to prove the second property stated in (15) above. Notice that $\gamma_l(\{R_i\})$ consists of $\binom{N-1}{l-1}$ terms, each term representing the contribution from one set of size l, which includes the element R_i. Moreover, the contribution of each such set (say κ) appears l times in the summation $\sum_{i=1}^{N} \gamma_l(\{R_i\})$. This is due to the fact that those indices i, which correspond to the records of κ contribute to the identical term. The total number of terms in the summation is $N \cdot \binom{N-1}{l-1}$, which is equivalent to $l \cdot \binom{N}{l}$. If we consider the grouping of terms as explained above, the result follows.[1]

The proof of the third property proceeds along the same lines as the previous proof. Once again, the number of terms in the term $\gamma_l(\{R_i, R_j\})$ is $\binom{N-2}{l-2}$. Therefore the total number

[1] An example which will help clarify the essential steps involved in the proof will be presented momentarily. We would rather maintain the formalism in the proof so as to retain the continuity of reasoning.

of terms in the summation $\sum_{j\neq i}\gamma_l(\{R_i,R_j\})$ is $(N-1)\cdot\binom{N-2}{l-2}$, or $(l-1)\cdot\binom{N-1}{l-1}$. Let κ be a set which contributes to the quantity $\gamma_l(\{R_i,R_j\})$. The contribution from the same set κ appears $l-1$ times in the summation $\sum_{j\neq i}\gamma_l(\{R_i,R_j\})$. Grouping the terms as indicated above leads to (16).

□

Example 1.

It is not inappropriate to study an example which can help clarify the notation, philosophy and the proof of the second property stated in Lemma 1. Consider the situation in which $N=4$, and $l=2$. The $\mathcal{P}$ vector in this case has the value $[p_1,p_2,p_3,p_4]^T$. For convenience, we shall define the following terms, which represent the contributions from each of the $\binom{N}{l}$, or $\binom{4}{2}=6$ sets.

$$\begin{aligned}
t_1 &= \frac{1}{\eta}p_1p_2(1-p_3)(1-p_4)\\
t_2 &= \frac{1}{\eta}p_1p_3(1-p_2)(1-p_4)\\
t_3 &= \frac{1}{\eta}p_1p_4(1-p_2)(1-p_3)\\
t_4 &= \frac{1}{\eta}p_2p_3(1-p_1)(1-p_4)\\
t_5 &= \frac{1}{\eta}p_2p_4(1-p_1)(1-p_3)\\
t_6 &= \frac{1}{\eta}p_3p_4(1-p_1)(1-p_2).
\end{aligned}\tag{17}$$

Note that $\gamma_2(\{\})$ is by definition $\sum_{i=1}^{6}t_i$.

Having defined these terms, we proceed to write the terms corresponding to $\gamma_2(\{R_i\})$ for $i=1,2,3,4$. These quantities can be written down in terms of the expressions t_1 through t_6 defined in (17) and have the form given below:

$$\begin{aligned}
\gamma_2(\{R_1\}) &= t_1+t_2+t_3\\
\gamma_2(\{R_2\}) &= t_1+t_4+t_5\\
\gamma_2(\{R_3\}) &= t_2+t_4+t_6\\
\gamma_2(\{R_4\}) &= t_3+t_5+t_6.
\end{aligned}\tag{18}$$

Observe from (18) that the terms t_i, for $1\leq i\leq 6$, appear in exactly two quantities of the form $\gamma_2(\{R_i\})$. In this example, the sum of the quantities on the LHS of (18) is $2\sum_{i=1}^{6}t_i$, which is the quantity $2\gamma_2(\{\})$. The generalization for other values of l should now be obvious to the reader.

By constructing a similar example, the reader can get a feeling for the correctness of the third property in Lemma 1. □.

We now make use the conditional probabilities γ_k's defined earlier, and derive a few more probabilities of interest. We now present our next result.

Lemma 2 *Let A_i denote the probability that the record R_i appears in the first position of a* newly generated *set query in SQS. Then the following results hold:*

$$1.\quad A_i = \sum_{l=1}^{N} \frac{\gamma_l(\{R_i\})}{l} \tag{19}$$

$$2.\quad \sum_{i=1}^{N} A_i = 1. \tag{20}$$

Proof:

The proof of the first result simply consists of enumerating all conditions under which the record R_i appears in the first position of a set. We note that the possible cardinalities for a query set are $1, 2, \ldots, N$. The probability of generating a set of size l, which includes the element R_i is given by $\gamma_l(\{R_i\})$. Also, given that R_i occurs in a set of length l, the probability that it occurs in the first position is simply $1/l$ since all $l!$ permutations of a set of cardinality l are equally likely. Hence the probability that R_i appears first, in a set of size l, is simply given by $\gamma_l(\{R_i\})/l$. Summing this quantity for all possible values of l in the interval $[1, N]$, yields the expression stated in (19).

The proof of the second part is intuitively obvious since *one* of the records of $\mathcal{R}$ must come in the first position of a newly generated query in the SQS. An alternate way to conclude this is as follows. The LHS of (20) can be written as:

$$\begin{aligned}
\sum_{i=1}^{N} A_i &= \sum_{i=1}^{N}\sum_{l=1}^{N} \frac{\gamma_l(\{R_i\})}{l} \quad \text{(from (19))} \\
&= \sum_{l=1}^{N} \frac{1}{l} \sum_{i=1}^{N} \gamma_l(\{R_i\}) \quad \text{(interchanging the order of summations)} \\
&= \sum_{l=1}^{N} \gamma_l(\{\}) \quad \text{(by (15))} \\
&= \sum_{l=1}^{N} y_l \\
&= 1.
\end{aligned}$$

The lemma is now completely proved. □

We state our next result which yields the asymptotic probability of observing the individual records of $\mathcal{R}$ in the SQS.

Theorem 3 *In the SQS, the asymptotic total probability of observing a record R_i is given by:*

$$p_i^* = \frac{p_i}{\sum_{j=1}^{N} p_j}$$

Proof:

Our aim is to compute the asymptotic probability[2] of observing the record R_i in the SQS. We shall compute this quantity as the limiting value of the observing R_i at an arbitrary time instant. Indeed, it remains to be proved that the limit exists. With this information, the asymptotic can be easily computed. In terms of notation, let:

$\alpha_i(n)$ denote the event "Record R_i is observed at time n".

$B(n)$ denote the event that a set boundary occurs at the time instant n in the SQS. Also $\overline{B(n)}$ denotes the complement of the event $B(n)$.

$\pi_0(n)$ denote the probability that the Markov chain, $\mathcal{M}$ is in the state ϕ_0 (i.e. at a state boundary) at the time instant n. Observe that $\pi_0(n)$ is well defined because of the Markov Chain, $\mathcal{M}$. Moreover, its limiting value was shown to exist in Theorem 2, and has the form given by (9).

Using the above notation, the probability of the event $\alpha_i(n+1)$ can be written down as:

$$\begin{aligned} Pr[\alpha_i(n+1)] &= \sum_{j=1}^{N} Pr[\alpha_i(n+1) \cap \alpha_j(n)] \\ &= Pr[\alpha_i(n+1) \cap \alpha_i(n)] + \sum_{j \neq i} Pr[\alpha_i(n+1) \cap \alpha_j(n)] \end{aligned} \tag{21}$$

Consider the evaluation of the first term of (21), namely the probability $Pr[\alpha_i(n+1) \cap \alpha_i(n)]$. In terms of the notation already introduced, this quantity can be expressed as:

$$\begin{aligned} &Pr[\alpha_i(n+1) \cap \alpha_i(n)] \\ &\quad = Pr[\alpha_i(n+1) \cap \alpha_i(n) \cap B(n)] + Pr[\alpha_i(n+1) \cap \alpha_i(n) \cap \overline{B(n)}], \end{aligned}$$

and since a single set in the TQS cannot include repeated elements, the second term on the RHS has a value of zero. Thus,

$$\begin{aligned} Pr[\alpha_i(n+1) \cap \alpha_i(n)] &= Pr[\alpha_i(n+1) \cap \alpha_i(n) \cap B(n)] \\ &= Pr[\alpha_i(n+1) | \alpha_i(n) \cap B(n)] \cdot Pr[\alpha_i(n) \cap B(n)] \end{aligned} \tag{22}$$

[2]In essence the result states that the asymptotic (total) probability vector of accesses to the elements of $\mathcal{R}$ is simply the *normalized* form of the vector $\mathcal{P}$. Moreover, contrary to intuition, this result holds only asymptotically and it converges in a Markovian fashion.

The first term in the RHS of (22) is simply the probability that record R_i occurs in the first position of a newly observed set, and is the quantity A_i derived in (19) as per Lemma 2. We now focus on the second term of (22). First of all observe that if $n < N$ (i.e. the chain has just started) the possible instants for a query set initiation are $1, 2, \ldots, n$. On the other hand, if $n > N$, the set boundary could have occurred at most N time units prior to n. Thus, in all cases, $\min(n, N)$ captures the number of time units in the past, where a new set could have begun.

$$Pr[\alpha_i(n) \cap B(n)] = \sum_{k=1}^{\min(n,N)} \pi_0(n-k) Pr\left[\begin{array}{l} R_i \text{ is the last element of set} \\ \text{of size } k \end{array}\right].$$

The probability of the event that R_i is the last element of a set of size k is quite simply $\gamma_k(\{R_i\})/k$. Therefore, we can write down:

$$Pr[\alpha_i(n) \cap B(n)] = \sum_{k=1}^{\min(n,N)} \pi_0(n-k) \frac{\gamma_k(\{R_i\})}{k}. \tag{23}$$

As a result of Theorem 2 we know that $\lim_{n\to\infty} \pi_0(n)$ is π_0^*. Hence, observing that the term $\frac{\gamma_k(\{R_i\})}{k}$ is independent of n, and taking the limits of (23) as $n \to \infty$ we obtain,

$$\begin{aligned} \lim_{n\to\infty} \{Pr[\alpha_i(n) \cap B(n)]\} &= \pi_0^* \sum_{k=1}^{N} \frac{\gamma_k(\{R_i\})}{k} \\ &= \pi_0^* A_i. \end{aligned}$$

Incorporating the above result in (22), we get:

$$\lim_{n\to\infty} \{Pr[\alpha_i(n+1) \cap \alpha_i(n)]\} = \pi_0^* A_i A_i. \tag{24}$$

We now turn our attention to the evaluation of the second term of (21) and write:

$$\begin{aligned} &Pr[\alpha_i(n+1) \cap \alpha_j(n)] \\ &\quad = Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap B(n)] + Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap \overline{B(n)}] \end{aligned} \tag{25}$$

By using arguments similar to those used to derive (24), the first term on the RHS of (25) can be shown to simplify to the following expression:

$$\lim_{n\to\infty} \{Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap B(n)]\} = \pi_0^* A_i A_j. \tag{26}$$

In the evaluation of the second term of (25), we know that a set boundary does not occur at time instant n. Therefore, the possible starting time instants for a set must belong to the interval $[n - \{\min(n, N) - 1\}, n - 1]$. Also note that when the set commences at the time instant $n - k$, the cardinality of the set is has to be in the closed interval $[k + 1, N]$.

The probability of a set of size l (containing the records R_i and R_j) is given by $\gamma_l(\{R_i, R_j\})$. Moreover, utilizing the information that the distribution of permutations is uniform, the probability that records R_i and R_j are adjacent in the generated permutation of a set of cardinality l is $(l-2)!/l!$ or $1/\{l(l-1)\}$. Therefore we obtain:

$$Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap \overline{B(n)}] = \sum_{k=1}^{\min(n,N)-1} \pi_0(n-k) \sum_{l=k+1}^{N} \frac{\gamma_l(\{R_i, R_j\})}{l(l-1)} \tag{27}$$

Taking the limits of both sides of (27) as $n \to \infty$ we obtain,

$$\begin{aligned} &\lim_{n\to\infty} \left\{ Pr[\alpha_i(n+1) \cap \alpha_j(n) \cap \overline{B(n)}] \right\} \\ &= \sum_{k=1}^{N-1} \pi_0^* \sum_{l=k+1}^{N} \frac{\gamma_l(\{R_i, R_j\})}{l(l-1)} \\ &= \pi_0^* \sum_{l=2}^{N} \frac{\gamma_l(\{R_i, R_j\})}{l} \end{aligned} \tag{28}$$

Combining (28), (26), and (25) we obtain:

$$\lim_{n\to\infty} \Pr[\alpha_i(n+1) \cap \alpha_j(n)] = \pi_0^* A_i A_j + \pi_0^* \sum_{l=2}^{N} \frac{\gamma_l(\{R_i, R_j\})}{l}.$$

Thus, the asymptotic probability of observing the record R_i can be written as:

$$\begin{aligned} p_i^* &= \lim_{n\to\infty} Pr[\alpha_i(n+1)] \\ &= \pi_0^* A_i A_i + \pi_0^* \sum_{j\neq i} A_i A_j + \sum_{j\neq i} \pi_0^* \sum_{l=2}^{N} \frac{\gamma_l(\{R_i, R_j\})}{l} \\ &= \pi_0^* A_i \sum_{j=1}^{N} A_j + \pi_0^* \cdot \sum_{l=2}^{N} \frac{1}{l} \sum_{j\neq i} \gamma_l(\{R_i, R_j\}) \end{aligned} \tag{29}$$

Using (20), (16), and (29), we see that:

$$\begin{aligned} p_i^* &= \pi_0^* A_i + \pi_0^* \cdot \sum_{l=2}^{N} \frac{(l-1)\gamma_l(\{R_i\})}{l} \\ &= \pi_0^* \left\{ \sum_{l=1}^{N} \frac{\gamma_l(\{R_i\})}{l} + \sum_{l=2}^{N} \frac{(l-1)\gamma_l(\{R_i\})}{l} \right\} \\ &= \pi_0^* \left\{ \sum_{l=1}^{N} \gamma_l(\{R_i\}) \right\} \\ &= \pi_0^* \frac{p_i}{\eta} \end{aligned}$$

But in Theorem 2, it was shown that $\pi_0^* = \eta/(\sum_i p_i)$. Hence, the expression for p_i* can be further simplified to the form:

$$p_i^* = \frac{p_i}{\sum_{j=1}^{N} p_j},$$

and the theorem is proved. □

Thus far we have completely characterized the SQS in terms of the asymptotic probabilities of accessing the individual records. Moreover, it is easy to see that successive record accesses are *statistically dependent.* Given this dependence between queries, the analysis of ergodic heuristics is far from simple. In fact the only results for dependent queries are due to Lam *et. al.* in their pioneering paper [12], and Valiveti *et. al.* in [18]. The general technique to model these systems is to study the composite chain, whose state space is the vector product of the $N!$ states of the linear list, and the N states of the query generator. The exact solution therefore involves the evaluation of $N \cdot N!$ stationary probabilities. This is impractical except for very small values of the parameter N, the number of records in the list. For the special case of the MTF heuristic, Lam *et. al.* were able to obtain closed form expressions for the expected cost, in terms of the mean times for first passage of the reversed time chain representing the query generator. The analysis of the TR heuristic remains open.

Hereafter, we shall refer to the classical MTF heuristic operating in conjunction with the SQS as MTF_SQS. In principle, we have the parameters needed to commence the analysis of MTF_SQS. Due to the complexity of the matrix capturing the dependence between queries, it is not feasible to follow the approach taken by Lam *et. al.* and compute the expected cost. Rather than explicitly analyze MTF_SQS we shall present a variant of the classical MTF, which interacts with the TQS. This heuristic will be called MTF_TQS. We shall first analyze the behaviour of MTF_TQS, and then compare it to the performance of MTF_SQS. In this special case, we can show without resorting to the techniques used by Lam *et. al.*, that MTF_TQS indeed yields a lower expected cost than MTF_SQS.

4 The Move-to-Front Heuristic for MAQs

We shall now describe the proposed MTF_TQS heuristic for the MAQG and subsequently proceed to analyze it. As is well known, the general characteristic of the MTF scheme is that it moves the accessed elements towards the front of the list. Since we are encountering the request for multiple elements in a single query it is natural to move all of them to the front of the list. Of course, if an element is already at the head of the list it need not be moved, and thus in such a case no list reorganization need be done. There is however one major note of difference between the current scheme and the traditional MTF. If we have a query accessing multiple elements of $\mathcal{R}$, there is no unique way to move them to the front of the list. In the absence of such a natural ordering our algorithm proposes that the sequence of moving the elements be chosen randomly in such a way that all permutations of a given query are

equally likely. The whole procedure is algorithmically given in Algorithm MTF_TQS and is illustrated in Figure 2.

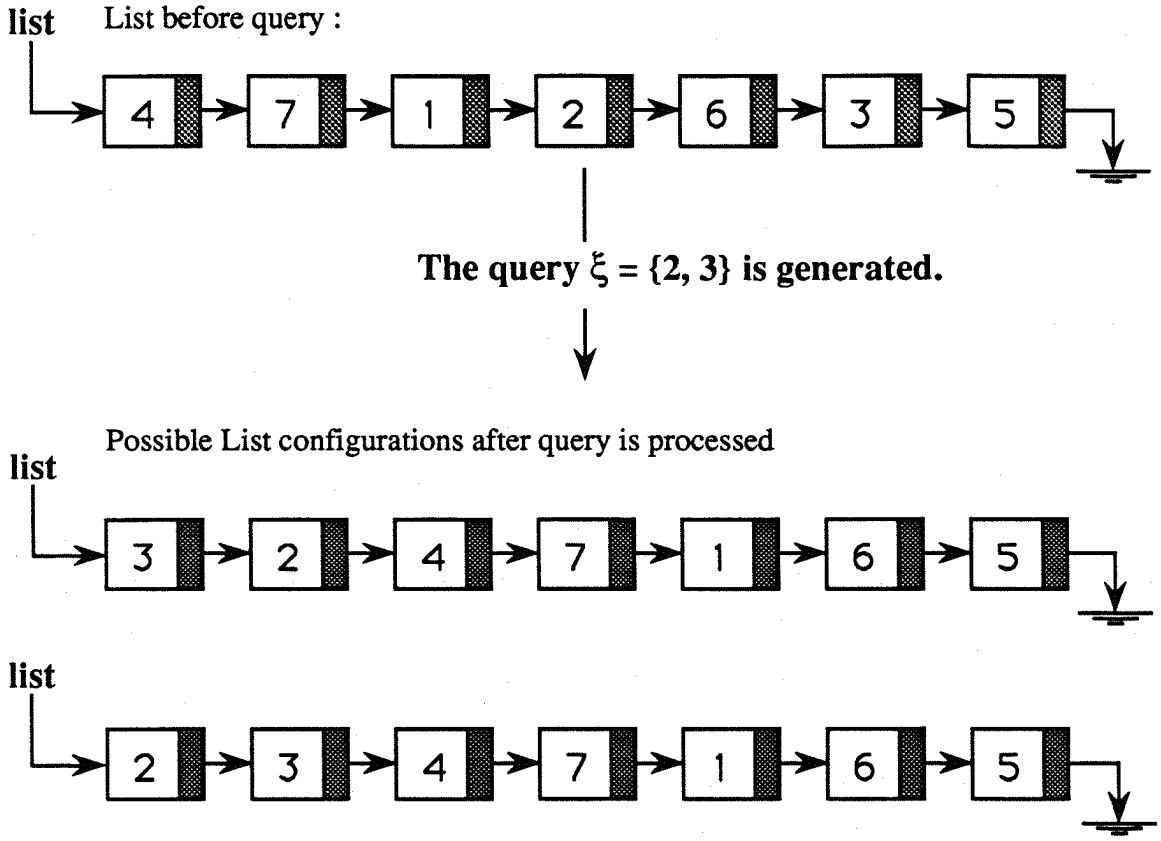

Figure 2: Diagram showing a sample seven element list before and after a query is processed. The reorganization is done using the MTF_TQS scheme. In this case the query is {2, 3} and both possible outcomes are shown. In the first case 2 is moved first then 3, in the second case 3 is moved first, followed by 2.

Observe that if the queries consist of only accesses to single elements of the list, this reorganization scheme is no different from the classical MTF. Given this qualitative description of the new MTF scheme we now proceed to examine the analytic properties of the scheme.

Theorem 4 *For the MTF_TQS heuristic responding to a MAQG the asymptotic probability of an element R_i preceding another element R_j is given by ${}_iP_j^*$, where:*

$$ {}_iP_j^* = \frac{p_i - 0.5p_ip_j}{p_i + p_j - p_ip_j} \tag{30} $$

Proof:

We prove this result by deriving a recursive relation for the probability of a record R_i

Algorithm MTF_TQS:

Input:

A list consisting of elements in the set R and an infinite sequence $\{\xi\}$ of MAQs. The algorithm gets the query ξ from the query processor by invoking GetNextQuery. ξ is itself stored as a list.

Output:

The reorganized list according to the MTF_TQS scheme.

Assumptions:

It is assumed that the following elementary procedures are accessible to the algorithm:

(i) Head (aList) - which returns the element at the head of the list.

(ii) DeleteHead (aList) - which removes the head of *aList* and returns the tail.

(iii) PermuteList (aList) -which uniformly permutes the given list.

(iv) Move-To-Front (aList, element) - which moves the only match found for *element* in *aList* to the front.

Method:

```
ξ := GetNextQuery
PermuteList (ξ)
while Not EmptyList (ξ) do
      Move-To-Front (aList, Head (ξ))
      ξ := DeleteHead (ξ)
endwhile
```

End Algorithm MTF_TQS

Program 1: The MTF_TQS heuristic for MAQGs

preceding the record R_j. Let ${}_iP_j(n)$ denote this probability at time instant n. We now calculate the conditional probability ${}_iP_j(n+1)$ given ${}_iP_j(n)$.

Let $\xi(n)$ be the query at time n. We have four mutually exclusive and collectively exhaustive cases. First of all, ${}_iP_j(n+1)$ is 0 if R_j was accessed and R_i was not accessed, since in that case R_j would be moved to the front while R_i would be unmoved. Thus, R_i will never precede R_j. In the second case, if R_i was accessed and R_j was not accessed the roles are interchanged and R_i will always precede R_j since it will be moved to the front at some point in the reorganization. The third case covers the possibility of neither R_j nor R_i being accessed in which case the probability of R_i preceding R_j is the same as it was before, since both elements R_i and R_j remain unmoved. Finally if both R_i and R_j are accessed, both of them will be moved to the front. However, since all permutations of $\xi(n)$ are equally likely to dictate the reorganization strategy, R_i will be moved after R_j with probability 0.5. Thus R_i will ultimately precede R_j with probability 0.5, and R_j will precede R_i with probability 0.5. Formalizing this we obtain:

$$ {}_iP_j(n+1) = \begin{cases} 0 & \text{if } R_j \in \xi(n),\ R_i \notin \xi(n) \\ 1 & \text{if } R_i \in \xi(n),\ R_j \notin \xi(n) \\ {}_iP_j(n) & \text{if } R_i, R_j \notin \xi(n) \\ 0.5 & \text{if } R_i, R_j \in \xi(n) \end{cases} \tag{31} $$

Using (1) we can calculate the probabilities with which each of these events can occur. Thus[3],

$$ {}_iP_j(n+1) = \begin{cases} 0 & \text{w.p. } (1-p_i)p_j \\ 1 & \text{w.p. } p_i(1-p_j) \\ {}_iP_j(n) & \text{w.p. } (1-p_i)(1-p_j) \\ 0.5 & \text{w.p. } p_ip_j \end{cases} $$

From the above the laws of probability yield that the quantity ${}_iP_j(n+1)$ given ${}_iP_j(n)$ has the form:

$$ \begin{aligned} {}_iP_j(n+1) &= p_i(1-p_j) + {}_iP_j(n)(1-p_i)(1-p_j) + 0.5p_ip_j \\ &= (p_i - 0.5p_ip_j) + {}_iP_j(n)(1-p_i)(1-p_j) \end{aligned} \tag{32} $$

Thus ${}_iP_j(n)$ obeys a simple first order difference equation. Its general solution is given by:

$$ {}_iP_j(n) = \{(1-p_i)(1-p_j)\}^n\, {}_iP_j(0) + \frac{p_i - 0.5p_ip_j}{p_i + p_j - p_ip_j} \cdot [1 - \{(1-p_i)(1-p_j)\}^n]. \tag{33} $$

[3]w.p. is an abbreviation for "with probability".

It can thus be seen that the sequence $\{{}_iP_j(n)\}$ converges when $0 < (1-p_i)(1-p_j) < 1$. Its asymptotic value can be easily shown to be given by (30).

We note that the above condition for convergence is not satisfied only when $p_i = p_j = 0$(or1). When $p_i = p_j = 0$, ${}_iP_j(n)$ is equal to ${}_iP_j(0)$, for all values of n. In the case when the common value of p_i, p_j is 1, ${}_iP_j(n)$ is equal to 0.5 for all values of n. □

Lemma 3 *For all i, j, whenever $p_i > p_j$, a lower bound for ${}_iP_j^*$ is given by:*

$$ {}_iP_j^* > \frac{p_i}{p_i + p_j} $$

Proof:

A simple rearrangement of the expression for ${}_iP_j^*$ in (30) yields,

$$\begin{aligned} {}_iP_j^* &= \frac{1}{1 + \frac{p_j - 0.5p_ip_j}{p_i - 0.5p_ip_j}} \\ &> \frac{1}{1 + \frac{p_j}{p_i}} \quad (\text{Since } \frac{p_j - 0.5p_ip_j}{p_i - 0.5p_ip_j} < \frac{p_j}{p_i} \text{ whenever } p_i > p_j.) \\ &> \frac{p_i}{p_i + p_j}, \end{aligned}$$

and the result follows immediately. In passing we note that whenever $p_i = p_j$ the quantity ${}_iP_j^*$ equals 0.5. □

Theorem 5 *The expected cost of the MTF_TQS scheme under the MAQG is at most $\pi/2$ times the optimal cost.*

Proof:

Let $C_{MTF\_TQS}(\mathcal{P})$ denote the expected cost of accessing elements of the linear list under the MAQG when the MTF heuristic is being used. $C_{OPT}(\mathcal{P})$ denotes the similar quantity for the optimal static ordering. Hence we wish to prove that:

$$ \frac{C_{MTF\_TQS}(\mathcal{P})}{C_{OPT}(\mathcal{P})} \leq \frac{\pi}{2}. $$

We first compute the expected cost under the MTF heuristic. To simplify the computation we compute the conditional expected cost, given that the ordering is a specific permutation Π. This quantity has been derived earlier and is:

$$ E[Cost|\Pi] = \sum_{i=1}^{N} \Pi^{-1}(i)p_i. \tag{34} $$

Taking expectations of (34) we obtain:

$$C_{MTF\_TQS}(\mathcal{P}) = E\left[E[Cost|\Pi]\right] = \sum_{i=1}^{N} E[\Pi^{-1}(i)]p_i. \tag{35}$$

We note that $E[\Pi^{-1}(i)]$ is simply the expected position of record R_i in the list, when the MTF heuristic is being used. Analogous to the derivation of Hendricks [8], we now extend our arguments to yield:

$$E[\Pi^{-1}(i)] = 1 + \sum_{j \neq i} {}_jP_i^*,$$

and hence,

$$C_{MTF\_TQS}(\mathcal{P}) = \sum_{i=1}^{N} p_i \left\{ 1 + \sum_{j \neq i} {}_jP_i^* \right\}. \tag{36}$$

Without any loss of generality let us assume that $p_i > p_j$ for all $i < j$. Also for all $i \neq j$, let ${}_i\underline{P}_j$ denote quantities which obey the following conditions:

$$\begin{aligned} {}_i\underline{P}_j + {}_j\underline{P}_i &= 1 \\ {}_iP_j^* &\geq {}_i\underline{P}_j \quad \forall\, i < j. \end{aligned}$$

Then following arguments similar to Hendricks [8] we can easily prove[4] that:

$$C_{MTF\_TQS}(\mathcal{P}) \leq \sum_{i=1}^{N} p_i \left\{ 1 + \sum_{j \neq i} {}_j\underline{P}_i \right\}. \tag{37}$$

By Lemma 3 we have identified lower bounds for the quantity ${}_iP_j^*$ for $i < j$. Thus, we can set:

$${}_i\underline{P}_j = \frac{p_i}{p_i + p_j} \quad \forall\, i < j,$$

and (37) becomes:

$$C_{MTF\_TQS}(\mathcal{P}) \leq \sum_{i=1}^{N} p_i \left\{ 1 + \sum_{j \neq i} \frac{p_j}{p_i + p_j} \right\}. \tag{38}$$

Notice that (38) is of a form similar to the cost of the classical MTF rule. (38) can be easily simplified to yield:

$$C_{MTF\_TQS}(\mathcal{P}) \leq 2 \sum_{i \leq j} \frac{p_i p_j}{p_i + p_j}. \tag{39}$$

But the optimal cost for the SLL under MAQG was previously derived and is given by:

[4]The new set of quantities ${}_i\underline{P}_j$ can be interpreted in Hendricks' proof as the set of probabilities of record R_i preceding R_j, according to some arbitrary heuristic.

$$C_{OPT}(\mathcal{P}) = E[Cost|\Pi_{opt}] = \sum_{i=1}^{N} ip_i,$$

and thus we have:

$$\frac{C_{MTF\_TQS}(\mathcal{P})}{C_{OPT}(\mathcal{P})} \leq \frac{2\sum_{i\leq j} \frac{p_i p_j}{p_i + p_j}}{\sum_{i=1}^{N} ip_i}. \tag{40}$$

Observe that Chung *et. al.* proved that the right hand side of (40) is always less than or equal to $\pi/2$ [5]. This proves the theorem. □

So far we have analyzed the performance of MTF_TQS heuristic when operating in the presence of set queries. The reader will recall this scheme had access to the TQS. Moreover, the heuristic utilized the information about the boundary between the various set queries. The issue which we investigate next concerns whether knowledge of these boundaries indeed results in a lower expected cost for query accesses. In other words, we wish to compare the performance of the two heuristics MTF_TQS and MTF_SQS.

Before we embark on a comparison of MTF_TQS and MTF_SQS, it is useful to form a clearer idea of their operation. Observe that in MTF_TQS, whenever the list configuration is $L(n)$, all the elements of a specific query $\xi(n)$ are searched in this list. In other words, the list reorganization takes place only when the desired elements have been located. Moreover, in the list reorganization process, the list "visits" several intermediate states of the underlying Markov Chain. The key characteristic which differentiates these heuristics is that in the case of MTF_TQS, these "intermediate states" are not observable. In the case of MTF_SQS, the list $L(n)$ is transformed *immediately* upon locating an element from $\xi(n)$, and the search for the remaining elements of $\xi(n)$ is done in the *transformed* list. Note however that after processing all the elements of a set query both heuristics will produce the same **final** list, since they perform the same sequence of data reorganization operations.

We now state our next result which claims the superiority of MTF_TQS over MTF_SQS. Informally speaking, the result claims that there is some latent information in the set boundaries and if this information is utilized, superior performance is attainable.

Theorem 6 *For all access distributions, the heuristic MTF_TQS yields a lower expected access cost than the heuristic MTF_SQS.*

Proof:

Let Π be any arbitrary permutation of elements in $\mathcal{R}$. Also, let Cost_{MTF_TQS} and Cost_{MTF_SQS} be the random variables denoting the access costs for queries, under the heuristics MTF_TQS and MTF_SQS respectively. The expected cost C_{MTF_TQS} can be computed as follows:

$$C_{MTF\_TQS} = E\left[E\left[\text{Cost}_{MTF\_TQS}|\xi, \Pi\right]\right]. \tag{41}$$

A similar procedure can be followed to obtain the expected cost C_{MTF_SQS} under the MTF_SQS heuristic.

We focus on the aspect of computing the conditional expected cost, given that the list is represented by Π. Using the notation that $\Pi(i)$ refers to the i^{th} element of Π, with no loss of generality, we can represent a general query set ξ as $\{\Pi(m_1), \Pi(m_2), \ldots, \Pi(m_{|\xi|})\}$, where $1 \leq m_i \leq N$, and $m_i < m_{i+1}$. In other words, m_i's are the indices of elements of Π which appear in ξ, and furthermore they represent the order in which they appear in Π.

Notice that since MTF_TQS uses only the list Π to locate the elements, $E[\text{Cost}_{MTF\_TQS}|\xi, \Pi]$ is clearly $\sum_{i=1}^{|\xi|} m_i$, where the expectation is done w.r.t. the uniform distribution of all $|\xi|!$ permutations of ξ. We now turn our attention to the behaviour of MTF_SQS. The cost incurred by MTF_SQS is greatly influenced by the order in which the elements of ξ appear in the SQS. In other words, we need to consider the $|\xi|!$ possible permutations (or serializations) of ξ.

First of all, consider the simple case in which the generated serialization of ξ corresponds exactly to the order in which elements are present in Π. In this case, although MTF_SQS moves the accessed records to the front of the list, the positions of all the other records are not altered, since they appear later in the list than the recently accessed element. Thus, in this case the cost incurred happens to be $\sum_{i=1}^{|\xi|} m_i$. In all other cases, the random permutation of ξ does not correspond to the order in which the elements are found in Π. The effect of accessing elements in a sequence other than in the order in which they are in Π, results in *at least* one record moving to position greater than its initial position. Consequently, we claim that the total access cost will be strictly greater than $\sum_{i=1}^{|\xi|} m_i$. This immediately yields the result that:

$$E[\text{Cost}_{MTF\_SQS}|\xi, \Pi] > \sum_{i=1}^{|\xi|} m_i.$$

Combining this equation with the similar cost for the MTF_TQS heuristic, we obtain:

$$E[\text{Cost}_{MTF\_SQS}|\xi, \Pi] > E[\text{Cost}_{MTF\_TQS}|\xi, \Pi]. \tag{42}$$

Taking expectations of (42), by considering the distribution of queries, we deduce:

$$E[\text{Cost}_{MTF\_SQS}|\Pi] > E[\text{Cost}_{MTF\_TQS}|\Pi]. \tag{43}$$

Taking expectations a second time, and observing that (41) is true, we get the desired result that:

$C_{MTF\_SQS} > C_{MTF\_TQS}$.

and the theorem is proved. □

5 The Transposition Heuristic for MAQs

In addition to the MTF_TQS heuristic already described we now present a variant of the classical TR heuristic where the accessed elements are transposed with the elements directly preceding them. This data reorganization scheme will be termed TR_TQS. As before, we encounter many elements in each query, and each of these elements has to be individually transposed. We define this heuristic, in a manner analogous to MTF_TQS and permute the *query* so that each element in the query has an equal probability of being transposed first, second, or last. Figure 3 depicts this scheme for a list of seven elements. Note that in certain cases the transposition scheme may leave the list unaffected, and this could occur whenever all elements of the query are adjacent in the list. Additionally if the element to be moved is already at the head of the list, the list is left untouched. The algorithm is formally given below.

In the traditional case the analysis for the TR heuristic is typically done by taking advantage of the time reversibility of the underlying Markov Chain. In fact, in the traditional case it can be shown that the chain is time reversible because the probabilities of going from one permutation to another in the forward direction exactly equal the probabilities of reversing the transition in the reverse chain.

This however, need not be the case when the TR_TQS heuristic operates under the MAQG. The reason for this is as follows. When a query ξ is presented to the scheme, if the query was processed in a particular order it would be equivalent to running the traditional TR heuristic for the sequence of queries specified by the order. Notice that in such a case one could have envisaged an argument to show that the probabilities associated with the reverse direction are identical if the query was processed in the reverse order. However, this argument could be fallacious because, whereas in the traditional TR scheme we are permitted to have consecutive accesses to the same element, in the current scheme it is assumed that the elements of ξ are all distinct. Thus it is not clear that the underlying Markov Chain is time reversible and consequently we believe that there is no straightforward technique to analyze the behavior of TR under an MAQG. Of course, the naive method is always possible which involves listing all the $N!$ possible states and subsequently computing all the $N!$ stationary probabilities.

Algorithm TR_TQS:

Input:

A list consisting of elements in the set R and an infinite sequence $\{\xi\}$ of MAQs. The algorithm gets the query ξ from the query processor by invoking GetNextQuery. ξ is itself stored as a list.

Output:

The reorganized list according to the scheme TR_TQS.

Assumptions:

It is assumed that the following elementary procedures are accessible to the algorithm:

(i) Head (aList) - which returns the element at the head of the list.

(ii) DeleteHead (aList) - which removes the head of *aList* and returns the tail.

(iii) PermuteList (aList) -which uniformly permutes the given list.

(iv) Transpose (aList, element) - which transposes the only match found for *element* in *aList* with the element directly preceding element.

Method:

ξ := GetNextQuery
PermuteList (ξ)
while Not EmptyList (ξ) do
    Transpose (aList, Head (ξ))
    ξ := DeleteHead (ξ)
endwhile

End Algorithm TR

Program 2: The TR_TQS heuristic for MAQGs

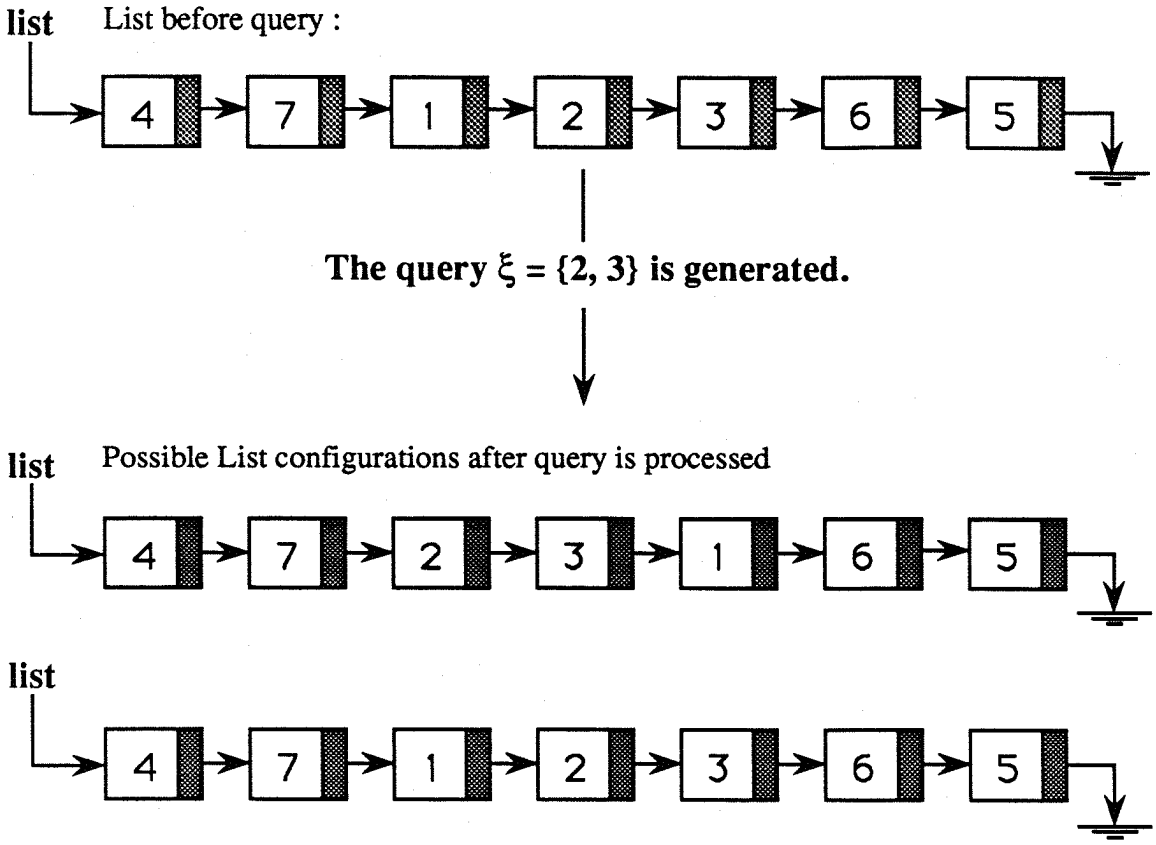

Figure 3: Diagram showing a sample seven element list before and after a query is processed. The reorganization is done using the TR_TQS scheme. In this case the query is $\{2,3\}$ and both possible list configurations are shown. Note that the list will be unchanged if $\xi = \{2,3\}$ and the transposition of 3 precedes the transposition of 2. If, on the other hand the query was $\{2,6\}$, both possible permutations lead to the same final outcome.

We are currently investigating two other possibilities. In the first, we are studying whether the states of the underlying Markov Chain are lumpable [10]. In other words we are investigating whether subsets of the states can be coalesced to yield a Markov Chain of smaller dimension. The other alternative is that of obtaining a difference equation involving the probability of R_i being immediately before R_j at the time instant n. This can possibly be a function all the possible sets of queries presented to the scheme. In either case we do not believe that the analysis of the TR_TQS heuristic will be straightforward.

While the explicit expected cost for the TR_TQS heuristic cannot be written, we nevertheless can compare its expected cost to the classical heuristic dealing with the SQS. We shall refer to *this* variant of TR as TR_SQS. We now state our final result which claims the superiority of TR_TQS over TR_SQS in all random environments. Just as in the case of the two variants of the MTF heuristic, we once again observe that the information about set boundaries can be utilized to attain a lower expected access cost for queries.

Theorem 7 *For all access distributions, the heuristic TR_TQS yields a lower expected access cost than the heuristic TR_SQS.*

Proof:

As in Theorem 6, let Π be any arbitrary permutation of elements in $\mathcal{R}$. Also, let Cost_{TR_TQS} and Cost_{TR_SQS} be the random variables denoting the access costs for queries, under the heuristics TR_TQS and TR_SQS respectively. The expected cost C_{TR_TQS} can be computed as follows:

$$C_{TR\_TQS} = E\left[E\left[\text{Cost}_{TR\_TQS}|\xi, \Pi\right]\right]. \tag{44}$$

A similar procedure can be followed to obtain the expected cost $C_{\text{Cost}_{TR\_SQS}}$ under the TR_SQS heuristic.

We focus on the aspect of computing the conditional expected cost, given that the list is represented by Π. Recapitulating the semantics of the notation used in Theorem 6, $\Pi(i)$ refers to the i^{th} element of Π, with no loss of generality, we can represent a general query set ξ as $\{\Pi(m_1), \Pi(m_2), \ldots, \Pi(m_{|\xi|})\}$, where $1 \leq m_i \leq N$, and $m_i < m_{i+1}$. In other words, m_i's are the indices of elements of Π which appear in ξ, and furthermore they represent the order in which they appear in Π.

Just like the MTF_TQS heuristic, TR_TQS uses only the list Π to locate the elements, and consequently $E[\text{Cost}_{TR\_TQS}|\xi, \Pi]$ is clearly $\sum_{i=1}^{|\xi|} m_i$, where the expectation is done w.r.t. the uniform distribution of all $|\xi|!$ permutations of ξ. We now turn our attention to the behaviour of TR_SQS. The cost incurred by TR_SQS is greatly influenced by the order in

which the elements of ξ appear in the SQS. In other words, we need to consider the $|\xi|!$ possible permutations (or serializations) of ξ.

Observe that TR_SQS swaps the most recently accessed record with the element which immediately precedes it, and hence it does not disturb the positions of the records which appear after the accessed record. Moreover, accesses to any of the records do not cause TR_SQS to move any other record which is yet to be accessed, closer to the beginning. As a result, the cost incurred in processing a query can never be less than $\sum_{i=1}^{|\xi|} m_i$. With this information alone we can conclude that TR_TQS performs at least as well as TR_SQS. Our aim is to exclude the possibility of equality in average access cost, thereby establishing that TR_TQS is strictly better than TR_SQS.

First of all, consider the simple case in which the generated serialization of ξ corresponds exactly to the order in which elements are present in Π. In this case, although TR_SQS moves the accessed records towards the front of the list, the positions of all the other records are not altered, since they appear later in the list than the recently accessed element. Thus, in this case the cost incurred is exactly $\sum_{i=1}^{|\xi|} m_i$. In all other cases, the random permutation of ξ does not correspond to the order in which the elements are found in Π, and our endeavour is now to demonstrate the existence of cases when the total access cost can be strictly greater than $\sum_{i=1}^{|\xi|} m_i$.

Consider a scenario in which the serialized permutation of ξ corresponds to the reverse of the order in which these elements occur in Π. The last two elements in this generated serialization are $\Pi(m_{|\xi|-1})$ and $\Pi(m_{|\xi|})$. If these two elements are not adjacent in Π, accessing $\Pi(m_{|\xi|})$ does not displace the other record and the total cost of accessing these records is not greater than the minimum possible cost. On the other hand, if these two records are adjacent, they will be swapped by TR_SQS resulting in the record $\Pi(m_{|\xi|-1})$ moving to the position $m_{|\xi|}$, thereby incurring a greater cost. Since we have shown that there exist serializations of ξ which incur a strictly greater cost than the lowest possible cost, the expected cost given a particular query and an ordering, will be strictly greater under TR_SQS. In other words,

$$E[\text{Cost}_{TR\_SQS}|\xi, \Pi] > \sum_{i=1}^{|\xi|} m_i.$$

Combining this equation with the similar cost for the TR_TQS heuristic, we obtain:

$$E[\text{Cost}_{TR\_SQS}|\xi, \Pi] > E[\text{Cost}_{TR\_TQS}|\xi, \Pi]. \tag{45}$$

The required result follows by taking the expectations of (45) two times and making use of (44). □

We shall now proceed to present simulation results which demonstrate the power of the MTF_TQS and TR_TQS heuristics when responding to an MAQG.

6 Experimental Results

Experiments were conducted to verify the theoretical results presented in this paper. The aim of the experiments was to compare the expected cost of queries under the MTF and TR heuristics when compared to the cost of the optimal (static) list. In summary, the experiments consisted of generating a large number of queries, based on a chosen vector $\mathcal{P}$, and then averaging the cost incurred using the different heuristics. The details of the experiments conducted are as follows.

Since the MAQG generation strategy was based on the parameter $\mathcal{P}$, the first step in the simulation involved assigning the vector a numerical value. Since there is no loss of generality in assuming that the components of $\mathcal{P}$ are in a descending order, the vectors used in the simulations obeyed this relation. As opposed to generating random vectors for $\mathcal{P}$, we chose three different types of variations in the components of $\mathcal{P}$. These variations are listed below:

(i) **Linear:** The components of $\mathcal{P}$ are given by

$$p_i = c - k \cdot i \tag{46}$$

where $i = 1, 2, \ldots, N$ and c, k are suitable positive constants satisfying $c - kN > 0$. The latter condition is imposed merely to ensure that the components of $\mathcal{P}$ are positive quantities.

(ii) **Zipf's:** The components of $\mathcal{P}$ are given by:

$$p_i = c + \frac{k}{i} \tag{47}$$

where c, and k are again suitable positive constants chosen so as to render each component of $\mathcal{P}$ to be a probability.

(iii) **Exponential:** In this case, the components of the $\mathcal{P}$ are given by:

$$p_i = c + \frac{k}{e^i} \tag{48}$$

where e is the base of the natural logarithms.

After determining the value of N, and the type of variation among the components of $\mathcal{P}$ (i.e. one of Linear/Zipf/Exponential), the components p_1 and p_N were fixed to be 0.9 and 0.4 respectively. This choice immediately fixes the values of the constants c and k in the equations (46) through (48) and these values subsequently assist in the evaluation of $p_2 \cdots p_{N-1}$.

As an example, consider the case when $N = 6$. The vectors obtained for the cases of Linear and Zipf's variation in the components of $\mathcal{P}$ are given below:

$$\begin{aligned} \mathcal{P}(\text{Linear}) &= [0.9,\ 0.8,\ 0.6,\ 0.7,\ 0.6,\ 0.5,\ 0.4]^T \\ \mathcal{P}(\text{Zipf's}) &= [0.9,\ 0.6,\ 0.5,\ 0.45,\ 0.42,\ 0.4]^T \end{aligned}$$

Using the vectors chosen as described above, queries were generated. The initial configuration of the list was randomly chosen to be one of the $N!$ permutations. Subsequently the heuristic being evaluated (i.e. either MTF or TR) was invoked for an initial period of 10,000 queries before the access costs were monitored. This initial waiting period served to ensure that the Markov chains had converged. The reported results consist of the ensemble average (over 100 experiments) of the time average of 2,500 queries following the above mentioned "transient" phase.

Various experiments were conducted for values of N ranging from 4 to 12, in steps of 2. The results for the MTF_TQS and TR_TQS heuristic, for the above values of N, and three types of distributions specified earlier, have been reported in Tables 1-6. Observe that that Tables 1-3 report the %-error between the *theoretical expected cost* for the MTF_TQS heuristic and the optimal cost.

Consider Table 1 which report the results obtained for the MTF_TQS heuristic when there was a linear gradation among the components of $\mathcal{P}$. The experimentally obtained average cost agrees well with the theoretically predicted value. Moreover, if we examine the case when $N = 6$, the expected cost under the MTF_TQS heuristic had a value of 13.0685, whereas the optimal cost was 11.9. This implies that the expected cost when using the MTF_TQS heuristic exceeds the optimal cost by merely 9.82%. To compare MTF_TQS and TR_TQS under identical settings consider the results presented in Table 4. From this table, we observe that the expected cost under the TR_TQS heuristic exceeds the optimal cost by an amazingly low 5.42%.

For all the results reported in Tables 1-6, the TR_TQS heuristic consistently outperforms the MTF_TQS heuristic. While we believe the result is true for all possible query distributions, a general proof of this statement remains open. This result, if true, will be a generalization of the analogous relation that exists between the TR and MTF heuristics, in

the case when the queries consist of statistically independent, single record accesses[16].

List Length	Experimental Expected Cost	Theoretical Expected cost	Optimal cost	% Error w.r.t. Optimal cost
4	6.1875	6.1941	5.6667	9.31
6	13.0765	13.0685	11.90	9.82
8	22.4534	22.4483	20.40	10.04
10	34.3542	34.3345	31.1667	10.16
12	48.7764	48.7275	44.20	10.24

Table 1: Results for the MTF_TQS heuristic, when the nature of the $\mathcal{P}$vector is "linear" as per (46).

List Length	Experimental Expected Cost	Theoretical Expected cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.4676	5.4753	4.9949	9.62
6	10.8685	10.8672	9.90	9.77
8	17.9378	17.9513	16.40	9.46
10	26.7414	26.7154	24.5	9.04
12	37.1686	37.1474	34.2	8.62

Table 2: Results for the MTF_TQS heuristic, when the nature of the $\mathcal{P}$ vector is "Zipf's" as per (47).

List Length	Experimental Expected Cost	Theoretical Expected cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.4427	5.4503	4.9698	9.67
6	10.4930	10.4959	9.5735	9.63
8	17.0088	17.0219	15.6338	8.88
10	25.1252	25.1065	23.2475	8.00
12	34.7915	34.7816	32.4506	7.18

Table 3: Results for the MTF_TQS heuristic, when the nature of the $\mathcal{P}$ vector is "Exponential" as per (48).

List Length	Experimental Expected Cost	Optimal cost	% Error w.r.t. Optimal cost
4	6.0219	5.6667	6.27
6	12.5452	11.90	5.42
8	21.338	20.40	4.60
10	32.4050	31.1667	3.97
12	45.7628	44.20	3.54

Table 4: Results for the TR_TQS heuristic, when the nature of the $\mathcal{P}$ vector is "linear" as per (46).

List Length	Experimental Expected Cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.3157	4.9949	6.42
6	10.4275	9.90	5.33
8	17.1139	16.40	4.35
10	25.4392	24.5	3.83
12	35.3430	34.2	3.34

Table 5: Results for the TR_TQS heuristic, when the nature of the $\mathcal{P}$ vector is "Zipf's" as per (47).

List Length	Experimental Expected Cost	Optimal cost	% Error w.r.t. Optimal cost
4	5.2864	4.9698	6.37
6	10.0519	9.5735	5.00
8	16.2221	15.6338	3.76
10	23.9537	23.2475	3.04
12	33.2091	32.4506	2.34

Table 6: Results for the TR_TQS heuristic, when the nature of the $\mathcal{P}$ vector is "Exponential" as per (48).

7 Conclusion

The problem of dynamically reorganizing a linear list, which is searched for individual list elements has been well studied in the literature. In this paper, we have proposed a model of query generation in which any subset of the elements in the linear list can be requested. Examples of the occurrences of such queries in the context of distributed systems, and database systems have been presented. An exact analytical description of this scenario necessarily involves 2^N parameters, where N is the number of elements in the list. This being impractical, we suggest a simple (and hence approximate) model for the query generation strategy based on a vector $\mathcal{P}$, of dimension N. We have derived the asymptotic record access distribution of the serial query stream emerging from such a query generator. Statistical dependence between successive records in this stream has also been established.

Under the new model of query generation, the optimal list configuration has been shown to be the order corresponding to the descending order of the components of $\mathcal{P}$. Additionally, two heuristics MTF_TQS and TR_TQS capable of dealing with set queries have been proposed which are generalizations of the MTF and TR heuristics for singly-linked lists. The cost of the MTF_TQS heuristic for our query model has been shown to be no more than $\pi/2$ times the expected cost for the optimal list arrangement. Furthermore, it is also shown that MTF_TQS and TR_TQS are superior to the corresponding serialized heuristics, MTF_SQS and TR_SQS respectively. Experimental results involving the MTF_TQS and TR_TQS heuristics have been presented. The question of proving that TR_TQS is always superior than MTF_TQS under the proposed model of query generation, remains open.

References

[1] B. Allen, and J.I. Munro, "Self-Organizing Binary Search Trees", *J. ACM*, Vol.25, 1978, pp.526-535.

[2] D.M. Arnow, and A.M. Tenenbaum, "An investigation of the Move-ahead-k Rules",*Proc. of the Thirteenth Southeastern Conference on Combinatorics, Graph Theory and Computing*, Florida, Feb. 1982, pp.47-65.

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

[4] R.P. Cheetham, B.J. Oommen, and D.T.H. Ng, "Adaptive Structuring Of Binary Search Trees Using Conditional Rotations", Technical Report SCS-TR-126, School Of Computer Science, Carleton University, October 1987.

[5] F.R.K. Chung, D.J. Hajela and P.D. Seymour, "Self- organizing Sequential Search and Hilbert's Inequalities", *Journal of Computer Sys. and Sci*, Vol.36, 1988, pp.148-157.

[6] G.H. Gonnet, J.I. Munro, and H. Suwanda, "The exegesis of Self-Organizing Linear Search",*SIAM J. Computing*, Vol.10, No.1 ,1981, pp. 613-637.

[7] K.N. Hendricks, "The Stationary Distribution of an Interesting Markov Chain", *J. Appl. Prob.*, 9, 1, (Mar. 1972), pp. 231-233.

[8] K.N. Hendricks, "An Account of Self-Organizing Systems", *SIAM J. Computing*, Vol.5, 1973, pp.715-723.

[9] J.H. Hester, D.S. Hirschberg, "Self-Organizing Linear Search", *ACM Computing Surveys*, Vol.17, No.3, Sept. 1985, pp. 295-311.

[10] J.G. Kemeny, and, J.L. Snell, *Finite Markov Chains*, Van Nostrand, New York, 1969.

[11] D.E. Knuth, *The Art Of Computer Programming. Vol. 3: Sorting and Searching*, Addison-Wesley, 1973.

[12] Lam, C.K., "File Reorganization with Dependent Accesses", *Journal of Appl. Prob.*, 1984.

[13] D. Matthews, D. Rotem, and E. BretHolz, "Self-Organizing Doubly-Linked Lists", *J. Comput. Maths.*, Sec.A, Vol.8 (1980), pp. 99-106.

[14] B.J. Oommen, and E.R. Hansen, "List organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-rear operations", *SIAM J. Computing*, Vol.16, No.6, August 1987, pp.705-716.

[15] B.J. Oommen, and D.T.H. Ng, "Ideal List Organization for Stationary Environments", Technical Report SCS-TR-154, School Of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6.

[16] R.L. Rivest, "On Self-Organizing Sequential Search Heuristics", *Comm. ACM*, Vol.19, No.2, Feb. 1976, pp.63-67.

[17] A.M. Tenenbaum , and R.M. Nemes, "Two Spectra of Self-Organizing Sequential Search Algorithms", *Journal of Sys. and Sciences*, Vol.36, 1988, pp.148-157.

[18] R.S. Valiveti, and B.J. Oommen, "Adaptive List Organizing for Non-stationary Query Distributions: Part I: The Move-to-Front Rule", Technical Report SCS-TR-168, School Of Computer Science, Carleton University, Jan. 1990.

[19] R.S. Valiveti, and B.J. Oommen, "Self-Organizing Doubly-Linked Lists", Technical Report SCS-TR-173, School Of Computer Science, Carleton University, May 1990.

School of Computer Science, Carleton University
Bibliography of Technical Reports

SCS-TR-140 **Computing the Configuration Space of a Robot on a Mesh-of-Processors**
F. Dehne, A.-L. Hassenklover and J.-R. Sack, June 1988.

SCS-TR-141 **Graphically Defining Simulation Models of Concurrent Systems**
H. Glenn Brauen and John Neilson, September 1988

SCS-TR-142 **An Algorithm for Distributed Mutual Exclusion on Arbitrary Networks**
H. Glenn Brauen and John E. Neilson, September 1988

SCS-TR-143 to 146 are unavailable.

SCS-TR-147 **On Transparently Modifying Users' Query Distributions**
B.J. Oommen and D.T.H. Ng, November 1988

SCS-TR-148 **An O(N Log N) Algorithm for Computing a Link Center in a Simple Polygon**
H.N. Djidjev, A. Lingas and J.-R. Sack, July 1988
Available in STACS 89, 6th Annual Symposium on Theoretical Aspects of Computer Science, Paderborn, FRG, February 16-18, 1989, Lecture Notes in Computer Science, Springer-Verlag No. 349

SCS-TR-149 **Smallscript: A User Programmable Framework Based on Smalltalk and Postscript**
Kevin Haaland and Dave Thomas, November 1988

SCS-TR-150 **A General Design Methodology for Dictionary Machines**
Frank Dehne and Nicola Santoro, February 1989

SCS-TR-151 **On Doubly Linked List ReOrganizing Heuristics**
D.T.H. Ng and B. John Oommen, February 1989

SCS-TR-152 **Implementing Data Structures on a Hypercube Multiprocessor, and Applications in Parallel Computational Geometry**
Frank Dehne and Andrew Rau-Chaplin, March 1989

SCS-TR-153 **The Use of Chi-Squared Statistics in Determining Dependence Trees**
R.S. Valiveti and B.J. Oommen, March 1989

SCS-TR-154 **Ideal List Organization for Stationary Environments**
B. John Oommen and David T.H. Ng, March 1989

SCS-TR-155 **Hot-Spot Contention in Binary Hypercube Networks**
Sivarama P. Dandamudi and Derek L. Eager, April 89

SCS-TR-156 **Some Issues in Hierarchical Interconnection Network Design**
Sivarama P. Dandamudi and Derek L. Eager, April 1989

SCS-TR-157 **Discretized Pursuit Linear Reward-Inaction Automata**
B.J. Oommen and Joseph K. Lanctot, April 1989

SCS-TR-158 (revised) **Parallel Fractional Cascading on a Hypercube Multiprocessor**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, May 1989 (Revised April 1990)

SCS-TR-159 **Epsilon-Optimal Stubborn Learning Mechanisms**
J.P.R. Christensen and B.J. Oommen, June 1989

SCS-TR-160 **Disassembling Two-Dimensional Composite Parts Via Translations**
Doron Nussbaum and Jörg-R. Sack, June 1989

SCS-TR-161 (revised) **Recognizing Sources of Random Strings**
R.S. Valiveti and B.J. Oommen, January 1990
Revised version of SCS-TR-161 "On the Data Analysis of Random Permutations and its Application to Source Recognition", published June 1989

SCS-TR-162 **An Adaptive Learning Solution to the Keyboard Optimization Problem**
B.J. Oommen, R.S. Valiveti and J. Zgierski, October 1989

SCS-TR-163 **Finding a Central Link Segment of a Simple Polygon in O(N Log N) Time**
L.G. Alexandrov, H.N. Djidjev, J.-R. Sack, October 1989

SCS-TR-164 **A Survey of Algorithms for Handling Permutation Groups**
M.D. Atkinson, January 1990

SCS-TR-165 **Key Exchange Using Chebychev Polynomials**
M.D. Atkinson and Vincenzo Acciaro, January 1990

SCS-TR-166 **Efficient Concurrency Control Protocols for B-tree Indexes**
Ekow J. Otoo, January 1990

SCS-TR-167 **A Hierarchical Stochastic Automaton Solution to the Object Partitioning Problem**
B.J. Oommen, January 1990

SCS-TR-168 **Adaptive List Organizing for Non-stationary Query Distributions. Part I: The Move-to-Front Rule**
R.S. Valiveti and B.J. Oommen, January 1990

SCS-TR-169 **Trade-Offs in Non-Reversing Diameter**
Hans L. Bodlaender, Gerard Tel and Nicola Santoro, February 1990

SCS-TR-170 **A Massively Parallel Knowledge-Base Server using a Hypercube Multiprocessor**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990

SCS-TR-171 **Parallel Processing of Quad Trees on the Hypercube (and PRAM)**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, April 1990

SCS-TR-172 **A Note on the Load Balancing Problem for Coarse Grained Hypercube Dictionary Machines**
Frank Dehne and Michel Gastaldo, May 1990

SCS-TR-173 **Self-Organizing Doubly-Linked Lists**
R.S. Valiveti and B.J. Oommen, May 1990

SCS-TR-174 **A Presortedness Metric for Ensembles of Data Sequences**
R.S. Valiveti and B.J. Oommen, May 1990

SCS-TR-175 **Separation of Graphs of Bounded Genus**
Ljudmil G. Aleksandrov and Hristo N. Djidjev, May 1990

SCS-TR-176 **Edge Separators of Planar and Outerplanar Graphs with Applications**
Krzystof Diks, Hristo N. Djidjev, Ondrej Sykora and Imrich Vrto, May 1990

SCS-TR-177 **Representing Partial Orders by Polygons and Circles in the Plane**
Jeffrey B. Sidney and Stuart J. Sidney, July 1990

SCS-TR-178 **Determining Stochastic Dependence for Normally Distributed Vectors Using the Chi-squared Metric**
R.S. Valivetei and B.J. Oommen, July 1990

SCS-TR-179 **Parallel Algorithms for Determining K-width- Connectivity in Binary Images**
Frank Dehne and Susanne E. Hambrusch, September 1990

SCS-TR-180 **A Workbench for Computational Geometry (WOCG)**
P. Epstein, A. Knight, J. May, T. Nguyen, and J.-R. Sack, September 1990

SCS-TR-181 **Adaptive Linear List Reorganization under a Generalized Query System**
R.S. Valiveti, B.J. Oommen and J.R. Zgierski, October 1990