Optimizing Self-organizing Lists-on-Lists Using Enhanced Object Partitioning

O. Ekaba Bisong, B. John Oommen

Optimizing Self-Organizing Lists-on-Lists using Enhanced Object Partitioning

O. Ekaba Bisong[1] ⋆ and B. John Oommen[2] ⋆⋆

School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6.
[1]ekaba.bisong@carleton.ca,[2]oommen@scs.carleton.ca

Abstract. The question of how to store, manage and access data has been central to the field of Computer Science, and is even more pertinent in these days when megabytes of data are being generated every second. This paper considers the problem of minimizing the cost of data retrieval from the most fundamental data structure, i.e., a Singly-Linked List (SLL). We consider a SLL in which the elements are accessed by a *Non-stationary* Environment (NSE) exhibiting so-called "Locality of Reference". We propose a solution to the problem by designing an "Adaptive" Data Structure (ADS) which is created by means of a composite of hierarchical data "sub"-structures to constitute the overall data structure. In this paper, we design an hierarchical Lists-on-Lists (LOLs) by assembling a SLL into an hierarchical scheme that results in a Singly-Linked List on Singly-Linked Lists (SLLs-on-SLLs) comprising of an outer-list and sublist context. The goal is that elements that are more likely to be accessed together are grouped within the same sub-context, while the sublists themselves are moved "en masse" towards the head of the list-context so as to minimize the overall access cost. This move is carried-out by employing the "de-facto" list re-organization schemes, i.e., the Move-To-Front (MTF) and Transposition (TR) rules. To achieve the clustering of elements within the sublists, we invoke the Object Migration Automaton (OMA) family of reinforcement schemes from the theory of Learning Automata (LA). They are introduced so as to capture the probabilistic dependence of the elements in the data structure as it receives query accesses from the Environment. In this paper, we show that SLLs-on-SLLs augmented with the Enhanced Object Migration Automaton (EOMA) minimizes the retrieval cost for elements in NSEs and are superior to the stand-alone MTF and TR schemes, *and also superior* to the OMA-augmented SLLs-on-SLLs operating in such Environments.

Keywords : *Learning Automata, "Adaptive" Data Structures, Hierarchical Singly-linked Lists, Object Migration Automaton*

⋆ *Member: IEEE*

⋆⋆ *Chancellor's Professor*; *Life Fellow: IEEE* and *Fellow: IAPR*. This author is also an *Adjunct Professor* with the University of Agder in Grimstad, Norway.

1 Introduction

The goal of this research endeavor is to further push the frontier of computational efficiency with regards to optimizing the speed of retrieving data from its data-structure. By considering the state-of-the-art, we address this issue by designing an *Adaptive* Data-Structure (ADS) that uses reinforcement learning schemes and their associated re-organization rules to update itself as it encounters query accesses from the Environment of interaction. The result of such a process is the subsequent minimization of the cost associated with query accesses.

To render the problem realistic, the Environments under consideration in this work are time-varying, i.e., they are *Non-stationary* Environments (NSEs), where the elements' access probabilities change with time. These Environments exhibit a particular dependency property called "Locality of Reference" where the events are probabilistically dependent on one another. In this work, we consider two such Environments, namely, the Periodic Switching Environment (PSE) and the Markovian Switching Environment (MSE).

The approach we adopt in designing these "Adaptive" Data Structures (ADSs) is to set up a hierarchy of data "sub"-structures. In this research, we employ hierarchical Lists-on-Lists (LOL) data-structures pioneered by Amer and Oommen in [1] for Singly-Linked Lists (SLLs) on Singly-Linked Lists. The LOL concept consists of an outer-list and many sublists, whose elements are called the outer and sublist contexts respectively. In this framework, elements that are more likely to be accessed together are grouped within the same sub-context, while the sublists are moved "en masse" towards the head of the list-context by following a re-organization rule.

In order to capture the probabilistic dependence of the elements in the data structure, based on the query accesses from the Environment, we employ a set of reinforcement learning schemes derived from the theory of Learning Automata (LA). These reinforcement schemes are variants of the so-called "Object Migration Automaton" (OMA).

The pioneering work of Amer and Oommen in [1] utilized the OMA algorithm to capture the probabilistic dependence of the queries coming from the Environment. The introduction of the OMA mitigated the static ordering of the sublists so that the elements can move freely from one sublist partition to another as the OMA learns the optimal sublist grouping. The addition of the OMA to the primitive hierarchical schemes, resulted in the MTF-MTF-OMA, and TR-MTF-OMA, where the third component in the triple is LA used.

Unfortunately, the OMA algorithm used in the literature [1] suffers from a deadlock[1] impediment that prevents it from converging to its optimal grouping. This is because the accessed element can be swapped from one sublist to another and then back to the original sublist. This deadlock phenomenon was mitigated by the Enhanced Object Migration Automaton (EOMA) in [8]. The EOMA forbids such "false alarm" swaps of elements between sublists, and also avoids pointless swaps between the various sublists themselves. Moreover, the EOMA

[1] Although this is referred to as a "deadlock" in the literature, it could probably, be better termed as a "livelock".

acknowledges that the sublist has converged when the elements are within a few of the most internal states. By this, it is certain about the identity of the elements that should constitute a sublist. This work augments the hierarchical SLLs-on-SLLs schemes with the EOMA. The design in this work yielded the MTF-MTF-EOMA, MTF-TR-EOMA, TR-MTF-EOMA and the TR-TR-EOMA schemes.

1.1 Contributions of this Paper

In summary, the novel contributions of this paper include:

- The design and implementation of the EOMA-enhanced SLLs-on-SLLs;
- The inclusion of the MTF-TR, and TR-TR enhanced hierarchical schemes as part of the SLLs-on-SLLs class of ADS design;
- Demonstrating the superiority of the EOMA-augmented hierarchical schemes to the MTF and TR rules when the outer-list context is the MTF;
- Demonstrating superiority of the EOMA-augmented hierarchical schemes to the original OMA-augmented schemes that pioneered the idea of a hierarchical LOL approach, with the "Periodic" and "UnPeriodic" versions;
- Showing that as the periodicity T increased in the PSE, the asymptotic cost is further minimized.

1.2 Outline of this Paper

Section 1 makes the case for minimizing retrieval costs in NSEs. Section 2 surveys[2] the theory of LA, which forms the framework for the EOMA used in learning the true partition of objects into groups. The section addresses the concept of "Locality of Reference" in NSEs and outlays the models of dependence used in this work to simulate state probabilities. Section 4 discusses the "de-facto" MTF and TR adaptive list organizing schemes for NSEs and why they constitute the primitive rules for the hierarchical LOL data-structures. Section 5 explains the rationale for using data "sub"-structures in designing the SLLs-on-SLLs giving rise to the MTF-MTF, MTF-TR, TR-MTF and TR-TR hierarchical schemes with static dependence capturing, making the case for an adaptive capturing mechanism. Section 6 explains the EOMA reinforcement algorithm and how it augments the Hierarchical SLLs. Section 7 presents the Results and Discussions, and Section 8 concludes the paper.

2 Theoretical Background

2.1 The Field of Learning Automata

An Automaton, by definition, models an autonomous agent, whose *behavior* manifests as a consequence of the interplay between a sequence of stimuli from

[2] Due to space limitations, it is obvious that the background material will be surveyed very briefly. More details of the various concepts concerning LA can be found in [12], and the details of the applications of the OMA, the EOMA and the state-of-the-art regarding ADSs etc. is found in the MCS thesis of the first author [5]. This thesis can be made available to the reader.

the Environment. The Automaton responds adaptively to the Environment and enforces the actions which fit the highest perceivable rewards from among a predetermined set of actions. Such an automaton is referred to as a *Learning Automaton* (LA) [5, 12]. Oommen and Ma proposed the Object Migrating Automata (OMA) [13, 14] to solve a special case of the OPP, namely the Equi-Partitioning Problem (EPP). The introduction of the OMA solution made real-life applications possible, because the prior art [20] was an order of magnitude slower. The OMA resolved the EPP both efficiently and accurately, and it could thus be easily incorporated into *many* real-life application domains [5–7].

2.2 The OMA

In the partitioning problem, the underlying distribution of the objects among the classes is unknown to the OMA, and its goal is to migrate the objects between its classes, using the incoming queries specified by an Environment, denoted as $\mathbb{E}$. This should be done in such a way that the partitioning error is minimized as the queries are encountered. Such an Environment and its associated query generating system can be characterized by three main parameters, namely, the number of objects, specified by W, the number of groups or partitions, specified by R, and a quantity 'p' , which is the probability specifying the certainty by which $\mathbb{E}$ pairs the elements in the query.

In our model, every query presented to the OMA by $\mathbb{E}$ consists of two objects, and this can be easily generalized for queries of larger sizes. Consider the case in which we have 3 classes with 3 objects, i.e., a system which has a total of 9 objects. This is depicted in Figure 1. $\mathbb{E}$ randomly selects an initial class with probability $\frac{1}{R}$, and it then chooses the first object in the query from it, say, q_1. The second element of the pair, q_2, is then chosen with the probability p from the same class, and with the probability $(1 - p)$ from one of the other classes uniformly, each of them being chosen with the probability of $\frac{1}{R-1}$.

In Figure 1, the three classes are named G_1, G_2 and G_3, and the objects inside them are represented by integers in $\{1, \cdots, 9\}$. The original distribution of the objects between the classes is shown in Figure 1, at the extreme left. This is the true unknown state of nature, i.e., Ω^*. The OMA is initialized in a purely random manner by the numbers within the range. This step is depicted at the right of Figure 1, and Ω_0 indicates the initial state of the OMA. At every iteration, a pair given by $\mathbb{E}$ is processed by the OMA, and it performs a learning step towards its convergence. The goal of the algorithm is for it to converge to a state, say Ω^+. In an optimal setting, we would hope that Ω^+ is identical to Ω^*.

The OMA is a *Fixed Structure Stochastic Automata (FSSA)* designed to solve the EPP. It is defined as a quintuple with R actions, each of which represents a specific class, and for every action there exists a fixed number of states, N. Every abstract object from the set $\mathcal{O}$ resides in a state identified by a state number, and it can move from one state to another, or migrate from one group[3] to another. Thus, if the abstract object O_i is located in state ξ_i belonging to a specific group (an action or class) α_k, we say that O_i is assigned to class k.

[3] As in the field of LA, we use the terms "action", "class" and "group" synonymously.

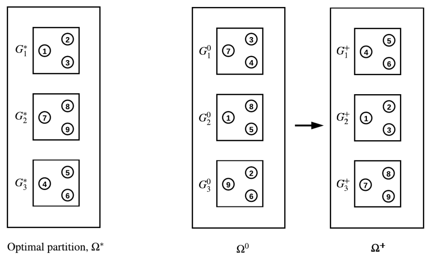

Fig. 1: A figure describing the partitioning of the objects.

If two objects O_i and O_j happen to be in the same class and the OMA receives a query $\langle A_i, A_j \rangle$, they will be jointly rewarded by the Environment. Otherwise, they will be penalized. Our task is to formalize the movements of the $\{O_i\}$ on reward and penalty.

For every action α_k, there is a set of states $\{\phi_{k1}, \cdots, \phi_{kN}\}$, where N is the fixed depth of the memory, and where $1 \leq k \leq R$ represents the number of desired classes. We also assume that ϕ_{k1} is the most internal state and that ϕ_{kN} is the boundary state for the corresponding action. The reward and penalty responses are defined as follows:

Reward: Given a pair of physical objects presented as a query $\langle A_i, A_j \rangle$, if both O_i,and O_j happen to be in the same class α_k, the reward scenario is enforced, and they are both moved one step toward the actions's most internal state[4], ϕ_{k1}.

Penalty: If, however, they are in different classes, α_k and α_m, (i.e., O_i, is in state ξ_i where $\xi_i \in \{\phi_{k1}, \cdots, \phi_{kN}\}$ and O_j, is in state ξ_j where $\xi_j \in \{\phi_{m1}, \cdots, \phi_{mN}\}$) they are moved away from ϕ_{k1} and ϕ_{m1} as follows:

1. If $\xi_i \neq \phi_{kN}$ and $\xi_j \neq \phi_{mN}$, then we move O_i and O_j one state toward ϕ_{kN} and ϕ_{mN}, respectively.
2. If $\xi_i = \phi_{kN}$ or $\xi_j = \phi_{mN}$ but not both (i.e., only one of these abstract objects is in the boundry state), the object which is not in the boundary state, say O_i, is moved towards *its* boundary state. Simultaneously, the object that is in the boundary state, O_j, is moved to the boundary state of O_j. Since this reallocation will result in an excess of objects in α_k, we choose one of the objects in α_k (which is not accessed) and move it to the boundary state of α_m. In this case, we choose the object nearest to the boundary state of ξ_i.
3. If $\xi_i = \phi_{kN}$ and $\xi_j = \phi_{mN}$ (both objects are in the boundary states), one object, say O_i, will be moved to the boundary state of α_m. Since this reallocation, will again, result in an excess of objects in α_m, we choose one of the objects in α_m (which is not accessed) and move it to the boundary state of α_k. In this case, we choose the object nearest to the boundary state of ξ_j.

The above rules are figuratively shown in [5]. The algorithm invokes the procedures "*ProcessReward*" and "*ProcessPenalty*" given algorithmically in [5].

[4] The actual figure describing the schematic of transitions of the LA is given in [5], and is omitted here in the interest of space.

3 Environments with Locality of Reference

Non-Stationary Environments (NSEs) deal primarily with learning in settings that change with time. Thus, in a NSE, $c_i(n)$ changes with time.

In the context of an ADS, this variation affects the expected query cost because the Environment exhibits so-called "Locality of Reference", or is characterized by dependent accesses. Locality of Reference occurs when there exists a probabilistic dependence between the consecutive queries [2]. Thus, there is a considerably small number of unrelated queries within a segment of the accesses.

Given a set of n distinct elements, if we split it into k disjoint and equal partitions with m elements where $n = k.m$, the k subsets can be considered to be local or "sub"-contexts. If the elements within a sub-context k_i exhibit Locality of Reference, it implies that if an element from set k_i is queried at time t, there exists a high likelihood that the next queried element will also arrive from the same set k_i. Thus, the Environment itself can be modeled to have a finite set of states $\{Q_i | 1 \leq i \leq k\}$, and the dependent model defines the transition from one Environmental state to another.

Learning schemes with fixed policies may become non-expedient over time, rendering them inadequate for such Environments. The goal is to have schemes which possess enough flexibility to choose actions that minimize the expected penalty. Two models of NSEs critical to this research are the Markovian Switching Environments (MSEs), and the Periodic Switching Environments (PSEs).

Markovian Switching Environments (MSEs): Consider an Environment with 128 distinct records, that are divided into $k = 4$ subsets, with 32 contiguous elements in each subset. In such a case, the set of states $\{Q_1, Q_2, Q_3, Q_4\}$, could be $Q_1 = \{1 \ldots 32\}$, $Q_2 = \{33 \ldots 64\}$, $Q_3 = \{65 \ldots 96\}$, and $Q_4 = \{97 \ldots 128\}$. The Markovian Switching Environment (MSE) models each subset as the states of the Environment as the states of a Markov chain. If the probability of the Environment choosing a record from the current subset is 0.9, the probability of switching to another subset is equally divided among the other three subsets. After a query is generated from Q_1, the Environment remains in that state with probability α and moves to a different state with probability $\frac{1-\alpha}{k-1}$.

Periodic Switching Environments (PSEs): The Periodic Switching Environment (PSE) on the other hand changes the state of the Environment in a round-robin fashion, i.e., after every T queries, the Environment changes state from Q_i to $Q_{i+1 \bmod k}$. This implies that each set of T consecutive queries belong to the same sub-context. Further, there are two variations that define the PSE model; the first is when the data structure is aware of the change of state in the query generator (*"Periodic"*), and the other is when the data structure is unaware of the state change (*"UnPeriodic"*). Understandably, the performance of the scheme is better when the ADS is aware of the Environment's state change.

3.1 Models of Dependence

The Environment generates queries according to a probability distribution. This work considered five different types of query distributions, namely, the Zipf, Eighty-Twenty, Lokta, Exponential and Linear distributions. For a given list of size J, divided into k sublists, with each sublist containing $\frac{J}{k}$ elements, the probability distribution $\{s_i\}$ where $1 \leq i \leq m$ describes the query accesses for the elements in the subset k. Thus, the total probability mass for the accesses in each group is the same, and the distribution within each group has the specified distribution. The distributions for these generators are described below.

1. **The Zipf distribution**: The access probabilities for the Zipf query generator is given as: $s_i = \frac{1}{iH_m}, \quad \text{for} \quad 1 \leq i \leq m$, where H_m is the m^{th} Harmonic number and defined as $H_m = \sum_{j=1}^{m}(\frac{1}{j})$. The Zipf distribution is the most commonly-used one for modelling real-life access probabilities.
2. **The 80-20 distribution**: The access probabilities for the 80-20 query generator is given as: $s_i = \frac{1}{i^{(1-\theta)}H_m^{(1-\theta)}}, \quad \text{for} \quad 1 \leq i \leq m \quad \text{and} \quad \theta = \frac{\log 0.80}{\log 0.20} \approx 0.1386$, where $H_m^{(1-\theta)}$ is the m^{th} Harmonic number of order $(1-\theta)$, and is given by $\sum_{j=1}^{m}(\frac{1}{j^{(1-\theta)}})$.
3. **The Lotka distribution**: The access probabilities for the Lotka query generator is given as: $s_i = \frac{1}{i^2 H_m^2}, \quad \text{for} \quad 1 \leq i \leq m$, where H_m^2 is the m^{th} harmonic number of order 2, and is given by $\sum_{j=1}^{m}(\frac{1}{j^2})$.
4. **The Exponential distribution**: The access probabilities for the Exponential query generator is given as: $s_i = \frac{1}{2^i K}, \quad \text{for} \quad 1 \leq i \leq m$, where $K = \sum_{j=1}^{m}(\frac{1}{2^j})$.
5. **The Linear distribution**: The access probabilities for the Linear query generator is given as: $s_i = K(m - i + 1), \quad \text{for} \quad 1 \leq i \leq m$, where K is determined as the constant which normalizes the $\{s_i\}$ to be a distribution.

A rationale for conducting the simulations with these query distributions is that, for the most part, they result in "L-shaped" graphs which assign high probabilities to a small number of the sublist elements. This is true for the Exponential and Lotka distribution, and to an extent, for the Zipf distribution.

4 Adaptive Lists-on-Lists (LOL)

Self-organization is the ability for a list to re-order its constituent elements in response to queries from the underlying query system, that serves as an Environment. The probability distribution of the query accesses is unknown to the list re-organization algorithm. The goal of this re-organization, among others, is to minimize the asymptotic cost or access-time of record retrieval.

The cost models employed in evaluating list access costs are the asymptotic cost, which is the ensemble mean of the final time-average cost after a convergence threshold, and the amortized cost, which is the mean overall query costs [4, 9, 17]. In studying ADSs, one assumes that the Environment will not request a record absent from the list, and that each record is retrieved at least once [9].

The simplest and yet most prominent Adaptive Lists are the Move-to-Front (MTF) and the Transposition rule (TR) adaptive schemes. The MTF update heuristic moves the queried element to the front of the list. In the TR, a queried record (if not at the front) is moved one position towards the front of the list.

For Environments with Locality of Reference, the MTF and TR have been shown to be superior to other deterministic schemes such as FC, MRI(0) and TS(0) [2]. Further, the time and space complexities involved in implementing other composite MTF and TR schemes (the details of which are omitted here) such as the MHD(k) [15], the POS(k) and the SWITCH(k) [18] and other probabilistic approaches such as the *SPLIT* algorithm [11], the JUMP [10], MTF2, Randomized MTF (RMTF), and the Randomized move ahead (RMHD) schemes [2] render most of them impractical for real-world settings.

5 Hierarchical Data "Sub"-structures

The novel idea that we propose is to combine the MTF and TR rules to take advantage of the quick updates of the MTF rule, and the asymptotically stable convergence of the TR rule, in designing the improved hierarchical strategies. The concept of a hierarchical data "sub"-structure involves dividing a list of size J into k sublists. A re-organization strategy is then hierarchically applied to the list by first considering the elements within the sublist (also called the sub-context) and then operating over the sublists (or sub-contexts) themselves.

As mentioned earlier, the primitive re-organization strategies involved are the MTF and TR rules. When used in a hierarchical scheme, this yields the MTF-preceding-MTF, (MTF-MTF), MTF-preceding-TR, (MTF-TR), TR-preceding-MTF, (TR-MTF), and TR-preceding-TR, (TR-TR) schemes. For example, in the case of MTF-TR, the element within a sub-context is first moved to the front of the list, and then the sub-context is moved to the front of the list context.

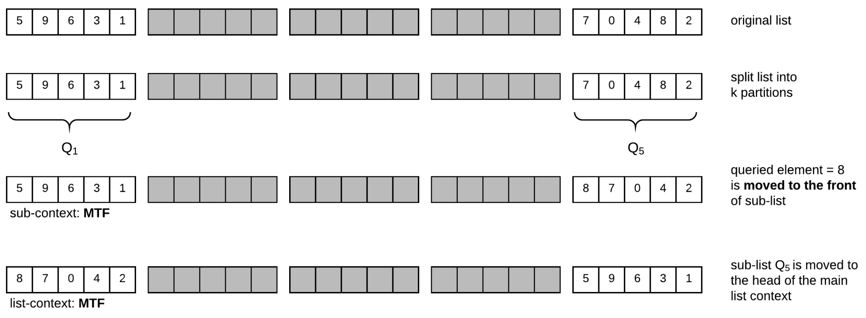
5 9 6 3 1
7 0 4 8 2
original list
split list into k partitions
Q₁
Q₅
queried element = 8 is moved to the front of sub-list
8 7 0 4 2
sub-context: MTF
sub-list Q₅ is moved to the head of the main list context
list-context: MTF

Fig. 2: A diagrammatic description of the MTF-MTF Hierarchical scheme

The hierarchical schemes on their own, however, perform worse than stand-alone schemes such as the MTF and TF in NSEs. The drawback is due to the fact that the hierarchical schemes make an assumption that the elements within a specific *a priori* sub-context have a probabilistic dependence. But this is often not the case as the elements in the list initially are ordered in an arbitrary manner. To mitigate this shortcoming, we will later argue that we must design a mechanism to adaptively group the elements that have a probabilistic dependence within the same sub-context.

6 EOMA-Augmented Hierarchical SLLs-on-SLLs

The "Enhanced" OMA (EOMA) is an upgraded embodiment of the OMA algorithm proposed by the authors of [8] to mitigate the susceptibility of the OMA algorithm to a "deadlock situation" which prevents the algorithm from converging to the objects' optimal partitioning. The deadlock condition is actually exacerbated when the algorithm is interacting with a near-optimal Environment (e.g., when $p = 0.9$) by considerably slowing down the convergence rate even if the problem complexity is small.

The deadlock phenomenon occurs when there is a query pair $\langle O_i, O_j \rangle$ in a stream of query pairs belonging to different actions, α_m and α_k. If one object is in the boundary state of its action, and the other is not, the query pairs are prevented from converging to their optimal ordering, and this can lead to an "infinite" loop scenario. To mitigate this, if there exists an object in the boundary state of the group containing O_j, the EOMA swaps O_i with the object in this boundary state (Figure 3). Otherwise, the update is identical to the OMA.

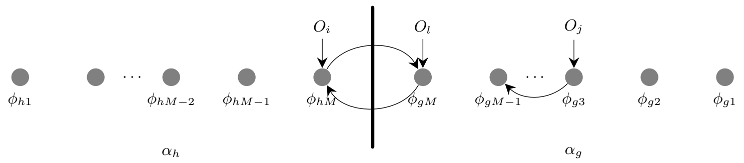

Fig. 3: Resolving the deadlock scenario with EOMA for the case when only one object is in the boundary state.

The EOMA also redefines the concept of the convergence condition so as to reduce the algorithms vulnerability to divergent queries. This modification designates the two-innermost states as the "final" states, as opposed to just the innermost state in the vanilla OMA. A marginally superior solution specifies a parameter m, to designate the m innermost states of each action to be the convergence condition. More details on the EOMA are found in [8, 16].

The augmentation of the hierarchical SLLs based on the EOMA reinforcement scheme results in a new set of hierarchical strategies, namely, the MTF-MTF-EOMA, MTF-TR-EOMA, TR-MTF-EOMA and the TR-TR-EOMA.

7 Results and Discussions

The experimental setup involved a list of size 128, split into k sublists, where $k \in \{2, 4, 8, 16, 32, 64\}$. In the MSE, the probability of subsequent query accesses coming from the same sublist, α, was set to 0.9, while the PSE had the hyper-parameter for the number of queries to arrive from the query space before switching to another pattern, $T = 30$. For all the results reported in this section, the simulation setup involved an ensemble of 10 experiments, each constituting 300,000 query accesses. In the interest of brevity, we report the results for $k = 8$.

Scheme	Zipf	80-20	Lotka	Exponential	Linear
MTF	43.35	43.76	39.30	8.72	43.60
TR	55.44	56.74	48.25	10.52	56.79
MTF-MTF-EOMA	19.14	19.23	18.70	12.34	19.31
MTF-TR-EOMA	27.80	27.77	27.17	16.89	28.04
TR-MTF-EOMA	18.84	18.99	18.37	12.87	18.96
TR-TR-EOMA	27.55	27.62	26.96	17.17	27.70
MTF	43.25	43.82	39.17	8.71	43.64
TR	55.85	56.96	48.66	10.93	57.26
MTF-MTF-EOMA	19.35	19.40	19.26	12.90	19.45
MTF-TR-EOMA	27.93	28.02	27.54	16.57	28.08
TR-MTF-EOMA	19.09	19.18	19.07	13.35	19.18
TR-TR-EOMA	27.72	27.80	27.25	17.10	27.87

Table 1: Asymptotic (top) and Amortized (bottom) costs in **MSE** with $\alpha = 0.9$ and $k = 8$.

Scheme	Zipf	80-20	Lotka	Exponential	Linear
MTF	49.64	50.24	44.52	8.46	50.08
TR	55.65	56.91	48.51	11.18	57.19
MTF-MTF-EOMA	14.63	14.70	14.12	8.59	14.72
MTF-TR-EOMA	25.82	25.90	25.32	13.88	25.92
TR-MTF-EOMA	14.39	14.49	13.76	8.92	14.50
TR-TR-EOMA	25.58	25.69	24.97	13.70	25.70
MTF-MTF-EOMA-P	7.16	7.24	6.66	6.14	7.26
MTF-MTF-EOMA-UP	7.69	7.78	7.19	8.90	7.79
MTF	49.62	50.23	44.53	8.48	50.06
TR	56.09	57.28	48.91	11.58	57.60
MTF-MTF-EOMA	14.76	14.84	14.31	8.68	14.84
MTF-TR-EOMA	25.93	26.01	25.44	12.49	26.02
TR-MTF-EOMA	14.54	14.62	14.03	9.69	14.63
TR-TR-EOMA	25.71	25.80	25.12	13.11	25.80
MTF-MTF-EOMA-P	7.28	7.38	6.82	7.53	7.40
MTF-MTF-EOMA-UP	7.86	7.95	7.48	10.57	7.92

Table 2: Asymptotic (top) and Amortized (bottom) costs in **PSE** with $T = 30$ and $k = 8$.

From the simulation results in Table 1, with $k = 8$, we observed that for the MSE, the hierarchical schemes with EOMA generally outperformed their stand-alone counterparts in both the asymptotic (top of the table) and amortized (bottom of the table) costs for all instances except the Exponential distribution. In the Exponential distribution, the stand-alone MTF and TR schemes had a slightly superior performance to the EOMA-augmented hierarchical schemes. This is because the MTF and TR rules are competitive in Environments with an L-shaped curve such as the Exponential distribution, because they assign higher probabilities to a small subset of the elements in the query system.

Table 2 compared the performance of the EOMA-augmented hierarchical schemes with the stand-alone MTF and TR schemes in PSEs when the number of sublists $k = 8$. Here we saw that the hierarchical schemes with the EOMA performed better than their stand-alone counterparts, except for the Exponential distribution with the same reason as in the MSE. However, when the concept of "periodicity" was introduced into the EOMA-augmented hierarchical schemes, the search cost is an order of magnitude superior to other schemes.

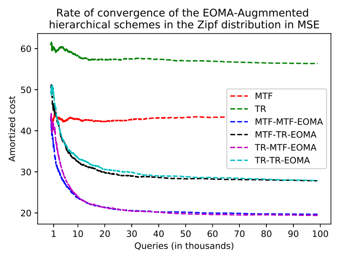

Fig. 4: Rate of convergence of the first 100,000 queries for the stand-alone and the EOMA-augmented hierarchical schemes in a MSE.

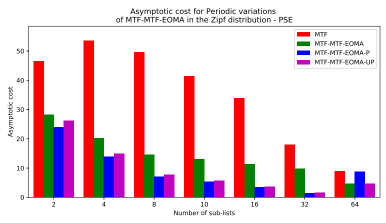

Fig. 5: Asymptotic cost of Periodic variations of MTF-MTF-EOMA in the Zipf distribution. PSE with period $T = 30$ and $k = \{k : 2, 4, 8, 10, 16, 32, 64\}$.

From Figure 4, it is easy to observe that from the first few queries, all the EOMA-augmented hierarchical schemes perform better than the TR rule in minimizing the amortized cost. Right about the $10,000^{th}$ query, the EOMA-augmented hierarchical schemes catches-up with the MTF rule in terms of performance and from thereon boasts a far superior performance compared to the MTF. A key observation is that the EOMA-augmented hierarchical schemes appeared to converge after about 30,000 queries. As opposed to this, the MTF and TR schemes quickly plateaued with no additional gains in performance with extended interactions with the Environment. Although Figure 4 shows the rate of convergence for the Zipf distribution, the observed phenomena were similar for the 80-20, Lotka, Exponential and Linear distributions.

In Periodic Environments, the hierarchical schemes that incorporated the EOMA were able to boost their performance if they possessed an insight into the period, T, of the Environment (see Figure 5). This implied that the schemes could preempt the EOMA's ordering by moving the first sublist to the end of the list after T queries. This move was predicated on the observation that the elements from the completed query space would not be requested again until after $(k-1)T$ queries. Schemes with such a prior awareness of period T are referred to by including the prefix "Periodic", leading to the MTF-MTF-EOMA-Periodic, MTF-TR-EOMA-Periodic, TR-MTF-EOMA-Periodic and TR-TR-EOMA- Periodic respectively.

Also, without explicitly knowing the value of T, the hierarchical schemes were able to infer the period, T, of the Environment by moving the first sublist to the end of the list if two successive queries to the EOMA were not in the same group. These periodic variations were suffixed by "UnknownPeriod", yielding the MTF-MTF-EOMA-UnknownPeriod, MTF-TR-EOMA-UnknownPeriod, TR-MTF-EOMA-UnknownPeriod and TR-TR-EOMA-UnknownPeriod schemes.

8 Conclusion

In this research we studied the area of Adaptive Data Structures (ADSs) and considered the relatively novel concept of having lists whose basic primitive elements were themselves sub-lists, with ADS operations being done on the elements *and* on the sublists. In order to break the static arrangement of their sublists, we incorporated the EOMA (from the field of Learning Automata (LA)) into the hierarchical schemes. The EOMA enabled the hierarchical schemes to capture the probabilistic dependence ordering of the query accesses from the Environment. Further, the paper discussed the performance of the MTF-MTF-EOMA, MTF-TR-EOMA, TR-MTF-EOMA and the TR-TR-EOMA for various sublist values of k, various distributions, and various types of non-stationarity.

The overall observation that we could make is that the MTF-MTF-EOMA and the TR-MTF-EOMA perform better than the MTF-TR-EOMA and the TR-TR-EOMA. One can almost categorically state that, the schemes having the TR as its outer-list re-organization strategy were inferior to the MTF, when we compared their asymptotic and amortized costs. However, the observed poor performance of the MTF-TR-EOMA and the TR-TR-EOMA schemes as k increases, were mitigated in the PSEs when a knowledge of the period, T, was incorporated into the hierarchical scheme.

A study of the various graphs that we have obtained seems to imply that there is a way by which we can group the various *schemes themselves* using a higher level statistical analysis. Such a study remains open.

References

1. A. Amer. Adaptive list organizing strategies for non-stationary distributions. 2004.
2. R. Bachrach, R. El-Yaniv, and M. Reinstadtler. On the competitive theory and practice of online list accessing algorithms. *Algorithmica*, 32(2):201–245, 2002.
3. R. Bellman. *Adaptive control processes: A guided tour*. Princeton University Press, Princeton, N.J., 1961.
4. J. L. Bentley and C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.
5. E. O. Bisong. *On Designing Adaptive Data Structures with Adaptive Data "Sub"-Structures*. MCS thesis, Carleton University, Ottawa, 2018..
6. E. Fayyoumi and B. J. Oommen. A fixed structure learning automaton micro-aggregation technique for secure statistical databases. In *International Conference on Privacy in Statistical Databases*, pages 114–128. Springer, 2006.
7. E. Fayyoumi and B. J. Oommen. Achieving microaggregation for secure statistical databases using fixed-structure partitioning-based learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(5):1192–1205, 2009.
8. W. Gale, S. Das, and C. T. Yu. Improvements to an algorithm for equipartitioning. *IEEE Transactions on Computers*, 39(5):706–710, 1990.
9. J. H. Hester and D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys (CSUR)*, 17(3):295–311, 1985.
10. J. H. Hester and D. S. Hirschberg. Self-organizing search lists using probabilistic back-pointers. *Communications of the ACM*, 30(12):1074–1079, 1987.
11. S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991.
12. K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*, Courier Corporation, 2012.
13. B. J. Oommen and D. C. Y. Ma. Deterministic learning automata solutions to the equipartitioning problem. *IEEE Transactions on Computers*, 37(1):2–14, 1988.
14. B. J. Oommen and D. C. Y. Ma. Stochastic automata solutions to the object partioning problem. *The Comput. Journal*, Vol. 35, A105A120, 1992.
15. R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, 19(2):63–67, 1976.
16. A. Shirvani. *Novel Solutions and Applications of the Object Partitioning Problem*. PhD thesis, Carleton University, Ottawa, 2018.
17. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
18. A. M. Tenenbaum and R. M. Nemes. Two spectra of self-organizing sequential search algorithms. *SIAM Journal on Computing*, 11(3):557–566, 1982.
19. C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.
20. C. Yu, M. Siu, K. Lam, and F. Tai, "Adaptive clustering schemes: General framework," in *Proc. of the IEEE COMPSAC Conference*, 1981, pp. 81–89.