A HIERARCHICAL STOCHASTIC AUTOMATON SOLUTION TO THE OBJECT PARTITIONING PROBLEM+

B.J. Oommen

SCS-TR-167, JANUARY 1990

+Partially supported by the Natural Sciences and Engineering Research Council of Canada.

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

A HIERARCHICAL STOCHASTIC AUTOMATON SOLUTION TO THE OBJECT PARTITIONING PROBLEM[+]

B. J. Oommen
School of Computer Science
Carleton University
Ottawa, K1S 5B6
CANADA.

ABSTRACT

In this paper we study the object partitioning problem in which the elements of a set Ω are accessed by users according to an unknown partitioning Θ. The joint access probabilities of the objects are unknown and furthermore, the objects are accessed in groups of unknown size which may or may not be equal. The object partitioning problem involves partitioning the objects into a partition Π isomorphic to Θ in such a way that the objects accessed more frequently together are located in the same class. It is well known that this problem is NP-hard [22,23]. In this paper, we propose a fast hierarchical stochastic learning automaton solution to the problem. Unlike the previously reported automaton solutions [24] the number of computations per iteration required by this method is logarithmic in the number of objects to be partitioned. Experimentally, this solution converges much faster than the best known algorithm in the literature which does not use learning automata [22,23].

Keywords : Learning Systems, Applications of learning, Adaptive Learning, Object Partitioning, Distributed File Allocation, Intelligent Partitioning of Attributes and Records.

+Partially Supported by the Natural Sciences and Engineering Research Council of Canada.

I. INTRODUCTION

Let $\Omega = \{A_1, ..., A_W\}$ be a set of W objects. We intend to partition Ω into R classes $\Pi=\{\Pi_1, ..., \Pi_R\}$ in such a way that the objects that are accessed (used) more frequently together lie in the same class. This problem is called the Object Partitioning Problem (OPP). Indeed, to render the problem non-trivial, we assume that their joint access probabilities are unknown.

In order to appreciate the problem and to motivate a need for studying it, we shall first informally state a variety of applications where the problem is encountered. Consider a library environment [17], in which the set Ω represents the set of disciplines (e.g. history, geography, biology, etc.) and the partitioning represents the physical location of the books and documents in these individual disciplines. Assuming that a person working in geography required books in geology more frequently than books in mathematics, it is desirable that books in the former two disciplines (i.e., geography and geology) be located on the same floor rather than books in the disciplines geography and mathematics. But apart from such conceptual applications, the OPP is also encountered in a number of computer applications. We list some of these below.

(i) Let Ω be a set of attributes belonging to a normalized relation. Because of the size of Ω, it may be that these attributes cannot all be physically incorporated in one single relation. In such a case it is desirable to partition the attributes into sub-relations in such a way that the attributes accessed simultaneously in a query are located in the same sub-relation [4,5,22]. This is indeed the traditional attribute partitioning problem.

(ii) Let Ω be a set of records stored on a disk. The record allocation problem involves partitioning these records so that the jointly accessed records are available on the same block (or track) of the disk. A solution to the OPP can directly be used to solve this problem. So too, can segment clustering in IMS trees [22,23] be considered as a record clustering problem [18].

(iii) In a distributed data base, the files that are distributed can be located at various sites. In this case, it is desirable that files which are jointly accessed more frequently together are physically located in the same geographical site. Obviously this would minimize expensive communication costs [9,23].

Various other applications of the OPP are also found in the literature [8,20,21]. Our aim is to obtain a general solution to the OPP which, with minor modifications, lends itself as a solution to any of the above mentioned problems.

A basic underlying assumption of the OPP is that the users who are accessing the elements of Ω are accessing them based on an underlying partition Θ. This, in a sense, is determined by the user's query (or access) patterns and is referred to as the grouping imposed on Ω. As far as the person who is solving the OPP is concerned, we assume that this grouping is unknown. A solution to the OPP aims to partition Ω into classes that mimic the underlying grouping.

To formalize the problem, let $\Omega = \{ A_1, \ldots, A_W\}$ be the set of objects to be partitioned. Let Θ be an unknown partitioning of Ω, where

$$\Theta = \{ \theta_1, \theta_2, \ldots, \theta_S\} \text{ such that for all } 1 \le i,j \le S,$$
$$\theta_i \cap \theta_j = \phi, \text{ and}$$
$$\bigcup_{i=1}^{S} \theta_i = \Omega.$$

Let Ψ be an infinite stream of queries, where each query is a pair (A_i, A_j), $i \neq j$, such that if $Pr(A_i,A_j)$ is the probability of A_i and A_j being accessed together, then

$$Pr(A_i, A_j) >> Pr(A_i, A_k) \quad \text{if} \quad A_i, A_j \in \theta_u \text{ and } A_k \in \theta_v \text{ for } u \neq v.$$

Thus, the queries come in such a way that the probability of the user concurrently accessing objects in the same group is much larger than the probability of the user accessing two objects from distinct groups.

The aim of the OPP is to learn the partition Θ. Indeed, by processing Ψ we intend to obtain a partitioning $\Pi = \{\Pi_1, \ldots, \Pi_R\}$ of Ω such that :

(i) $S = R$

(ii) $\Pi_i \cap \Pi_j = \phi$ for all $1 \le i, j \le R$

(iii) $\bigcup_{i=1}^{R} \Pi_i = \Omega$, and,

(iv) There exists an isomorphism $m(i)$, where $1 \le m(i) \le S$, such that for all i, $\Pi_i = \theta_{m(i)}$.

Observe that the OPP assumes that S, the number of groups in the user's partition, is unknown.

Even the simplified versions of the Object Partitioning Problem (OPP) have been shown to be NP-hard [22,23] due to the exponential growth in the number of partitions of the objects. Our aim is to try to use an adaptive scheme which converges to the exact underlying partition with a relatively high probability.

The most obvious method of solving this problem is to process the queries and to maintain a frequency count of the accesses for various combinations of the objects. Since the number of combinations is exponential, the use of such statistics is impractical and infeasible. Observe too that strategies (such as the Move-to-Front, Transposition, Move-to-Root, Splaying etc.) which adaptively reorganize lists and trees [1-3,6,10,15,16] are inappropriate for the OPP because the assumption that the objects are accessed independently is quite meaningless in this context.

Further, the question of physically moving the objects after every query (or access) seems quite unreasonable, especially in the case of attribute partitioning or distributed file allocation. Thus, the the research in this field has taken alternate directions.

In this paper we shall present some hierarchical learning automata solutions to the OPP. Learning automata were introduced in the literature about three decades ago and have since then been used to model biological learning systems and also to determine the optimal action which an environment offers. The learning is achieved by the automaton interacting with a random environment and processing the responses that the environment gives. Various applications of learning automata have been reported in the literature and among these are parameter optimization, statistical decision making and telephone routing [7,11,12]. The reader is referred to the following review paper by Narendra and Thathachar [11] and two comprehensive books in the field [7,12] for an overview of the families and applications of learning automata.

The learning process of an automaton can be described as follows: The automaton is offered a set of actions by the environment with which it interacts, and it is constrained to choose one of these actions. When an action is chosen, the automaton is either rewarded or penalized by the environment, and the environment provides this response stochastically. A learning automaton is one which learns the optimal action, i.e. the action which has the minimum penalty probability, and eventually chooses this action more frequently than the others. Although a few extremely fast learning automata solutions to the OPP have been reported in the literature [24], these schemes possess inherent drawbacks which can be quite prominent. These drawbacks are overcome here by presenting a new hierarchical automaton solution for the OPP.

For the rest of this section we briefly discuss the fundamentals of learning automata. In the next section we shall discuss the BAM and some existing automata solutions to the OPP. We shall then present the non-hierarchical solution to the OPP which will serve as the as the basis for the hierarchical automata solution proposed. We conclude this paper with a presentation of the experimental results which we have obtained.

I.1 Fundamentals of Learning Automata

Stochastic learning automata can be classified into two main classes :

(a) Fixed Structure Stochastic Automata (FSSA), and,

(b) Variable Structure Stochastic Automata (VSSA).

Some examples of the former type are the Tsetlin, Krinsky and Krylov automata [19]. Automata of the latter type are completely defined in terms of action probability updating rules which are either of a continuous [7,11,12] or discrete nature [13].

The automaton-environment interaction is described as follows. The automaton selects an action based on its structure and state, and this action serves as the input to the environment which gives out a response $\beta(n) \in \{0,1\}$ at time 'n'. The environment penalizes (i.e., $\beta(n) = 1$) the automaton with the penalty c_i, where,

$$c_i = \Pr [\beta(n) = 1 \mid \alpha(n) = \alpha_i] \qquad (i = 1,..,R).$$

Thus the environment characteristics are specified by the set of penalty probabilities $\{c_i\}$ (i = 1,..,R). On the basis of the response $\beta(n)$ the automaton performs some internal computations and a new action is chosen at time (n+1). Of course, to render the problem meaningful, the $\{c_i\}$ are unknown initially and it is desired that as a result of interaction with the environment the automaton arrives at the action which gives it the minimum penalty response. Let c_L be the minimum penalty probability, and let $P_i(n) = \Pr [\alpha(n) = \alpha_i]$. Then, the convergence of $P_L(n)$ to unity and the convergence of $P_i(n)$ to zero for $i \neq L$ achieves this result. Automata are designed with this solution in view.

As mentioned earlier, a Variable Structure Stochastic Automaton (VSSA) is essentially [7,11,12] an updating rule by which the action probability vector at the $(n+1)^{st}$ time instant is computed using the probability vector at the n^{th} time instant and the automaton-environment interaction at the n^{th} time instant. Thus, if $\mathbf{P}(n)$ is the action probability vector at time 'n', the VSSA is fully defined by specifying a function H, where,

$$\mathbf{P}(n+1) = H(\mathbf{P}(n), \alpha(n), \beta(n)).$$

Let E[M(n)] be the expected penalty at time 'n'. An automaton is said to learn expediently if, as time tends towards infinity, the expected penalty is less than the initial penalty. It is said to be optimal if E[M(n)] asymptotically equals the minimum penalty probability as time approaches infinity. Finally, it is ε-optimal if in the limit, for any arbitrary $\varepsilon > 0$, $E[M(n)] < c_L + \varepsilon$, where c_L is the minimum penalty probability. This could be achieved by a suitable choice of some parameter of the scheme, such as the multiplying constants characterizing the updating functions.

II. PREVIOUS SOLUTIONS TO THE OPP

To present our solution in the right perspective, we shall first present a brief summary of the some of the adaptive solutions to the problem. This is done, not only for the sake of completeness, but also because, in this present solution, we have tried to incorporate various advantages of these solutions, and have simultaneously tried to eliminate the disadvantages that the previous schemes have possessed.

The earliest adaptive solution for the OPP is probably due to Hammer *et al* [4,5]. This solution is clever algorithm, which is conceptually a "hill-climbing" technique that tries to converge to the optimal partitioning by using a variety of heuristics. The best known adaptive solution, which does not use learning automata, is the one due to Yu *et al* [22,23] which has been clearly demonstrated to be superior to the algorithm in [4].

The method due to Yu *et. al.* [22,23] is called the Basic Adaptive Method (BAM), and is a truly ingenious scheme. The three major advantages of the BAM are (i) the method is adaptive, (ii) it does not involve computing any query statistics and (iii) as opposed to traditional data reorganization strategies (such as the Move-to-Front, Transposition, Move-to-Root, Splaying etc.), the objects are not moved on processing every query. The BAM works as follows.

Associated with the set of *physical* objects $\Omega=\{A_1, A_2,...,A_W\}$ is a set of *abstract* objects $\{O_1,O_2, ...,O_W\}$. The BAM manipulates these abstract objects as the queries are processed. Notice that this can be done in the background, in a fashion which is invisible to the user. Periodically, the set of physical objects Ω are re-partitioned (typically, when the queries on the actual physical objects are not expected, for example, at back-up times) so as to conform to the partitioning dictated by the BAM. The way the abstract objects are manipulated is as follows. Initially the BAM assigns for each abstract object O_i a real number X_i. On processing a query requesting the physical objects A_i and A_j, the quantities X_i and X_j are moved toward their centroid by a small constant Δ_1*. This tends to group the points into clusters. Subsequently, two distinct random indices p and q are chosen, $1\leq p,q \leq W$, and X_p and X_q are drawn away from their centroid by a constant Δ_2*. Generally speaking, it is possible that the indices p and q assume the particular values of i and j in the same iteration. The techniques of "determining" Δ_1, Δ_2 and the initial X_i's are informally discussed in [22,23]* . After a fairly long time (T accesses), the partitions are created based on the proximity of the X_i's. The technique for creating these partitions is also discussed in [22,23] for the case when the number of partitions is known *a priori*. In the absence of any *a priori* information, heuristics are used to decide on the actual partitioning based on the value of the $\{X_i\}$. An algorithmic version of the BAM is given in [14,24].

The first attempt to solve the OPP using stochastic automata was presented by Oommen *et. al.* [14] for a special case of the OPP, the Equi-Partitioning Problem, in which the underlying groups of the objects were of the same size. Of the three deterministic automata solutions to the EPP presented in [14] the most elegant was a new machine called the Object Migrating Automaton [14]. For the case when W=4 it was shown to be expedient in all random environments. Experimental results involving the OMA demonstrated its excellent accuracy and showed that it is an order of magnitude faster than the BAM.

* Δ_1 and Δ_2 need not necessarily be constants. In general, they can be quantities proportional to the proximity of the numbers X_1 and X_2.

The first reported automata solutions for the general OPP were presented in [24]. Initially, a variant of the Object Migrating Automaton called the Stochastic Move Automaton was suggested and its capability in solving the OPP was demonstrated. The latter automaton converged much faster than the BAM although, in terms of accuracy, it was inferior in some environments. Finally, in [24] a new VSSA similar to the traditional Linear Reward Penalty Scheme was presented. This automaton, termed as the Modified Linear Reward Penalty (ML_{RP}) scheme, was shown to be expedient in all random environments. Experimentally, it was shown to possess a remarkable accuracy in that it converged to the exact underlying partitioning almost all the time. Yet it is about ten times faster (and sometimes even about twenty times faster) than the BAM.

Although the ML_{RP} scheme had the advantages mentioned above, it possesses one fundamental disadvantage, and that is that for each iteration a linear (in the number of objects to be partitioned) number of probabilities have to updated. This is unfortunately a marked disadvantage. With this in mind, in this paper, we shall tackle the general OPP again. However, rather than have a single ML_{RP} aiming to dictate the partitioning, we shall use a technique which utilizes a hierarchy of automata. Indeed, in this case, rather than execute a linear number (measured in terms of the number of objects to be partitioned) of probability updates for each iteration, only a logarithmic number of probabilities are updated. This scheme is again expedient and quite accurate. Experimentally, if the rate of convergence is measured in terms of the number of iterations, this scheme is still much faster than the BAM and is almost as fast as the ML_{RP} scheme.

III. THE ML_{RP} SCHEME

The scheme which we introduce in this paper involves a hierarchy of learning automata, which, at each level uses machines which are analogous in operation to the ML_{RP} scheme referred to in the previous section. Since the understanding of our present scheme requires a rather thorough understanding of the latter, in this section we shall briefly describe the philosophy behind the ML_{RP} scheme and explain its inherent drawbacks.

Unlike the BAM, in the use of automata, we assume that the W abstract objects have to each choose a class (partition) to be in. In the case of the scheme suggested for the Equi-Partitioning Problem this choice is made using a fixed structure automaton. But for the general OPP the choice is made on the basis of a VSSA. On processing a user's query for objects A_i and A_j, this scheme makes a decision as to which classes the abstract objects O_i and O_j should reside. If both O_i and O_j choose to be in the same class, then the choice of this partitioning is rewarded. Otherwise, it is penalized. Based on this feedback the automaton updates its internal structure (which in this case is a probability vector) and awaits the next user's query. It must be noted that the main motivation for the use of VSSA for the OPP (as opposed to the fixed structure scheme suggested for the Equi-

Partitioning Problem) is essentially the fact that these automata allow objects to choose two different classes at two consecutive time instants.

The Modified Linear Reward-Penalty (ML_{RP}) is quite similar to the traditional Linear Reward-Penalty (L_{RP}) scheme in which the automaton chooses its action based on the distribution $P(n) = [p_1(n), p_2(n), ..., p_R(n)]^T$. If α_i is the action chosen and the environment penalizes the automaton, the action probability is updated by by multiplying $p_i(n)$ by a constant $b < 1$, and the probability mass $(1-b)p_i(n)$ is divided up between the other R-1 action probabilities. Similarly, if the action is rewarded the action probabilities of the remaining R-1 classes are decreased by a constant $a < 1$, and the probability $p_i(n)$ is incremented accordingly. Explicitly, the L_{RP} scheme is defined below for $1 \leq i, j \leq R$:

$$\begin{aligned}
p_i(n+1) &= p_i(n) + \Sigma_{j \neq i}\ a.p_j(n) \\
&= p_i(n) + a.(1 - p_i(n)) && \text{if } \alpha_i \text{ is chosen and } \beta = 0 \\
p_i(n+1) &= p_i(n) - a.p_i(n) \\
&= (1 - a).p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 0 \\
p_i(n+1) &= p_i(n) - \Sigma_{j \neq i} \{[b/(R-1)] - b.p_j(n)\} \\
&= p_i.(1 - b) && \text{if } \alpha_i \text{ is chosen and } \beta = 1 \\
p_i(n+1) &= p_i(n) + [b/(R-1)] - b.p_i(n) \\
&= [b/(R-1)] + (1 - b).p_i(n) && \text{if } \alpha_j \text{ is chosen and } \beta = 1
\end{aligned} \tag{1}$$

The rationale for the ML_{RP} scheme is as follows. The abstract object O_i chooses to be in a particular action based on the probability vector $\mathbf{P}_i(n) = [p_{i1}(n), p_{i2}(n), ..., p_{iW}(n)]^T$. Whenever a pair of objects (A_i, A_j) is accessed if O_i and O_j choose the same action, say α_k, then they will be rewarded by the interacting environment. Just like the L_{RP} scheme, this is achieved by increasing the probability of *them* choosing α_k and at the same time decreasing the probabilities of them choosing all other actions. On the other hand, if O_i and O_j choose actions α_p and α_q respectively, where $\alpha_p \neq \alpha_q$, then this choice will be penalized. Since it may be the case that A_i and A_j should be in the same class, both O_i and O_j will increase the probability of "trying" to choose a common action. In such a case, instead of letting O_i move to α_q and O_j move to α_p, we modify the probabilities so that O_i chooses α_q and O_j chooses α_p with higher probabilities. Since the other actions α_k (for $k \neq p,q$) are not involved, the probabilities of them being chosen by O_i and O_j are not changed. To be more specific, since α_p is considered as an inappropriate action for O_i and α_q is the one that O_i should try, p_{ip} (the probability of O_i choosing α_p) is decremented, and this decrement is added to p_{iq} (the probability of O_i choosing α_q). Analogously, the value of p_{jq} is also decreased and the value of p_{jp} is increased by the corresponding amount. However, since the query system is stochastic, the request for the pair of objects (A_i, A_j) may be a "misleading" query. By this we mean that the pair of objects (A_i, A_j) may not actually belong to the same underlying partition, but are still requested by the user, and so we must be conservative in forcing them to be in the same partition. To achieve this, the amount of change upon receiving a penalty is rendered small so that a recovery due to an erroneous choice can be more easily achieved. The above informal discussion is formalized below.

If $\{A_1, ..., A_W\}$ is the set of W given objects, there will be W actions offered to the ML_{RP} automaton, where each action represents a certain class. The ML_{RP} is defined as a quadruple

$$(\alpha, \beta, \underline{\mathbf{P}}, F), \text{ where,} \tag{2}$$

(i) $\alpha = \{\alpha_1, ..., \alpha_W\}$ is the set of actions from which the abstract objects must choose.

(ii) $\beta = \{0, 1\}$ is the set of inputs. The value '0' represents a reward, '1' is a penalty.

(iii) $\underline{\mathbf{P}} = \{\mathbf{P}_1, ..., \mathbf{P}_W\}$ is the set of action probability vectors such that $\mathbf{P}_i = [p_{i1},...,p_{iW}]^T$, for all $1 \le i \le W$. Each component p_{ij}, $1 \le j \le W$, represents the probability that action α_j will be chosen by O_i. Also, for all i, $\Sigma_{1\le j \le W}\ p_{ij} = 1.0$

(iv) F is the probability updating scheme. If A_i and A_j are the accessed objects and their abstract counterparts choose actions α_m and α_n, respectively, $1 \le m, n \le W$, then the ML_{RP} scheme is defined as follows, for $0 < \lambda_R, \lambda_P < 1$.

If $\beta = \mathbf{0}$ (i.e., $\alpha_m = \alpha_n$)

$$\begin{aligned} p_{im}(n+1) &= 1 - \lambda_R * (1 - p_{im}(n)) \\ p_{ik}(n+1) &= \lambda_R * p_{ik}(n) \qquad 1 \le k \le W \text{ and } k \ne i \end{aligned} \tag{3}$$

If $\beta = \mathbf{1}$ (i.e., $\alpha_m \ne \alpha_n$)

$$\begin{aligned} p_{in}(n+1) &= p_{in}(n) + (1 - \lambda_P) * p_{im}(n) \\ p_{im}(n+1) &= \lambda_P * p_{im}(n) \end{aligned} \tag{4}$$

The probabilities $\{p_{jk}\}$ are updated in an analogous manner. Also, the way the constants λ_R and λ_P are chosen is described in [24].

Since the objects must be given the complete freedom to arbitrarily choose their actions, and since the cardinality of any particular class could be as small as two, we initially assign each of the actions to be in a class of its own. Thus, if O_i and O_j are two distinct objects, with no loss of generality, we can assume that O_i is assigned to choose α_i (i.e., $p_{ii} = 1.0$ and $p_{ij} = 0.0$ for $1 \le i, j \le W$ and $i \ne j$). Also, just as in the case of the BAM, after T accesses, the physical objects will be partitioned based on the actions chosen by the abstract objects. For the sake of completeness, we present below an algorithmic version of the ML_{RP} automaton.

Algorithm ML_{RP} Automaton

Input : The abstract objects $\{O_1, ..., O_W\}$, the reward parameter λ_R, the penalty parameter λ_P, the periodicity constant T, and a stream of queries $\{(A_i, A_j)\}$.

Output : A periodic partitioning of the objects.

Notation: α_p and α_q are the actions chosen by O_i and O_j at a given instant. They are integers between 1 and W. p_{ij}, defined as in (2), is the probability that O_i chooses class j at a given instant.

Method :

```
Initialize $p_{ii} = 1.0$ and $p_{ij} = 0.0$ for all $1 \le i \le W$.
Repeat
    For a sequence of T queries Do
        ReadQuery ($A_i$, $A_j$)
        If ($\alpha_p = m$ and $\alpha_q = m$)          (* Reward *)
            $p_{im} = 1 - \lambda_R ( 1 - p_{im})$
            $p_{jm} = 1 - \lambda_R ( 1 - p_{jm})$
            For ( $k = 1$ to W, $k \neq m$) Do
                $p_{ik} = \lambda_R p_{ik}$
                $p_{jk} = \lambda_R p_{jk}$
            Endfor
        Endif
        If ( $\alpha_p = m$ and $\alpha_q = n$ )         (* Penalty *)
            $p_{in} = p_{in} + (1 - \lambda_P) p_{im}$
            $p_{im} = \lambda_P p_{im}$
            $p_{jm} = p_{jm} + (1 - \lambda_P) p_{jn}$
            $p_{jn} = \lambda_P p_{jn}$
        Endif
    Endfor
    Print out partitions based on the probabilities $p_{ij}$
Forever
End Algorithm $ML_{RP}$.
```

With this understanding of the ML_{RP} scheme we shall now present the Hierarchical Modified Linear Reward-Penalty (HML_{RP}) which is the fundamental contribution of this paper.

III. THE HIERARCHICAL ML_{RP} SCHEME

The ML_{RP} scheme discussed above is very accurate and it converges to the exact underlying partition almost all the time. Also, in terms of speed it converges in a remarkably small number of iterations. Thus, for example, when fifteen objects were partitioned into four classes of unequal size, whereas the BAM converged after 3507 iterations the ML_{RP} automaton converged in only 383 iterations when the parameters were $\lambda_R = 0.5$ and $\lambda_P = 0.975$. This represents an increase in speed of the order of 9.1. However, as mentioned earlier, the ML_{RP} automaton has one major drawback. This drawback is that whenever the scheme is rewarded there is linear number of computations involved in evaluating the probability vectors for the next time instant. This drawback is unfortunately quite a significant drawback.

To overcome this drawback we shall perform the partitioning using a hierarchy of automata. The philosophy behind this hierarchy is quite straightforward. Let us suppose that W, the number of objects to be partitioned, satisfies $W = 2^J$ for some J. The partition or class to which an object O_i belongs is represented by a binary string $B_{i1}B_{i2}...B_{iJ}$. Rather than have a single automaton decide on the class of the object O_i, each abstract object is associated with J distinct automata. Each of these automata has two actions which is the set {0,1}. When the kth automaton for O_i chooses a binary value as its action, it is deciding on the value of the bit B_{ik}. Observe that once all the J automata for a particular object O_i have made their choices, the partition chosen by O_i is well defined, and thus the process of penalizing and rewarding the automata can be executed just as in the case of the previous automata solutions.

To be more specific, each abstract object O_i has J automata $\{M_{i1}, M_{i2}, .. M_{iJ}\}$ associated with it. The automaton M_{ik} is a 4-tuple $(\alpha, \beta, \mathbf{P}_{ik}, F)$, where,

(i) $\alpha = \{0,1\}$ is the set of actions. The action chosen is the value of the bit B_{ik}.

(ii) $\beta = \{0, 1\}$ is the set of inputs. The value '0' represents a reward, '1' is a penalty.

(iii) $\mathbf{P}_{ik} = [p_{ik0}, p_{ik1}]^T$ is the action probability vector, where p_{ik0} is the probability of M_{ik} choosing the action $\alpha = 0$, and, p_{ik1} is the probability of M_{ik} choosing the action $\alpha = 1$.

(iv) F is the probability updating scheme. If A_i and A_j are the accessed objects, let their abstract counterparts choose classes $\alpha_m = B_{i1}B_{i2}...B_{iJ}$ and $\alpha_n = B_{j1}B_{j2}...B_{jJ}$ respectively, where $1 \le m, n \le W$. Then the M_{ik} automaton updates the probability vector $\mathbf{P}_{ik}$ as follows :

If $\beta = \mathbf{0}$ (i.e., $B_{ik} = B_{jk}$).

Let $d = B_{ik} = B_{jk}$, and d' be the complement of d. Then,

$$\begin{aligned} p_{ikd}(n+1) &= 1 - \lambda_R * (1 - p_{ikd}(n)) \\ p_{ikd'}(n+1) &= \lambda_R * p_{ikd'}(n) \end{aligned} \tag{5}$$

If $\beta = \mathbf{1}$ (i.e., $B_{ik} \neq B_{jk}$).

Let $B_{ik} = d$ and $= B_{jk} = d'$, where d' is the complement of d.

$$\begin{aligned} p_{ikd'}(n+1) &= p_{ikd'}(n) + (1-\lambda_{P,k}) * p_{ikd}(n) \\ p_{ikd}(n+1) &= \lambda_{P,k} * p_{ikd}(n) \\ p_{jkd}(n+1) &= p_{jkd}(n) + (1-\lambda_{P,k}) * p_{jkd'}(n) \\ p_{jkd'}(n+1) &= \lambda_{P,k} * p_{jkd'}(n) \end{aligned} \qquad (6)$$

Before we explain the philosophy behind the algorithm, a word concerning the initialization process is not out of place. As in the case of the ML_{RP} scheme, since the objects must be given the complete freedom to arbitrarily choose their actions, and since the cardinality of any particular class could be as small as two, we initially assign each of the actions to be in a class of its own. Thus, if O_i and O_j are two distinct objects, with no loss of generality, we can assume that O_i is assigned to choose α_i and O_j to α_j where $i \neq j$. As in (3) and (4), for any binary digit d, let d' be its complement. Then, the values of p_{ikd} and $p_{ikd'}$ are set to zero and unity in such a way that each object is placed in a class of its own. This is trivially done since each action can be associated with the binary string representing the index of the action, and thus the corresponding binary bits B_{ik} can be easily assigned their values. Thus, for example, if the number of objects was four, the hierarchy will be of two levels, and the binary bits representing the actions will be the binary strings {00, 01, 10, 11}. Thus, in this case, the probability vectors associated with the action α_3 = "10" will be assigned the values :

$p_{311} = 1.0$, $p_{310} = 0.0$, $p_{321} = 0.0$, and $p_{320} = 1.0$.

The elements of the automaton, M_{ik}, namely, α, β, P_{ik} are easily understood. However to understand how the function F operates, it is well worth studying equations (5) and (6). Let us suppose the user requests A_i and A_j together, and let their abstract counterparts choose classes $\alpha_m = B_{i1}B_{i2}...B_{iJ}$ and $\alpha_n = B_{j1}B_{j2}...B_{jJ}$, respectively. Clearly, if every $B_{ik} = B_{jk}$, all the automata have made a choice which has to be rewarded, because their choice supports the query system. But if the actions α_m and α_n are not the same, clearly the automata which have made the choice must be penalized. However, in this case, it is easy to see that *some* of the automata may have made a correct choice, and indeed these are ones for which $B_{ik} = B_{jk}$. These automata are rewarded, and this is achieved using (3). Thus, whenever the choices are to be reinforced the probability of choosing the particular action is increased, and the probability of choosing the complementary action has to be decreased. This increment and decrement are done using a linear function, where λ_R is a constant, referred to as the Reward Constant.

The case of penalizing the actions chosen, however, is also straightforward. In this case, the actions chosen by the abstract objects are not the same. Thus, the individual automata which have made "erroneous" choices are penalized. Since it may be the case that A_i and A_j should be in the same class, both O_i and O_j will increase the probability of "trying" to choose a common action.

This is achieved by modifying the probabilities so that M_{ik} and M_{jk} choose the same action with an increased probability. Let us suppose that M_{ik} chose the action d and M_{jk} chose d', where d and d' are complementary. Then, the probability of M_{ik} choosing d and M_{jk} choosing d' are decremented, and the corresponding probabilities of M_{ik} choosing d' and M_{jk} choosing d are incremented. In this case, this is achieved using (6) using again a linear updating rule in the spirit of the L_{RP} scheme. Observe that in this case the constant $\lambda_{P,k}$ is the penalty constant.

Rather than use a single constant λ_P as in the case of the ML_{RP} scheme, in this case we have chosen to allow the penalty constant $\lambda_{P,k}$ to vary with the depth of the automaton, k. The reason for this is as follows. The higher the automaton is in the hierarchy the greater will be its influence on the choice of the action. This is primarily because a smaller value of k indicates that the automaton in question controls a more significant bit of the bit string $B_{i1}B_{i2}...B_{iJ}$. Thus changes made to the probability vector $\mathbf{P}_{i1}$ should be more conservative than the changes made to $\mathbf{P}_{iJ}$, and this is achieved by decreasing the value of $\lambda_{P,k}$ as k increases. We have opted to achieve this be defining $\lambda_{P,k}$ in terms of $\lambda_{P,k+1}$ as follows :

$$\lambda_{P,k} = \lambda_{P,k+1} + 0.5.(1-\lambda_{P,k+1}). \quad (7)$$

Notice that in this case the value of $\lambda_{P,k}$ is increasingly closer to unity than $\lambda_{P,k+1}$. Observe too that if $\lambda_{P,k}$ is sufficiently close to unity,(5) can be equivalently written as :

$$\lambda_{P,k+1} = 2.\lambda_{P,k}-1. \quad (8)$$

For the sake of completeness the pseudo-code for the hierarchical system is given below. Observe that in the pseudo-code the value of $\lambda_{P,k}$ is defined using (8).

Throughout the above discussion we have assumed that W, the number of objects to be partitioned, satisfies $W = 2^J$ for some J. The extension for the case when the number of objects is not an exponent of two is handled in an identical way, because dummy objects can be introduced to increase the number of abstract objects so as to render it to be an exponent of two. However, since these dummy objects are **never** accessed, they never come into any formal computation.

Algorithm HML_{RP} Automaton (* The Hierarchical ML_{RP} Scheme*)

Input : The abstract objects $\{O_1, ..., O_W\}$, the reward parameter λ_R, the penalty parameter $\lambda_{P,1}$, the periodicity constant T, and a stream of queries $\{(A_i, A_j)\}$.

Output : A periodic partitioning of the objects.

Notation : α_i and α_j are the nodal partitions chosen by O_i and O_j at a given instant. They are strings of length J, where $J = Log_2(W)$ containing binary directions taken at each junction of a node. p_{ikd} is the probability that the automaton M_{ik} chooses direction $d \in \{0,1\}$, or equivalently that O_i chooses direction $d \in \{0,1\}$ at a node of depth k at a given instant.

Method :

```
For k = 1 to Log2(W) Do
    Initialize $p_{ikd}$ and $p_{ikd'}$ to 0 or 1 so as to place each object in a class of its own.
Repeat
    For a sequence of T queries Do
        ReadQuery $(A_i, A_j)$
        For k = 1 to J Do                      (* For each depth *)
            Choose $B_{ik}$ and $Bjk$ using the vectors $\mathbf{P}_{ik}$ and $\mathbf{P}_{jk}$
            If $B_{ik} = B_{jk}$ Then              (* Reward $M_{ik}$ and $M_{jk}$*)
                $d = B_{ik}$ ; $d'$= Complement (d)
                $p_{ikd} = 1 - \lambda_R * (1 - p_{ikd})$ ; $p_{ikd'} = \lambda_R * p_{ikd'}$
                $p_{jkd} = 1 - \lambda_R * (1 - p_{jkd})$ ; $p_{jkd'} = \lambda_R * p_{jkd'}$
            Else                               (* Penalize $M_{ik}$ and $M_{jk}$*)
                $d = B_{ik}$ ; $d' = B_{jk}$
                If (k=1) Then                  (* Optional : Modify the Penalty Constant *)
                    $\lambda_P = \lambda_{P,1}$
                Else
                    $\lambda_P = 2.\lambda_P - 1$
                Endif
                $p_{ikd'} = p_{ikd'} + (1 - \lambda_P) * p_{ikd}$ ; $p_{ikd} = \lambda_P * p_{ikd}$
                $p_{jkd} = p_{jkd} + (1 - \lambda_P) * p_{jkd'}$ ; $p_{jkd'} = \lambda_P * p_{jkd'}$
            Endif
        Endfor
    Endfor
    Print out partitions based on the actions chosen by $M_{ik}$'s.
Forever
```

End Algorithm HML_{RP} Automaton

We shall now present a result concerning the hierarchical ML_{RP} scheme.

Theorem I

The Hierarchical ML_{RP} automaton is expedient if for all k, $0 < \lambda_R, \lambda_{P,k} < 1.0$.

Proof :

As explained above, since the objects must be given the complete freedom to arbitrarily choose their actions, and since the cardinality of any particular class could be as small as two, we initially assign each of the actions to be in a class of its own. Thus, each object O_i is initially assigned to the distinct action (representing a class) α_i, $1 \leq i \leq W$, where the binary representation of i exactly specifies P_{ikd} and $P_{ikd'}$, which are the probability assignments for all the J automata associated with O_i. Thus, since each O_i is in a distinct class, the probability of an accessed pair of objects being rewarded is zero. Thus, the initial expected penalty, denoted as M_0, is unity.

Suppose at the first time instant the pair (A_i, A_j) is accessed. Since both O_i and O_j are in different classes O_i and O_j will not be able to choose the same class. Thus if $\alpha_i = B_{i1}B_{i2}...B_{iJ}$ and $\alpha_j = B_{j1}B_{j2}...B_{jJ}$, there will be at least one bit position, say k, for which $B_{ik} \neq B_{jk}$. Let us suppose that $B_{ik} = d$ and that $B_{jk} = d'$. This implies that since the choice of the hierarchy of automata will be penalized, at least the automata M_{ik} and M_{jk} will be penalized. According to the hierarchical ML_{RP} scheme, $p_{ikd'}$ is updated to be $p_{ikd'} + (1 - \lambda_P)*p_{ikd}$ and that p_{ikd} is updated to be λ_P*p_{ikd}. Similarly, p_{jkd} is updated to be $p_{jkd} + (1 - \lambda_P)*p_{jkd'}$, and $p_{jkd'}$ is updated to be $\lambda_P*p_{jkd'}$. Thus, after the first query, the expected probability that O_i and O_j will be rewarded if (A_i, A_j) is accessed will become non-zero, if the parameters of the scheme are as in the statement of the theorem. Thus, if $E[M(1)]$ denotes the expected penalty at the first time instant, $E[M(1)] < M_0$.

Notice too that after the first instant, independent of whether O_i and O_j are rewarded or penalized, if the parameters of the scheme are as given in the statement of the theorem, the values of the probabilities $\{p_{ikd}$ and $p_{ikd'}\}$ and $\{p_{jkd}$ and $p_{jkd'}\}$ cannot be changed back to zero and unity so as to assure that O_i and O_j *always* choose distinct actions. Therefore, the limiting expected penalty, $E[M(\infty)]$, will be less than M_0, and hence the theorem is proved. ♦ ♦ ♦

IV. EXPERIMENTAL RESULTS

Experimental results comparing the ML_{RP} and the BAM were described in great detail in [24], and since the ML_{RP} has already been shown to be superior to the BAM both in terms of speed and accuracy, in this paper, in the interest of brevity we shall merely compare the performance of the Hierarchical ML_{RP} discussed above with the ML_{RP} scheme. The experiments were done primarily to compare the rate of convergence and the accuracy of the HML_{RP} automaton with the ML_{RP} as a function of the number of objects to be partitioned. In order to standardize the query system, the queries were generated based on an underlying grouping, and the assumptions on the distribution of the underlying groups were identical to those made in [14,24] and are explained below.

In all the experiments performed, queries were generated according to an underlying distribution having the following properties :

(a) The probability of accessing any pair of objects in one group is the same as that of accessing any other pair of objects in the same group.

(b) The probability of accessing any pair of objects in different groups is the same as that of accessing any other pair of objects in different groups. In other words, for all j and k, $Pr(A_j, A_k)$ is the same whenever A_j and A_k are not in the same group.

(c) The probability of accessing a pair of objects in the same group is greater than that of accessing a pair of objects in different groups.

Thus, if π_i was the group in which the physical object A_i is located, then these queries obeyed the following distribution:

$$\sum_{A_j \in \pi_i} \Pr[A_i, A_j \text{ accessed together}] = \delta, \quad i \neq j \tag{9}$$

Observe that if $\delta = 1$, then the partitioning is ideal, which implies that queries will involve only objects in the same underlying grouping. However, as the value of δ decreases, the queries will be decreasingly informative about the underlying distribution governing the queries. Indeed, the value of δ should be greater than 0.5 due to assumption (c). Also, to render the problem non-trivial, the underlying distribution which generated the queries was unknown to the algorithms that adaptively obtained the clusters.

In each set up, one hundred experiments were performed in order to obtain accurate results. The parameters determining the ML_{RP} scheme and the HML_{RP} scheme, and the probability δ defined by (9) were assigned as follows:

(i) λ_R and λ_P for the ML_{RP} scheme were set to 0.5 and 0.975 respectively as recommended in [24].

(ii) λ_R and $\lambda_{P,1}$ for the HML_{RP} scheme were set to 0.5 and 0.975 respectively.

(iii) In all the experiments reported the value of the joint access probability δ defined by (9) was set to be 0.9.

(iv) Rather than have the penalty parameter vary with the depth of the automaton, various simultaneous experiments were also conducted for the case when $\lambda_{P,k} = \lambda_{P,1}$ for all k. Whereas these experiments have been reported in the tables under the columns "Constant λ", the case when $\lambda_{P,k}$ varies with the depth of the automaton has been reported under the columns "Graded λ".

The experiments were done for various values of W, the number of objects, which varied from four to twelve. Additionally an automaton was considered to have converged if any action probability attained a value greater than or equal to 0.95. This is quite realistic because when such a situation occurs the automaton will not be able to change its action all too easily. A summary of the results that have been obtained is found in Table I.

***************** Insert Table I *****************

From Table I, we observe that the number of iterations required by the hierarchical ML_{RP} scheme is more than the number of iterations required by the ML_{RP}, but is of the same order. For example, when twelve objects were partitioned into three classes of size four, the ML_{RP} automaton converged in 239 iterations with an accuracy of 98%. The corresponding figure for the case of the HML_{RP} scheme with constant penalty parameters is 1519, the accuracy being only 50%. However, if $\lambda_{P,k}$ is made to obey (8) the accuracy increases to 90% and the number of iterations required for convergence is only 463. Notice that in this case, the number of iterations required by the BAM is as high as 2925 [24]. Thus the improvement obtained by using the HML_{RP} is a factor of about 6.3.

To compare the effect of *individually* penalizing all the automata associated with an object, the concept of rewarding and penalizing the entire hierarchy of automata collectively was also investigated. Thus, if the object O_i selected $\alpha_m = B_{i1}B_{i2}...B_{iJ}$ based on the choices of the automata $\{M_{ik}\}$and O_j selected $\alpha_n = B_{j1}B_{j2}...B_{jJ}$ based on the choices of the automata $\{M_{jk}\}$, then all the automata were collectively rewarded if for every k, $B_{ik} = B_{jk}$. Similarly, all the automata were collectively penalized if for any k, $B_{ik} \neq B_{jk}$. These results are given in Table II. For the same example mentioned above, when twelve objects were partitioned into three classes of size four, the hierarchical ML_{RP} automata with constant λ_P converged in 285 iterations with an accuracy of 90%. The corresponding figures for the case of the HML_{RP} scheme in which $\lambda_{P,k}$ is made to obey (8) has an accuracy of 84% and required 338 iterations.

***************** Insert Table II *****************

IV.1 Comments

From the results reported in Tables I and II and the results given in [24] involving the BAM we can conclude that, generally speaking, the HML_{RP} scheme requires fewer iterations than the BAM. However, it is always less accurate than the ML_{RP} scheme. Also, in terms of the number of computations required per iteration the HML_{RP} is far more efficient because it requires only a logarithmic number (in terms of the number of objects to be partitioned) of computations. It is also categorically seen that if all the automata associated with an object are collectively rewarded or penalized, then the penalty constant must not decrease with the depth of the automata. However, if M_{ik} and M_{jk} are penalized if and only if $B_{ik} \neq B_{jk}$, it pays to vary the penalty parameter $\lambda_{P,k}$.

In the case of the ML_{RP}, from the experimental evidence we have, the best results recommend that λ_R should take a value in the neighborhood of 0.5 and λ_P should take a value close to unity. In the case of λ_P, using a larger value (i.e., closer to unity) is quite plausible intuitively because, this implies that on being penalized only small changes are made on the action probabilities. Therefore, any erroneous choice can be rectified easily. On the other hand, the fact that λ_R should take on values near 0.5 seems to contradict our intuition because this causes substantial changes on the action probabilities upon receiving rewards. These values for the parameters seem to yield fairly good results for the case of the HML_{RP} scheme too.

Unlike the ML_{RP} scheme we do not have experimental evidence that suggests that the HML_{RP} is ε-optimal. But it may be possible that by suitably varying the parameters in the parameter space such an ε-optimal behavior can be obtained. This is still being investigated.

Unfortunately, the accuracy of the HML_{RP} scheme, measured in terms of the number of times the computed partition is **exactly** the underlying grouping, falls as the number of objects increases. From a pragmatic point of view we believe that this index of measuring the accuracy of a object partitioning algorithm is unrealistic. Indeed, as the number of objects increases, the number of possible classes increases exponentially, and it is unrealistic to expect that the class learnt is **exactly** the underlying group. A more realistic measure is to quantify the probability of objects which are in the same underlying grouping being in the final learnt (computed) class. We are currently investigating the existence of such measures.

Learning automata have been shown to possess superior learning characteristics if the probability space in which they operate is discretized [13]. In this regard we are currently investigating the possibility of discretizing both the ML_{RP} and the HML_{RP} schemes. We believe that the resulting discretized learning mechanisms will be far superior to their continuous counterparts and even more superior to the BAM.

V. CONCLUSIONS

In this paper, we have studied the Object Partitioning Problem (OPP). This problem involves partitioning a set of objects $\{A_1, A_2,\ldots,A_W\}$ into classes whose sizes are not necessarily equal. The intention is to partition this set in such a way that the objects accessed together are found in the same class.

Earlier, in [24], various stochastic automata solutions for the OPP were presented. One of these automata was a Variable Structure Stochastic Automaton (VSSA) solution, closely related to the traditional Linear Reward-Penalty Scheme. This automaton, the ML_{RP} scheme, was extremely accurate, but converges an order of magnitude faster than the Basic Adaptive Method (BAM) presented by Yu *et. al.*[22,23]. Indeed, in certain environments, the latter converges about 20 times faster than the BAM.

In this paper, we have proposed a fast hierarchical VSSA solution to the problem. It is first of all shown to be expedient. Experimentally, this solution converges with an accuracy which is less than the accuracy of the ML_{RP}. But in terms of the number of iterations required, it converges much faster than the BAM, and yet as opposed to the ML_{RP}, it requires only a logarithmic number (in terms of the number of objects to be partitioned) of computations per iteration.

Acknowledgements : The author is grateful to Jack Zgierski who implemented the algorithms presented here. He is also grateful to Mr. Valiveti who helped prepare the final manuscript.

REFERENCES

1. Arnow, D.M., and Tenenbaum, A. M., "An Investigation of the Move-Ahead-k-Rules", *Congressus Numerantium*, Proc. of the Thirteenth Southeastern Conference on Combinatorics, Graph Theory and Computing, Florida, Feb. 1982, pp.47-65.
2. Bitner, J.R.,"Heuristics that Dynamically Organize Data Structures", *SIAM J. Computing*, Vol. 8, 1979, pp. 82-110.
3. Gonnet, G.H., Munro, J.I., and Suwanda, H., "Exegesis of Self Organizing Linear Search", *SIAM J. Computing*, Vol. 10, 1981, pp. 613-637.
4. Hammer, M., and Niamir, B., "A Heuristic Approach to Attribute Partitioning", *Proc. of the ACM SIGMOD Conference*, 1979, pp.93-101.
5. Hammar, M., and Chan, A., "Index Selection in a Self-adaptive Database Management System", *Proc. of the ACM SIGMOD Conference,* 1976, pp.1-8.
6. Hendricks, W. J.,"An Account of Self-Organizing Systems", *SIAM J. Computing*, Vol. 5, 1976, pp. 715-723.
7. Lakshmivarahan, S., *Learning Algorithms Theory and Applications*, Springer-Verlag, New York, 1981.
8. Lam, K. and Yu, C.T., *A Clustered Search Algorithm Incorporating Arbitrary Term Dependencies*, Tech. Report, Dept. of Information Eng., Univ. of Illinois at Chicago Circle, 1978.
9. Lam, K. and Yu, C.T., "An Approximation Algorithm for a File Allocation Problem in a Hierarchical Distributed System", *Proc. of the ACM SIGMOD Conference*, 1980, pp.125-132.

10. McCabe, J., "On Serial Files with Relocatable Records", *Operations Research*, Vol. 12, 1965, pp.609-618.
11. Narendra, K.S., and Thathachar, M.A.L., "Learning Automata -- A Survey", *IEEE Trans. Syst. Man and Cybern.*, Vol.SMC-4, 1974, pp. 323-334.
12. Narendra, K.S., and Thathachar, M.A.L., *Learning Automata*, Prentice-Hall, 1989.
13. Oommen, B.J., "Absorbing and Ergodic Discretized Two Action Learning Automata", *IEEE Transactions on Systems, Man and Cybernetics*, March/April 1986, pp.282-293.
14. Oommen, B. J., and Ma, D. C. Y., "Deterministic Learning Automata Solutions to the Equi-Partitioning Problem", *IEEE Transactions on Computers*, Vol. 37, January 1988, pp.2-14.
15. Oommen, B.J., and Hansen E.R., "List Organizing Strategies using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations", *SIAM Journal of Computing*, Vol. 16, No. 4, August 1987, pp.705-716.
16. Sleator, D. and Tarjan, R., "Amortized Efficiency of List Update Rules", *Proc. of the Sixteenth Annual ACM Symposium on Theory of Computing*, April 1984, pp.488-492.
17. Salton, G., *Dynamic Information and Library Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 1975.
18. Schkolnick, M., "A Clustering Algorithm for Hierarchical Structures", *ACM Trans. on Database Systems*, 1977, pp.27-44.
19. Tsetlin, M.L., *Automaton Theory and Modelling of Biological Systems*, Academic Press, New York and London, 1973.
20. Van Rijsbergen, C.J., "A Theoretical Basis for the Use of Co-occurrence Data in Information Retrieval", *J. Documentation*, 1977, pp.106-119.
21. Yu, C.T., and Salton, G., "Precision Weighting - An Effective Automatic Indexing Method", *J. ACM*, 1976, pp.76-78.
22. Yu, C.T., Suen, C.M., Lam, K. and Siu, M.K., "Adaptive Record Clustering ", *ACM Trans. on Database Systems*, 1985, pp.180-204.
23. Yu, C.T., Siu, M.K., Lam K., and Tai, F., "Adaptive Clustering Schemes : General Framework", *Proc. of the IEEE COMPSAC Conference*, 1981, pp.81-89.
24. Oommen, B. J., and Ma, D. C. Y., "Stochastic Automata Solutions to the Object Partitioning Problem". To appear in *The Computer Journal.* Also available as a Tech. Report from the School of Computer Science, Carleton University, Ottawa, Canada.

Table I: Comparison of the rate of convergence of the ML_{RP} and the HML_{RP} automaton solutions for the OPP. In both cases, the values of λ_R and $\lambda_{P,1}$ are 0.5 and 0.975 respectively. Also the joint access probability δ defined by (9) is 0.9. Each automaton for an object is individually rewarded or penalized. In the graded λ_P case, $\lambda_{P,k}$ obeys (8).

No of Objects	Partition	No of groups	ML_{RP}		HML_{RP} Constant λ_P		HML_{RP} Graded λ_P	
			Iterations	Accuracy	Iterations	Accuracy	Iterations	Accuracy
4	2,2	2	(24,42)	100	(33,54)	86	(26,44)	97
6	3,3	2	(81,106)	100	(149,194)	70	(150,184)	94
	2,4	2	(81,102)	95	(138,184)	66	(240,271)	79
	2,2,2	3	(46,72)	99	(85,133)	78	(49,86)	97
9	3,3,3	3	(137,177)	100	(554,728)	56	(361,526)	75
	2,3,4	3	(130,167)	100	(609,768)	56	(398,521)	78
	4,5	2	(208,241)	98	(769,905)	63	(408,511)	96
12	2,2,2,2,2,2	6	(108,167)	100	(488,637)	27	(162,207)	49
	2,2,2,3,3	5	(152,199)	100	(703,926)	37	(388,699)	69
	3,3,3,3	4	(194,249)	100	(955,1181)	39	(582,733)	61
	2,3,3,4	4	(198,235)	99	(902,1179)	46	(409,460)	70
	4,4,4	3	(189,239)	98	(1156,1519)	50	(398,463)	90
	6,6	2	(259,312)	99	(1676,1807)	40	(**,**)	**

Notation: Number of iterations is given as the pair (x,y), where x is the average number of iterations for the partitioning to be obtained, and y is the average number of iterations for the algorithm to converge to the partitioning. In the case of (**) the results are not presently available.

Table II: Comparison of the rate of convergence of the ML_{RP} and the HML_{RP} automaton solutions for the OPP. In both cases, the values of λ_R and $\lambda_{P,1}$ are 0.5 and 0.975 respectively. Also the joint access probability δ defined by (9) is 0.9. All the automaton for an object are collectively rewarded or penalized. In the graded λ_P case, $\lambda_{P,k}$ obeys (8).

No of Objects	Partition	No of groups	ML_{RP}		HML_{RP} Constant λ_P		HML_{RP} Graded λ_P	
			Iterations	Accuracy	Iterations	Accuracy	Iterations	Accuracy
4	2,2	2	(24,42)	100	(22,38)	100	(23,38)	100
6	3,3	2	(81,106)	100	(67,89)	97	(82,103)	97
	2,4	2	(81,102)	95	(81,97)	67	(103,112)	51
	2,2,2	3	(46,72)	99	(43,71)	100	(41,69)	100
9	3,3,3	3	(137,177)	100	(113,147)	85	(157,186)	90
	2,3,4	3	(130,167)	100	(166,189)	81	(117,151)	76
	4,5	2	(208,241)	98	(167,205)	96	(208,296)	83
12	2,2,2,2,2,2	6	(108,167)	100	(133,163)	44	(125,152)	43
	2,2,2,3,3	5	(152,199)	100	(137,186)	80	(183,233)	82
	3,3,3,3	4	(194,249)	100	(163,213)	89	(231,275)	92
	2,3,3,4	4	(198,235)	99	(187,229)	72	(224,255)	66
	4,4,4	3	(189,239)	98	(234,285)	90	(291,338)	84
	6,6	2	(259,312)	99	(271,316)	92	(424,446)	58

Notation: Number of iterations is given as the pair (x,y), where x is the average number of iterations for the partitioning to be obtained, and y is the average number of iterations for the algorithm to converge to the partitioning.

School of Computer Science, Carleton University
Bibliography of Technical Reports

SCS-TR-127 **On the Packet Complexity of Distributed Selection**
A. Negro, N. Santoro and J. Urrutia, November 1987.

SCS-TR-128 **Efficient Support for Object Mutation and Transparent Forwarding**
D.A. Thomas, W.R. LaLonde and J. Duimovich, November 1987.

SCS-TR-129 **Eva: An Event Driven Framework for Building User Interfaces in Smalltalk**
Jeff McAffer and Dave Thomas, November 1987.

SCS-TR-130 Out of print **Application Frameworks: Experience with MacApp**
John R. Pugh and Cefee Leung, December 1987.
Available in an abridged version in the Proceedings of the Nineteenth ACM SIGSCE Technical Symposium, February 1988, Atlanta, Georgia.

SCS-TR-131 **An Efficient Window Based System Based on Constraints**
Danny Epstein and Wilf R. LaLonde, March 1988.

SCS-TR-132 **Building a Backtracking Facility in Smalltalk Without Kernel Support**
See Third International Conference on OOPSLA, San Diego, Calif., Sept. '88.
Wilf R. LaLonde and Mark Van Gulik, March 1988.

SCS-TR-133 **NARM: The Design of a Neural Robot Arm Controller**
Daryl H. Graf and Wilf R. LaLonde , April 1988.

SCS-TR-134 **Separating a Polyhedron by One Translation from a Set of Obstacles**
Otto Nurmi and Jörg-R. Sack, December 1987.

SCS-TR-135 **An Optimal VLSI Dictionary Machine for Hypercube Architectures**
Frank Dehne and Nicola Santoro, April 1988.

SCS-TR-136 **Optimal Visibility Algorithms for Binary Images on the Hypercube**
Frank Dehne, Quoc T. Pham and Ivan Stojmenovic, April 1988.

SCS-TR-137 **An Efficient Computational Geometry Method for Detecting Dotted Lines in Noisy Images**
F. Dehne and L. Ficocelli, May 1988.

SCS-TR-138 **On Generating Random Permutations with Arbitrary Distributions**
B. J. Oommen and D.T.H. Ng, June 1988.

SCS-TR-139 **The Theory and Application of Uni-Dimensional Random Races With Probabilistic Handicaps**
D.T.H. Ng, B.J. Oommen and E.R. Hansen, June 1988.

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 **Parallel Fractional Cascading on a Hypercube Multiprocessor**
Frank Dehne, Afonso Ferreira and Andrew Rau-Chaplin, May 1989

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

SCR-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