Solving Multiconstraint Assignment Problems Using Learning Automata

Geir Horn and B. John Oommen, *Fellow, IEEE*

***Abstract*—This paper considers the *NP*-hard problem of object assignment with respect to multiple constraints: assigning a set of elements (or objects) into mutually exclusive *classes* (or groups), where the elements which are "similar" to each other are hopefully located in the same class. The literature reports solutions in which the similarity constraint consists of a *single* index that is inappropriate for the type of multiconstraint problems considered here and where the constraints could simultaneously be *contradictory*.[1] Such a scenario is illustrated with the *static mapping problem*, which consists of distributing the processes of a parallel application onto a set of computing nodes. This is a classical and yet very important problem within the areas of parallel computing, grid computing, and *cloud* computing. We have developed four learning-automata (LA)-based algorithms to solve this problem: First, a fixed-structure stochastic automata algorithm is presented, where the processes try to form pairs to go onto the same node. This algorithm solves the problem, although it requires some centralized coordination. As it is desirable to avoid centralized control, we subsequently present *three* different variable-structure stochastic automata (VSSA) algorithms, which have superior partitioning properties in certain settings, although they forfeit some of the scalability features of the fixed-structure algorithm. All three VSSA algorithms model the processes as automata having first the hosting nodes as possible actions; second, the processes as possible actions; and, third, attempting to estimate the process communication digraph prior to probabilistically mapping the processes. This paper, which, we believe, comprehensively reports the pioneering LA solutions to this problem, unequivocally demonstrates that LA can play an important role in solving complex combinatorial and integer optimization problems.**

***Index Terms*—Learning automata (LA), multiconstraint assignment, object partitioning, static mapping problem (SMP).**

I. Introduction

OBJECT assignment is the problem of partitioning a set $\mathbb{P}$ of $|\mathbb{P}|$ elements into $|\mathbb{N}|$ *classes* satisfying certain capacity constraints on these classes. Within the context of our problem, it is assumed that there are weighted directed relations between the elements of $\mathbb{P}$ and that there is no relation from an object to itself. Furthermore, each element of the object set carries a weight.

This is a combinatorial optimization problem, also known as integer optimization, because the solutions are restricted to the integer domain. This class of optimization problems is, in general, computationally very "hard" owing to the vast number of possible configurations and the fact that working in a discrete solution space renders it difficult to infer information on nearby solutions from a given configuration. The higher the dimension of the solution space is, the more difficult it will be to obtain a solution. Indeed, in most cases, determining the *optimal* solution becomes computationally intractable, although there might be heuristics available that permit the determination of a *feasible* solution.

The problem is made even worse when considering that some of the constraints can be mutually conflicting.[2] One constraint may require that two objects P_i and P_j should be assigned to the same group because they have a strong relation with each other, whereas an alternate constraint may require that they be allocated to separate classes because their combined weight is prohibitive. In this paper, we analyze such a classical object assignment problem from parallel computing and propose learning-automata (LA)-based heuristic algorithms to find a solution for this problem. The problem is generic enough to serve as an introduction to the general solution approach. This example problem will be introduced in Section II, highlighting its connection to the multidimensional knapsack problem.

The objective of the classical assignment problem is to minimize a linear sum over the integer cost of assigning object i to class j, whose optimal solution can be found by the Hungarian method [2]. The problem considered here is much more complex, and the constraints on the classes combined with the weights of the objects to be partitioned indicate that the problem is more alike a multidimensional knapsack problem, which is known to be NP-hard in the strong sense [3]. It can therefore be expected that algorithms finding exact solutions will scale exponentially with the problem size, making them intractable for practical use.

The equipartitioning problem is the archetypical problem for assessing NP-completeness [4, p. 223], and, given that the most efficient algorithm for equipartitioning [5] is based on LA, we submit that it is a good starting point for a heuristic

Manuscript received December 31, 2008; revised July 29, 2009. Current version published October 30, 2009. This paper was recommended by Associate Editor M. S. Obaidat.

G. Horn was with the Simula Research Laboratory, 1325 Lysaker, Norway, and also with Carleton University, Ottawa, ON K1S 5B6, Canada. He is now with SINTEF, 0314 Oslo, Norway (e-mail: Geir.Horn@sintef.no).

B. J. Oommen is with the School of Computer Science, Carleton University, Ottawa, ON K1S 5B6, Canada, and also with the University of Agder, 4876 Grimstad, Norway (e-mail: oommen@scs.carleton.ca).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2009.2032528

[1]This feature, where we permit possibly *contradictory* constraints, distinguishes this paper from the state of the art. Indeed, we are aware of no learning automata (or other heuristic) solutions which solve this problem in its most general setting.

[2]The scenario is akin to the one which deals with a multiteacher environment [1].

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

for the problem at hand. LA have also proven successful in dealing with stochastic problems, and, in a practical setting, one will typically be uncertain about the involved weights, and, consequently, the best one can often do is to take these to be average values. A solution based on fixed-structure stochastic automata (FSSA) learning is introduced in Section III. Alternative approaches based on variable-structure stochastic automata (VSSA) learning and no centralized control are introduced in Section IV. Section VI introduces the simulation methodology used to assess the different algorithms, and the results are further discussed in Section VIII. Although the algorithms discussed in this paper have been presented previously at various conferences [6]–[9], this paper provides a consolidated and coherent presentation, adding unprecedented formalism with respect to the problem statement, complexity, and the objective function used in Section II and the construction of the baseline partition in Section VII.

II. Static Mapping Problem

The example application used in this paper is a classical problem well known since the early days of parallel computing. The challenge is to partition a set $\mathbb{P}$ of processes in $|\mathbb{N}|$ subsets to be allocated to computing nodes such that the combined computation load of the processes in one set does not exceed the capacity of the nodes and such that the combined communication weight between any two subsets also does not exceed the capacity of the inbound and outbound communication links of the nodes. In other words, the mapping is the act of assigning the processes to the nodes of the available parallel computer. Without loss of generality, it will here be assumed that all the nodes of the computer are equal in capacities.

This problem is known in the literature as the *static* mapping problem (SMP) [10] since it is assumed that the set of processes remains fixed during the execution of the parallel application. The SMP deals with data parallel applications, where all processes are ready to run at startup. A wide range of algorithms has been proposed over the years, and the interested reader is referred to [10] and the references therein for an overview and to [11] for a comparison of the different strategies. The SMP is called a static *scheduling* problem if one adds the constraint that there are data dependencies among the processes such that the objective of the placement of the processes on the nodes is to find the minimum completion time (makespan) of the parallel application (see the excellent survey [12] for alternative algorithms).

The mapping discussed here is known as the problem of *cardinality variation* [13] since it only groups processes together on unlabeled nodes. It is a deferred task to assign a position for these nodes in a physical network or to design or reconfigure the interconnect for the resulting communication pattern. Given that the processes are grouped onto unlabeled nodes, the former is a graph isomorphism problem [14], and a solution to the latter can be found in [15].

The problem is trivial if $|\mathbb{P}| \leq |\mathbb{N}|$. In such a case, each node will host at most one process. The number of ways to partition the set of processes grows combinatorially when $|\mathbb{P}| > |\mathbb{N}|$, and the mapping problem has been shown to be NP-hard.

A. General Concepts

Each process P_i requires a certain computation time τ_i. The computation requirements of a process can, without loss of generality, be normalized to the computational power of the nodes such that $0 < \tau_i \leq 1$. Although commonly assumed known, it could, in practice, be difficult to estimate τ_i for a parallel application. The discussion in [16] ends with the recommendation to take τ_i to be equal to the expected load of the process, accepting that some of the nodes could be oversubscribed owing to the statistical variation of the execution times of the assigned processes around their average loads. If strict real-time response is required, it is better to assign τ_i to a value that is equal to the upper bound of the execution time. However, this difficulty shows why it is important to deploy algorithms that operate well in stochastic environments. This is a capacity of LA, making them suitable for this situation, and, to the authors' knowledge, this is the first solution of the mapping problem allowing stochastic process parameters.

In the same way as for the computational power, also the inbound and outbound capacity of a node can be assumed normalized to unity. Associated with each process P_i, there is a vector $\mathbf{w}_i$ whose elements $w_{i,j}$ indicate how much process P_i, *on average*, communicates *to* process P_j. Since a process can be the only one allocated to a node and the total outbound communication from a node is restricted to unity, the sum of the element in $\mathbf{w}_i$ must be less or equal to unity. Note also that a process will not communicate with itself, so $w_{i,i} = 0$.

A particular mapping is called *feasible* if and only if no node's capacity is exceeded. In other words, the computational time of the processes allocated to a node must be less or equal to unity, and their combined outbound and inbound communication must be less or equal to unity in both directions. Hence, the *feasible region* is the unit cube in the three constraint dimensions.

Note that a *feasible* configuration is only a useful configuration. This is not a quality assessment since some feasible configurations will clearly be better than others. Finding the better configurations has to be traded against available time to search. However, given the complexity of the problem to be developed in the next section, it will become clear that even finding the first feasible configuration is in itself a challenging problem. Finding the *optimal* configuration involves solving the integer optimization problem in Section II-C which is subject to the objective function in Section II-D. This is intractable as the system size grows; hence, there has, for a long time, been a quest for good heuristics to find feasible mappings.

B. Complexity

Let $\mathcal{S}_2(|\mathbb{P}|, |\mathbb{N}|)$ be the number of configurations. Given that a process can be assigned to one and only one node, the location of every process can be assigned $|\mathbb{N}|$ values. The same goes for all the other processes, so the upper limit for the number of configurations must be the total number of different possible configurations

$$\mathcal{S}_2\left(|\mathbb{P}|, |\mathbb{N}|\right) < |\mathbb{N}|^{|\mathbb{P}|}. \tag{1}$$

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

This is an upper limit for $\mathcal{S}_2(|\mathbb{P}|, |\mathbb{N}|)$ since it allows some nodes to be left empty. Furthermore, this simple calculation does not take into consideration the symmetries of the problem caused by assuming that all the nodes are equal, and, consequently, their labels can easily interchange.

However, it is easy to generate all possible configurations by recursive application of the following two rules.

1) Allocate the first process to the first node, and restart the allocation with the remaining processes and nodes. Thus, the number of configurations from this rule will be $1 \times \mathcal{S}_2(|\mathbb{P}| - 1, |\mathbb{N}| - 1)$.
2) Keep the first process on hold, and then restart the allocation of the remaining processes on all the nodes. Then, allocate in turn the process on hold to each of the configurations produced by the reduced process set. Hence, this will contribute $|\mathbb{N}| \times \mathcal{S}_2(|\mathbb{P}| - 1, |\mathbb{N}|)$.

Adding the contributions of the two rules together, one arrives at the following recursive expression for the number of configurations:

$$\mathcal{S}_2(|\mathbb{P}|, |\mathbb{N}|) = \mathcal{S}_2(|\mathbb{P}|-1, |\mathbb{N}|-1) + |\mathbb{N}|\mathcal{S}_2(|\mathbb{P}|-1, |\mathbb{N}|). \quad (2)$$

This is recognized as the recurrence relation for the Stirling number of the second kind [17]; hence

$$\mathcal{S}_2(|\mathbb{P}|, |\mathbb{N}|) = \sum_{k=1}^{|\mathbb{N}|} (-1)^{|\mathbb{N}|-k} \frac{k^{|\mathbb{P}|-1}}{(k-1)!\,(|\mathbb{N}|-k)!}. \quad (3)$$

C. Mathematical Formulation

Let $x_{i,n}$ be an indicator variable that takes the value of unity if process P_i is allocated to node N_n, otherwise zero. Consider first the outbound communication from a node assuming that P_i is allocated to this particular node. Then

$$\sum_{j=1}^{|\mathbb{P}|} (1 - x_{j,n}) w_{i,j}$$

is the external communication from process P_i to all the other processes P_j. If any process P_j is also allocated to node N_n, then $x_{j,n} = 1$, and this process gives no contribution to the aforementioned sum. To find the outbound communication from the node, similar sums must be added together but only for the processes P_i allocated to N_n

$$\sum_{i=1}^{|\mathbb{P}|} \left[x_{i,n} \sum_{j=1}^{|\mathbb{P}|} (1 - x_{j,n}) w_{i,j} \right].$$

Multiplying and rearranging give the outbound communication constraint (5).

Observe that the aforementioned one adds together the communication from a process hosted on a given node to another process *not* hosted on the node. In other words, it splits the set of processes into two subsets and adds the communication from one subset to the other. The only "directional" quantity is the amount of communication $w_{i,j}$, indicating that the communication goes from the process P_i on the node to a remote process P_j not on the node. Thus, the inbound communication can be found by the same formula but adding the communication going from the remote processes *to* the processes on the node $w_{j,i}$, resulting in (6).

Therefore, the problem can be stated as follows:

$$\begin{aligned}
&\text{minimize} \quad f(\mathbf{x}) && (4)\\
&\text{subject to} &&\\
&\sum_{i=1}^{|\mathbb{P}|}\sum_{j=1}^{|\mathbb{P}|} (x_{i,n} - x_{i,n}x_{j,n}) w_{i,j} \le 1, \quad n = 1, \ldots, |\mathbb{N}| && (5)\\
&\sum_{i=1}^{|\mathbb{P}|}\sum_{j=1}^{|\mathbb{P}|} (x_{i,n} - x_{i,n}x_{j,n}) w_{j,i} \le 1, \quad n = 1, \ldots, |\mathbb{N}| && (6)\\
&\sum_{i=1}^{|\mathbb{P}|} x_{i,n}\tau_i \le 1, \quad n = 1, \ldots, |\mathbb{N}| && (7)\\
&\sum_{n=1}^{|\mathbb{N}|} x_{i,n} = 1, \quad i = 1, \ldots, |\mathbb{P}| && (8)\\
&x_{i,n} \in \{0, 1\}. && (9)
\end{aligned}$$

The set of constraints (7) restricts the total computational time for the processes allocated to any node to the normalized capacity of the node. The constraint set (8) ensures that a process will be placed on one and only one node. The problem is called *hard* if the constraints (5) and (6) are all equal to unity and *relaxed* if they are less than unity.

As argued previously, a process can be the only one allocated to a node, and, therefore, its total outbound communication cannot exceed unity. This restricts the process' capacity to communicate to other processes allocated to the same node, although there is no explicit constraint to this effect. Again, the local communication is said to be *relaxed* if the total communication from a node is less than unity and *hard* if it is exactly unity.

D. Objective Function

The feasible region for each node is the unit cube in the three constraint dimensions (7), (5), and (6). Denote the left-hand side of the constraints L_n, O_n, and I_n, respectively, for $n = 1, \ldots, |\mathbb{N}|$. Thus, any given allocation of processes for a given node corresponds geometrically to a point (L_n, O_n, I_n). In the following, a single node will be considered, and the subscripts are omitted without ambiguity.

The task is to find an objective function that penalizes all points outside of the unit cube, and the more, the longer away from the unit cube the point is. The Euclidean distance would have been a good choice had only the feasible region been a unit ball. However, the square sides of the region make the problem slightly more difficult. The proposed solution is nevertheless inspired by the distance metric.

Consider first the degenerate case where $I = 0$ and the point is in a 2-D plane, and convert the point (L, O) to polar coordinates (r, ϕ). Here, r is the length of the vector from the origin to the point (L, O), and let $r_1(\phi)$ be the length of this vector inside the unit square. The fundamental idea is that $r - r_1(\phi)$

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

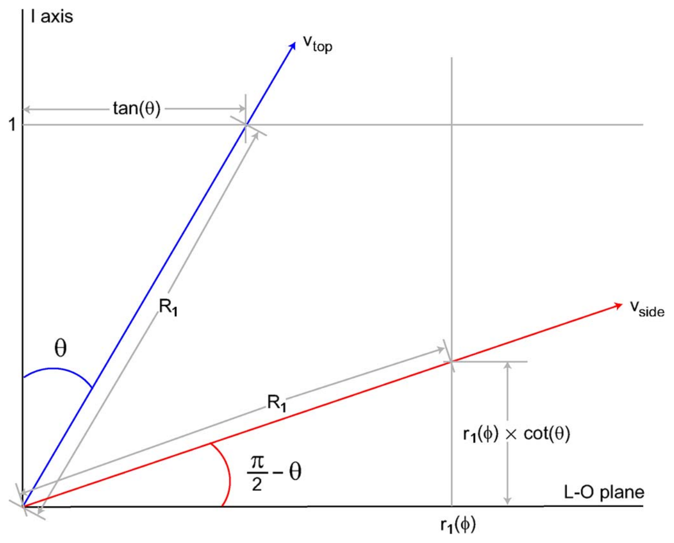


Fig. 1. Two possibilities for the configuration vector to cross the boundary of the feasible unit cube shown in a plane rotated an angle ϕ.

will represent how much outside the unit square the point is and will serve as a distance metric in the objective function.

Start with the case when $\phi \le \pi/4$ so that the vector to (L, O) leaves the unit square, where $L = 1$. Recall the fundamental trigonometric identity for the cosine in a right triangle

$$\cos(\phi) = \frac{L}{r_1(\phi)} = \frac{1}{r_1(\phi)}, \qquad \text{when} \quad \phi \le \frac{\pi}{4}.$$

Thus

$$r_1(\phi) = \frac{1}{\cos(\phi)} = \sec(\phi), \qquad \text{when} \quad \phi \le \frac{\pi}{4}.$$

Similarly, for the case when $\phi > \pi/4$, the vector crosses the side of the unit square, where $O = 1$, and from the trigonometric identity for the sinus in a right triangle

$$\sin(\phi) = \frac{O}{r_1(\phi)} = \frac{1}{r_1(\phi)}, \qquad \text{when} \quad \phi > \frac{\pi}{4}.$$

Hence

$$r_1(\phi) = \frac{1}{\sin(\phi)} = \csc(\phi), \qquad \text{when} \quad \phi > \frac{\pi}{4}.$$

Therefore, the length of the vector inside the unit square is

$$r_1(\phi) = \begin{cases} \sec(\phi), & \phi \le \pi/4 \\ \csc(\phi), & \phi > \pi/4. \end{cases} \tag{10}$$

The same idea can be used for the 3-D vector: The starting point is to express the Cartesian point (L, O, I) in spherical coordinates as (R, θ, ϕ). In this case, there are two possibilities: The vector may exit the feasible unit cube through one of the sides, or it can cut through the ceiling. Fig. 1 shows this situation in the plane rotated the angle ϕ. The goal is to find the length $R_1(\theta, \phi)$ representing the length of the vector inside the feasible unit cube.

Consider first the case where the vector crosses one of the sides of the cube. Given that θ is the angle of the vector with the positive I-axis, the minor angle of the right triangle is $\pi/2 - \theta$. Let I_{unit} be the I value where the vector crosses the side, and apply the basic trigonometric identity

$$\tan\left(\frac{\pi}{2} - \theta\right) = \cot(\theta) = \frac{I_{\text{unit}}}{r_1(\phi)}.$$

Therefore, from the Pythagorean theorem

$$\begin{aligned} R_1(\theta, \phi) &= \sqrt{[r_1(\phi)]^2 + I_{\text{unit}}^2} \\ &= \sqrt{[r_1(\phi)]^2 + [r_1(\phi)\cot(\theta)]^2} \\ &= r_1(\phi)\sqrt{1 + \cot^2(\theta)} = r_1(\phi)\csc(\theta). \end{aligned}$$

The situation where the vector crosses the top side of the unit cube is simpler. In this case, $I_{\text{unit}} = 1$, and the length of the vector inside the cube is directly given as

$$R_1(\theta, \phi) = \sqrt{1 + \tan^2(\theta)} = \sec(\theta).$$

It is readily seen from Fig. 1 that the vector crosses the sides of the cube if $\tan(\theta) \ge r_1(\phi)$, and, otherwise, it will cross the top side. Combining the aforementioned results gives

$$R_1(\theta, \phi) = \begin{cases} \sec(\theta), & \tan(\theta) < r_1(\phi) \\ r_1(\phi)\csc(\theta), & \tan(\theta) \ge r_1(\phi). \end{cases} \tag{11}$$

Therefore, the obvious cost of a certain allocation of processes on a node (L_n, I_n, O_n) can be defined for each node $n = 1, \ldots, |\mathbb{N}|$, after converting the configuration point to spherical coordinates, as

$$c_n(L_n, I_n, O_n) = \begin{cases} 0, & R_1(\theta, \phi) \ge R \\ R - R_1(\theta, \phi), & R_1(\theta, \phi) < R. \end{cases} \tag{12}$$

These costs are all positive and can be combined as necessary to give the overall cost of the configuration (4) for all nodes. The obvious approach to combining the cost of various nodes taken here is just to add them together, even though one could argue that higher costs should be penalized more, and, thus, an enforcement function (like squaring) should be used on the c_n elements before adding them.

III. FSSA

A. *Fundamental Concepts of Learning*

The functionality of LA can be described in terms of a sequence of repetitive feedback cycles in which the automaton interacts with an environment [18]–[21]. During a cycle, the automaton chooses an action, which triggers a response from the environment, where the response can be either a reward or a penalty. The automaton uses this response and the knowledge acquired in the past actions to determine which is the next action. The goal of LA is to determine the optimal action out of a set of allowable actions, where the optimal action is defined as the action that maximizes the probability of being rewarded. By learning to choose the optimal action, the automaton adapts itself to the environment. The learning paradigm as modeled by

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

LA has found applications[3] in systems that possess incomplete knowledge about the environment in which they operate.

Of all the classes of LA, the pioneering ones are those which belong to the FSSA families. These FSSA have the property that their transition and output functions do not change with time. They have powerful applications in solving various *NP*-hard problems, and fundamental to the solution proposed here to the SMP is a subclass of LA solutions which has been used to solve the object partitioning problem. In the special case when all the groups are required to possess the same number of objects, the problem is also referred to as the *equipartitioning* problem. The most efficient solution to this involves a learning automaton [5], and some modifications to the original version were proposed by Oommen and Ma in [5, the Appendix] and also by Gale *et al.* [22].[4]

Inspired by this, the present authors developed a similar FSSA algorithm to solve the problem at hand [6]. This will be described in the following. In contrast to the equipartitioning algorithm, this algorithm allows an unequal number of objects in each class; however, it is similar in modeling the objects as having a set of states that can be thought of as the level of *confidence* the object has in its present allocation. Here, it is taken that an object has eight confidence states, and it has converged to a good allocation if it is in one of the two last states. If an object has zero confidence in its present allocation, it is a candidate to be *moved* to another object class.

B. Object Pairing Process

The Oommen–Ma algorithm [5] works on a sequence of *queries*, each in the form of object pairs $\langle P_k, P_l \rangle$ indicating that these two objects have something in common. The algorithm will, in general, try to group and allocate the two objects of a query to the same class. It is imperative for the convergence of the algorithm that the random sequence of queries reflects an underlying optimal partition. The optimal partition is, in general, not known for the mapping problem at hand, nor there is any oracle that may provide a sequence of pairs to go together. Thus, the Oommen–Ma algorithm cannot be used for the problem at hand; however, it has inspired the algorithm developed here, which is also based on pairing, exploiting the fact that the outbound communication $\mathbf{w}_i$ is known for all processes as follows.

1) First, the node that violates the most the constraint (7) is selected.
2) A process, for example, P_A, allocated to this node is selected randomly among the processes on the node according to the empirical distribution of their average weights τ_i.
3) Then, the process P_A selects randomly another process P_B according to the probability distribution $\mathbf{w}_A$.

The set of processes $\langle P_A, P_B \rangle$ is, in the following, referred to as a (unordered) pair.

[3]For a comprehensive list of the applications, we refer the reader to [18]–[21] and to the references cited in these publications.

[4]This paper also specifies a modified "criterion for convergence," by which the OMA demonstrates a higher accuracy.

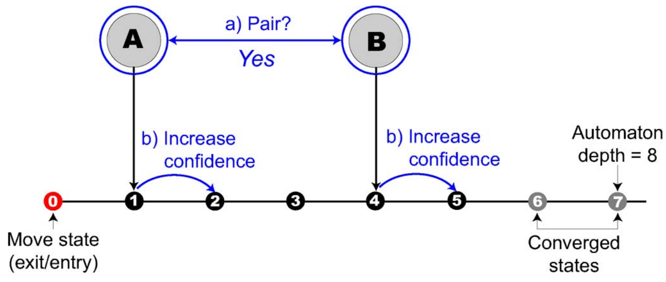


Fig. 2. Successful pairing of two processes A and B belonging to the same node will increase their confidence in a correct placement and move them one step closer to a converged configuration. Here, the convergence is said to have occurred if the depth is greater than or equal to six.

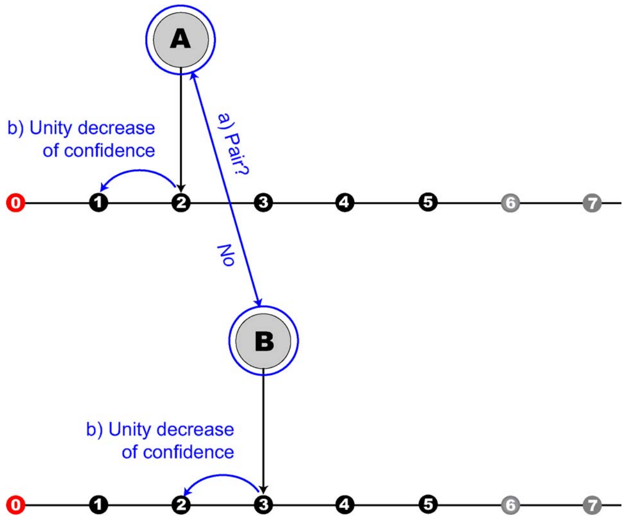

A
b) Unity decrease of confidence
a) Pair?
No
B
b) Unity decrease of confidence
0
1
2
3
4
5
6
7

Fig. 3. Decreasing the confidence of two paired processes, where neither is in the exit state.

The pairing is said to be *successful*, and the two processes will be awarded if the two processes already belong to the same node; otherwise, it is unsuccessful, and the two processes are penalized. In the case of a successful pairing, both the two involved processes will increase their confidence in their current allocation. The situation is graphically illustrated in Fig. 2.

C. Unsuccessful Pairing

When there is an unsuccessful pairing, the two processes are not on the same node. However, this is an indication that the processes might have intensive communication partners with whom they are not currently colocated. Consequently, the confidence in their present location should be decreased, and this situation is illustrated in Fig. 3.

Consider next an unsuccessful pairing, where process A is in the boundary (exit) state of its node and has zero confidence in its location, whereas B has some confidence in its location. In this case, one would desire to bring A to the node of B. This situation is shown in Fig. 4.

Finally, there is the situation where both process A and process B are in the exit states of their nodes. Then, there are three possible alternative migrations.

1) Both A and B may group on A's node.
2) Both A and B may group on B's node.
3) A and B group on a third node.

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

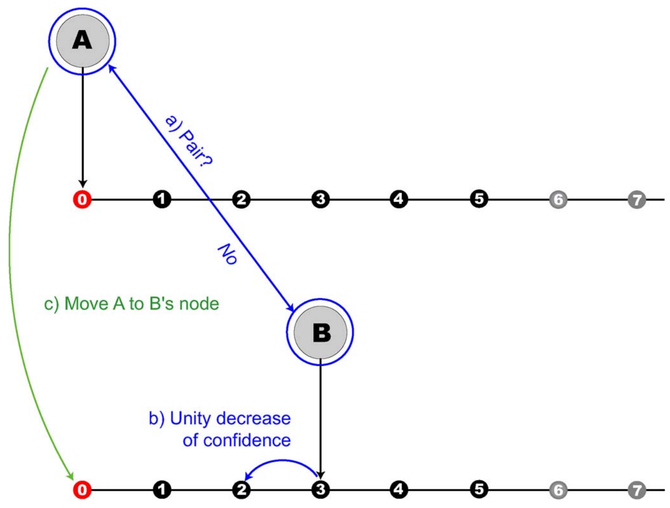


Fig. 4. A and B are unsuccessfully paired: B reduces its confidence as usual, and A will migrate to the node of B.

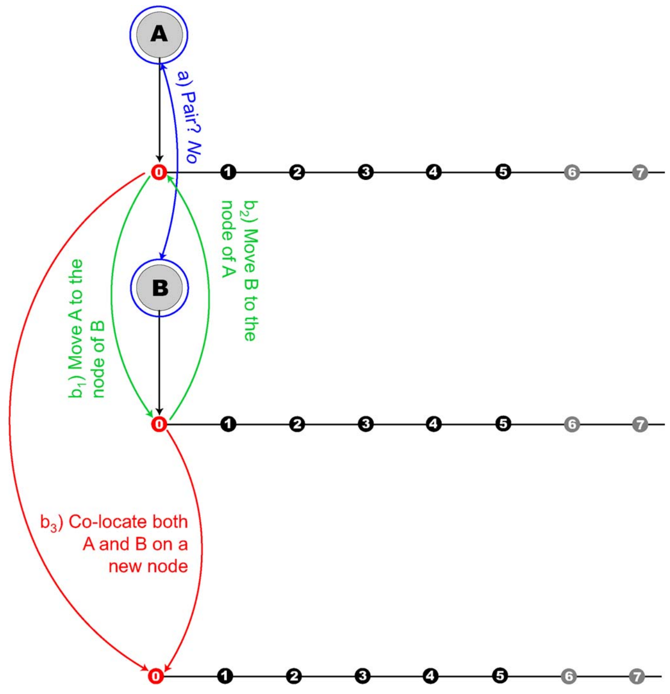

A
a) Pair? No
B
b_1) Move A to the node of B
b_2) Move B to the node of A
b_3) Co-locate both A and B on a new node
0
1
2
3
4
5
6
7

Fig. 5. Three alternatives for colocating two processes A and B both having zero confidence in their present location.

The three alternative moves are shown in Fig. 5. The decision about which move to make is based on how much the involved nodes of the processes A and B violate the constraint (7).

The first alternative is really not an option given that P_A was chosen to be on the node violating most the constraint (7). Therefore, both processes will be assigned to B's node if this is possible without violating the constraint (7); otherwise, both processes will be placed together on the node with the least value for the constraint (7).

D. Algorithm

Fig. 6 shows the pseudoalgorithm used to solve the mapping problem with FSSA. How the paired processes are moved was described in the preceding sections.

```
    Input
        The number of nodes, $|\mathbb{N}|$
        The number of processes, $|\mathbb{P}|$
        The communication patterns $\mathbf{p}_i$ for $i = 1, \ldots, |\mathbb{P}|$
    Assumptions
        There is at least one solution to the mapping problem
    Output
        A feasible process assignment $\mathbf{x}$
1:  begin
2:      do
3:          $N_n$ = The node most violating the constraint (7).
4:          $P_A$ = Randomly selected according to the distribution of the $\tau_i$ for which $x_{i,n} = 1$.
5:          $P_B$ = Randomly selected process based on $\mathbf{p}_A$
6:          Present the Learning Automata with the request to pair $\langle P_A, P_B\rangle$ and move one or both of the processes accordingly
7:      until
8:          None of the constraints (5)–(7) are violated and all process automata are in converged states.
11: end
```

Fig. 6. Pseudocode for the basic algorithm to partition the processes based on first selecting a node and then pairing one process on this with a remote communication partner.

IV. VSSA

The alternative to FSSA is VSSA whose transition and output matrices are time varying [18], [23]. In practice, they are defined in terms of an action probability vector $\mathbf{p}(k)$ with $p_i(k)$ being the probability that action i in the action set $\mathbb{A}$ will be selected at time k out of the $|\mathbb{A}|$ available actions. Hence, $\sum_i p_i(k) = 1$ for all k. The updating rules for the probability vector are either continuous or discrete. The fastest converging LA belong to the VSSA family. Adapting these families of automata to solve the problem considered could hopefully improve the speed of obtaining a solution.

Based on this, Thathachar and Sastry [24] introduced what is known as *estimator algorithms*: The main feature of these algorithms is that they maintain the running estimates for the reward probability of each possible action and use them in the probability-updating equations. Typically, in the first step of the functional cycle, the automaton chooses an action, and the environment generates a response to this action. Based on this response, the estimator algorithm updates the estimate of the reward probability for that action. The change in the action probability vector $\mathbf{p}(k)$ is based on $\hat{\mathbf{d}}(k)$, the vector of the running estimates of the reward probabilities that was updated according to the feedback received from the environment.

A. Pursuit Automata

The pursuit algorithm first introduced by Thathachar and Sastry [24] is a rule for updating the probability vector directly to "pursue" the currently *estimated* best action. Hence, after receiving the feedback on a proposed action, the second step

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

```
    Parameters
        $|\mathbb{A}|$ Number of actions
    Initialisation
1:      Estimates $\hat{\mathbf{d}}$ initialized according to the definition of the estimator.
2:      $p_i(0) = 1/|\mathbb{A}|$ for $i = 1, \ldots, |\mathbb{A}|$
3:  begin
4:      do
5:          Update action probabilities $\mathbf{p}(k)$ as a function of $\mathbf{p}(k-1)$ and $\hat{\mathbf{d}}(k)$.
6:          Select next action according to the probability distribution defined by $\mathbf{p}(k)$.
7:          Execute next action.
8:          Obtain the binary feedback from the environment.
9:          Update the estimates of reward probabilities, $\hat{\mathbf{d}}$, based on the feedback.
11:     until problem solved
12: end
```

Fig. 7. Fundamental pursuit learning algorithm.

is to increase the component of $\mathbf{p}(k)$ whose reward estimate is maximal (the current optimal action) and to decrease the probability of all the other actions. The last step is to update the running estimates for the probability of being rewarded based on the actual reward or penalty received from the environment. The fundamental steps of the algorithm are illustrated in Fig. 7.

The differences between the discrete and continuous versions of the pursuit algorithms occur only in the updating rules for the action probabilities. The discrete pursuit algorithms make changes to the probability vector $\mathbf{p}(k)$ in discrete steps, whereas the continuous versions use a continuous function to update $\mathbf{p}(k)$.

The discrete generalized pursuit automata (DGPA) [25] is the fastest converging automata known. Fundamentally, it is a way to update the elements of the action probability vector. The general idea is that all action probabilities corresponding to actions with a higher reward estimate than the currently chosen action should be increased. The action probabilities corresponding to reward estimates that are less than the currently chosen action should similarly be decreased. Finally, the probability for the chosen action is updated to normalize the probability vector. The algorithm is given in Fig. 8.

Traditionally, the reward estimates $\hat{\mathbf{d}}(k)$ are obtained by maximum likelihood estimation (MLE). The automaton keeps two counters for each action, both initially set to zero. One counts the number of times this action has been selected, and the other counts the number of times the task of selecting this action has been rewarded. Then, the estimated reward probability of a given action is simply the ratio of the number of rewarded selections to the number of selections. Observe that the estimates obtained by MLE do not necessarily sum to unity, but they converge asymptotically to the true values of the d_i's. Observe also that the MLE is optimal if and only if the *true* reward probabilities estimated by $\hat{\mathbf{d}}$ are stationary and do not change over time. If one action is optimal for some time before another action becomes optimal, the automaton will continue to select wrongly the first action until the ratio of the number of rewards to the number of selections has decreased sufficiently for the second action to be chosen.

```
    Parameters
        $|\mathbb{A}|$      Number of actions
        $\epsilon$          Resolution parameter
        $\Delta$            Smallest step size
        $\hat{\mathbf{d}}(k)$   Estimated reward probabilities
    Initialisation
1:      $\Delta = 1/(\epsilon|\mathbb{A}|)$
2:      $_i(0) = 1/|\mathbb{A}|$ for $i = 1, \ldots, |\mathbb{A}|$
3:  begin UPDATE ACTION PROBABILITIES
4:      $K(k)$ = The number of actions with higher reward estimates than the currently chosen action $\alpha$
5:      $p_j(k) = \min\left(p_j(k-1) + \frac{\Delta}{K(k)}, 1\right)$ for $j \neq \alpha : \hat{d}_j > \hat{d}_\alpha$
6:      $p_j(k) = \max\left(p_j(k-1) - \frac{\Delta}{|\mathbb{A}| - K(k)}, 0\right)$ for $j \neq \alpha : \hat{d}_j < \hat{d}_\alpha$
7:      $p_\alpha(k) = 1 - \sum_{j \neq i} p_j(t)$
8:  end
```

Fig. 8. DGPA subroutine to update the action probabilities.

To account for this, a recently introduced family of weak estimators [26], referred to as stochastic learning weak estimates (SLWEs) that are suitable for time-varying distributions, may be successfully used to obtain the probability estimates for the problem at hand [7]–[9]. The SLWE is itself computed using the principles of LA, and it will, in contrast with the MLE, always be a proper probability vector trying to estimate the underlying distribution. Thus, as one of the estimates is increased due to a reward, the other elements will be decreased. This estimation process leads to estimates that do not converge with probability one but rather converge with regard to the first and second moments.

There are basically two options for selecting the automata for the SMP: Each node may be an automaton, or each process is modeled as an automaton as before. In the first case, the "actions" of the nodes would be to select the processes belonging to that class. However, this would be impossible because of the uniqueness constraint (8) since several nodes independently may decide that a certain process is their better "action." A process may either have the nodes as its "actions" or may try to select other peer processes as "actions" as a sort of pairing without involving the nodes. These two strategies are discussed in the following sections.

Thathachar and Sastry [21, pp. 76–82] prove that games of automata like the one suggested here will converge for stationary environments, although pursuit algorithms seem to require full information about the past configuration history in order to converge. This is clearly infeasible, and, therefore, the approach can be seen similar to extending the principles used in [27], where Batalov and Oommen demonstrated that learning machines can be taught to play games even if they are not explicitly provided with the rules of the game.

B. Node Action: Selecting the Better Hosts

In this case, $\mathbb{A} = \mathbb{N}$, and $p_{i,n}$ is the probability for process P_i to be rewarded if it is allocated to node N_n. However, the

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

concept of the reward needs definition. The approach used here is a game of independently moving automata, and moving any process from one node and allocating it to another at time k create a new configuration $\mathbf{x}(k)$. This move is hence rewarded if and only if this *decreases* the objective function (4), i.e., $f(\mathbf{x}(k)) \leq f(\mathbf{x}(k-1))$.

Seen from the moving process P_i, this means that the action corresponding to node N_j can be rewarded some times, while, other times, the very same move will create a penalty depending on where the other processes are at the time of the move. It is, therefore, obvious that the rewards are time dependent, and an estimator like the SLWE supporting time variate distribution should be preferred.

Two innovations suggested in [7] are likely to further improve the situation: It obviously makes no sense to update the reward estimates at the end of the iteration loop as in step 9 in Fig. 7 since the configuration has likely changed the next time the process will move. Thus, the estimator update of step 9 in the algorithm in Fig. 7 is moved before step 5 of the same algorithm.

Second, it makes no sense to update the reward estimates $\hat{\mathbf{d}}(k)$ based on the reward given to the process at some past execution as the configuration has probably changed in the meantime. Instead, a sequence of synthetic rewards is generated by trying to predict what would be a good move for the process P_i as follows: For each of the elements in the communication vector, $\mathbf{w}_i$, one decides if the communication is active or not with probability $w_{i,j}$. If the communication is taken as active, check if P_i can be allocated to the same node as P_j without violating the constraints (5)–(7). If this is the case, then the reward estimates are updated as if the automaton received a reward for choosing the node hosting P_j. Note that, if several of the communication partners of P_i are already allocated to the same node, one may have several rewards for this node, enforcing it as a good choice for reallocating P_i.

This strategy has two benefits: It will better reflect the current situation in the update of the reward estimates, and it avoids computing the objective function (4) but rather queries a subset of nodes for their capacities before moving P_i randomly based on $\mathbf{p}_i$.

C. *Process Action: Autonomous Process Flocking*

A fundamental observation is that it will be beneficial to group together processes with mutual communication on one node since this will limit the strain on the communication link from the node to the other nodes. The idea is to attempt taking advantage of the fact that a process' affinity with the other processes is a static relationship based on the underlying communication digraph of the application. Consequently, it should be possible to *learn* these relations in a more stable way.

Let a process have the other processes as possible actions. When a process P_i arrives to a node and that move is rewarded, i.e., the value of the objective function (4) is decreased, it is an indication that P_i should increase its affinity with the other processes hosted by that node. However, it is also an indication that every other process on that node should increase its affinity with P_k. Indeed, the reward is symmetric, so all processes on the node should reciprocally increase their affinities. In practical terms, a sequence of awards is given to every process on the node for all the actions corresponding to the processes on the node.

Then, there is a need to identify the mapping from the process probabilities to the next host node for a process. Reference [9] proposes an intuitive solution to this by assigning each node a probability that is equal to the sum of action probabilities a process has for the other processes currently hosted on that node. However, also the weights of the processes allocated to a node must be taken into account since the search is for a feasible configuration, and it makes little sense moving a process to a node already overloaded. Details on this enhanced process placement can be found in [9].

V. Digraph Estimation

A general problem with using automata to learn affinities among processes like in the previous approach is that each automaton has the goal to select the *best* action and not a set of good actions. In the stationary case, the probability for the best action will converge to unity, whereas all the other action probabilities will go to zero. Thus, one can never learn the static relations of the process communication digraph this way.

Using a traditional *estimator* might overcome this restriction: The inbound relations will be *learned* through a sequence of interactions with the other processes. In a round-robin fashion, the other processes are queried if they have a relation with the "proposing" process P_i. Consequently, another object P_j provides it with a positive feedback with probability $w_{j,k}$ (which represents a *reward*), and, with probability $1 - w_{j,i}$, it will provide it with a *penalty*. Based on this feedback, P_i will update its estimate $\hat{w}_{j,i}$.

The reward estimates that process P_i obtains $\hat{w}_{\cdot,i}$ can best be produced by an MLE scheme since the relation among the processes is static. If this is the case, the automaton keeps two counters for each of the other processes, both initially set to zero. One counts the number of times this process has been selected, and the other counts the number of times the action of selecting this process has been rewarded by the system. Thus, the estimated reward probability is simply the ratio of the number of rewarded selections to the number of selections. Each individual estimate satisfies $0 \leq \hat{w}_{\cdot,i} \leq 1$. Observe that the estimates obtained by the MLE do not necessarily sum to unity, which is, indeed, valid because the inbound relations are not normalized like the outbound ones.

With the availability of the estimates of the inbound communication, one has again the same need to assign the process to a node. The previously described heuristic proposed in [9] was slightly modified in [8] to account differently for the loading of the node and derive a set of node probabilities used to randomly allocate a process.

VI. Simulation

Extensive simulations were carried out in order to test the different learning algorithms outlined previously. A dedicated

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

TABLE I
FRACTION OF CONVERGED SIMULATIONS FOR DIFFERENT SYSTEM SIZES AND APPLICATION SIZES. MORE THAN 300 SIMULATIONS ARE PERFORMED FOR EACH VALUE PRESENTED. A HYPHEN MEANS THAT AN EXTENSIVE SIMULATION EXPERIMENT HAS NOT BEEN PERFORMED. FOR THE VSSA ALGORITHMS, IT IS BECAUSE OF THE LACK OF CONVERGENCE, AND, FOR THE FSSA ALGORITHM, THERE IS NO REASON TO SIMULATE SMALLER SYSTEMS WHEN IT CONVERGES FOR LARGER AND MORE CHALLENGING ONES

$\|\mathbb{N}\|$	$\|\mathbb{P}\|$	$\frac{\|\mathbb{P}\|}{\|\mathbb{N}\|}$	FSSA	Node action	Process action	Digraph estimation
2	4	2	—	99.7%	96.6%	96.7%
	6	3	—	98.3%	98.0%	98.3%
	8	4	—	98.3%	97.0%	97.3%
4	8	2	100%	98%	85.6%	82.3%
	12	3	—	91%	74.0%	84.0%
	16	4	—	86%	65.0%	82.0%
6	12	2	—	87.3%	60.0%	71.3%
	18	3	—	80.3%	18.6%	12.0%
	24	4	—	36.3%	0.0%	0.0%
16	64	4	100%	—	—	—
	96	6	100%	—	—	—
	128	8	100%	—	—	—
	160	10	100%	—	—	—
32	128	4	100%	—	—	—

simulator was implemented in C++, and, for each run, the following steps were carried out:

1) Initialization of the random number generator with the current time of the computer to ensure that each random sequence was unique. Furthermore, the best known random generator was used [28, pp. 282–286].
2) Assignment of a feasible random baseline partition as it would be useless to seek a solution to the problem unless it was known that a solution would exist. The construction of the baseline partition is the subject in Section VII.
3) Randomly rehash the baseline partition to create an infeasible starting configuration for the learning algorithms.
4) Execute the partitioning algorithm, and stop as soon as a feasible configuration was detected. If the simulation *converged* on a solution, the number of iterations, i.e., a process moves from one node to another, was recorded.
5) Stop if no solution was found after a fixed number of iterations. Various values were used for the iteration limit, increasing with the system size to be comparable in scale with the number of possible configurations, (3).

At least 300 simulation runs were carried out for each problem size and for each algorithm, and the percentage of converged simulations is reported in Table I. Not all problem sizes were simulated for all algorithms since it would be pointless to simulate larger systems for algorithms failing smaller ones. Seven hundred additional simulations were executed for the FSSA algorithm to confirm the reported results. The simulations were executed on a wide range of Linux workstations at the University of Oslo using Condor.[5]

VII. BASELINE PARTITION

The purpose of the baseline partition is to ensure that there is at least one solution to the partition problem. Without this baseline partition, it could be that the problem had no solution and that no algorithm would ever converge. However, defining this baseline partition is nontrivial, and this section defines a solution to this problem and is included here for the completeness of this paper.

A. Allocating Processes to Nodes

The starting point is a partition, i.e., an allocation of all processes onto the nodes. In order to verify if the algorithm finds the baseline partition or some other feasible partition, the processes are allocated in subsequences such that the indexes of the processes on a given node form a sequence. This is achieved as follows.

1) For each process, a random integer is drawn in the interval $[1, \ldots, |\mathbb{N}|]$, representing the initial assignment of this process. If the resulting configuration leaves one or more nodes without any processes, the random assignment is repeated for all processes.
2) The number of processes assigned to each node is counted.
3) A sequence of processes of the same length as the random assignment is allocated to each node.

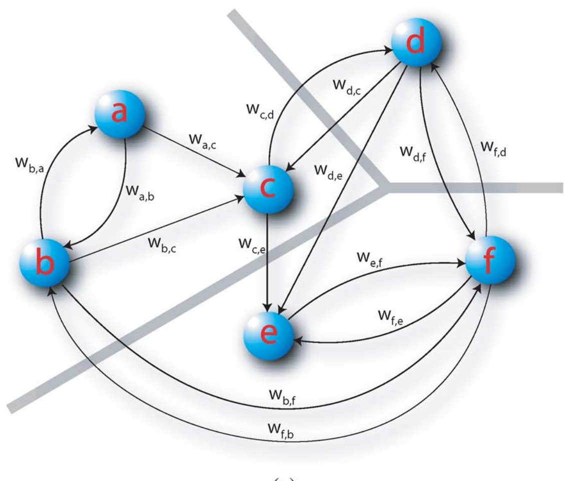
Once the processes have been allocated to the nodes, their weights τ_i are assigned. For each node, a random vector with as many elements as there are assigned processes to the node is formed and normalized. If the constraint (7) is *hard*, i.e., equal to unity, the assigned weights are the elements of this vector; otherwise, the sum of (7) is assumed to be a random number in the interval [0, 1], and the random vector is multiplied with this number before the weights are assigned to the processes.

B. Node Communication

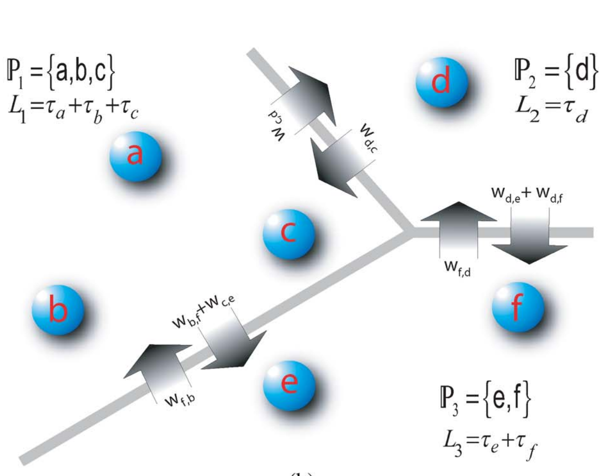
Fig. 9 illustrates a situation where six processes are assigned to three nodes, resulting in a new set of communication vectors among the nodes. When forming the baseline partition, this

[5]See http://www.cs.wisc.edu/condor/.

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.



(a)



(b)

Fig. 9. Six processes with directed communication being partitioned onto three nodes, where the communication among the nodes is based on the original communication digraph. The load of each node is the sum of the weight of the processes. (a) Communication among the processes with the partition of the process set indicated as gray lines. (b) Communication among the nodes under a given partition.

process must be reversed: After allocating the processes onto the nodes, the communication digraph of the nodes must be defined as that in Fig. 9(b), before splitting the outbound and inbound communication of a single node on the processes allocated to that node as that in Fig. 9(a).

The key point is to observe the flow *invariant*: Whatever flows out of one node must flow into all the others. Consequently, this principle establishes a link between the outbound flows and the inbound flows and explains why the flows cannot be selected entirely at random.

Consider the node-to-node communication matrix $\mathbf{W}$, where the rows represent the outbound communication from the nodes and the columns represent the inbound communication. This matrix will be zero diagonal since a node will not communicate with itself, and it will have the following structure:

$$\begin{array}{c|cccccc} & I_1 & I_2 & I_3 & \cdots & I_{|\mathbb{N}|-1} & I_{|\mathbb{N}|} \\ \hline O_1 & 0 & W_{1,2} & W_{1,3} & \cdots & W_{1,|\mathbb{N}|-1} & W_{1,|\mathbb{N}|} \\ O_2 & W_{2,1} & 0 & W_{2,3} & \cdots & W_{2,|\mathbb{N}|-1} & W_{2,|\mathbb{N}|} \\ O_3 & W_{3,1} & W_{3,2} & 0 & \cdots & W_{3,|\mathbb{N}|-1} & W_{3,|\mathbb{N}|} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ O_{|\mathbb{N}|-1} & \cdots & \cdots & \cdots & \cdots & 0 & W_{|\mathbb{N}|-1,|\mathbb{N}|} \\ O_{|\mathbb{N}|} & \cdots & \cdots & \cdots & \cdots & W_{|\mathbb{N}|,|\mathbb{N}|-1} & 0 \end{array}$$

where $O_i = \sum_{j=1}^{|\mathbb{N}|} W_{i,j}$ is the total outbound communication from node N_i and $I_i = \sum_{j=1}^{|\mathbb{N}|} W_{j,i}$ is the total inbound communication. Then, $\sum_i O_i = \sum_i I_i$ from the flow invariant and the fact that the sum over all matrix elements is the same. Furthermore, the inflow to N_i cannot be larger than the combined outflow of all the others with the limiting case corresponding to the situation where all nodes are sending all their output to N_i

$$I_i \leq \left(\sum_{j=1}^{|\mathbb{N}|} O_j\right) - O_i. \tag{13}$$

If the problem is *hard*, then $I_i = 1$ and $O_i = 1$ for all $i = 1, \ldots, |\mathbb{N}|$; otherwise, the total flow is taken as a random number drawn uniformly over the interval $[0, |\mathbb{N}|]$, i.e., $\mathcal{F} \sim U(0, |\mathbb{N}|)$. Then, $O_i \sim U(0,1)$ for all $i = 1, \ldots, |\mathbb{N}|$ such that $\sum_{i=1}^{|\mathbb{N}|} O_i = \mathcal{F}$. The numbers are made to satisfy the latter equality by first drawing a random vector with elements $U(0,1)$ and then normalizing and scaling this to $\mathcal{F}$. Finally, the column sums are drawn uniformly between zero and the upper limit given by (13), i.e., $I_i \sim U(0, \mathcal{F} - O_i)$, again, such that $\sum_{i=1}^{|\mathbb{N}|} I_i = \mathcal{F}$.

The matrix $\mathbf{W}$ is solved recursively. The idea is to randomly draw the elements of the first column and the first row and then compute the new row and column sums for the minor matrix of size $(|\mathbb{N}|-1) \times (|\mathbb{N}|-1)$. The first column elements are taken $W_{i,1} \sim U(0, O_i)$ for $i = 2, \ldots, |\mathbb{N}|$ such that $\sum_{i=2}^{|\mathbb{N}|} W_{i,1} = I_1$. In order to preserve (13) also for the minor matrix, there must be a lower bound for the values drawn for the first row in the major matrix. Since $O_i^{[|\mathbb{N}|-1]} = O_i^{[|\mathbb{N}|]} - W_{i,1}$ and $I_i^{[|\mathbb{N}|-1]} = I_i^{[|\mathbb{N}|]} - W_{1,i}$ for $i = 2, \ldots, |\mathbb{N}|$, the constraint for the minor matrix yields the following lower bound for the major matrix elements in the first row:

$$I_i^{[|\mathbb{N}|-1]} \leq \left(\sum_{j=2}^{|\mathbb{N}|} O_j^{[|\mathbb{N}|-1]}\right) - O_i^{[|\mathbb{N}|-1]}$$

$$I_i^{[|\mathbb{N}|]} - W_{1,i} \leq \left(\sum_{j=2}^{|\mathbb{N}|} O_j^{[|\mathbb{N}|]} - W_{j,1}\right) - \left(O_i^{[|\mathbb{N}|]} - W_{i,1}\right)$$

$$W_{1,i} \geq I_i^{[|\mathbb{N}|]} - \left[\left(\sum_{j=2}^{|\mathbb{N}|} O_j^{[|\mathbb{N}|]}\right) - I_1^{[|\mathbb{N}|]}\right] - \left(O_i^{[|\mathbb{N}|]} - W_{i,1}\right). \tag{14}$$

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

Hence, the first row elements are drawn $W_{1,i} \sim U(W_{1,i}^{\min}, I_i)$ for $i = 2, \ldots, |\mathbb{N}|$, where $W_{1,i}^{\min}$ is the right-hand side in (14).

This process continues down to the 3×3 matrix. In this case, there will be six elements to be chosen in the matrix and $3 + 3$ constraints in the row and column sums. Essentially, this is a fully determined set of linear equations. However, since the sum of the row sums is equal to the sum of the column sums, there is one degree of freedom. Given the matrix

$$\mathbf{W}^{[3]} = \begin{pmatrix} 0 & W_{1,2} & W_{1,3} \\ W_{2,1} & 0 & W_{2,3} \\ W_{3,1} & W_{3,2} & 0 \end{pmatrix}$$

the element $W_{3,2}$ is taken as the free element randomly chosen within a region defined by the row and column sums. Then, the rest of the elements follow from solving the linear system.

C. Process Communication

A process allocated to a node has two types of communication: node external communication with processes assigned to other nodes and node internal communication with the processes on the same node.

With the node communication matrix $\mathbf{W}$ given, the total amount of outbound traffic from N_i to N_j given by $W_{i,j}$ is split randomly over the processes on N_i in a two-step process:

1) A random vector with as many elements as there are processes on N_i is formed and normalized and then scaled with $W_{i,j}$, giving each process' share of the external communication.
2) The process is repeated for each process on N_i, forming a normalized random vector with as many elements as there are destination processes hosted by N_j and scaling this vector to the process' share of N_i's external communication with N_j.

Since a process should be allowed to be a single process on any node without violating the capacity constraints, neither its total outbound communication nor its total inbound communication can exceed unity. The problem is *hard* if the total communication is exactly unity in both directions for all processes and *relaxed* otherwise.

With the upper limit for the communication given for all processes, the node external part can be subtracted, leaving the process' capacity for communicating with the other processes on the same node. Knowing these capacities, the node internal communication among the processes can be allocated essentially by solving the same constrained flow problem previously solved for the nodes, producing a process-to-process communication matrix $\mathbf{w}$.

VIII. Discussion

Fig. 10 shows the number of times any process is moved from one node to another before the feasible configuration has been found. The FSSA algorithm converges for all system sizes and manages to find a feasible configuration even when

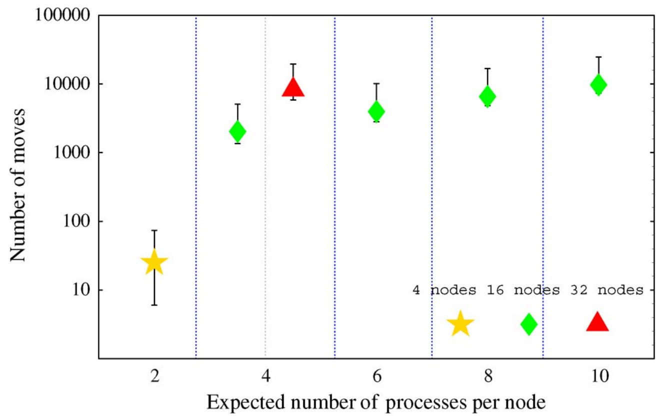


Fig. 10. Median number of moves, with the 95% quantiles, necessary to achieve a feasible mapping using the described FSSA algorithm.

the number of possible configurations is huge.[6] The "level" is mainly given by the number of nodes involved, and, as expected from Section II-B, there is an exponential increase in the number of moves necessary to find a solution with the system size. Remarkably is also the fact that the number of moves needed seems to be fairly constant over all the 1000 simulations performed, with a slightly longer tail in the distribution. This implies that, for all systems, a minimal number of moves must be expected before the convergence starts; however, this is a general aspect of all learning algorithms.

It is surprising that the best learning method gives unacceptable results while an older FSSA method converged perfectly on a feasible solution. From the results, it is evident that the approaches based on the variable-structure pursuit automata, although novel, are efficient only for very small class sizes. In their current form, they do not scale well and fail to find solutions even for moderate system sizes (which could possibly be solved with exact integer programming methods). There may be several reasons for this phenomenon, and this section examines the most obvious ones while describing subjects for further investigations.

A possible problem with the estimators used is that they only update the estimates in response to a *reward*. Given that most of the possible configurations are infeasible and that it is increasingly harder to find a better configuration than the present one, selecting a new host node will most likely give an unfavorable response. The same is the case when trying to select another process in a group or to estimate inbound relations since most of the inbound relations carry a relatively small weight because the weights of all the outbound relations from an object are limited by unity. Hence, by virtue of such an updating strategy, most of the information obtained in an iteration will be ignored, rendering the convergence slower. This may partially explain the poor results seen.

A further complication is that most estimators assume that the observations are independent and identically distributed. This is not the case when the estimator values are used to select the next configuration, which, in turn, generate the feedback

[6]Evaluating (3) for the largest system yields approximately 9.75993×10^{156} configurations, and finding a feasible configuration in about 10 000 iterations as indicated in Fig. 9 is impressive.

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

and the next observation for the estimator. In other words, a new observation is conditioned on the past history of observations. This in-loop estimation violates the premises of the estimators.

The present authors are therefore currently working to develop learning-based estimators that may explore both rewards and penalties and that are suitable for in-loop estimation where the estimated values are used to make decisions that influence future system configurations.

A potential issue with the digraph estimation or with having the processes as actions of the automata is that there is no direct way to assign the new host node to a moving process. Making this assignment only on the grouping of processes seems insufficient in larger systems. When forming the node probability vector for finding the next host for a process, the nodes with insufficient capacity were either excluded or given very low probabilities. This may confine the allocation of the processes to a limited region, preventing a full exploration of all possible configurations. In other words, it might be necessary to overload temporarily a node for the system to be able to pass to a better configuration.

The aforementioned strategies for moving processes seem to indicate that the game of independent automata *may* not converge for this problem. Thathachar and Sastry [21] prove that a game of automata will converge even in the case when they are given individual feedback on their choices. A fundamental assumption in that work, when transcribed to the setting of our current problem, is that the automata should select a single *configuration*. With combinatorial growth in the number of possible configurations, even enumerating them is intractable.

Finally, all processes were allowed to move all the time for all the VSSA approaches. This may not be the ideal strategy since there is no reason why a process should be moved away from a node that is neither overloaded nor exceeding its communication capacity. Rather, it is probably advantageous to move only the processes on the overloaded nodes. Note that this violates the original motivation for the VSSA approaches to find completely decentralized algorithms that are capable of solving the partitioning problem without central coordination. The FSSA solution has an element of global coordination since the processes are moved away only from the most overloaded node. The need for coordination has long been recognized for multiagent systems [29], and future improvements of the algorithms presented here could benefit from existing frameworks [30] and consensus protocols [31]. LA can be considered a special branch of reinforcement learning [32], and, although distributed reinforcement learning is in its infancy, the need for central coordination is already recognized [33].

Abandoning the idea that one game of moving objects may be capable of identifying a feasible configuration (except by pure chance) opens several avenues for further research. One avenue would be to regard this as a game with two groups of learning players in which the processes attempt to learn the digraph of relations, while the nodes, at the same time, actively move the processes between themselves. In such a model, learning can possibly be applied to the selection of the process that has to move out of a node if the node is overloaded. This too is currently being investigated.

IX. Conclusion

This paper considered the problem of partitioning a set $\mathbb{P}$ of $|\mathbb{P}|$ elements (or objects) into $|\mathbb{N}|$ mutually exclusive *classes* (or groups), with the goal of having "similar" elements cluster with each other in the same class. The objects could be linked together in a multiconstraint (and possibly, contradictory) manner. This was motivated and illustrated by the SMP of assigning a set of processes of a parallel application to a set of computing nodes.

After formalizing the problem, we first presented an FSSA algorithm to solve it. The solution, although elegant and efficient, required some centralized coordination. In an attempt to determine a solution that is void of a centralized control mechanism, we subsequently proposed *three* different VSSA algorithms. However, by virtue of gaining these advantages, they forfeit some of the scalability features of the fixed-structure algorithm.

Our conjecture is that a fully decentralized LA solution does not exist. Rather, since it is doubtful that a fully distributed algorithm can be found, the question of arriving at a scalable solution by utilizing some form of coordination (i.e., achieving the goal with the minimum centralized control) remains open.

Acknowledgment

The authors would like to thank Dr. D. Calitoiu who is now with the Bank of America in Ottawa for the fruitful discussions when he was at Carleton and Prof. O. Lysne of the University of Oslo and Simula Research Laboratories for the continued support.

References

[1] N. Baba and Y. Mogami, "A consideration on the learning behaviors of the HSLA under the nonstationary multiteacher environment and their application to simulation and gaming," in *Proc. KES, Knowledge-Based Intell. Inf. Eng. Syst.*, Sep. 2004, pp. 792–798.

[2] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[3] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Berlin, Germany: Springer-Verlag, 2004.

[4] M. Garey and D. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.

[5] B. J. Oommen and D. C. Y. Ma, "Deterministic learning automata solutions to the equipartitioning problem," *IEEE Trans. Comput.*, vol. 37, no. 1, pp. 2–13, Jan. 1988.

[6] G. Horn and B. J. Oommen, "A fixed-structure learning automaton solution to the stochastic static mapping problem," in *Proc. 19th IEEE Int. Parallel Distrib. Process. Symp.*, H. J. Siegel, D. A. Bader, and J.-L. Gaudiot, Eds., Denver, CO, Apr. 2005, p. 297.2.

[7] G. Horn and B. J. Oommen, "Generalised pursuit learning automata for non-stationary environments applied to the stochastic static mapping problem," in *Proc. 11th Int. Conf. ISAS, 2nd Int. Conf. CITSA*, J. Aguilar, H.-W. Chu, J. St-Amand, I. Galkin, and P. Chantzimisios, Eds., Orlando, FL, Jul. 2005, vol. 1, pp. 91–97.

[8] G. Horn and B. J. Oommen, "Towards a learning automata solution to the multi-constraint partitioning problem," in *Proc. IEEE Conf. CIS*, Bangkok, Thailand, Jun. 2006, pp. 1–8.

[9] G. Horn and B. J. Oommen, "An application of a game of discrete generalised pursuit automata to solve a multi-constraint partitioning problem," in *Proc. IEEE Int. Conf. SMC*, Taipei, Taiwan, Oct. 2006, vol. 2, pp. 1042–1049.

[10] T. Muntean and E.-G. Talbi, "Métodes de placement statique des processus sur architectures parallèles," *T.S.I—Technique et Sicence Informatiques*, vol. 10, no. 5, pp. 355–373, 1991.

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.

[11] M. A. Senar, A. Ripoll, A. Cortes, and E. Luque, “Clustering and reassignment-based mapping strategy for message-passing architectures,” *J. Syst. Archit.*, vol. 48, pp. 267–283, 2003.

[12] Y.-K. Kwok and I. Ahmad, “Static scheduling algorithms for allocating directed task graphs to multiprocessors,” *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[13] F. Berman and L. Snyder, “On mapping parallel algorithms into parallel architectures,” *J. Parallel Distrib. Comput.*, vol. 4, no. 5, pp. 439–458, Oct. 1987.

[14] S. H. Bokhari, “On the mapping problem,” *IEEE Trans. Comput.*, vol. C-30, no. 3, pp. 207–214, Mar. 1981.

[15] G. Horn, O. Lysne, and T. Skeie, “The existence of a network of fixed-sized switches that satisfies any communication needs,” in *Proc. Int. Conf. PDPTA*, 2004, pp. 1056–1062.

[16] J. E. Boillat and P. G. Kropf, *A Fast Distributed Mapping Algorithm*, vol. 457, *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 1990, pp. 405–416.

[17] L. Comtet, *Advanced Combinatorics: The Art of Finite and Infinite Expansions*. Dordrecht, The Netherlands: Reidel, 1974.

[18] K. S. Narendra and M. A. L. Thathachar, *Learning Automata*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

[19] M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis, “Learning automata: Theory, paradigms and applications,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. SMC-32, no. 6, pp. 706–709, Dec. 2002.

[20] A. S. Poznyak and K. Najim, *Learning Automata and Stochastic Optimization*. Berlin, Germany: Springer-Verlag, 1997.

[21] M. A. L. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Boston, MA: Kluwer, 2004.

[22] W. Gale, S. Das, and C. T. Yu, “Improvements to an algorithm for equipartitioning,” *IEEE Trans. Comput.*, vol. 39, no. 5, pp. 706–710, May 1990.

[23] V. I. Varshavskii and I. P. Vorontsova, “On the behaviour of stochastic automata with a variable structure,” *Autom. Remote Control*, vol. 24, pp. 327–333, Mar. 1963.

[24] M. A. L. Thathachar and P. S. Sastry, “A new approach to designing reinforcement schemes for learning automata,” *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, no. 1, pp. 168–175, Jan./Feb. 1985.

[25] M. Agache and B. J. Oommen, “Generalized pursuit learning schemes: New families of continuous and discretized learning automata,” *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 32, no. 6, pp. 738–749, Dec. 2002.

[26] B. J. Oommen and L. Rueda, “Stochastic learning-based weak estimation of multinomial random variables and its applications to pattern recognition in non-stationary environments,” *Pattern Recognit.*, vol. 39, no. 3, pp. 328–341, Mar. 2006.

[27] D. V. Batalov and B. J. Oommen, “On playing games without knowing the rules,” in *Proc. Joint 9th IFSA World Congr., 20th NAFIPS Int. Conf.*, M. H. Smith, W. A. Gruver, and L. O. Hall, Eds., Vancouver, BC, Canada, Jul. 2001, vol. 4, pp. 1862–1868.

[28] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[29] V. R. Lesser, “Cooperative multiagent systems: A personal view of the state of the art,” *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 133–142, Jan. 1999.

[30] V. R. Lesser, “Reflections on the nature of multi-agent coordination and its implications for an agent architecture,” *Auton. Agents Multi-Agent Syst.*, vol. 1, no. 1, pp. 89–111, Mar. 1998.

[31] W. Ren, R. W. Beard, and E. M. Atkins, “A survey of consensus problems in multi-agent coordination,” in *Proc. Amer. Control Conf.*, Portland, OR, Jun. 2005, vol. 3, pp. 1859–1864.

[32] R. Sutton and A. G. Barto, *Reinforcement Learning*. Boston, MA: MIT Press, 1998.

[33] J. Dowling, E. Curran, R. Cunningham, and V. Cahill, “Using feedback in collaborative reinforcement learning to adaptively optimize MANET routing,” *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 35, no. 3, pp. 360–372, May 2005.

Geir Horn was born in Oslo, Norway, on May 25, 1967. He received the Cand. Scient. degree in theoretical cybernetics from the University of Oslo, Oslo, in 1993.

In 1993, he joined the Center for Industrial Research, Oslo. In 1996, he was responsible for the IT sector with the industrial attaché’s office of the Norwegian embassy in Paris. From 1997 to 2002, he was the Research Director with SINTEF Electronics and Cybernetics. Since 1997, he has been heavily used by the European Commission as a Scientific Evaluator of research proposals and a Reviewer of ongoing research projects. From 2003 to 2006, he was with the Simula Research Laboratory for fundamental research. Since 2007, he has been with SINTEF, Oslo, as a Senior Scientist. He has been the Project Manager of nine European collaborative research consortia involving industry, research institutes, and universities. In addition, he has been actively contributing to another four projects, where he led the SINTEF team. His research started with computer communication that took him into parallel high-performance computing and further to cluster computing with a special interest in optimization techniques. Since mathematical programming does not scale to the size of the problems considered, he took interest in automata learning as an optimization heuristic. His current focus is on using learning to support adaptation and reconfiguration in distributed and mobile ubiquitous autonomous systems. He has authored and coauthored more than 30 peer-reviewed papers.

Mr. Horn was leading the East Norway Chapter of the Norwegian Computer Society in 2002 and was the Elected President of the Norwegian Computer Society for the period 2006–2008.

B. John Oommen (F’03) was born in Coonoor, India, on September 9, 1953. He received the B.Tech. degree from the Indian Institute of Technology, Madras, India, in 1975, the M.E. degree from the Indian Institute of Science, Bangalore, India, in 1977, and the M.S. and Ph.D. degrees from Purdue University, West Lafayette, IN, in 1979 and 1982, respectively.

In 1981, he joined the School of Computer Science, Carleton University, Ottawa, ON, Canada, where he is currently a Full Professor. His research interests include automata learning, adaptive data structures, statistical and syntactic pattern recognition, stochastic algorithms, and partitioning algorithms. He is the author of more than 310 refereed journal and conference publications.

Dr. Oommen has been awarded the honorary rank of *Chancellor’s Professor*, which is a lifetime award from Carleton University, since July 2006. He is the holder of an *Adjunct Professorship* with the Department of Information and Communication Technology, University of Agder, Grimstad, Norway. He is a Fellow of the International Association for Pattern Recognition. He has been on the Editorial Board of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS and Pattern Recognition.

Authorized licensed use limited to: UNIVERSITY OF AGDER. Downloaded on May 19,2010 at 07:15:00 UTC from IEEE Xplore. Restrictions apply.