ON SOLVING THE CAPACITY ASSIGNMENT PROBLEM USING CONTINUOUS LEARNING AUTOMATA[*]

B. John Oommen and T. Dale Roberts

School of Computer Science
Carleton University
Ottawa ; Canada : K1S 5B6.

Abstract. The Capacity Assignment problem focuses on finding the best possible set of capacities for the links that satisfy the traffic requirements in a prioritized network while minimizing the cost. Apart from the traditional methods for solving this NP-Hard problem, one new method that uses Learning Automata (LA) strategies has been recently reported. This method uses discretized learning automata [1]. The present paper shows how the problem can be solved using continuous learning automata. The paper considers the realistic scenario when different classes of packets with different packet lengths and priorities are transmitted over the networks. After presenting the two well-known solutions to the problem (due to Marayuma and Tang [2], and Levi and Ersoy [3]) we introduce our new method that uses continuous LA, which is comparable to the discretized version, and is probably the fastest and most accurate scheme available.

I. INTRODUCTION

In this paper we study the Capacity Assignment problem which focuses on finding the best possible set of capacities for the links that satisfies the traffic requirements in a prioritized network while minimizing the cost. Unlike most approaches, which consider a single class of packets flowing through the network, we base our study on the more realistic assumption that different classes of packets with different packet lengths and priorities are transmitted over the networks. Apart from giving a brief overview of the problem and a report of the existing schemes, we present a new continuous learning automata strategy for solving the problem. This strategy is analogous to the discretized version already reported [1] except that it operates in a continuous probability space, and is thus easier to comprehend since it is more akin to the well-studied families of learning automata.

Data networks are divided into three main groups which are characterized by their size, these are Local Area Networks (LANs), Metropolitan Area Networks (MANs) and Wide Area Networks (WANs). An Internetwork is comprised of several of these networks linked together, such as the Internet. Most applications of

[*] Partially supported by the Natural Sciences and Engineering Research Council of Canada. Contact e-mail address : oommen@scs.carleton.ca.

computer networks deal with the transmission of logical units of information or messages, which are sequences of data items of arbitrary length. However, before a message can be transmitted it must be subdivided into packets. The simplest form of a packet is a sequence of binary data elements of restricted length, together with addressing information sufficient to identify the sending and receiving computers and an error correcting code.

There are several tradeoffs to be considered when designing a network system. Some of these are difficult to quantify since they are criteria used to decide whether the overall network design is satisfactory. This decision is based on the designer's experience and familiarity with the requirements of the individual system. As there are several components to this area, a detailed examination of the pertinent factors, which are primarily cost and performance, can be found in [4] and [5].

In the process of designing computer networks the designer is confronted with a trade-off between costs and performance. Some of the parameters effecting the cost and performance parameters used in a general design process are listed above, but, in practice, only a subset of these factors are considered in the actual design. In this paper we study scenarios in which the factors considered include the location of the nodes and potential links, as well as possible routing strategies and link capacities.

The **Capacity Assignment (CA) Problem** specifically addresses the need for a method of determining a network configuration that minimizes the total cost while satisfying traffic requirements across all links. This is accomplished by selecting the capacity of each link from a discrete set of candidate capacities that have individual associated cost and performance attributes. Although problems of this type occur in all networks, in this paper, we will only examine the capacity assignment for prioritized networks. In prioritized networks, packets are assigned to a specific priority class which indicates the level of importance of their delivery. Packets of lower priority will be given preference and separate queues will be maintained for each class.

The currently acclaimed solutions to the problem are primarily based on heuristics that attempt to determine the lowest cost configuration once the set of requirements are specified. These requirements include the topology, the average packet rate, or the routing, for each link, as well as the priorities and the delay bounds for each class of packets. The result obtained is a capacity assignment vector for the network, which satisfies the delay constraints of each packet class at the lowest cost.

The primary contribution of this paper is to present a continuous Learning Automaton (LA) solution to the CA problem. Apart from this fundamental contribution of the paper, the essential idea of using LA which have actions in a "meta-space" (i.e., the automata decide on a *strategy* which in turn determines the physical action to be taken in the real-life problem) is novel to this paper and its earlier counterpart [1]. This will be clarified in Section IV.

I.1 ASSUMPTIONS AND DELAY FORMULAE

The model used for all the solutions presented have the following features [3]

:

1. Standard Assumptions : (a) The message arrival pattern is Poissonly distributed, and (b) The message lengths are exponentially distributed.
2. Packets : There are multiple classes of packets, each packet with its own (a) Average packet length, (b) Maximum allowable delay and (c) Unique priority level, where a lower priority takes precedence.
3. Link capacities are chosen from a finite set of predefined capacities with an associated fixed setup cost, and variable cost/km.
4. Given as input to the system are the (a) Flow on each link for each message class, (b) Average packet length measured in bits, (c) Maximum allowable delay for each packet class measured in seconds, (d) Priority of each packet class, (e) Link lengths measured in kilometers, and (f) Candidate capacities and their associated cost factors measured in bps and dollars respectively.
5. A non-preemptive FIFO queuing system [6] is used to calculate the average link delay and the average network delay for each class of packet.
6. Propagation and nodal processing delays are assumed to be zero.

Based on the standard network delay expressions [3], [6], [7], all the researchers in the field have used the following formulae for the network delay cost :

$$T_{jk} = \frac{h_j \cdot \left(\sum_l \frac{l_{jl} \cdot m_l}{h_j \cdot C_j}\right)^2}{(1-U_{r-1})(1-U_r)} + \frac{m_k}{C_j} \tag{1.1}$$

$$U_r = \sum_{l \in V_r} \frac{l_{jl} \cdot m_l}{C_j} \tag{1.2}$$

$$Z_k = \frac{\sum_j T_{jk} \cdot l_{jk}}{g_k}. \tag{1.3}$$

In the above, T_{jk} is the Average Link Delay for packet class k on link j, U_r is the Utilization due to the packets of priority 1 through r (inclusive),V_r is the set of classes whose priority level is in between 1 and r (inclusive), Z_k is the Average Delay for packet class k, $h_j = \sum_l l_{jl}$ is the Total Packet Rate on link j, $g_k = \sum_j l_{jk}$ is the Total Rate of packet class k entering the network, l_{jk} is the Average Packet Rate for class k on link j, m_k is the Average Bit Length of class k packets, and C_j is the Capacity of link j. As a result of the above, it can be shown that the problem reduces to an integer programming problem, the details of which can be found in [4].

II. Previous Solutions

II.1 The Marayuma -Tang Solution

The Marayuma/Tang (MT-CA) solution to the Capacity Assignment (CA) problem [2] is based on several low level heuristic routines adapted for total network cost optimization. Each routine accomplishes a specific task designed for the various phases of the cost optimization process. These heuristics are then combined, based on the results of several experiments, to give a composite algorithm. We briefly describe each of them below but the details of the pseudocode can be found in [2], [3] and [4].

There are two initial capacity assignment heuristics, SetHigh and SetLow:

(a) **SetHigh**: In this procedure each link is assigned the maximum available capacity.

(b) **SetLow**: On invocation each link is assigned the minimum available capacity.

The actual cost optimization heuristics, where the motivating concept is to decide on increasing or decreasing the capacities using various cost/delay trade-offs, are:

(a) **Procedure AddFast**: This procedure is invoked when all of the packet delay requirements are not being satisfied and it is necessary to raise the link capacities while simultaneously raising the network cost, until the bounds are satisfied.

(b) **Procedure DropFast**: This procedure is invoked when all of the packet delay requirements are being satisfied but it is necessary to lower the link capacities, and thus lower the network cost, while simultaneously satisfying the delay bounds.

(c) **Procedure Exc**: This procedure attempts to improve the network cost by pairwise link capacity perturbations.

To allow the concatenation of these heuristics, the algorithm provides two interfaces, ResetHigh, used by DropFast, and ResetLow, used by AddFast below :

(a) **ResetHigh**: Here the capacity of each link is increased to the next higher one.

(b) **ResetLow**: Here the capacity of each link is decreased to the next lower one.

After performing several experiments using these heuristics on a number of different problems, Marayuma/Tang determined that a solution given by one heuristic can often be improved by running other heuristics consecutively. The MT-CA algorithm is the best such composite algorithm (see [2], [3] and [4]).

II.2 THE LEVI/ERSOY SOLUTION

To our knowledge the faster and more accurate scheme is the Levi/Ersoy solution to the CA problem (LE-CA) [3], based on simulated annealing. The process begins with an initial random, feasible solution and creates neighbor solutions at each iteration. If the value of the objective function of the neighbor is better than that of the previous solution, the neighbor solution is accepted unconditionally. If, however, the value of the objective function of the neighbor solution is worse than the previous solution it is accepted with a certain probability. This probability is the **Acceptance Probability** and is lowered according to a distribution called the **Cooling Schedule**.

Since the simulated annealing process is a multi-purpose method, its basic properties must be adopted for the CA problem. In this case, the solution will be a **Capacity Assignment Vector**, C, for the links of the network. Therefore, $C = (C_1, C_2, C_3, ..., C_i, ..., C_m)$ where m is the total number of links and C_i takes a value from the set of possible link types/capacities. The objective function is the minimization of the total cost of the links. Neighbor solutions, or assignment vectors, are found by first selecting a random link and randomly increasing or decreasing its capacity by one step. Feasibility is constantly monitored and non-feasible solutions are never accepted. The pseudocode for the actual algorithm is given in [3] and [4].

III. LEARNING AUTOMATA

Learning Automata have been used to model biological learning systems and to find the optimal action which is offered by a random environment. The learning is accomplished by actually interacting with the environment and processing its responses to the actions that are chosen, while gradually converging toward an ultimate goal. A complete study of the theory and applications of this subject can be found in [8], [9].

The learning loop involves two entities, the **Random Environment** (RE) and a **Learning Automaton** (LA). Learning is achieved by the automaton interacting with the environment by processing responses to various actions and the intention is that the LA learns the optimal action offered by the environment.

The actual process of learning is represented as a set of interactions between the RE and the LA. The LA is offered a set of actions $\{a_1, ..., a_r\}$ by the RE it interacts with, and is limited to choosing only one of these actions at any given time. Once the LA decides on an action a_i , this action will serve as input to the RE. The RE will then respond to the input by either giving a **reward**, (‘0’), or a **penalty,** (‘1’), based on the **penalty probability** c_i associated with a_i . Based upon the response from the RE and the current information it has accumulated so far, the LA decides on its next action and the process repeats. The intention is that the LA learns the **optimal action** (that is, the action which has the minimum **penalty probability**), and eventually chooses this action more frequently than any other action.

Variable Structure Stochastic Automata (VSSA) can be described in terms of time-varying transition and output matrices. However, they are usually completely defined in terms of **action probability updating schemes** which are either **continuous** (operate in the continuous space [0, 1]) or **discrete** (operate in steps in the [0, 1] space). The action probability vector $P(n)$ of an r-action LA is $[p_1(n), ..., p_r(n)]^T$ where, $p_i(n)$ is the probability of choosing action a_i at time ‘n’, and satisfies $0 \leq p_i(n) \leq 1$, and the sum of $p_i(n)$ is unity.

A VSSA can be formally defined as a quadruple (a, P, b, T), where a, P, b, are described above, and T is the updating scheme. It is a map from P ´ b to P, and defines the method of updating the action probabilities on receiving an input from the RE. Also they can either be ergodic or absorbing in their Markovian behavior. Since we require an absorbing strategy, the updating rule we shall use is for the Linear Reward-Inaction (L_{RI}) scheme. The updating rules for the L_{RI} scheme are as follows, and its e-optimal and absorbing properties can be found in [8], [9].

$$p_i(n+1) = 1 - \sum_{j^1 i} l_r\, p_j(n) \qquad \text{if} \qquad a_i \text{ is chosen and } b=0$$
$$p_j(n+1) = l_r\, p_j(n) \qquad \text{if} \qquad a_i \text{ is chosen and } b=0$$

$p_j(n+1) = p_j(n)$ if a_i , a_j chosen, and b=1,

where l_r ($0 < l_r < 1$) is the parameter of the scheme. Typically, l_r is close to unity.

IV. THE CONTINUOUS AUTOMATA SOLUTION TO CA

We now propose a continuous LA which can be used to solve the CA problem. The Continuous Automata Solution to CA (CASCA) algorithm is faster than the MT and LE algorithms and also produces superior cost results.

This solution to the CA problem utilizes the capacity assignment vector nomenclature discussed for the Levi/Ersoy solution [3]. The capacities of the links are represented by a vector of the form $(C_1, C_2, ..., C_i, ..., C_n)$,
where C_i is chosen from a finite set of capacities (e.g. 1200, 2400, ..., etc.), and n is the maximum number of links.

In this solution each of the possible link capacities of the capacity assignment vector has an associated **probability vector** of the form (I_{ij}, S_{ij}, D_{ij}), where

I_{ij} is the probability that the current capacity *j* of link *i* is *increased*,

S_{ij} is the probability that the current capacity *j* of link *i* is *unchanged*, and

D_{ij} is the probability that the current capacity *j* of link *i* is *decreased*.

The final solution vector will be comprised of the capacities, C_i, that exhibit S_{ij} probability values that are closest to the converging value of unity. In a practical implementation this value is specified by the user and is reasonably close to unity. Indeed, the closer this value is to unity, the higher the level of accuracy.

We now present the various initial settings for the probability vector. By virtue of the various values for C_i, there are three possible settings for the initial probability vector (I_{ij}, S_{ij}, D_{ij}) given below as Init1, Init2 and Init3 respectively. To explain how this is done, we shall refer to the index of a capacity as its “capacity-index”. Thus, if the set of possible capacities for a link are {1200, 2400, 3600, 4800, 9600} the corresponding capacity-indices are {0, 1, 2, 3, 4} respectively. Using this terminology we shall explain the initial settings and the updating strategies for our scheme.

Init 1: This is the scenario when the capacity-index of the link is at the *lowest* possible value, 0, called the left boundary state. This means that the capacity cannot be lowered further. In such a case,

$I_{i0} = 1/2$, $S_{i0} = 1/2$, $D_{i0} = 0$,

because the value can be increased or stay the same, but cannot be decreased.

Init 2: This is the scenario where the capacity-index of the link is at the *highest* possible value, n, called the right boundary state. This means that the capacity cannot be raised further. Thus,

$I_{in} = 0$, $S_{in} = 1/2$, $D_{in} = 1/2$,

because the value can be decreased or stay the same, but cannot be increased.

Init 3: This is the scenario where the capacity-index of the link is at one of the interior values, referred to as the interior state. This means that the capacity can be raised or lowered or maintained the same, and hence,

$I_{ij} = 1/3$, $S_{ij} = 1/3$, $D_{ij} = 1/3$ for $0 < j < n$.

The next problem that arises is that of determining when, and how, to modify the probability values for a given link/capacity combination. Initially, a random feasible capacity assignment vector is chosen and assumed to be the current best solution. After this step, the algorithm enters the learning phase which attempts to find a superior cost by raising/lowering the capacities of the links using the L_{RI} strategy. At each step of this process the capacity of *every single link* is raised, lowered or kept the same based on the current action probability vector associated with the link. Based on the properties of the new capacity vector, the associated probability vector for this assignment is modified in two cases to yield the updated solution and the new capacity probability vector. We consider each of these cases individually.

Case 1 : The new capacity assignment is feasible. Since this means that no delay constraints are violated, the probability vector is modified in the following manner :

(a) If the capacity was increased we raise D_{ij}, the Decrease probability of the link,
(b) If the capacity stayed the same we raise S_{ij}, the Stay probability of the link, and,
(c) If the capacity was decreased we raise D_{ij}, the Decrease probability of the link.

Case 2 : The new capacity assignment is feasible *and* the cost of the network has been reduced. Since this means that the new assignment results in a lower cost than the previous best solution the probability vector is modified as :

(a) If the capacity was increased we raise D_{ij}, the Decrease probability of the link,
(b) If the capacity stayed the same we raise S_{ij}, the Stay probability of the link, and,
(c) If the capacity was decreased we raise S_{ij}, the Stay probability of the link.

It is important to remember that we are always trying to minimize cost, we thus never attempt to reward an increase in cost, by raising the increase probability, I_{ij}.

The next question we encounter is one of determining the degree by which the probability vectors are modified. These are done in terms of two user defined quantities - the first, λ_{R1}, is the reward parameter to be used when a feasible solution is reached, and the second, λ_{R2}, is used when the solution *also* has a lower cost. As in learning theory, the closer these values are to unity, the more accurate the solution. It should also be noted that the rate of convergence to the optimal probability vector decreases as the parameters increase.

As stated previously, this paper not only presents a new solution for the CA problem, but also introduces a new way of implementing LA. Unlike traditional LA, in our current philosophy we proceed from a “meta-level” whereby each LA chooses the *strategy* (increase the capacity, decrease the capacity, or let the capacity remain unchanged) which is then invoked on the selected link to set the new capacity. In this way the LA always selects its choice from a different action space rather than from a vector consisting of all the available capacities. The actual algorithm is in [4], [5].

V. EXPERIMENTAL RESULTS

In order to evaluate the quality of potential solutions to the CA problem an experimental test bench [5] must be established. This mechanism will establish a base from which the results of the algorithms can be assessed in terms of the comparison criteria. In this case the comparison criteria is the cost of the solution and the execution time. The test bench consists of the two main components described below.

First, the potential link capacities and their associated cost factors are specified as inputs. Each link capacity has two cost entries - the initial setup cost of establishing the link, and a cost per kilometer of the length of the link. Each of these cost factors increases as the capacity of the link increases.

The next step is to establish a set of sample networks that can be used to test the various solution algorithms. Each network will possess certain characteristics that remain the same for each algorithm, and therefore allow the results of the solutions to be compared fairly. The set of networks that will be used in this paper are shown in Table 5.1 below. Each network has a unique I.D. number given in column 1 and is composed of a number of nodes connected by a number of links, given in column 2, with the average length of the links given in column 3. Each network will carry multiple classes of packets with unique priority levels. The classes of packets which the network carries is given in column 4 while the average packet rate requirements, for each class over the entire network, is given in column 5.

In the suite of networks used in the test bench the network I.D. indicates the average size and complexity of the network. This means that network 4 is substantially more complex when compared with network 1 in terms of the number of links and the type and quantity of packet traffic carried.

NET I.D.	# LINK	AVE: LINK LEN:	PACKET CLASSES	AVE: PACKET RATES	PACKET PRIORITY	DELAY BOUND	PACKET LEN
1	6	54.67	1	13	3	0.013146	160
			2	13.5	2	0.051933	560
			3	14.5	1	0.914357	400
2	8	58.75	1	14.375	3	0.013146	160
			2	15.625	2	0.051933	560
			3	15.125	1	0.914357	400
			4	15.5	4	0.009845	322
3	12	58.08	1	15.417	3	0.053146	160
			2	15	2	0.151933	560
			3	15.5	1	0.914357	400
			4	17.083	4	0.029845	322
4	12	58.08	1	15.417	3	0.053146	160
			2	17	2	0.151933	560
			3	15.5	1	0.914357	400
			4	17.083	4	0.029845	322
			5	17.33	5	0.000984	12
5	48	54.67	1	13	3	0.013146	160
			2	13.5	2	0.051933	560
			3	14.5	1	0.914357	400

Table 5.1 : Characteristic values of the networks used in the test-bench.

Each of the sample networks that is used to test the algorithms carry a distinct type of packet traffic, and these are catalogued in Table 5.1 above. Each network, given by the network I.D. in column 1, carries a number of different packet classes, given in column 4. Each packet class has its own distinct priority, given in column 6, delay bound, given in column 7, and length, given in column 8. The delay bound indicates the maximum amount of time that the packet can stay undelivered in the network. This type of network carries packets of three different types:

1. Packet class one has a priority level of three. Each packet of this class has an average length of 160 bits with a maximum allowable delay of 0.013146 seconds.
2. Packet class two has a priority level of two. Each packet of this class has an average length of 560 bits with a maximum allowable delay of 0.051933 seconds.
3. Packet class one has a priority level of one. Each packet of this class has an average length of 400 bits with a maximum allowable delay of 0.914357 seconds.

A sample network similar to Network Type 1 has nodes {1,2,3,4,5}, and edges

{(1,2), (1,4), (3,4), (3,5), (4,5) and (2,5)}

with edge lengths L1, L2, L3, L4, L5 and L6 respectively. Each of the six links, L1 - L6, can be assigned a single capacity value from Table 5.1 and the average of the lengths will be specified by the quantity "average length" of Network Type 1. Additional details of the setup/variable costs are included in [4], [5].

In order to demonstrate that the new algorithm achieved a level of performance that surpassed both the MT and LE algorithms, an extensive range of tests were performed. The best results obtained are given in the table below. The result of each test is measured in terms of two parameters, Cost and Time which are used for comparison with the previous algorithms. In the interest of time we have only considered one large network in this series of tests, namely Network #5 which consists of 48 links, since the execution times of the previous solutions, especially the MT algorithm, take a considerable time to produce results for larger networks.

Scheme	**Category**	**Net 1**	**Net 2**	**Net 3**	**Net 4**	**Net 5**
MT-CA	Cost ($) Time (sec)	5735.04 0.22	12686.10 0.77	11669.30 1.86	53765.90 2.08	43341.90 67.45
LE-CA	Cost ($) Time (sec)	5735.04 0.22	7214.22 1.21	10295.70 1.10	45709.60 1.93	39838.40 4.01
CASCA	Cost ($) Time (sec)	4907.68 0.11	6937.90 0.39	9909.11 0.70	40348.90 1.33	37307.40 4.12

Table 5.2 : Best results for all algorithms.

The results displayed in the tables demonstrates that the LA solution to the CA problem produces superior results when compared with both the MT and LE solutions. The new algorithm also has consistently lower execution times in most cases when compared to either of the previous solutions. For example, we consider the tests for Network #4 [5, 4]. The MT algorithm finds its best cost as $53,765.90 and takes 2.08 seconds while the LE algorithm finds a best cost of $45,709.60 and takes 1.93 seconds. The CASCA algorithm finds a best cost of $40348.90 and takes only 1.33 seconds which is superior to either of the previous best costs. Hundreds of other experiments have been carried out which demonstrate identical properties. More detailed results can be found in [4] and [5].

It is obvious that as the reward values get closer to unity the accuracy of each cost value improves but the execution times also increases. This means that the algorithm can be optimized for speed (decrease the parameters λ_{R1}, λ_{R2}), accuracy (increase the parameters λ_{R1}, λ_{R2}) or some combination of the two that the user finds appropriate for his particular requirements. Also, the value of λ_{R2} should always be set lower than, or equal to, λ_{R1} since this is only invoked when a lower cost solution is found and is not used as much as λ_{R1} invoked when any feasible solution is found.

VI. CONCLUSIONS

In this paper we have studied the Capacity Assignment (CA) problem. This problem focuses on finding the lowest cost link capacity assignments that satisfy certain delay constraints for several distinct classes of packets that traverse the network. Our fundamental contribution has been to design the Continuous Automata Solution to CA (CASCA) algorithm which is the LA solution to the problem. This algorithm generally produces superior low cost capacity assignment when compared with the MT-CA and LE-CA algorithms and also proves to be substantially faster. Indeed, to the best of our knowledge, the LA automata solutions are the fastest and most accurate schemes available.

The problem of incorporating topology and routing considerations in the network design remains open. Also, the problem of relaxing the Kleinrock assumption is still a challenge.

REFERENCES

[1] Oommen, B. J. and Roberts, T. D., "A Fast and Efficient Solution to the Capacity Assignment Problem using Discretized Learning Automata, *Proceedings of IEA/AIE-98, the Eleventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems,* Benicassim, Spain, June 1998, Vol. II, pp. 56-65.

[2] Maruyama, K., and Tang, D. T., "Discrete Link Capacity and Priority Assignments in Communication Networks", *IBM J. Res. Develop*., May 1977, pp. 254-263.

[3] Levi, A., and Ersoy, C., "Discrete Link Capacity Assignment in Prioritized Computer Networks: Two Approaches", *Proceedings of the Ninth International Symposium on Computer and Information Services*, November 7- 9, 1994, Antalya, Turkey, pp. 408-415.

[4] Oommen, B. J. and Roberts, T. D., "Continuous Learning Automata Solutions to the Capacity Assignment Problem". Unabridged version of this paper. Submitted for Publication. Also available as a technical report from the School of Computer Science, Carleton University, Ottawa ; Canada : K1S 5B6.

[5] Roberts, T. D., *Learning Automata Solutions to the Capacity Assignment*

Problem, M.C.S. Thesis, School of Computer Science, Carleton University, Ottawa, Canada : K1S 5B6.

[6] Bertsekas, D. and Gallager, R., *Data Networks* Second Edition, Prentice-Hall, New Jersey, 1992.

[7] Kleinrock, L., *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill Book Co., Inc., New York, 1964.

[8] Narendra, K. S., and Thathachar, M. A. L., *Learning Automata*, Prentice-Hall, 1989.

[9] Lakshmivarahan, S., *Learning Algorithms Theory and Applications*, Springer-Verlag, New York, 1981.

[10] Oommen, B. J., and Ma, D. C. Y., "Deterministic Learning Automata Solutions to the Equi-Partitioning Problem", *IEEE Trans. Comput*., Vol. 37, pp 2-14, Jan. 1988.

[11] Ellis, Robert L., *Designing Data Networks*, Prentice Hall, New Jersey, 1986, pp. 99-114.

[12] Etheridge, D., Simon, E., *Information Networks Planning and Design*, Prentice Hall, New Jersey, 1992, pp. 263-272.

[13] Gerla, M., and Kleinrock, L., *On the Topological Design of Distributed Computer* Networks, IEEE Trans. on Comm., Vol. 25 No. 1, 1977, pp. 48-60.