A Fast and Efficient Solution to the Capacity Assignment Problem Using Discretized Learning Automata *

B. John Oommen and T. Dale Roberts
School of Computer Science
Carleton University
Ottawa ; Canada : K1S 5B6.
e-mail address : oommen@scs.carleton.ca.

Abstract. The Capacity Assignment (CA) problem 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. Most approaches consider a single class of packets flowing through the network, but in reality, different classes of packets with different packet lengths and priorities are transmitted over the networks. In this paper we assume that the traffic consists of different classes of packets with different average packet lengths and priorities. Marayuma and Tang [7] proposed a single algorithm composed of several elementary heuristic procedures. Levi and Ersoy [6] introduced a simulated annealing approach which produced substantially better results. A new method that uses continuous learning automata was introduced in a previous paper [12]. In this paper we introduce a new method which uses discretized learning automata to solve the problem. Indeed, to the best of our knowledge, this is the fastest and most accurate scheme currently available.

I. Introduction

I.1 Data Networks And Design Considerations

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 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 packet is a sequence of binary data elements of restricted length, with addressing information sufficient to identify the sending/receiving computers and an error correcting code.

There are several tradeoffs that must be considered when designing a network system. Some of these are difficult to quantify since they are the criteria used to decide whether the overall network design is satisfactory or not. 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 list of the pertinent factors, which involve cost and performance, can be found in [2,12,13].

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,

* Supported in part by the Natural Sciences and Engineering Research Council of Canada.

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 the 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. Lower priority packets are 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 main contribution of this paper is to present a *discretized* Learning Automata (LA) solution to the CA problem. Apart from this 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. This will be clarified in Section IV.

I.2 Assumptions and Delay Formulae

The network model used for all the solutions in the following sections have the following features [6] :

1. Standard Assumptions : We assume that the message arrival pattern is Poissonly distributed, and that 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 [1] is used to calculate the average link delay for each class of packet and the average network delay for each class.
6. Propagation and nodal processing delays are assumed to be zero.

Based on the standard network delay expressions [1,4,6,7], all the researchers in the field have used the following formulae for the network delay cost :

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

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

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

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, λ_{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 [12,13].

II. Previous Solutions

II.1 The Marayuma -Tang Solution

The Marayuma/Tang (MT-CA) solution to the Capacity Assignment (CA) problem [7] 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 [6,7,12,13].

First, there are two initial CA 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 in which the fundamental 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 in a situation 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 each packet's delay bound is satisfied.

(b) *Procedure DropFast*: This procedure is invoked in a situation 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 bound for each packet.

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

link capacity perturbations.

To allow the concatenation of the heuristics mentioned above the algorithm provides two interfaces, ResetHigh and ResetLow. ResetHigh is the interface used by DropFast and ResetLow is the interface used by AddFast. They are:

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

(b) *ResetLow*: This procedure decreases the capacity of each link to the 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. The pseudocode with the details of how the procedures are invoked is given in [6,7,12,13].

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) [6] which is based on the concept of simulated annealing. *Simulated Annealing* is an iterative, heuristic search paradigm, based on statistical physics, that has been used to solve a number of combinatorially explosive different problems. 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 called the *Acceptance Probability* and is lowered according to a distribution called the *Cooling Schedule* as the iterations continue.

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, $(C_1, C_2, ..., C_i, ..., C_m)$,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 [6,12,13].

III. Learning Automata

Learning Automata (LA) 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. There are a variety of applications that use automata [5,8] including parameter optimization, statistical decision making, telephone routing, pattern recognition, game playing, natural language processing, modeling biological learning systems, and object partitioning [10]. In this section we shall provide a basic introduction to LA and show how they can be used to solve the CA problem.

The learning loop involves two entities, the *Random Environ***ment** (RE) and a *Learning Automaton*. Learning is achieved by the automaton interacting with the

environment, and processing the responses it gets to various actions chosen. The intention is that the LA learns the optimal action offered by the environment. A complete study of the theory and applications of LA can be found in two excellent books by Lakshmivarahan [5] and by Narenda and Thathachar [8].

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 $\{\alpha_1, ..., \alpha_r\}$ 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 α_i , this action will serve as input to the RE. The RE will then respond to the input by either giving a *Reward*, signified by the value '0', or a *Penalty*, signified by the value '1', based on the *penalty probability* c_i associated with α_i . This response serves as the input to the automaton. 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 the time-varying transition and output matrices. They are, however, 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 α_i at time 'n', and satisfies $0 \leq p_i(n) \leq 1$, whose components sum to unity.

A VSSA can be formally defined as a quadruple (α, P, β, T), where α, P, β, are described above, and T is the updating scheme. It is a map from $P \times \beta$ 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.

Thathachar and Oommen (see [13]) first suggested that VSSA could be improved if the probability space could be rendered discrete. This would increase the rate of convergence and also eliminate the assumption that the random number generator could generate real numbers with arbitrary precision. This idea was implemented by restricting the probability of choosing an action to only a finite number of values from the interval [0, 1] with changes in probability made not continuously, but in a step-wise manner. By making the probability space discrete, a minimum step size is obtained and if the LA is close to an end state it forces the LA to this state with just a few more favorable responses. Therefore, once the optimal action has been determined, and the probability of selecting that action is close to unity, the discrete LA increases this probability directly rather than approach it asymptotically.

In a sense, discrete VSSA are a hybrid of fixed structure automata and VSSA, because, like the former, they consist of finite sets, but like the latter are characterized by a probability vector that evolves with time. Since we require an absorbing strategy we use the updating rules for the Discrete Linear Reward-Inaction (DL_{RI}) scheme. The details of this scheme can be found in [9,11].

IV. The Discrete Automata Solution to CA

The Discrete Automata Solution to CA (DASCA) algorithm is faster than either the MT-CA or LE-CA algorithms, and also produces superior cost results that

are generally closer to the optimal cost value. This solution is also faster than the CASCA algorithm presented earlier [12].

This solution to the CA problem utilizes the capacity assignment vector nomenclature previously discussed for the LE-CA and CASCA solutions where the capacities of the links are represented by a vector of the following 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 the CASCA solution [12] a probability vector was used to accomplish the learning process. This is unsuitable in the case of the discretized solution, DASCA, because we are now using integers and not real numbers. As a result, each of the link capacities of the capacity assignment vector will now have an associated *discrete convergence vector*, that has the following form.

$$(\iota_{ij}, \sigma_{ij}, \delta_{ij})$$

where, ι_{ij} is the discretized Increase parameter of link i with current capacity j, σ_{ij} is the discretized Stay parameter of link i with current capacity j, and δ_{ij} is the discretized Decrease parameter of link i with current capacity j.

The discrete convergence vector is related to the original probability vector in the following manner :

(i) $I_{ij} = \iota_{ij}$ / total-steps, (ii) $S_{ij} = \sigma_{ij}$ / total-steps, and (iii) $D_{ij} = \delta_{ij}$ / total-steps, where total-steps is the number of partitions of the probability space [0, 1].

The final solution vector will now be comprised of the capacities that exhibit σ, or stay, parameters that are closest to the converging value of the total number of steps, which is specified by the user in a practical implementation. The larger the number of steps, the higher the level of accuracy, which will result in a superior final capacity vector and associated network cost. The use of this convergence vector is similar to that used for the probability vector.

We now present the various initial settings for the convergence vector. If we assume that there are a total number of steps given by the variable total-steps, there are three possible settings for the initial convergence vector, given as Init1, Init2 and Init3 respectively.

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

$$\iota_{i0} = \text{total-steps}/2, \sigma_{i0} = \text{total-steps}/2, \delta_{i0} = 0,$$

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

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

$$\iota_{in} = 0, \sigma_{in} = \text{total-steps}/2, \delta_{in} = \text{total-steps}/2$$

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

Init 3: This is the scenario where the capacity of the link is at one of the interior capacities, called the interior state. This means that the capacity can be raised or lowered or maintained the same, and hence,

$$\iota_{ij} = \text{total-steps}/3, \sigma_{ij} = \text{total-steps}/3, \delta_{ij} = \text{total-steps}/3 \quad \text{for } 0 < j < n.$$

The next problem that arises is that of determining when, and how, to modify the convergence vector and also the probability values for a given link/capacity

combination. The procedure is identical to that of the CASCA [12] with the exception that the convergence vector is modified which then yields the associated probability vector. The convergence vector for each capacity assignment is modified in two cases to yield the updated solution and the new convergence vector. Each of these cases individually is explained below.

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

(a) if the capacity was increased we raise the δ_{ij} entry in the convergence vector,

(b) if the capacity stayed the same we raise the σ_{ij} entry in this vector, and

(c) if the capacity was decreased we raise the δ_{ij} entry in the convergence vector.

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 convergence vector is modified in the following manner :

(a) if the capacity was increased we raise the δ_{ij} entry in the convergence vector,

(b) if the capacity stayed the same we raise the σ_{ij} entry in this vector, and

(c) if the capacity was decreased we raise the σ_{ij} entry in the convergence vector.

It is important to remember that we are always trying to minimize the cost. We never attempt to reward an increase in cost by raising the increase entry, ι_{ij}, of the convergence vector at any point in time. Also, the degree by which the convergence vector is modified remains fixed for any resolution, total-steps since entries are always decreased by unity or increased by subtracting the total of the decreased entries from the total number of steps.

As stated previously, this paper not only represents 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, decrease 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 included in the [2,13].

V. Experimental Results

In order to evaluate the quality of potential solutions to the CA problem an experimental test bench was established. This set-up 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 are used in this paper are shown in Table 5.1.1 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.

Net. I.D.	Size	Ave. Link Len.	Packet Classes	Ave. Packet Rates	Packet Priority	Delay Bound	Packet Length
1	6	54.67	1 2 3	13 13.5 14.5	3 2 1	0.013146 0.051933 0.914357	160 560 400
2	8	58.75	1 2 3 4	14.375 15.625 15.125 15.5	3 2 1 4	0.013146 0.051933 0.914357 0.009845	160 560 400 322
3	12	58.08	1 2 3 4	15.417 15 15.5 17.083	3 2 1 4	0.053146 0.151933 0.914357 0.029845	160 560 400 322
4	12	58.08	1 2 3 4 5	15.417 17 15.5 17.083 17.33	3 2 1 4 5	0.053146 0.151933 0.914357 0.029845 0.000984	160 560 400 322 12
5	48	54.67	1 2 3	13 13.5 14.5	3 2 1	0.013146 0.051933 0.914357	160 560 400

Table 5.1.1 Characteristic values of the networks used in the test-bench.

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.

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 5.1.1 above. Each network, given by the network I.D. in column 1, carries a number of different packet classes, given in column 5. 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. For example, Network #1 has six links with an average link length of 54.67 Km. 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.

Figure 5.1.1 below shows a sample network similar to Network Type 1. Each of the six links, L1 - L6, can be assigned a single capacity value from Table 5.1.1 and

the average of the lengths will be specified by the quantity "average length" of Network Type 1. More details of the setup costs and variable costs are found in [13].

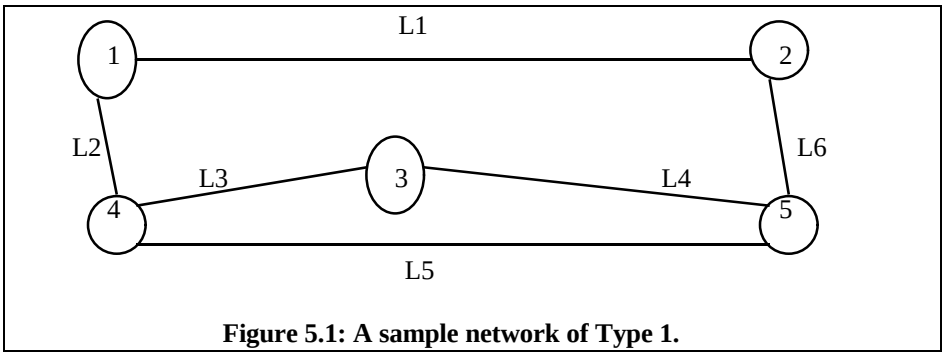

Figure 5.1: A sample network of Type 1.

In order to demonstrate that the new algorithm achieved a level of performance that surpassed the MT, LE and the continuous LA schemes, an extensive range of tests were performed [2,13]. 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 that 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 with 50 links. This is because the previous schemes, especially the MT algorithm, take a considerable time to produce results for larger networks.

Algorithm	**Category**	**Net 1**	**Net 2**	**Net 3**	**Net 4**	**Net 5**
MT-CA	Cost ($)	5735.04	12686.10	11669.30	53765.90	43341.90
	Time (sec)	0.22	0.77	1.86	2.08	67.45
LE-CA	Cost ($)	5735.04	7214.22	10295.70	45709.60	39838.40
	Time (sec)	0.22	1.21	1.10	1.93	4.01
CASCA	Cost ($)	4907.68	6937.90	9909.11	40348.90	37307.40
	Time (sec)	0.11	0.39	0.70	1.33	4.12
DASCA	Cost ($)	4907.68	6937.90	9909.11	40348.90	37307.40
	Time (sec)	0.10	0.33	0.51	1.32	2.51

Table 5.2.1 : Best results for all algorithms.

The results clearly demonstrate that the discrete scheme is faster than the non-automata and the CASCA scheme. Thus, for example, the best result obtained by the CASCA algorithm for Network #3 is a final cost of $9,909.11 which took 0.70 seconds. The DASCA algorithm produces the same final cost value of $9,909.11, but takes only 0.51 seconds. This improvement becomes more pronounced as the size of the network increases. For example, the best result obtained by the CASCA algorithm for Network #5 is a final cost of $37,307.4 which took 4.12 seconds. The DASCA algorithm produces the same final cost ($37,307.4), but takes only 2.51 seconds. The superiority of DASCA over MT-CA and LE-CA is even *more pronounced*.

It is clear from the results that as the total number of steps gets larger, the accuracy of each final cost value improves but the execution times and the number of iterations increase proportionally. This means that the algorithm can be adjusted to be optimized for speed (decrease the total number of steps), accuracy (increase the total number of steps) or some combination of the two that the user finds appropriate.

VI. Conclusions

In this paper we have discussed 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 traversing the network. Our fundamental contribution has been to design the Discrete Automata Solution to CA (DASCA) algorithm. This algorithm generally produces superior low cost capacity assignment when compared with the non-automata algorithms and also proves to be substantially faster. The DASCA algorithm also proves to be faster than the CASCA algorithm which was presented in a previous paper [12]. Indeed, to the best of our knowledge, this is the fastest and most accurate scheme currently available. The problem of incorporating topology and routing considerations and of relaxing the Kleinrock assumption in the design remains open.

References

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

[2] 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, 1997.

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

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

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

[6] 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.

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

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

[9] Oommen, B.J. and Lanctôt, J.K., "Discretized Pursuit Learning Automata", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-20, July/August 1990, pp. 431-438.

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

[11] Oommen, B.J., "Absorbing and Ergodic Discretized Two Action Learning Automata", *IEEE Transactions on Systems, Man and Cybernetics*, 1986, pp. 282-293.

[12] Oommen, B. J. and Roberts, T. D., "Continuous Learning Automata Solutions to the Capacity Assignment Problem". (Submitted for Publication).

[13] Oommen, B. J. and Roberts, T. D., "Discrete Learning Automata Solutions to the Capacity Assignment Problem". (Submitted for Publication).