CONTINUOUS LEARNING AUTOMATA SOLUTIONS TO THE CAPACITY ASSIGNMENT PROBLEM*

B. John Oommen[1] **and T. Dale Roberts**
School of Computer Science
Carleton University
Ottawa ; Canada : K1S 5B6.

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. We shall look at three different solutions to this problem.

Marayuma and Tang [MT77] proposed a single algorithm composed of several elementary heuristic procedures. Levi and Ersoy [LE94] introduced a simulated annealing approach that produced substantially better results. In this paper we introduce a new method which uses continuous learning automata to solve the problem. Our new schemes produce superior results when compared with either of the previous solutions and is, to our knowledge, currently the best known solution.

I. INTRODUCTION

I.1 DATA NETWORKS AND DESIGN CONSIDERATIONS

Data networks are divided into three main groups which are characterized by their size ; these groups 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, together with addressing information sufficient to identify the sending and receiving computers and an error detecting code. Typically, however, packets also contain an error correcting code.

---

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

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 we will examine in detail only the factors that are pertinent to the problem which we study, and will briefly state the others.

I.1.1 COST

Costs are measured in terms of three main considerations.

(i) **Line Costs**: These are the charges imposed by the carrier to connect the nodes of the network and the communications center(s) as required.

(ii) **Equipment Costs**: These are the cost of the devices used in the network to accommodate its users (for example, terminals), to transmit the packets, and to monitor performance.

(iii) **Software Costs**: These are the costs incurred in obtaining and maintaining the required software packages. It is important to note that the more complex the design of the network, the more intricate will be the software required to operate and control it.

I.1.2 PERFORMANCE

Performance is measured in terms of four main criteria. **Response Time** refers to the percentage of packets in an inquiry/response system to be handled by the system within a certain time period. This is sometimes a goal for the network's peak hour. **Throughput** specifies the portion of a line's capacity that must be carrying packets of value to the network's users. It is usually measured in Throughput Rate in Information Bits (TRIB). Some factors that improve throughput lead to an improvement in response time, for example, a TRIB of 2100 bps on a 2400 bps line. **Utilization** specifies the maximum portion of a line's capacity that can be used, counting both data and overhead. A major component of response time is the wait time that is directly dependent on a line's utilization. **Busy Rate** applies to systems that use primarily voice and data, and that use switched or measured services of common carriers. It specifies the maximum percentage of packets that will find the network's facilities busy and thus unable to serve them. Other factors that might influence performance include reliability and vulnerability, although these are generally considered to be of lesser importance.

Response time is the primary area of interest and there are many factors that can affect it.

(i) **Terminal Buffering** refers to the size of the buffer that the node has for accommodating a packet. If the buffer is adequate in size, only a single transmission will be required. If multiple transmissions are required the response time will be lengthened by the amount of time needed for the individual transmissions to be sent and acknowledged.

(ii) **Line Speed** can have a significant effect on response time. The effect must be measured by considering every individual leg of the packet's route through the network, and determining the minimum, average, and peak-load-time transmission rates in order to determine the overall impact. Possible alternative paths for the original message and acknowledgment considerations must also be examined whenever queuing or line problems cause rerouting.

(iii) **Quantity of Paths** is critical to response time. Routing selections must be done intelligently in order to determine whether it is feasible (or desirable) to send long transmissions over the same paths as shorter ones.

(iv) **Queuing at Transmission Nodes and Communication Centers** is also very important and can have a significant effect on response time. Since almost all transmission nodes are store-and-forward devices[2], the entire message must be received and validated before it can be transmitted to the next destination. Consequently, the possibility of malfunctioning equipment and/or queues building up at nodes must also be examined. As the queues become longer, the wait time increases, and the longer it will take for a message to be received and retransmitted by a node.

(v) **The Length of the Transmission** must also be considered because as the message lengths increase, the time of transmission also increases, regardless of line speed. This does not mean that messages should be shortened as this would be unacceptable in most systems. However, the designer should be aware that occasional large messages transmitted over a low capacity line will result in delays that would not normally occur if the line is primarily used by messages of short lengths.

(vi) **Processing Delay** refers to the actual work that must be done at a node in order to retransmit a message. This factor is sometimes ignored in the design phase only to be re-encountered when the network is in operation. The speed of processing is determined by several factors such as the speed of the input and output devices, the quality of the transmission media, the traffic through the node, and even the type of software. Every facility will have individual preferences in this regard, and each may require a careful and extensive planning effort to account for all the variables involved.

I.1.3 OTHER CRITERIA

The other factors that must be considered in the design process include ease of implementation, cohesion, maintenance, servicing and support, security, reliability, redundancy, robustness, flexibility, financing, and compatibility. These are all important factors but are not explained in detail because they are not crucial to the problems that we study in this paper.

I.2 THE CAPACITY ASSIGNMENT PROBLEM

In the process of designing computer networks the designer is confronted with a trade-off between costs and performance. In the previous section we catalogued the cost and performance parameters used in a general design process, 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. 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 only solutions (to our knowledge) to the CA problem have been developed by Marayuma/Tang[MT77] and Levi/Ersoy[LE94]. The Marayuma/Tang algorithm is a heuristic based solution that results in a high quality solution but proves to be quite slow for large networks. The Levi/Ersoy algorithm uses a Simulated Annealing approach, and yields better results than the Marayuma/Tang algorithm for both cost and performance criteria. However, the strengths of these results diminish for larger networks. In this paper, a new algorithm is proposed that uses Learning Automata to generate superior results (when compared with the previous methods) in terms of cost and performance. Apart from the fundamental contribution of the paper being a new technique to solve the CA problem, 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.

---

[2] The increasing popularity of the ATM and the Frame Relay schemes are making the store-and-forward strategies less and less universal. We are grateful to an anonymous referee who brought this to our attention.

As the results of this study are quite recent (and since the Levi/Ersoy paper is published only in the Proceedings of a conference [LE94]) we believe that a fairly detailed report on these algorithms is not out of place. Indeed, we hope that our paper will be used as a survey and a reference for all future researchers and for designers in the real world.

I.2.1 ASSUMPTIONS AND DELAY FORMULAE

The network model that will be used for all the solutions in the following sections have the following features [LE94].

1. Standard Assumptions
   (a) The message arrival pattern is Poissonly distributed, and
   (b) The message lengths are exponentially distributed.
2. There are multiple classes of packets. Each packet class has its own:
   (a) Average packet length measured in bits,
   (b) Maximum allowable delay measured in seconds, and
   (c) Unique priority level, where a lower priority takes precedence.
3. Link capacities are chosen from a finite set of predefined capacities. For each capacity there are:
   (a) Associated fixed setup cost, and
   (b) Variable cost/km.
4. Given as input to the system are the:
   (a) Flow on each link for each message class,
   (b) Average length of each packet class, 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 [BG92] is used to calculate the average link delay for each class of packet, and also 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 [LE94], [BG92], [Kl64], all the researchers in the field have used the following formulae for the delay cost incurred in the network:

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

$$U_r = \sum_{l \in V_r} \frac{\lambda_{jl} \cdot m_l}{C_j} \quad (1.2)$$

$$Z_k = \frac{\sum_j T_{jk} \cdot l_{jk}}{g_k}. \quad (1.3)$$

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,

λ_{jk} is the Average class k Packet Rate 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 the problem reduces to the integer programming problem which is to minimize the network cost,

$$D = \sum_i \sum_j d_j C_j x_{ij}$$

subject to,

$$Z_k \le B_k, \forall k ,$$

where, d_j is the discrete link cost-capacity function for link j,

B_k is the maximum allowable average delay for packet class k, and

x_{ij} is unity if link type i is chosen for link j, and zero otherwise.

Consequently,

$$\sum_i x_{ij} = 1, \forall j .$$

II. PREVIOUS SOLUTIONS

II.1 THE MARAYUMA -TANG SOLUTION

The Marayuam/Tang (MT-CA) solution to the Capacity Assignment (CA) problem [MT77] 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.

Some additional notation is required before the heuristic routines can be introduced. These are :

C_j is the capacity currently assigned to link j,

C_j^+ is the next higher capacity available to link j,

C_j^- is the next lower capacity available to link j,

D_j is the cost of link j when C_j is used,

D_j^+ is the cost of link j when C_j^+ is used,

D_j^- is the cost of link j when C_j^- is used,

T_{jk} is the average link delay for packet class k when C_j is used on link j,

T_{jk}^+ is the average link delay for packet class k when C_j^+ is used on link j,

T_{jk}^- is the average link delay for packet class k when C_j^- is used on link j,

Z_{jk} is the average delay for packet class k when C_j is used on link j,

Z_{jk}^+ is the average delay for packet class k when C_j^+ is used on link j, and finally,

Z_{jk}^- is the average delay for packet class k when C_j^- is used on link j.

The Marayuam/Tang (MT-CA) solution to the CA problem is based on the philosophy of a few procedures. We describe each of them below.

First, there are two initial capacity assignment heuristics, SetHigh and SetLow:

(a) **SetHigh**: In this procedure each link is assigned the maximum available capacity.
Observe that if there is a class for which the delay bound is smaller than the average delay then the problem is not feasible.

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

We now describe 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. This is done using some fundamental procedures explained in the next few subsections.

II.1.1 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.

The first step is to determine the class with the smallest difference between the average delay of the network and the delay bound of the packet. This identifies the packet class that benefits the most from an

increase in link capacity. Next, the link that gives the maximum performance improvement per unit cost for the packet class is found and *its* capacity is increased to the next higher one. This process continues until the delay requirements of each packet class are satisfied. The procedure is formally described below.

Procedure AddFast
Input: No parameters. State of the network is required as input.
Output: The modified state of the network in which the cost of network is increased.
BEGIN
Repeat

Find the packet class k which maximizes $\{Z_k / B_k\}$

Find the link j which carries k class packets and maximizes $\frac{\lambda_{jk}.\ (T_{jk} - T_{jk}^{+})}{D_j^{+} - D_j}$

Capacity(Link j) := C_j^{+}

Until ($Z_k \leq B_k$ is true for all k)
END Procedure AddFast

II.1.2 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 average delay bound for each packet class.

The first step is to determine the class with the maximum difference between the average delay of the network and the delay bound of the packet. Next, the link that gives the minimum performance degradation per unit cost for the packet class is found, and *its* capacity is decreased to the next lower one. This process continues as long as the delay constraint of any packet class is not violated. The pseudo-code of this procedure is given below.

Procedure DropFast
Input: No parameters. State of the network is required as input.
Output: The modified state of the network in which the cost of network has been decreased.
BEGIN
Repeat

Find the packet class k which minimizes $\{Z_k / B_k\}$

Find the link j which carries k class packets and minimizes $\frac{\lambda_{jk}.\ (T_{jk}^{-} - T_{jk})}{D_j - D_j^{-}}$

Capacity(Link j) := C_j^{-}

Until ($Z_k^{-} \leq B_k$ holds for all k)
END Procedure DropFast

II.1.3 PROCEDURE EXC

This procedure listed below attempts to improve the network cost by pairwise link capacity perturbations. For any two links i and j we reassign the capacities (using the notation described earlier) as

follows, $C_i := C_i^+$ and $C_j := C_j^-$. This reassignment must not violate any delay constraint and must satisfy $D_i + D_j > D_i^+ + D_j^-$. The formal code is omitted here.

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

(a) **ResetHigh**: In this procedure the capacity of each link is increased to the next higher one, C^+.

(b) **ResetLow**: In this procedure the capacity of each link is decreased to the next lower one, C^-.

II.1.4 THE MARAYUMA/TANG ALGORITHM

Marayuma/Tang determined that a solution given by one heuristic can often be improved by running other heuristics consecutively. The MT-CA algorithm, given below, is the best such composite algorithm. It yielded the best overall performance based on both the solution's accuracy and efficiency.

```
MT-CA //Due to Marayuma and Tang
Input:      The network characteristics and packet types.
Output:     The lowest cost network capacity assignment vector.
BEGIN-MAIN //Algorithm MT-CA
        SetHigh()
        previous-cost := calculate-network-cost()
        DropFast()
        current-cost := calculate-network-cost()
        While (current-cost < previous-cost) Do
                ResetHigh()
                DropFast()
                Exc()
                ResetLow()
                AddFast()
                DropFast()
                Exc()
                previous-cost := current-cost
                current-cost := calculate-network-cost()
        End-While
END-MAIN //Algorithm MT-CA
```

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) [LE94] 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 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, $C = (C_1, C_2, C_3, ..., C_i, ..., C_m)$ 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 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.

II.2.1 THE LEVI/ERSOY ANNEALING HEURISTIC

The simulated annealing heuristic used by Levi/Ersoy has three major steps listed below.

1. Generate a sequence of neighbor CA vectors, C_n, to the currently accepted solution , C_a.

2. Calculate the cost difference, ΔD, between C_a and C_n.

3. If the difference is negative (implying that the cost decreases) C_n is accepted unconditionally. If the difference is positive (implying that the cost increases) C_n is accepted with a certain probability.

This probability is given by the expression $\exp(-\Delta D/t_k)$, where t_k is the **Control Parameter** and is gradually decreased as the algorithm proceeds according to the cooling schedule. Note that even if the cost increases there is a chance that C_n will be accepted. This allows the algorithm to search for a global minimum, because if such configurations were always rejected only a portion of the search space would be scanned and only convergence to a local minimum could be guaranteed.

In the previous subsection we discussed how neighbor solutions are created. We now examine the criteria for the cooling schedule. There are four basic considerations listed below.

(i) The first parameter considered is t_0, the initial value of the control parameter t_k.

The initial value of the control parameter, t_0, is determined so that virtually all feasible neighbors of the initial capacity assignment vector, C_i, are acceptable. This means that t_0 is chosen so that $P_0 = \exp(-\Delta D_{mean}/t_0) \cong 1$, for all possible feasible neighbors. ΔD_{mean} is the average increase in cost and is calculated by looking at W random neighbors of C_i. Therefore t_0 becomes $\Delta D_{mean}/\ln(1/P_0)$. In our implementation, the value of P_0 is set to 0.99. W is set to 30 if the number of links is less than 16, otherwise it is assigned a value which is twice the number of links.

(ii) The second parameter considered is the number of iterations for each value of the control parameter t_k, which is equivalently the criterion to change t_k. In the current implementation the control parameter is

updated after five neighbors have been accepted or the number of iterations for that value of t_k is equal to twice the number of links.

(iii) The rule, or formula, used to update the control parameter t_k.

The initial value of k is unity. At each change of t_k, the value of k is incremented by unity and the following formula is applied.

$$t_k = t_0 * e^{-\alpha k}$$

where, α is a small positive number. In this implementation it is set to 0.5.

(iv) The stopping criteria for the algorithm.

The algorithm terminates if the cost remains the same for a few consecutive values of t_k. In our implementation we required that the cost should remain the same for three consecutive values of t_k.

The formal algorithm is quite straightforward and is omitted here in the interest of brevity.

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. There are a variety of applications that use automata [NT89, La81] including parameter optimization, statistical decision making, telephone routing, pattern recognition, game playing, natural language processing, modeling biological learning systems, and object partitioning [OM88]. In this section we shall provide a basic introduction to the topic and show how these principles can be used to formulate a solution to the CA problem.

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. A complete study of this subject can be found in the book '*Learning Automata: An Introduction*' by Narenda and Thathachar [NT89] and '*Learning Automata'* by Lakshmivarahan [La81] which contains a detailed analysis and examples of the types and applications of learning automata.

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.

This interaction between the two is shown diagramatically below.

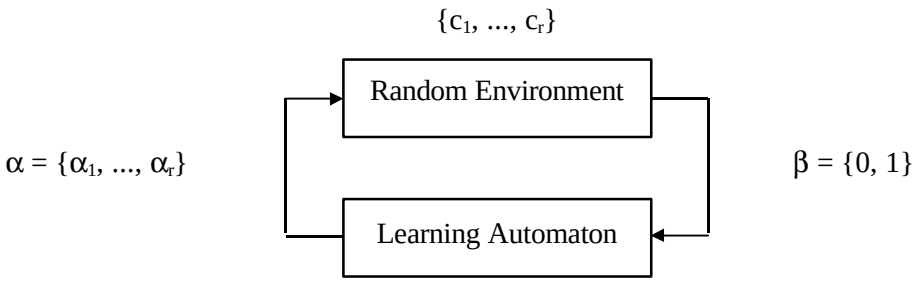

Figure 3.1: The Automaton - Environment Feedback Loop

Based on the structure of the transition and output matrices LA can be classified into the following two categories, mainly (i) Fixed Structure Stochastic Automata (FSSA), and (ii) Variable Structure Stochastic Automata (VSSA). FSSA exhibit transition and output matrices that are time invariant as opposed to VSSA which possess exhibit transition and output matrices that change with time. We shall work with the latter class of automata.

Although the VSSA can be described in terms of the transition and output matrices 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 defined as:

$$[p_1(n), ..., p_r(n)]^T$$

where, $p_i(n)$ is the probability of choosing action α_i at time 'n',

$$0 \le p_i(n) \le 1, \text{ and } \sum_{i=1}^{r} p_i(n) = 1.$$

Since the VSSA chooses an action based on a distribution specified by its action probability vector it is implemented using a random number generator. Since a random number generator and the action probability vector are used to decide which action is to be chosen, the VSSA has the added flexibility of choosing two different actions at two consecutive time instants. Also, action probability updating schemes are mainly divided into two mutually exclusive classes.

(a) A scheme with no absorbing barrier, or an **ergodic scheme**, has a distribution of the limiting action probabilities that is independent of the initial distribution of values. This type of scheme is

primarily used if the RE is non-stationary as the LA will not get locked into any of the given actions. An example of this type of scheme is the Linear Reward-Penalty (L_{RP}) scheme.

(b) A scheme which has an absorbing barrier, or an **absorbing scheme**, has values of limiting action probabilities that depend on the initial distribution of action probabilities. This type of scheme is primarily used if the RE is stationary. An example of this type of scheme is the Linear Reward-Inaction (L_{RI}) scheme.

A VSSA can be formally defined as a quadruple (α, P, β, T), where,

(i) $\alpha = \{\alpha_1, ..., \alpha_r\}$ is the set of r actions offered by the RE that the LA must choose from.

(ii) $P = [p_1(n), ..., p_r(n)]$ is the action probability vector where p_i represents the probability of choosing action α_i at the n^{th} time instant.

(iii) $\beta = \{0, 1\}$ is the set of inputs from the RE where '0' represents a reward and '1' a penalty.

(iv) T is the updating scheme. This 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.

Since we require an absorbing strategy, the updating rule we shall use is for the Linear Reward-Inaction (L_{RI}) scheme explained below.

Consider a VSSA that has r possible actions, $\{\alpha_1, ..., \alpha_r\}$ with penalty probabilities $\{c_1, .., c_r\}$. If these actions are equally likely at first, then the initial probability vector is $[1/r, ..., 1/r]^T$. The updating rules for the L_{RI} scheme are as follows.

$p_i(n+1) = 1 - \sum_{j \neq i} \lambda_r p_j(n)$ if α_i is chosen and $\beta=0$

$p_j(n+1) = \lambda_r p_j(n)$ if α_i is chosen and $\beta=0$

$p_j(n+1) = p_j(n)$ if α_i , α_j chosen, and $\beta=1$,

where $\lambda_r (0 < \lambda_r < 1)$ is the parameter of the scheme. Typically, λ_r is chosen to be close to unity.

Notice that only rewards are processed in this scheme. Therefore, if α_i is chosen and it receives a reward the probability of choosing this action on the next iteration, $p_i(n+1)$, must be increased. This is accomplished in two steps. First, the probabilities of choosing any other action α_j, for all $j \neq i$, on the next iteration are reduced by setting $p_j(n+1)$ to $\lambda_r p_j(n)$ for all $j \neq i$. Next, the probability of choosing α_i on the next iteration, $p_i(n+1)$, is increased by subtracting the sum of all $p_j(n+1)$ for $j \neq i$, from unity.

It can be shown that as this process continues, a unit vector will eventually be reached since the chain is absorbing. Indeed, the L_{RI} scheme can be shown to be ε-optimal - it converges to the optimal action with a probability as close to unity as desired. It is thus ideal for solving the CA problem.

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 either 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. 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 this solution each of the possible link capacities of the capacity assignment vector has an associated **probability vector** of the following form:

(I_{ij}, S_{ij}, D_{ij})

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

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

D_{ij} is the probability that the current capacity j of the link i should be *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. The closer this value is to unity, the higher the level of accuracy, which will result in a superior final capacity vector and associated network cost.

We now present the various initial settings for the probability vector. Indeed, by the 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,

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

The next problem that arises is that of determining when, and how, to modify the probability values for a given link/capacity combination. We shall explain this in detail presently. 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 and lowering the capacities of the links using the L_{RI} learning 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. Thus, at every single instant, the algorithm examines a (possible completely) different capacity assignment vector. 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 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 S_{ij}, the Stay probability of the link.

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

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. The closer they are to unity, the slower the algorithms are.

As stated previously, this paper not only represents a new solution for the CA problem, but also introduces a new way of implementing LA. Traditionally, each LA would choose an action directly from the set of options available in the application domain. For example, in the case of the CA problem, traditional LA would choose the new capacity of the selected link from the total set of available capacities. 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.

We clarify the operation of the scheme by the following example.

Example 4.1

Let us assume that the capacity j of a link i has been lowered, such that j = j - 1, resulting in a lower cost feasible solution. This means that we take the following steps.

(i) Since the solution vector is feasible, we shall raise the decrease probability, D_{ij} , as:

$$I_{ij}(t+1) = \lambda_{R1} * I_{ij}(t)$$

$$S_{ij}(t+1) = \lambda_{R1} * S_{ij}(t)$$

$$D_{ij}(t+1) = 1 - (I_{ij}(t+1) + S_{ij}(t+1)).$$

(ii) Now, since the solution vector results in a lower network cost it implies that the stay probability, S_{ij}, should be raised. This is done as follows:

$$I_{ij}(t+1) = \lambda_{R2} * I_{ij}(t)$$

$$D_{ij}(t+1) = \lambda_{R2} * D_{ij}(t)$$

$$S_{ij}(t+1) = 1 - (I_{ij}(t+1) + D_{ij}(t+1)).$$

Similar steps are performed for each of the possible scenarios that are examined during the execution of the algorithm. This process continues until all the link capacities, C_i, have S_{ij} probability values that are close enough to unity.

If the test for feasibility fails then the link capacities are reset to their last lowest cost values. This enables the algorithm to examine all possible configurations, even more costly configurations, while maintaining a base solution to which it can return to if there are no feasible, lower cost solutions encountered in the search. The algorithm is stated formally below.

THE CASCA ALGORITHM

Input: (i) The network characteristics and packet types.
(ii) num-iterations - the total number of iterations.
(iii) required-accuracy - the value to which the probability must reach before we accept it as being converged.
(iv) Parameters λ_{R1}, λ_{R2}.

Output: The lowest cost network capacity assignment vector.

Method

START-MAIN ***//Algorithm CASCA***
For (i=1 to maxlinks)
    For (j=1 to maxcaps)
        If ($link_i$ = left-boundary-state) **Then** I_{ij} = 1/2, S_{ij} = 1/2, D_{ij} = 0 **End-If**
        If ($link_i$ = right-boundary-state) **Then** I_{ij} = 0, S_{ij} = 1/2, D_{ij} = 1/2 **End-If**
        If ($link_i$ = internal-state) **Then** I_{ij} = 1/3, S_{ij} = 1/3, D_{ij} = 1/3 **End-If**
    End-For
End-For
Repeat For (i=1 to maxlinks) C_i = RAND(1, maxcap) **End-For**
Until (network is feasible)
current-cost = calculate-network-cost()
For (i=1 to maxlinks) best-C_i = C_i **End-For**
best-cost = current-cost
While (count < num-iterations) AND (accuracy-level(all links) < required-accuracy)
    For (i=1 to maxlinks)
        $Action_i$ = RAND($Increase_{ij}$, $Stay_{ij}$, $Decrease_{ij}$)
        If ($Action_i$ = $Increase_{ij}$) **Then** $C_i = C_i^+$
        Else If ($Action_i$ = $Decrease_{ij}$) **Then** $C_i = C_i^-$ **End-if**
        End-if
    End-For
    current-cost = calculate-network-cost()
    For (i=1 to maxlinks)
        j = C_i
        If (network is feasible)
        Then If ($Action_i$ = $Increase_{ij}$) **Then** Raise(D_{ij}, λ_{R1}) **End-If**
            If ($Action_i$ = $Stay_{ij}$) **Then** Raise(S_{ij}, λ_{R1}) **End-If**
            If ($Action_i$ = $Decrease_{ij}$) **Then** Raise(D_{ij}, λ_{R1}) **End-If**
        Else Reset all links to best-cost capacities.
        End-If
        If (network is feasible) AND (current-cost < best-cost)
        Then If ($Action_i$ = $Increase_{ij}$) **Then** Raise(D_{ij}, λ_{R2}) **End-If**
            If ($Action_i$ = $Stay_{ij}$) **Then** Raise(S_{ij}, λ_{R2}) **End-If**
            If ($Action_i$ = $Decrease_{ij}$) **Then** Raise(S_{ij}, λ_{R2}) **End-If**
            For (i=1 to maxlinks) best-C_i = C_i **End-For**
            best-cost = current-cost
        End-If
    End-For
End-While
END-MAIN ***//Algorithm CASCA***

Procedure Raise

Input: (i) $Action_{ij}$: either - Increase, Stay, Decrease - performed on link i with current capacity j.

(ii) λ_R, which is the learning scheme modification parameter. λ_R is set to λ_{R1} or λ_{R2} depending on whether the new solution is feasible or both feasible and superior.

(iii) The current probability vector for link i with current capacity j, (I_{ij}, S_{ij}, D_{ij}).

Output:

The modified probability vector for link i with current capacity j, (I_{ij}, S_{ij}, D_{ij}).

Method

BEGIN

If (Action = Increase)

Then $D_{ij} = \lambda_R * D_{ij}$

$S_{ij} = \lambda_R * S_{ij}$

$I_{ij} = 1 - (D_{ij} + S_{ij})$

Else If (Action = Stay)

Then $I_{ij} = \lambda_R * I_{ij}$

$D_{ij} = \lambda_R * D_{ij}$

$S_{ij} = 1 - (I_{ij} + D_{ij})$

End-If

Else If (Action = Decrease)

Then $I_{ij} = \lambda_R * I_{ij}$

$S_{ij} = \lambda_R * S_{ij}$

$D_{ij} = 1 - (I_{ij} + S_{ij})$

End-If

End-If

END Procedure Raise

V. EXPERIMENTAL RESULTS

V.1 EXPERIMENTAL TEST BENCH

In order to evaluate the quality of potential solutions to the CA problem an experimental test bench 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 for the CA problem consists of two main components which are described below.

First, the potential link capacities and their associated cost factors are specified as inputs, and in our case the specific values we have used are shown in Table 5.1.1 below. The potential link capacities refers to a finite set of capacities, measured in bits per second, that are available for each link which are given in column 1. Each link capacity has two cost entries - the initial setup cost of establishing the link which is given in column 3, and a cost per kilometer of the length of the link which is given in column 2. 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 of these networks 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 5.1.2 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.

CAPACITY (bps)	COST PER KM ($)	FIXED COSTS ($)
9600	0.31	750.00
19200	1.31	850.00
50000	2.63	850.00
108000	2.63	2400.00
230000	13.10	7300.00
460000	37.50	8300.00

Table 5.1.1 Set of possible Link Capacities and Costs that are used for all networks.

NET I.D.	NUMBER OF LINKS	AVERAGE LINK LENGTH	PACKET CLASSES	AVERAGE PACKET RATES
1	6	54.67	1 2 3	13 13.5 14.5
2	8	58.75	1 2 3 4	14.375 15.625 15.125 15.5
3	12	58.08	1 2 3 4	15.417 15 15.5 17.083
4	12	58.08	1 2 3 4 5	15.417 17 15.5 17.083 17.33
5	48	54.67	1 2 3	13 13.5 14.5

Table 5.1.2 Characteristic values of the networks.

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 5.1.3 below. Each network, given by the network I.D. in column 1, carries a number of different packet classes, given in column 2. Each packet class has its own distinct priority, given in column 3, delay bound, given in column 4, and length, given in column 5. The delay bound indicates the maximum amount of time that the packet can stay undelivered in the network.

NET I.D.	PACKET CLASS	PACKET PRIORITY	DELAY BOUND	PACKET LENGTH
1	1	3	0.013146	160
	2	2	0.051933	560
	3	1	0.914357	400
2	1	3	0.013146	160
	2	2	0.051933	560
	3	1	0.914357	400
	4	4	0.009845	322
3	1	3	0.053146	160
	2	2	0.151933	560
	3	1	0.914357	400
	4	4	0.029845	322
4	1	3	0.053146	160
	2	2	0.151933	560
	3	1	0.914357	400
	4	4	0.029845	322
	5	5	0.000984	12
5	1	3	0.013146	160
	2	2	0.051933	560
	3	1	0.914357	400

Table 5.1.3 Characteristic values of packet classes for each 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 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 length will be specified by the

quantity “average length” of Network Type 1 taken from Table 5.1.2 . Such a network will carry traffic that exhibit characteristics similar to those shown in Table 5.1.2 and Table 5.1.3 for Network Type 1.

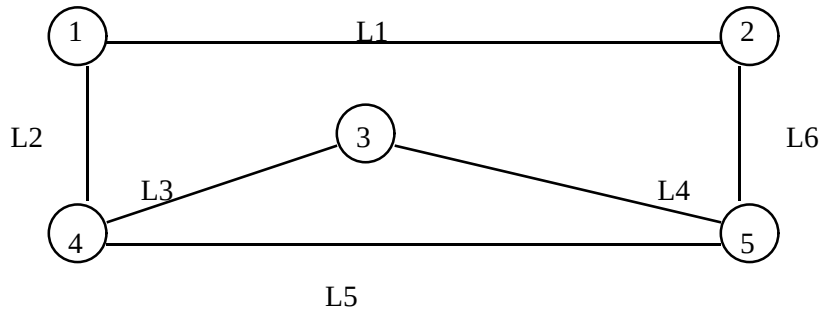

Figure 5.1: A sample network of Type 1.

V.2 TEST RESULTS

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 and a sample of these tests are shown in the tables below. Each table contains the results obtained when the algorithm was run five consecutive times for each network using each algorithm. The result of each test is measured in terms of two parameters, Cost and Time which are used for comparison with the MT and LE algorithms. In the interest of time we have only considered one large network in this series of tests, namely Network #5 which consists of 50 links, since the execution times of the previous solutions, especially the MT algorithm, take a considerable time to produce results for larger networks. A comparison with the MT and LE solutions concludes the section.

Test	Category	Net 1	Net 2	Net 3	Net 4	Net 5
1	Cost ($)	5735.04	12686.10	11669.30	53765.90	43341.90
	Time (sec)	0.22	0.77	1.86	2.14	78.05
2	Cost ($)	5735.04	12686.10	11669.30	53765.90	43341.90
	Time (sec)	0.22	0.77	1.93	2.14	83.04
3	Cost ($)	5735.04	12686.10	11669.30	53765.90	43341.90
	Time (sec)	0.22	0.77	1.92	2.14	67.45
4	Cost ($)	5735.04	12686.10	11669.30	53765.90	43341.90
	Time (sec)	0.22	0.77	1.92	2.08	67.45
5	Cost ($)	5735.04	12686.10	11669.30	53765.90	43341.90
	Time (sec)	0.22	0.77	2.03	2.14	67.45

Table 5.2.1 : Results for MT-CA Algorithm tests using the networks described above.

Test	Category	Net 1	Net 2	Net 3	Net 4	Net 5
1	Cost ($)	5735.04	12686.10	12978.80	65501.20	45538.40
	Time (sec)	0.22	0.49	0.93	1.37	5.55
2	Cost ($)	5735.04	15786.10	11386.60	49467.90	47360.60
	Time (sec)	0.22	0.38	0.83	1.59	5.98
3	Cost ($)	5735.04	7214.22	11416.80	49509.80	43400.40
	Time (sec)	0.22	0.55	1.37	1.71	5.39
4	Cost ($)	5735.04	12686.10	10289.70	47097.70	39919.60
	Time (sec)	0.22	0.49	1.10	1.21	5.38
5	Cost ($)	5735.04	12686.10	12990.00	45709.60	39838.40
	Time (sec)	0.22	0.38	0.99	1.93	4.01

Table 5.2.2 : Results for LE-CA Algorithm tests using the networks described above.

Test	Category	Net 1	Net 2	Net 3	Net 4	Net 5
1	Cost ($)	4907.68	8843.10	12961.60	49876.90	40406.90
	Time (sec)	0.05	0.33	0.66	1.16	1.65
2	Cost ($)	5020.00	10144.60	10006.60	49221.60	48688.90
	Time (sec)	0.11	0.72	0.66	1.31	1.32
3	Cost ($)	4919.68	7265.26	11744.70	48376.00	37809.00
	Time (sec)	0.05	0.22	0.60	1.37	3.13
4	Cost ($)	5161.20	6940.22	10779.20	45709.60	37771.00
	Time (sec)	0.06	0.33	0.66	1.38	2.14
5	Cost ($)	4919.68	7448.10	10194.70	43189.50	37723.80
	Time (sec)	0.11	0.33	0.83	1.05	2.37

Table 5.2.3 : Results for CASCA Algorithm tests using the networks described above. The parameters are λ_{R1} = 0.9, λ_{R2} = 0.9.

Test	Category	Net 1	Net 2	Net 3	Net 4	Net 5
1	Cost ($)	4907.68	7016.78	10458.80	41888.40	39303.40
	Time (sec)	0.22	0.87	1.37	2.58	2.08
	Iterations	160	422	429	467	194
2	Cost ($)	4907.68	7214.22	9939.27	44138.60	40866.20
	Time (sec)	0.22	0.55	1.16	2.80	2.14
	Iterations	163	244	389	510	203
3	Cost ($)	4907.68	7214.22	10725.90	40348.90	37862.00
	Time (sec)	0.11	0.66	1.81	3.40	3.46
	Iterations	138	300	629	637	348
4	Cost ($)	4907.68	7186.38	10479.70	45385.00	37629.40
	Time (sec)	0.28	0.55	1.10	3.08	1.82
	Iterations	214	285	348	565	176
5	Cost ($)	4907.68	6988.94	10707.30	41888.40	38957.80
	Time (sec)	0.28	0.72	1.10	2.80	1.87
	Iterations	190	325	384	520	180

Table 5.2.4 : Results for CASCA Algorithm tests using the networks described above. The parameters are $\lambda_{R1} = 0.95$, $\lambda_{R2} = 0.95$. In this case, unlike the other tables, we have also given the mean number of iterations required for the scheme to converge.

The results displayed in the tables demonstrate that the LA solution to the CA problem produces superior results when compared with both the MT and LE solutions. There is always a set of reward values that can be used to produce superior results, but in every case the cost value produced by the new algorithm falls in a range that is generally much closer to the optimal value. Indeed, the MT and LE solutions categorically produce values that generally lie much further from the optimal value. The new algorithm also has consistently lower execution times in most cases when compared to either of the previous solutions. For example, if we examine the results for the set of five tests where λ_{R1} is set to 0.9 and λ_{R2} is set to 0.9, the results clearly demonstrate the power of the scheme in both time and accuracy. Consider the tests for Network #4. The MT algorithm finds its best cost as $53,765.90 and takes 2.14 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 $43,189.50 and takes only 1.05 seconds which is superior to either of the previous best costs. For this series of tests, the best set of reward values are 0.9/0.9 which always yield near-optimal solutions and have lower execution times than either the MT or LE solutions.

As a point of interest, note that for the series of tests where λ_{R1} is set to 0.95 and λ_{R2} is set to 0.95 the CASCA algorithm always finds a superior cost value, even though the execution times are *sometimes* larger than those produced by the MT or LE solutions. In this case (see Table 5.2.4) apart from the accuracy and time results which can be used to compare the various algorithms, we have also given the mean number of iterations needed for the algorithm to converge. The table demonstrates the power of the CASCA algorithm: for example, in Test 5, when the algorithm examined a network of Type 5, the algorithm converged to its final value with an average of only 180 iterations, and the entire computation took only 1.87 seconds.

Hundreds of other experiments have been carried out which demonstrate identical properties of the CASCA algorithm. They are omitted here in the interest of brevity, but can be found in [DR97].

It is obvious that as the reward values get closer to unity the accuracy of each cost value improves but the execution time 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 their 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}, which is invoked when any feasible solution is found. This means that the algorithm can make

larger jumps once it has found a lower cost solution since the main goal is to find the lowest cost solution possible.

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.

The first reported solution to the CA problem is due to Marayuma and Tang (MT) and it uses a set of heuristic procedures to find the lowest cost link capacity configuration. This algorithm is limited by large execution times that increase as the network becomes larger and more complex. The second algorithm, due to Levi and Ersoy (LE), is based on the concept of simulated annealing. It produces a superior low cost capacity assignment, and has much faster execution time than the MT algorithm.

Our fundamental contribution has been to design the Continuous Automata Solution to CA (CASCA) algorithm, a LA solution to the problem. This algorithm generally produces superior low cost capacity assignment when compared with the above algorithms and also proves to be substantially faster. To the best of our knowledge, this is the fastest and most accurate scheme currently 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.

Acknowledgements : We are *extremely* grateful to the referees of the paper for their appreciative and critical remarks on the original manuscript. These comments definitely improved the paper's quality.

REFERENCES

[BG92] Bertsekas, D., Gallager, R., *Data Networks Second Edition*, Prentice-Hall, New Jersey, 1992.

[DR97] 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.

[El86] Ellis, Robert L., *Designing Data Networks*, Prentice Hall, New Jersey, 1986, pp99-114.

[ES92] Etheridge, D., Simon, E., *Information Networks Planning and Design*, Prentice Hall, New Jersey, 1992, pp 263-272.

[GK77] Gerla, M., Kleinrock, L., *On the Topological Design of Distributed Computer Networks*, IEEE Trans. on Comm., Vol. 25 No. 1, 1977, pp. 48-60.

[Kl64] Kleinrock, L., *Communication Nets: Stochastic Message Flow and Delay*, McGraw-Hill Book Co., Inc., New York, 1964.

[La81] Lakshmivarahan, S., *Learning Algorithms Theory and Applications*, Springer-Verlag, New York, 1981.

[LE94] Levi, A., 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.

[MT77] Maruyama, K., Tang, D. T., *Discrete Link Capacity and Priority Assignments in Communication Networks*, IBM J. Res. Develop., May 1977, pp. 254-263.

[NT89] Narendra, K. S., Thathachar, M. A. L., *Learning Automata*, Prentice-Hall, 1989.

[OM88] Oommen, B. J., Ma, D. C. Y., *Deterministic Learning Automata Solutions to the Equi-Partitioning Problem*, IEEE Trans. Comput., Vol. 37, pp 2-14, Jan. 1988.