INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6” x 9” black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700  800/521-0600
LEARNING AUTOMATA SOLUTIONS TO THE CAPACITY ASSIGNMENT PROBLEM

by

T. DALE ROBERTS, B.Sc.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Department of Computer Science

Carleton University
Ottawa, Ontario
April 1997

©copyright 1997, T. Dale Roberts
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni desextraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.
The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Learning Automata Solutions to the Capacity Assignment Problem

submitted by T. Dale Roberts, B.Sc.
in partial fulfillment of the requirements for
the degree of Master of Computer Science

Director, Department of Computer Science

Thesis Supervisor

CARLETON UNIVERSITY

APRIL 1997
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 thesis we assume that the traffic consists of different classes of packets with different average packet lengths and priorities. We shall look at four different solutions to this problem.

Marayuma and Tang proposed a single algorithm composed of several elementary heuristic procedures. Levi and Ersoy introduced a simulated annealing approach which produced substantially better results. In this thesis we introduce new methods which use continuous and discretized learning automata to solve the problem. Our new schemes produce superior results when compared with either of the previous solutions.

The CA problem can be modified to include assigning optimal priority levels to the packet classes. This is called the Capacity Assignment with Priority Assignment (CAPA) problem and we will look at two solutions to this problem. The first (and only known previous solution) is due to Marayuma and Tang in which they provide an extension to their solution to the CA problem. Their algorithm produces adequate results but is very slow. The second method is an extension of our previously mentioned learning automata algorithms and produces superior cost results, and much lower execution times, when compared with the Marayuma-Tang solution.
ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. John Oommen, my supervisor, for his inspiration, ideas and enthusiasm. He spent many hours patiently discussing the thesis with me and his thoughtful insights and advice proved to be extremely valuable. I would also like to thank Dr. Oommen for the extra time and effort he put in to help me finish my thesis on time.

Thanks to my Sally (my Mom) and Grandfather who both advised me to return to University and complete my Masters degree. They provided a great deal of support and encouragement over my two years at Carleton which I really appreciate and without which this thesis would not have been possible. This thesis is dedicated to them.

I would like to extend my thanks to the staff of the Computer Science office, especially Marlene Wilson who helped me with all the various administrative requirements. Finally, thanks to Jason Morrison for his comments and useful suggestions.
# Table of Contents

Chapter 1 - Communication Networks

1.1 Introduction ........................................................................................................... 1

1.2 Network Types ....................................................................................................... 2
  1.2.1 Local Area Networks ..................................................................................... 2
  1.2.2 Wide Area Networks and Metropolitan Area Networks ......................... 3
  1.2.3 Internetworks ................................................................................................. 4

1.3 Network Technologies .......................................................................................... 5
  1.3.1 Circuit Switched WANs .............................................................................. 5
  1.3.2 Data Packets .................................................................................................. 6
  1.3.3 Packet Switched WANs ............................................................................... 7
  1.3.4 Packet Switched LANs ................................................................................ 8
  1.3.5 Virtual Circuits and Datagrams .................................................................. 8

1.4 Network Architecture ........................................................................................ 10

1.5 Design Considerations ....................................................................................... 11
  1.5.1 Cost ............................................................................................................. 12
  1.5.2 Performance ................................................................................................. 12
  1.5.3 Other Criteria ............................................................................................... 14

1.6 The Capacity Assignment Problem .................................................................... 15

1.7 The Priority Assignment Problem ..................................................................... 16

1.8 Basic Search Techniques used in Artificial Intelligence ................................. 17

1.9 Contributions of the Thesis ............................................................................ 18

1.10 Conclusion ......................................................................................................... 19
Chapter 6 - The CA Problem with Priority Assignments

6.1 Introduction................................................................................................................120

6.2 Assumptions, Delay Formulae and Experimental Test Bench..................................121

6.3 Problem Complexity..................................................................................................122

6.4 The Marayuma-Tang Solution to the CAPA Problem...............................................124
   6.4.1 Priority Preference Among Classes-The H4 Heuristic......................................124
   6.4.2 Determination of the Number of Priority Classes..............................................126
   6.4.3 The MT-CAPA Algorithm...................................................................................128
   6.4.4 Experimental Results.........................................................................................129

6.5 The Continuous Automata Solution to the CAPA Problem.....................................131
   6.5.1 The CASCAPA Algorithm................................................................................135
   6.5.2 Experimental Results.......................................................................................137

6.6 The Discrete Automata Solution to the CAPA Problem..........................................149
   6.6.1 The DASCAPA Algorithm..............................................................................152
   6.7.2 Experimental Results.......................................................................................154

6.8 Conclusion................................................................................................................165

Chapter 7 - Conclusions

7.1 Summary..................................................................................................................166

7.2 Future Work.............................................................................................................170

References

References.....................................................................................................................171
LIST OF TABLES

Table 3.3.1 Set of possible Link Capacities and Costs that are used for all networks ........................................................................................................... 48

Table 3.3.2 Characteristic values of the networks ................................................................................................................................. 49

Table 3.3.3 Characteristic values of packet classes for each network ............................................................................................................. 50

Table 3.4.1 Results for MT-CA Algorithm tests using the networks described in Section 3.3 ............................................................................................................. 60

Table 3.5.1 Average results for LE-CA Algorithm tests using the networks described in Section 3.3 ............................................................................................................. 66

Table 3.5.2 Best Results for LE-CA Algorithm tests using the networks described in Section 3.3 ............................................................................................................. 66

Table 4.5.1 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are \( \lambda_{R_1} = 0.8, \lambda_{R_2} = 0.8 \) .................................................................................................................. 88

Table 4.5.2 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are \( \lambda_{R_1} = 0.9, \lambda_{R_2} = 0.8 \) .................................................................................................................. 90

Table 4.5.3 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are \( \lambda_{R_1} = 0.9, \lambda_{R_2} = 0.9 \) .................................................................................................................. 91

Table 4.5.4 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are \( \lambda_{R_1} = 0.95, \lambda_{R_2} = 0.9 \) .................................................................................................................. 92
Table 4.5.5 Results for the CASCA Solution to the CA Problem.
The networks that are used are described in Section 3.3 and the
parameters are $\lambda_{R1} = 0.95$, $\lambda_{R2} = 0.95$ ................................................................. 93

Table 4.5.6 Results for the CASCA Solution to the CA Problem.
The networks that are used are described in Section 3.3 and the
parameters are $\lambda_{R1} = 0.99$, $\lambda_{R2} = 0.99$ .................................................................. 94

Table 4.5.7 Best Results for the CASCA Solution to the CA
Problem using the networks described in Section 3.3 .................................................................... 95

Table 5.4.1 Results for the DASCA Solution to the CA Problem
with 20 steps. The networks used are described in Section 3.3 .......................................................... 112

Table 5.4.2 Results for the DASCA Solution to the CA Problem
with 40 steps. The networks used are described in Section 3.3 .......................................................... 113

Table 5.4.3 Results for the DASCA Solution to the CA Problem
with 60 steps. The networks used are described in Section 3.3 .......................................................... 114

Table 5.4.4 Results for the DASCA Solution to the CA Problem
with 80 steps. The networks used are described in Section 3.3 .......................................................... 115

Table 5.4.5 Results for the DASCA Solution to the CA Problem
with 100 steps. The networks used are described in Section 3.3 .......................................................... 116

Table 5.4.6 Results for the DASCA Solution to the CA Problem
with 500 steps. The networks used are described in Section 3.3 .......................................................... 117

Table 5.4.7 Best Results for the DASCA Solution to the CA
Problem using the networks described in Section 3.3 .................................................................... 118

Table 6.2.1 - Average Packet Path Lengths used for packets
traversing the networks described in Section 3.3 ............................................................................. 121
Table 6.4.4.1 Results for the MT-CAPA Algorithm when used to solve the CAPA problem. The networks used are described in Section 3.3 and Section 6.2.

Table 6.5.2.1 Results for CASCAPA Solution the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.8 \lambda_{R2} = 0.8 \lambda_{R3} = 0.8 \lambda_{R4} = 0.7$.

Table 6.5.2.2 Results for CASCAPA Solution the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.9 \lambda_{R2} = 0.8 \lambda_{R3} = 0.8 \lambda_{R4} = 0.7$.

Table 6.5.2.3 Results for CASCAPA Solution the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.9 \lambda_{R2} = 0.9 \lambda_{R3} = 0.8 \lambda_{R4} = 0.7$.

Table 6.5.2.4 Results for CASCAPA Solution the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95 \lambda_{R2} = 0.9 \lambda_{R3} = 0.8 \lambda_{R4} = 0.7$.

Table 6.5.2.5 Results for CASCAPA Solution the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95 \lambda_{R2} = 0.95 \lambda_{R3} = 0.8 \lambda_{R4} = 0.7$.

Table 6.5.2.6 Results for CASCAPA Solution for the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.9 \lambda_{R2} = 0.8 \lambda_{R3} = 0.9 \lambda_{R4} = 0.8$.

Table 6.5.2.7 Results for CASCAPA Solution for the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.9 \lambda_{R2} = 0.9 \lambda_{R3} = 0.9 \lambda_{R4} = 0.8$. 

xi
Table 6.5.2.8 Results for CASCAPA Solution for the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95$ $\lambda_{R2} = 0.9$ $\lambda_{R3} = 0.9$ $\lambda_{R4} = 0.8$.

Table 6.5.2.9 Results for CASCAPA Solution for the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95$ $\lambda_{R2} = 0.95$ $\lambda_{R3} = 0.9$ $\lambda_{R4} = 0.8$.

Table 6.6.2.1 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1 = 20$, num-steps$_2 = 20$.

Table 6.6.2.2 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1 = 40$, num-steps$_2 = 20$.

Table 6.6.2.3 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1 = 60$, num-steps$_2 = 20$.

Table 6.6.2.4 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1 = 80$, num-steps$_2 = 20$.

Table 6.6.2.5 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1 = 100$, num-steps$_2 = 20$.

Table 6.6.2.6 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1 = 40$, num-steps$_2 = 40$. 

xii
Table 6.6.2.7 Results for DASCAPA Solution to the CAPA Problem.
The networks used are described in Section 3.3 and Section 6.2 and
the parameters are \(num\text{-steps}_1 = 60, num\text{-steps}_2 = 40\).................................................................161

Table 6.6.2.8 Results for DASCAPA Solution to the CAPA Problem.
The networks used are described in Section 3.3 and Section 6.2 and
the parameters are \(num\text{-steps}_1 = 80, num\text{-steps}_2 = 40\).................................................................162

Table 6.6.2.9 Results for DASCAPA Solution to the CAPA Problem.
The networks used are described in Section 3.3 and Section 6.2 and
the parameters are \(num\text{-steps}_1 = 100, num\text{-steps}_2 = 40\).................................................................163

Table 7.1.1 Best Results for CA Problem for all the algorithms studied.
The networks that were used are described in Section 3.3.................................................................168

Table 7.1.2 Best Results for CAP problem for all the algorithms studied.
The networks that were used are described in Section 3.3 and Section 6.2............................................168
LIST OF FIGURES

Figure 3.3.1: A Network of Type 1 ................................................................. 51

Figure 4.2.1: The Automaton -Environment Feedback Loop .................................. 71

Figure 4.5.1 The Decrease in Cost with the Reward Parameters
for the CASCA Algorithm ....................................................................................... 87

Figure 4.5.2 The Increase in Execution Time
with the Reward Parameters ................................................................................... 88

Figure 5.4.1 The Decrease in Cost with the Reward Parameters
for the DASCA Algorithm ...................................................................................... 110

Figure 5.4.2 The Increase in Execution Time with the Reward Parameters
for the DASCA Algorithm ...................................................................................... 111
CHAPTER 1

COMMUNICATION NETWORKS

1.1 Introduction

In the fast-paced, high technology world of the 1990's, communication networks have been developed in order to provide a reliable, cost-effective, and secure method by which information can be transmitted around an office, city, country, or around the world. In their early development, these networks fell into many categories such as telephone, television, and computer networks. Each of these would have their own distinct primary function. For example, a telephone network primarily carried speech while a computer network would carry primarily data and text. In recent years the dividing lines between the various forms of networks have been almost completely erased, and most of today's communication network systems are capable of transmitting any form of data ranging from text to video and sound.

In this chapter a general overview of the topic will be provided defining the various types of computer networks along with brief sketches of their implementation and architectural features.
1.2 Network Types

1.2.1 Local Area Networks

Networks originally linked large mainframe computers that provided access to users through terminals. As the quality of PC applications surpasses that of software previously developed for centralized mainframe computers, the mainframes are being increasingly replaced by PCs. However, PCs lack the inter-connectivity that mainframe terminals offer. To regain this ability, the Local Area Network (LAN) was developed. There is now a very large installed base of LANs, encompassing virtually all working environments that contain more than one or two computers.

The LAN is a data communications system that provides high speed connections among PCs and their peripherals within a limited area, such as a single building or campus. A communication medium such as a fiber-optic or coaxial cable is generally used to connect the computers. No routing of messages is required, since the medium provides direct connections between all computers in the network.

At the heart of the LAN is the Network Server, which is a powerful PC whose sole purpose is to monitor network requests and process them. Additional servers on the network share resources such as printers or large databases. Users’ PCs connected to the network are called Work-Stations or Nodes.

The network server uses a Network Operating System to control the flow of information over the network. All PCs use disk operating systems to control the flow of information internal to themselves. The network operating system works one level higher in the processing scheme, routing and processing network requests to and from the individual PC.
Chapter 1: Communication Networks

The Topology of a LAN describes its physical layout, the connection of wires and cables, while also determining how the flow of data through the LAN should be handled. There are many types of LAN topologies including the Star, Bus, and Token-Ring topologies.

1.2.2 Wide Area Networks and Metropolitan Area Networks

Local area networks are restricted by their nature to data transmission within a small area. LANs can be expanded to Wide Area Networks (WANs) by adding Bridges, Routers, or Gateways. All three of these devices process data transmission between LANs. In addition, routers and gateways provide varying degrees of translation between LANs of different types.

WANs carry messages at lower speeds between computers that are separated by large distances. The computers that are interconnected by a WAN are called Hosts. They may be located in different cities, countries, or continents, and transmit using either Circuit-Switched or Packet-Switched telecommunications lines.

The Metropolitan Area Network (MAN) is now emerging, and is not as widely based as the other classes. It is based on the fiber-optic cabling of towns and cities for the transmission of video, voice and other data over distances of up to 50 kilometers. Message routing and other delays are much shorter than for WANs. MANs meet needs similar to those currently met by LANs while spanning greater distances.
1.2.3 Internetworks

An internetwork is a communication subsystem in which several networks are linked together to provide common data communication facilities that conceal the technologies and protocols of the individual component networks and the methods used for their interconnection.

Internetworks are implemented by linking component networks with dedicated packet routing computers called Routers or by general-purpose computers called Gateways, and by adding protocols that support the addressing and transmission of data to computers throughout the internetwork.

A Router is used to link two or more networks, which may be of different types. Routers pass packets from one network to another, using a strategy that enables messages to reach their correct destinations anywhere in the internetwork. To make this possible, routers must hold tables that describe part of the structure of the internetwork beyond the networks to which they are directly connected.

Gateways have a similar function to routers, but since they are normally used to link only two networks their routing tables are simpler. Thus the workload involved in acting as a gateway is small enough to permit one computer to act as a host for other work.

Many WANs and LANs can be combined to produce a single internetwork – a communication system that interconnects large collections of geographically dispersed computers. Internetworks can also be constructed on a single site such as an office building, factory, or university campus, to achieve interconnection between several LANs on the site. Such local internetworks are relatively simple to implement, especially when the interconnected networks are all based on a single LAN technology, and their end-to-end performance is similar
Chapter 1: Communication Networks

to that of the individual component networks. Bridges are used to connect networks of the same type.

The connected Internet is a particular instance of a wide area internetwork. It is a single world-wide collection of interconnected networks that shares a uniform scheme for addressing host computers and a suite of agreed protocols.

1.3 Network Technologies

1.3.1 Circuit Switched WANs

Circuit-switched lines are a temporary dedicated connection between two LANs. Using a circuit-switched line, the sending LAN dials the number of the receiving LAN. When the receiving LAN answers, the data transfer begins and the connection is broken once the transfer is completed. Data is transmitted over circuit-switched lines from the sender’s LAN to a switching station, and subsequently passed to a switching station closest to the receiving LAN.

The connection between the LAN and the circuit-switched telecommunications networks is processed by two devices. First, a node on the sending LAN originates a call. The call request is then received by a WAN device (either a bridge, a router, or a gateway) for processing. If the two LANs that make up the WAN use different network operating systems, a router or a gateway is used. If the two LANs use the same network operating system, a bridge can be used instead. After the request is processed by the WAN device, it is passed to a modem, which initiates the call.

When a circuit-switched WAN transfers data over digital phone lines (instead of the standard analog telephone lines) an additional piece of equipment is needed which is either the
Data Service Unit (DSU) or the Channel Service Unit (CSU). This device performs most of the same functions as a regular digital modem, except for the conversion (digital to analog) of the data to be transmitted. The type of digital service being employed determines whether the network should use a DSU or a CSU. A modem transmits data asynchronously, which means that the time interval varies between the transmission of each character. The beginning and ending of each character is identified by additional start and stop bits. In contrast, a DSU/CSU device transmits data synchronously. In other words, it uses timing signals to synchronize the sending and receiving units, and the characters are transmitted at regular intervals. This technique avoids the wastage of time involved in transmitting additional start and stop bits.

1.3.2 Data Packets

In most applications of computer networks the requirement is for 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. Packets of restricted lengths are used for the following reasons:

(i) to enable each computer in the network to allocate sufficient buffer storage to hold the largest possible incoming packet, and,

(ii) to avoid undue delays that would occur in waiting for communications channels to become free if long messages were transmitted in their entirety

The packet may also contain an error correcting code. The receiving PC computes its own error correcting code based on the actual data it receives and verifies it against the packet's
error correcting code. This technique allows the receiving PC to verify that it has received the data intact.

### 1.3.3 Packet-Switched WANs

Instead of using circuit-switched phone lines, most WANs use Packet-Switched telecommunications lines that work like "giant" networks. The host computers are connected to the network through packet switches, and messages or packets are routed to their destinations by packet switches until they reach the particular network to which the packet is addressed. Using packet-switching, several different LANs can be interconnected to the same WAN. Packets from different users share transmission lines in an economical and orderly manner that is designed to be transparent to the users of the network.

A WAN consists of a collection of communications channels linking special-purpose computers, known as Packet Switches or Packet Switching Exchanges (PSEs), first introduced in the ARPA network with the name Interface Message Processors (IMPs). A PSE is located at each node in the network and is dedicated to the task of data communication. Collectively, they send and receive packets of data through the network on behalf of other computers. For many end systems and terminal types that do not have this capability, data is processed in a device called a Packet Assembler/Disassembler (PAD) before being passed to the PSE.

The PSEs operate the network by forwarding packets from one PSE to another along a route from the sender to the recipient. They are responsible for defining the route taken by each packet. This mode of network operation is referred to as the Store-and-Forward communication, because every packet of data is stored temporarily by each PSE along its route before it is forwarded to another PSE.
Computers that use a network to send and receive data are called **Hosts**. Hosts are normally located close to a PSE and are connected directly to it. They pass packets of data to the PSEs for transmission through the network to other hosts, and in turn, receive packets from the PSEs that are addressed to them. In general, message transmission times are relatively long for WANs and may depend on the route taken by each message.

### 1.3.4 Packet-Switched LANs

LANs are structured as either buses or rings with dedicated communication circuits, normally on a single site and extending, at most, over a few kilometers. Messages are transmitted directly from the source computer to the destination computer without intermediate storage or processing. There are no PSEs in LANs. Instead, the host computers are collectively responsible for the management of traffic on the network using special-purpose hardware interfaces to transmit and receive data on the network circuits. The mode of operation is based on **Broadcast Communication** rather than the store-and-forward mode. This means that each packet is transmitted to all of the computers in the network and each computer is responsible for identifying and receiving the packets that are addressed to it.

### 1.3.5 Virtual Circuits and Datagrams

There are two basic approaches to packet switching. The first method used is to create a **Virtual Circuit** between the users. The latter appears to be a dedicated physical circuit between the source and destination, although the circuit is actually shared among multiple users. Initial connecting and disconnecting setup phases are required and the packets use relatively short
headers that indicate only the identification of the virtual circuit rather than a complete
destination address. Extra features such as error control, guaranteed delivery, and sequencing of
packets can also be provided which improve the quality of service provided by the virtual circuit
even when compared to the service provided by a dedicated physical circuit. The result is that the
delivery of packets in proper sequence, and with essentially no errors, is guaranteed, and
congestion control to minimize queuing is common. However, delays are more variable when
compared to those for dedicated circuits since several virtual circuits may compete for the same
facilities.

In contrast, for Datagram transmission, each packet is treated as a separate entity with
no prior route information, and so packets may follow different routes to the destination. In this
case, delivery is not guaranteed, and consequently, packets may arrive out of sequence or be lost
or duplicated. Error handling is the responsibility of the user as are any enhancements of the
basic service.

The choice of approach impacts protocols operating both between intermediate nodes
and end-to-end. Networks may use datagrams between intermediate nodes but provide additional
functions such as end-to-end error checking and sequence control. This allows intermediate
nodes to use relatively simple protocols at the expense of more complex end-to-end protocols, if
end-to-end virtual circuit service is required.

Since the communications subnetwork handles most communications functions when
virtual circuits are used, sophisticated users prefer to use the datagram service since it allows
more flexibility, allowing them to implement the features they require with their own software.
However, there are situations where a compromise between virtual circuits and datagrams are
best.
1.4 Network Architecture

When one PC needs to send data to another on the LAN, specific protocols are used. These are the parts of the network operating system that define processes for preparing data packets and moving them through the LAN. The network operating system also processes requests for data that is maintained on the server's hard drive. The network operating system consists of various operating layers, each playing a part in the overall task of data movement.

In the early 1970's the International Standards Organization (ISO) developed a standard model of network communications. The Open Systems Interconnection (OSI) model defined the specific function of each network operating system layer. The lowest layer of the network operating system is the Physical Layer. This layer handles the physical connection of the PC to the network, and the network's connection, if any, to other LANs. The Data Link Layer controls the movement of data over the LAN. This layer controls the contents of the LAN's data packets. This includes pieces of information such as the amount of data the PC could hold, the location of the sender's address, and the amount and structure of the error correction information. This layer of the operating system also determines how and when data packets can be placed on the network. In addition, the data link layer handles the procedures for error control and the methods for acknowledging safe receipt of data.

The Network Layer determines the best path of the data packet through the network, based on current network activity. In addition to assigning a relative priority for each particular data packet, the network layer also controls the transport of data to other LANs on the same WAN.

The Transport Layer of the network operating system is the layer which checks arriving data packets to determine whether it arrived safely. After the safe arrival of data, this layer
packages and proceeds to transmit a message of acknowledgment. If the data arrives damaged, the transport layer controls the details of retransmissions. Also, if the network fails, the transport layer retains the PC's data until it can be retransmitted.

The part of the network operating system that controls the establishment, and termination, of communications is the Session Layer. Generally, this layer also handles the transmission of security passwords.

The Presentation Layer is concerned with data conversion of arriving data packets and the proper formatting of data packets to be transmitted. If, for security reasons, the data should be encrypted before being placed on the network, the presentation layer invokes the necessary encryption modules.

The Application Layer of the network operating system is the layer that caters to the users’ needs. This layer understands the standards for shared-file access, printer sharing, e-mail, and database access. The network operating system connects to the PC's disk operating system through the application layer.

1.5 Design Considerations

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 that are 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. There are several components to this area so we will examine in detail only the factors that are pertinent to the problem, which we study, and will briefly state the others.
1.5.1 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.

1.5.2 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 systems, 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 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 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.

1.5.3 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, 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 thesis.
Chapter 1: Communication Networks

1.6 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 thesis 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 Problem (CA) 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 thesis, 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.
Chapter 1: Communication Networks

Two of the most popular solutions to the CA problem have been developed by Marayuma/Tang[MT76] and Levi/Ersoy[LE94]. The Marayuma/Tang algorithm is a heuristic based solution that results in a near-optimal 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. Furthermore, the magnitude of these results increase for larger and denser networks. In this thesis, a new algorithm is proposed that uses Learning Automata to generate an $\varepsilon$-optimal solution and gives superior results (when compared with the previous methods) in terms of cost and performance.

The problem will be discussed in greater detail in Chapter 3 where the Marayuma/Tang and Levi/Ersoy solutions are also examined.

1.7 The CA with Priority Assignments Problem

In the Marayuma/Tang solution to the CA problem, known as the MT-CA algorithm, it was shown that a substantial reduction in network cost could be achieved by introducing different classes of packets based on fixed delay requirements and priority of service. In certain cases, the packet classes are not assigned fixed priorities, and must be assigned priorities for passage through the network. A further network cost reduction can be achieved by mapping different classes of packets into different priority levels depending on various criteria such as the delay requirements of the packets. This is known as the Capacity Assignment with Priority Assignments (CAPA) Problem.

Unless an explicit priority assignment on packet classes is specified, the priority assignment becomes part of the network design problem. In general, there is no guaranteed method of determining an optimal priority assignment without exhaustively solving the capacity
assignment problem for all possible priority assignments. Since this approach is not generally feasible, a heuristic-based method was developed by Marayuma/Tang [MT77] based on the ordering of packet classes according to various parameters such as delay constraints, average packet length, and average packet rate. When this Priority Assignment algorithm is combined with the Capacity Assignment algorithm the resulting solution provides near-optimal capacity assignments and priority assignments.

Although it is possible to consider a different priority assignment on the packet classes for each node, such a system is not desirable and is therefore not considered in this thesis.

1.8 Basic Search Techniques used in Artificial Intelligence

Various strategies for effective search have emerged from the fields of mathematics and computer science. These range from totally uninformed search methods with no knowledge of the search domain (called weak methods) to well informed techniques in which knowledge of the domain is used effectively to speed up the search. The principle contribution of AI to the science of searching is the concept of knowledge based heuristics that are used in constraining and directing the search. Problems are generally formulated by developing a decision tree where the root is the initial state of the problem and branches which are composed of steps leading toward a potential solution located at the leaf. Heuristics are used to “prune” the search tree, reducing the search space, and finding the best path down the branches that leads to the optimal solution. Examples of search techniques include the well known Depth-First Search and Breadth-First Search algorithms while examples of heuristics include the Hill Climbing and Best-First algorithms. A more detailed discussion of this topic can be found in [Fi89].
Chapter 1: Communication Networks

It is easy to see that the problems studied in this thesis fall under the general category of optimization problems for which search techniques can be readily applied. Unfortunately, no such techniques have been applied to this specific problem, and we believe that various search and heuristic techniques (including genetic algorithms) can be used to solve it.

1.9 Contributions of the Thesis

In this thesis we present two new solutions to the CA problem. The first, the Continuous Automata Solution to CA (CASCA) algorithm, produces superior cost results and is faster than either of the previous solutions. The second, the Discrete Automata Solution to CA (DASCA) algorithm, is faster than the CASCA algorithm while maintaining the quality of the cost results.

We also present two new solutions to the CAPA problem. The Continuous Automata Solution to CAPA (CASCAPA) algorithm produces superior cost results and has substantially lower execution times when compared with the MT-CAPA algorithm. The Discrete Automata Solution to CAPA (DASCAPA) algorithm is faster than the CASCAPA algorithm and maintains the quality of the cost results.
Chapter 1: Communication Networks

1.10 Conclusion

In this chapter we have briefly introduced the reader to the area of computer networks. Section 1.1 provides an introduction as to why communication networks have become a common part of everyday life. Section 1.2 introduced the various types of networks -- LAN, MAN, WAN, and the methods by which data is transmitted through the network (Circuit Switching, Packet Switching) have been discussed in Section 1.3. A summary of various underlying network architectures has been included in Section 1.4.

The topic of Network Design has also been introduced in Section 1.5 where we have discussed the various factors that contribute to the cost and performance analysis phases. This provides a foundation for understanding the Capacity Assignment (CA) problem (described in Section 1.6), where we briefly list the two reported methods for solving it. The CA problem will be discussed in detail in Chapter 3. Section 1.7 introduces the Priority Assignment problem and alludes to a solution presented by Marayuma/Tang. This problem will be discussed in detail in Chapter 4. Section 1.8 discusses the basic search techniques used in AI and Section 1.9 gives a synopsis of the contributions that are provided by this thesis.
CHAPTER 2

QUEUING THEORY AND SYSTEMS

2.1 Introduction

Queues and queuing systems have been the subject of considerable research since the appearance of the first telephone systems and queuing theory was originally developed by the Danish mathematician A.K. Erlang for analyzing traffic within these systems. It was later discovered that models of the reliability of complex systems could well be formulated in terms of queues, and so queuing theory has been adapted, and extended, to become a tool in the design and development of computer networks.

This subject has given rise to an enormous amount of research and publications on the optimization problems for particular queuing models. The modeling of computer and data transmission systems has opened the way for the study of queues characterized by complex service disciplines, and has created the need to analyze interconnected systems. In the computer industry, queuing network models have resulted in software packages for the automatic solution of problems arising in the design of new computers, and in the evaluation and improvement of existing systems. The methods of queuing networks have always been a basic component of the study of telecommunication systems and new results on queuing networks have been used in studies of the performance of large communication networks.

In this thesis the performance of a given network will be measured primarily by the average delay required to deliver a packet from its source to a given destination. Queuing theory is the method by which the network is analyzed, via modeling, for such a delay. However, in the interest of decreasing the level of complexity in the computations, some simplification of the
basic assumptions are introduced. As a result it is sometimes impossible to obtain precise delay predictions using queuing models. However the results generated do provide adequate delay approximations and valuable insights into the performance of the system. The main focus of these delay approximations will be for packets in the network layer of the OSI model. This delay is given by the total delay on each link, which consists of four components explained below.

The processing delay is the time difference between the time the packet is received by a node and the time at which the packet is placed in the outgoing queue for re-transmission. The queuing delay is the time difference between the time when the packet is assigned to a queue and the time its begins to receive service. The transmission delay is the time difference between when the first and last bits are transmitted. The propagation delay is the time difference between the time the last bit is transmitted from a source (or intermediate) node, and the time the first bit is received at a destination (or intermediate) node.

This chapter will provide a brief introduction to queuing theory, particularly with regard to network delay models, and the analysis required for finding solutions to the DLA problem.

### 2.2 Queuing Theory

A queue is a waiting-line, and, in communications and computer engineering, queuing theory is the study of waiting-line and line-serving phenomena. Queues usually consist of requests waiting to be processed by the system and may be nested at several levels in large and/or busy systems.

In the case of computer networks the request is usually a message to be transmitted in the form of a packet. The request is usually routed to a service facility where there are one or more servers. If all the servers are busy when a request enters the system it joins the queue until
Chapter 2: Queuing Theory and Systems

A server becomes available. The simplest queuing system is the single-server system which can serve only one request at a time. A multi-server system has \( n \) identical servers that can serve up to \( n \) requests simultaneously. In an infinite-server system every arriving request is immediately served.

In some queuing systems, the queue capacity is assumed to be infinite. This means that every arriving request is allowed to wait until service can be provided. Other systems, called ‘loss systems’, have zero waiting line capacity. This means that if a request arrives when the service facility is fully utilized it is turned away. Still other systems have a positive, but not an infinite, capacity.

The queuing (or service) discipline is the rule for selecting the next request to be serviced. The most common discipline is the First In First Out (FIFO) or First Come First Served (FCFS). Another is Last In First Out (LIFO) or Last Come First Served (LCFS). Yet another type of discipline is Random Selection for Service (RSS) or Service in Random Order (SRO). Finally, there is the Priority Service (PRI) in which the requests are processed in the order of their assigned priorities.

In the store-and-forward model of computer networks a packet that is transmitted from a given source node must arrive completely at a given destination node before it is further re-transmitted. This means that any other arriving packets must be placed in a waiting queue, usually based on a FIFO system, until the node is free to receive and transmit them toward their destination. Generally speaking, we assume that packets arrive at random times and the average number of packets that arrive per unit time will be known as the arrival rate. The average number of packets that the system serves per unit time is called the service rate and the time taken between the initial arrival of the packet and the final re-transmission is known as the service time and is given by, \( \tau \), where
\[ \tau = \frac{L}{C} \quad (2.1) \]

in which, \( L \) is the packet length (bits), and \( C \) is the transmission capacity of the link, measured in bits per second.

Generally speaking in the analysis of queuing systems we would like to obtain expressions for the average number of packets in the system, which is the average number of packets either waiting or being serviced, along with the average packet delay, which is the average time a packet is waiting plus the service time.

### 2.3 Random Variables and Distributions

We now catalogue some of the random variables and distributions encountered in the study of computer networks. The material is not intended to be a complete study of the topic but rather a summary of the types of distributions that are used later to describe arrival/departure and service statistics at nodes of the network.

Consider the following random variables,

\[ X(t) = \text{number of customers (waiting or in service) in the system at time } t \]

In this situation the average number of customers in the system at time \( t \) is given by:

\[ N(t) = E[X(t)] \]

that is, the expected value of the random variable \( X(t) \) is given as,

\[ N(t) = \sum_{n=0}^{\infty} n \Pr[X(t) = n] = \sum_{n=0}^{\infty} np_n(t) \]

where the probability distribution is,

\[ p_n(t) = \Pr[X(t) = n]. \]
In typical systems $X(t)$ depends on the initial value $X(0)$ and the initial distribution $p_0(0), p_1(0), p_2(0), \ldots, p_n(0)$.

On reaching steady-state, since the random variables converge in distribution, the system can be described as,

$$\lim_{t \to \infty} X(t) = X$$

$$\lim_{t \to \infty} p_n(t) = p_n,$$

and hence,

$$\lim_{t \to \infty} \Pr[X(t) = n] = \Pr[X(t) = n].$$

If $N$ is the average number of customers in the system at steady state, we have,

$$N = \lim_{t \to \infty} N(t) = \lim_{t \to \infty} \sum_{n=0}^{\infty} n \Pr[X(t) = n] = \sum_{n=0}^{\infty} n \lim_{t \to \infty} p_n(t) = \sum_{n=0}^{\infty} np_n$$

We now consider the random variable $Y(k)$ defined as:

$Y(k) =$ time delay of the $k^{th}$ customer

If the probability distribution is given by $q(k) = \Pr[Y(k) = t]$ then the average delay of the $k^{th}$ customer can be given by:

$$T(k) = E[Y(k)] = \sum_{n=0}^{\infty} \Pr[Y(k) = t] = \sum_{n=0}^{\infty} t q(k).$$
Chapter 2: Queuing Theory and Systems

Since $Y(k)$ will also depend on the initial variable $Y(0)$ and the initial distribution, and will also be reaching steady state as $k \to \infty$,

$$T = \lim_{k \to \infty} T(k) = \sum_{i=0}^{\infty} iq_i$$

where,

$$q_i = \lim_{k \to \infty} q_i(k).$$

Under these assumptions Little’s formula $N = \lambda T$ holds where,

$$\lambda = \lim_{t \to \infty} \frac{E[Z(t)]}{t}$$

and, $Z(t)$ is the number of arrivals in the interval $[0, t]$.

Examples of random variables typically encountered in the study of networks are the Bernoulli, Binomial, Geometric and Poisson random variables.

In particular, a random variable $X$ with values $0, 1, 2, \ldots$ is called Poisson with parameter $\lambda > 0$ if,

$$\Pr[X = k] = \frac{\lambda^k}{k!} e^{-\lambda}.$$  

Data traffic in communication networks is often modeled as a Poisson process.

### 2.3.1 The Exponential Distribution

A continuous random variable $X$ has an exponential distribution with parameter $\lambda > 0$ if the probability density function is given by the following expression.

$$p(u) = \Pr[X = u] = \lambda e^{\lambda u} \quad \text{if } u \geq 0, \ 0 \text{ otherwise}$$
This implies that the cumulative density function can be given as,

\[ F(u) = \Pr[X \leq u] = \int_{-\infty}^{u} p(x)dx = 1 - e^{-\lambda u} \quad \text{if } u \geq 0, \quad 0 \text{ otherwise} \]

The main property of the distribution is that it is memoryless. This means that,

\[ \Pr[X > s + t | X > t] = \Pr[X > s] \quad \text{for all } s, \ t \geq 0 \]

This is equivalent to,

\[ \Pr[X > s + t] = \Pr[X > t] \cdot \Pr[X > s]. \]

### 2.3.2 The Poisson Process

A counting process \( \{N(t) : t \geq 0\} \) is a process that counts the total number of events that have occurred up to time \( t \). A counting process is said to have ‘independent increments’ if the number of events that occur in disjoint number intervals are independent.

The counting process possesses ‘stationary increments’ if the distribution of the number of events depends only on the length of the interval.

A Poisson process of rate \( \lambda > 0 \) is a counting process \( \{N(t) : t \geq 0\} \), such that.

1. \( N(0) = 0 \)

2. The process \( N(t) \) has independent increments.

3. The number of events in any interval of length \( t \) is Poisson distributed with mean \( \lambda t \).

In other words, for all \( s, t \geq 0 \),

\[ \Pr[N(t + s) - N(s) = n] = e^{-\lambda s} \frac{(\lambda t)^n}{n!} \quad \text{where } n = 0, 1, 2, ... \]
2.3.3 Interarrival Times

Consider a Poisson process and let $T_n$ be the time elapsed between the $(n-1)\text{th}$ and $n\text{th}$ events. This is known as the sequence of interarrival times. Since $N(t) = 0$ means that there are no events in the time interval $[0, t]$ then,

$$\Pr[T_1 > t] = \Pr[N(t) = 0] = e^{-\lambda t}.$$ 

Similarly,

$$\Pr[T_2 > t] = \int_0^t \Pr[T_2 > t \mid T_1 = s] ds = \int_0^t \Pr[T_2 > t \mid T_1 = s] \Pr[T_1 = s] ds.$$ 

From this result it can be shown that,

$$\Pr[T_2 > t \mid T_1 = s] = e^{-\lambda t}.$$ 

Similarly, it can be proved that for all $n$,

$$\Pr[T_n > t \mid T_1 = s] = e^{-\lambda t}.$$ 

This yields in the following theorem.

**Theorem of Interarrival Times**

The interarrival times of a Poisson distribution are independent, identically distributed, exponential random variable having mean $1/\lambda$.

Another useful result concerns the waiting time until the $n\text{th}$ event $W_n$, where,

$$W_n = T_1 + T_2 + \ldots + T_n.$$ 

Note that, $N(t) \geq n \Rightarrow W_n \leq t$.

**Theorem of Waiting Time**

The waiting time of a Poisson distribution satisfies,

$$\Pr[W_n \leq t] = \lambda e^{-\lambda t} \frac{(\lambda t)^{n-1}}{(n-1)!}.$$
2.3.4 Partitioning a Poisson Process

Let $N(t)$ be a Poisson process with mean $\lambda$ and at each time $t$ let event 1 occur with probability $p$ and event 2 with probability $1 - p$. Also assume that events 1 and 2 are independent.

- $N_1(t) =$ number of customers of type 1
- $N_2(t) =$ number of customers of type 2

This means that,

\[
\text{Mean} \{ N_1(t) \} = \lambda p \\
\text{Mean} \{ N_2(t) \} = \lambda (1 - p)
\]

Finally, one must also observe that the sum of independent Poisson processes with rates $\lambda_1, \lambda_2, \ldots, \lambda_n$ is Poisson with rate $\lambda_1 + \lambda_2 + \ldots + \lambda_n$.

2.4 Little’s Theorem

Fundamental to the analysis of queuing systems is a theorem proposed by Little. Suppose that an average system is observed for an indefinite amount of time, recording, in particular the following values.

- Let $N_t =$ Number of packets in the system at time $t$, and be given by,

- $N_t = \frac{1}{t} \int_0^t N(\tau) d\tau$

- $N_t$ tends to its steady-state $N$ as $t$ increases, and so, it is possible to state the following formula where $N$ is the steady-state value of $N(t)$.

\[
N = \lim_{t \to \infty} N_t
\]
We also define $\alpha(t)$ to be the number of packets that arrive in the time interval $[0, t]$ and $\lambda_t$ to be the packet arrival rate over time interval $[0, t]$, where

$$\lambda_t = \frac{\alpha(t)}{t}$$

The steady-state arrival rate as given in the formula below.

$$\lambda = \lim_{t \to \infty} \lambda_t$$

If $T_i$ is the time spent in the system by $i^{th}$ arriving packet the time average of the packet delay up to time $t$ can be seen to be,

$$T_i = \frac{\sum_{i=0}^{a(t)} T_i}{\alpha(t)}$$

Therefore the steady-state average customer delay, $T$, is,

$$T = \lim_{t \to \infty} T_i$$

These quantities are related by a simple formula that makes it possible to determine one, given the others and is known as **Little's Formula**, stated below.

$$N = \lambda T \quad (2.2)$$

This formula implies that in busy systems, where $N$ is large, there are resulting long packet waiting times. Conversely, if there are long packet wait times, the system will be correspondingly busy. Little's theorem provides the basis for all of the delay formulae that are defined in following sections.
2.5 Queuing System Models

There are several different factors that must be considered when trying to simulate a network queuing system. Historically (and probably primarily to render the analysis tractable) the most important factors are considered, and typically constitute a set of three quantities that could be used to characterize different categories of queuing systems. First, it is generally assumed that successive interarrival times and service times are statistically independent of each other, where the interarrival time is defined as the time between successive packet arrivals. The relevant quantities referred to above are defined as follows.

(i) The nature of the arrival process.

(ii) The nature of the probability distribution of the service times.

(iii) The number of servers.

In cases (i) and (ii) the representations are as follows.

a) M (Memoryless or Poisson) implies that the arrival times obey an exponential distribution of arrival times.

b) G (General) implies that the arrival times obey an arbitrary general distribution of arrival times.

c) D (Deterministic) implies that the interarrival times are deterministic.

By this nomenclature it is possible to define different categories of queuing systems.

Examples of such queuing systems are,

(i) M/M/1

\[ \text{queue} \rightarrow \text{one server} \]

(ii) M/G/1
(iii) M/M/m

(iv) M/M/∞

Thus, for example in an M/G/1 queue, the arrival times are exponential, the service times are arbitrary, and the number of servers is unity. We shall briefly follow the analysis of one of these systems.

2.5.1 The M/M/1 Queuing System

This system is made up of one server with a Poisson packet arrival process and an exponential distribution of service times.

For this system the following quantities are of prime interest.

\[ N = \text{Average number of packets in the system} \]

\[ T = \text{Average packet time in the system} \]

\[ N_Q = \text{Average number of packets waiting in queue} \]

\[ W = \text{Average packet waiting time in the queue} \]

These quantities are related in the following manner, due to Little’s theorem.

\[ N = \lambda T \]

\[ N_Q = \lambda W \]

(2.3)

Let \( p_n \) be the probability of \( n \) packets being in the system where \( n = 0, 1, 2, ... \). It is now possible to state the following.

\[ N = \sum_{n=0}^{\infty} np_n \]
Let $\rho$ be the utilization factor which is defined as the proportion of time that the server is busy.

$$\rho = \frac{\lambda}{\mu}$$

where,

$$\frac{1}{\mu}$$

is the average service time

Then $p_n = \rho^n (1 - \rho)$ where $n = 0, 1, 2, ...$

This implies that, $N = \frac{\rho}{1 - \rho}$

Substituting $\rho = \frac{\lambda}{\mu}$, $N = \frac{\lambda}{\mu - \lambda}$

By Little's theorem, $T = \frac{N}{\lambda}$, which yields,

$$T = \frac{\rho}{\lambda(1 - \rho)}$$

Using $\rho = \frac{\lambda}{\mu}$, yields,

$$T = \frac{1}{\mu - \lambda}$$

Thus, the average waiting time in queue, $W$, is given by the formula below.

$$W = \frac{1}{\mu - \lambda} - \frac{1}{\mu} = \frac{\rho}{\mu - \lambda}$$

Finally, by invoking Little's theorem again we get,

$$N_0 = \lambda W = \frac{\rho^2}{1 - \rho}$$

(2.5)
2.5.2 The M/G/1 Queuing System

This M/G/1 system has a single server, with a Poisson packet arrival process and a general distribution for service times.

Let $X_i$ be the service time of the $i$th arriving packet. We assume that the random variables \( \{X_1, X_2, \ldots\} \) are identically distributed, mutually independent, and independent of the interarrival times.

Let $\frac{1}{\mu}$ be the average service time and $\overline{X^2} = E\{X^2\}$ be the second moment of the service times.

The Pollaczek-Khinchin (P-K) formula states:

$$W = \frac{\lambda \overline{X^2}}{2(1 - \rho)}$$  \hspace{1cm} (2.6)

$W$ is the expected packet waiting time in queue and $\rho = \frac{\lambda}{\mu} = \lambda \overline{X}$.

Given the P-K formula, the total waiting time in queue and in service is,

$$T = \overline{X} + \frac{\lambda \overline{X^2}}{2(1 - \rho)}$$

Applying Little's formula to $T$ and $W$, we can see that,

$$N_Q = \frac{\lambda^2 \overline{X^2}}{2(1 - \rho)}$$

and,

$$N' = \rho + \frac{\lambda^2 \overline{X^2}}{2(1 - \rho)}$$
Chapter 2: Queuing Theory and Systems

As mentioned above, crucial to the analysis of the M/G/1 queue is a result due to Pollaczek and Kninchin. We shall prove that result here.

The proof of the P-K formula is based on the concept of the mean residual time. The residual time refers to the amount of time remaining in the time needed to provide service to a packet when another packet arrives and is placed in the queue. The mean residual time is the asymptotic expected residual time and is used in the analysis of M/G/1 queues and queues with priorities.

We define:

- $W_i$: Waiting time in queue of the $i^{th}$ arriving packet
- $R_i$: Residual service time
- $X_i$: Service time of the $i^{th}$ arriving packet
- $N_i$: Number of packets waiting in the queue by the $i^{th}$ arriving packet

Therefore, taking expectations and using the independence of the random variables $N_i$ and $X_{i-1}, ..., X_{i-N}$ and taking limits as $i \to \infty$, we obtain,

$$W = R + \frac{1}{\mu} N_Q$$

By Little's formula, $N_Q = \lambda W$, where, by substitution, $W = R + \rho W$

Substituting $\rho = \frac{\lambda}{\mu}$, we get

$$W = \frac{R}{1 - \rho}$$

(2.7)

During the period $[0, t]$, the time average of the residual service time $r(t)$ is

$$\frac{1}{t} \int_0^t r(t) \, dt = \frac{1}{t} \sum_{i=1}^{M(t)} \frac{X_i^2}{2} = \frac{1}{2} \frac{M(t) \sum_{i=1}^{M(t)} X_i^2}{M(t)}$$

(2.9)
Chapter 2: Queuing Theory and Systems

Taking the limit as \( t \to \infty \) and equating time and ensemble averages, we get,

\[
R = \frac{1}{2} \lambda X^2
\]  

(2.10)

where \( R \) is the mean residual service time, defined as, \( R = \lim_{t \to \infty} E\{R_t\} \).

The P-K formula follows by a direct substitution.

Observe that this derivation is based on two assumptions:

1. The existence of the steady-state averages \( W, R, \) and \( N_Q, \) and,

2. The equality of the long-term time and the corresponding ensemble averages.

2.6 Priority Queuing

In many queuing systems packets are divided into priority classes. This means that each class is assigned a unique priority level (say, in a range of 1 to \( n \)) with the lower numbered implying a higher priority. Thus, packets of priority class \( i \) are given preference over packets with priority \( j, \) provided that \( i < j. \) In a priority queuing system a separate queue is maintained for every packet priority class. When the server becomes free the packet waiting in the queue of the highest priority receives service.

A potential problem arises in these systems when a class \( i \) packet arrives while a class \( j \) packet is being served, where \( i < j. \) There are two types of control policies that manage these situations called the preemptive and nonpreemptive policies.

In a preemptive priority system, service is interrupted and the new arrival, with higher priority, begins service. At this point there are two further options that can be used once that packet has been served. If the system is a preemptive-resume priority system the lower priority
packet will continue service from the point at which service was interrupted once it had access to the server. However, if the system is a preemptive-repeat priority system the lower priority packet repeats its entire service from the beginning.

In a nonpreemptive priority system the newly arriving packet of higher priority is placed in the queue until the packet currently being served completes service before it is allowed access to the server. This type of system is also called a Head-Of-Line (HOL) system. Therefore, the nonpreemptive priority rule states that a packet undergoing service is allowed to finish the service routine without interruption even if a higher priority packet arrives.

We now proceed to derive an equation for the average delay of each priority class. As before we first define some elementary quantities, as follows.

$\lambda_k$: Arrival rate of packet class $k$

$\bar{X}_k$: First moment of service time for packet class $k$

$\bar{X}_k^2$: Second moment of service time for packet class $k$

$N^k_Q$: Average number of packets of priority $k$ in the queue

$W_k$: Average wait time in the queue for packets of priority $k$

$\rho_k$: System utilization for packets of priority $k$

We assume that the overall system utilization is less than unity, and hence, $\rho_1, \rho_2, ..., \rho_n < 1$. We intend to compute $R$, the mean residual service time.

For the highest priority class,

$$W_1 = R + \frac{1}{\mu} N^1_Q$$

By Little's theorem,

$$N^1_Q = \lambda_1 W_1$$
Substituting,

\[ W_1 = R + \rho_1 W_1 \]

where, for the highest priority class, we get:

\[ W_1 = \frac{R}{1 - \rho_1} \quad \text{(2.11)} \]

The same method is followed for the second priority class but we have to take into account the additional delay due to arriving packets of higher priority while waiting in the queue.

Therefore,

\[ W_2 = R + \frac{1}{\mu_1} N_Q^1 + \frac{1}{\mu_2} N_Q^2 + \frac{1}{\mu_2} \lambda_i W_1 \]

Again, by Little's theorem,

\[ W_2 = R + \rho_1 W_1 + \rho_2 W_2 + \rho_1 W_2, \]

hence, we get,

\[ W_2 = \frac{R + \rho_1 W_1}{1 - \rho_1 - \rho_2} \]

Using (2.11),

\[ W_2 = \frac{R}{(1 - \rho_1)(1 - \rho_1 - \rho_2)} \quad \text{(2.12)} \]

Arguing in a similar fashion, for any packet class priority \( k > 1 \), we can see that,

\[ W_k = \frac{R}{(1 - \rho_1 - \cdots - \rho_{k-1})(1 - \rho_1 - \cdots - \rho_k)} \quad \text{(2.13)} \]

This means that,

\[ T_k = \frac{1}{\mu_k} + W_k \quad \text{(2.14)} \]

As in the case of the M/G/1 queue we now derive an expression for \( R \).
Chapter 2: Queuing Theory and Systems

During the period \([0, t]\), the time average of the residual service time \(r(t)\) is

\[
\frac{1}{t} \int_0^t r(t) \, dt = \frac{1}{t} \sum_{i=1}^{M(t)} \frac{1}{2} X_i^2 = \frac{1}{2} \frac{M(t)}{t} \sum_{i=1}^{M(t)} \frac{X_i^2}{M(t)}
\]

Taking the limit as \(t \to \infty\) and equating time and ensemble averages, we get,

\[
R = \frac{1}{2} \lambda \overline{X^2}
\]

where \(R\) is the mean residual service time, defined as, \(R = \lim_{t\to\infty} E\{R_t\}\)

Hence, we have,

\[
R = \frac{1}{2} \sum_{i=1}^{n} \lambda_i \overline{X_i^2}
\]

Substituting \(R\) in (2.13), yields:

\[
W_k = \frac{\sum_{i=1}^{n} \lambda_i \overline{X_i^2}}{2(1 - \rho_1 - \ldots - \rho_k - \ldots)(1 - \rho_1 - \ldots - \rho_k)}
\]

(2.15)

2.7 The Kleinrock Independence Assumption

Much of the difficulty in solving computational issues in networks arises from the assignment of permanent lengths to packets since this assignment results in a dependency between interarrival times and the lengths of adjacent packets as they travel within the network. If this dependency could be eliminated (or its effects could be assumed to be negligible) the mathematics would be greatly simplified.

In the study of networks it is generally assumed that a packet's length is independent of its time of arrival/entry in the network from an external source. This is true for any external traffic entering the network; therefore, it is important to define what properties of the external
traffic contribute to this independence. An external packet source consists of a large number of
subscribers, each generating packets at a small rate. While it is true that individual packet
interarrival times and lengths are dependent, the collective interarrival times and lengths of
packets generated by the entire group of subscribers tend to exhibit an independence. This is due
to the fact that the length of an individual packet is completely independent of the arrival times
of packets from other individuals.

A similar situation arises for internal traffic in store-and-forward networks. In general,
there is more than one link delivering packets to any particular node, which is considered in
addition to the external source feeding the node. Also, in general, there is more than one link
transmitting packets out of the node, this is considered in addition to the ‘virtual’ link that
removes packets which have reached the node as their final destination.

In networks there is a high level of interaction between transmission queues whereby
packets departing one queue enters (one or more) other queue(s) after merging with portions of
other packet streams. This has the effect of complicating the nature of the arrival process of
subsequent queues along the path of the packet stream. The difficulty that arises is that the packet
interarrival times become strongly dependent upon the length of the packet once the packet
moves beyond the queue which serves as its entry junction into the network. This means that it is
impossible to carry out an accurate analysis analogous to the one done for the M/M/1 system.

This can be clarified by the following example. Consider two tandem transmission lines
with packet streams that have independent exponentially distributed packet lengths and
interarrival times at the first queue. This means that the first queue can be modeled as M/M/1.
However, strictly speaking the second queue cannot be an M/M/1 because the interarrival times
at the second queue are dependent upon the length of the packet transmitted from the first queue.
The reasoning for this is simple. The interarrival time of two packets at the second queue is
greater than, or equal to, the transmission time of the second packet at the first queue. As a result, longer packets will typically wait less time at the second queue than shorter packets, since their transmission at the first queue takes longer, providing the second queue adequate time to empty.

Consider a network consisting of several interlinked nodes with multiple possible routes from one node to another. Assume that there are several packet streams that each follow a unique path through the network. This type of model is well suited for virtual circuit networks where each packet stream follows a separate virtual circuit. For packet networks it may become necessary to use a more general model that allows the packet streams to be split into (two or more) components that follow distinct paths through the network. This means that once again there will be several packet streams, each having a unique source and destination. There may be several paths followed by the packets of a stream, but no packets travel in a loop.

It has been shown that even if the packet streams are Poisson with independent packet lengths at their point of entry, this property is lost after the first transmission line. To resolve this problem Kleinrock [KL75] suggested that merging several packet streams on a single transmission line has an effect equivalent to restoring the independence of interarrival times and packet lengths. For example, if a transmission line received a substantial portion of additional Poisson traffic, then the dependence of interarrival times and packet lengths is greatly reduced. Kleinrock concluded that it is appropriate to adopt an M/M/1 queuing model for each link regardless of the interaction of traffic on the link with traffic on other links. This is known as the Kleinrock Independence Assumption. This results in a reasonable approximation for network systems that are densely connected with moderate to heavy traffic loads, with Poisson arrival patterns at all entry points, and exponentially distributed packet lengths.
Chapter 2: Queuing Theory and Systems

The mathematical consequences of this assumption is a model which describes the behavior of the message delay in networks fairly accurately, and which simultaneously does not impact the utility of the numerical results.

Consider a network $N = \langle V, E \rangle$ consisting of Vertices and Edges. There are several packet streams each following a unique path $p$ which consists of a sequence of edges across the network. Let $x_s$ be the arrival rate of the packet streams. Then, the total packet arrival rate of link $(i, j)$, $\lambda_{ij}$, is given by:

$$\lambda_{ij} = \sum_{\text{all } s \text{ across link } i, j} x_s$$

This model is best suited for virtual circuit networks. For datagram networks a model which allows bifurcation of the traffic of packet streams should be used.

Let $f_s(i)$ denote the fraction of the packets of stream $s$ that go through link $(i, j)$ and $x_s$ denote the arrival rate of packet stream $s$. Then, the total arrival rate of link $(i, j)$ has the following expression.

$$\lambda_{ij} = \sum_{\text{all } s \text{ across link } i, j} f_s(i)x_s$$

Based on the M/M/1 queue model discussed above the average number of packets at link $(i, j)$ that are either in queue or being serviced is thus $N_{ij}$, where,

$$N_{ij} = \frac{\lambda_{ij}}{\mu_{ij} - \lambda_{ij}}$$

where $1/\mu_{ij}$ is the average transmission time of packets over link $(i, j)$.

Therefore the average number of packets over all queues is be given by:

$$N = \sum_{(i,j)} N_{ij} = \sum_{(i,j)} \frac{\lambda_{ij}}{\mu_{ij} - \lambda_{ij}}$$
By Little's theorem, the average delay per packet is

\[ T = \frac{N}{\text{Total Arrival Rate}}. \]

Substituting,

\[ T = \frac{\sum \frac{\lambda_j}{\mu_j - \lambda_i}}{\sum x_s} \]

(2.16)

2.8 Conclusion

Queuing theory and systems provide the framework by which the performance of a network can be measured. The primary quantity analyzed is the queuing, or delay, model which takes into account the arrival and service rates as well as the number of servers being utilized. The formulae used are essentially based on Little's Theorem which, when combined with a model, gives an approximation of the delay experienced by a packet that is transmitted across the network. In this chapter we have briefly stated the expressions derived for the various delay times that will be used by the various heuristic algorithms in the subsequent chapters to yield solutions to the problems studied in the thesis such as the Capacity Assignment problem.
CHAPTER 3
THE CAPACITY ASSIGNMENT PROBLEM

3.1 Introduction

In the process of network analysis and design the designer is confronted with many problems which are to be solved. These problems vary depending on the size and function of the network in question, but there are some types of design problems that are common to any network. One of the major problems that a designer encounters is one of determining how to resolve cost and performance issues. These two issues usually conflict with each other, and resolving them typically results in a performance-cost trade-off occurring in the design process. There are also other critical issues that must be examined which are more directly related to the cost and performance factors, such as the location of nodes, potential links, routing strategies, and link capacities. Also, in priority networks, different classes of packets have different lengths and priorities, which result in a considerable effect on performance issues. These performance issues are primarily measured in terms of delay constraints imposed by the traffic on the network while cost is measured in terms of setup and maintenance of the links.

Most network design problems may be formulated as linear/integer programming problems in which the cost (or delay) is minimized subject to various cost/delay constraints. In this process, special attention must also be paid to feasibility and consistency factors. The capacity of links is a vendor enforced constraint on networking and telecommunications, and results in a discrete and finite set of possible links with various capacity and cost values. One such problem, which is the focus of this chapter, is the Capacity Assignment (CA) problem. In this problem the designer is required to determine the least expensive assignment to the links from the potential link types that simultaneously satisfies the traffic requirements of the network.
in question. As a result, this becomes a more complex integer programming problem which is very difficult to solve. This problem is NP-complete and therefore heuristic based solutions are developed. Solutions proposed in the literature have been based on combinatorial optimization methods and heuristic search methods. One of the best known set of heuristics for the discrete link capacity and the priority assignment problems are due to Marayuma and Tang [MT76]. Their solution is presented in Section 3.3 of this chapter and the results obtained provide the comparison basis for all subsequent methods that are discussed. Another solution, based on simulated annealing, was developed by Levi and Ersoy [LE94] is presented in Section 3.4 of this chapter. This method is faster, and gives better results, than the Marayuma-Tang solution, especially for large networks.
3.2 Assumptions and Delay Formulae

The network model that will be used for all the solutions in the following sections has the following features.

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 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 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 expressions derived in Chapter 2, all the researchers in the field have used the following formulae for the delay cost incurred in the network.

\[ T_{jk} = \frac{\eta_p \cdot \left( \sum_{i \in S^i} \frac{\lambda_{il} \cdot m_i}{\eta_p C_j} \right)^2}{(1 - U_r - 1)(1 - U_r)} + \frac{m_k}{C_j} \]  \hspace{1cm} (3.1)

\[ U_r = \sum_{i \in S^i} \frac{\lambda_{il} \cdot m_i}{C_i} \]  \hspace{1cm} (3.2)

\[ Z_k = \frac{\sum_j T_{jk} \cdot \lambda_{jk}}{\gamma_k} \]  \hspace{1cm} (3.3)

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 \( k \),
- \( \eta_p = \sum_{i} \lambda_{il} \) is the Total Packet Rate on link \( j \),
- \( \gamma_k = \sum_{j} \lambda_{jk} \) is the Total Rate of packet class \( k \) entering the network,
- \( \lambda_{jk} \) 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 the problem reduces to the integer programming problem which is to minimize the network cost,
\[
D = \sum \sum d_i \cdot C_j \cdot x_{ij}
\]
subject to,
\[
Z_k \leq B_k, \forall k,
\]
where, \( d_i \) 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 \( j \), zero otherwise.
Consequently,
\[
\sum x_{ij} = 1, \forall j
\]

3.3 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 3.3.1 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.

<table>
<thead>
<tr>
<th>CAPACITY (bps)</th>
<th>COST PER KM ($)</th>
<th>FIXED COSTS ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>0.31</td>
<td>750.00</td>
</tr>
<tr>
<td>19200</td>
<td>1.31</td>
<td>850.00</td>
</tr>
<tr>
<td>50000</td>
<td>2.63</td>
<td>850.00</td>
</tr>
<tr>
<td>108000</td>
<td>2.63</td>
<td>2400.00</td>
</tr>
<tr>
<td>230000</td>
<td>13.10</td>
<td>7300.00</td>
</tr>
<tr>
<td>460000</td>
<td>37.50</td>
<td>8300.00</td>
</tr>
</tbody>
</table>

Table 3.3.1 Set of possible Link Capacities and Costs that are used for all networks.

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 thesis are shown in Table 3.3.2 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.
<table>
<thead>
<tr>
<th>NET L.D.</th>
<th>NUMBER OF LINKS</th>
<th>AVERAGE LINK LENGTH</th>
<th>PACKET CLASSES</th>
<th>AVERAGE PACKET RATES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>54.67</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>13.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>14.5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>58.75</td>
<td>1</td>
<td>14.375</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>15.625</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>15.125</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>15.5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>58.08</td>
<td>1</td>
<td>15.417</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>15.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>17.083</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>58.08</td>
<td>1</td>
<td>15.417</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>15.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>17.083</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>17.33</td>
</tr>
<tr>
<td>5</td>
<td>48</td>
<td>54.67</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>13.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>14.5</td>
</tr>
</tbody>
</table>

Table 3.3.2 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 3.3.3 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 4. The delay bound indicates the maximum amount of time that the packet can stay undelivered in the network.

<table>
<thead>
<tr>
<th>NET I.D.</th>
<th>PACKET CLASS</th>
<th>PACKET PRIORITY</th>
<th>DELAY BOUND</th>
<th>PACKET LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0.013146</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0.051933</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0.914357</td>
<td>400</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0.013146</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0.051933</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0.914357</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>0.009845</td>
<td>322</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0.053146</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0.151933</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0.914357</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>0.029845</td>
<td>322</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>0.053146</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0.151933</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0.914357</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4</td>
<td>0.029845</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>5</td>
<td>0.000984</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>0.013146</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2</td>
<td>0.051933</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0.914357</td>
<td>400</td>
</tr>
</tbody>
</table>

Table 3.3.3 Characteristic values of packet classes for each network.
Chapter 3: The Capacity Assignment Problem

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 3.3.1 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 3.3.1 and the average of the lengths will be specified by the quantity “average length” of Network Type 1 taken from Table 3.3.2. This type of network will carry traffic that exhibit characteristics similar to those shown in Table 3.3.2 and Table 3.3.3 for Network Type 1.

Figure 3.3.1: A sample network of Type 1.
3.4 The Marayuma-Tang Solution

The Marayuma-Tang (MT-CA) solution to the Capacity Assignment (CA) problem 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 terms are listed below:

- $C_j$ 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,
- $T_j$ is the average link delay for packet class $k$ when $C_j$ is used on link $j$,
- $T_j^+$ is the average link delay for packet class $k$ when $C_j^+$ is used on link $j$,
- $T_j^-$ is the average link delay for packet class $k$ when $C_j^-$ is used on link $j$,
- $Z_j$ is the average delay for packet class $k$ when $C_j$ is used on link $j$,
- $Z_j^+$ is the average delay for packet class $k$ when $C_j^+$ is used on link $j$, and finally,
- $Z_j^-$ is the average delay for packet class $k$ when $C_j^-$ is used on link $j$.

The Marayuma-Tang (MT-CA) solution to the CA problem is based on the philosophy of a few procedures. We describe each of them below.
Chapter 3: The Capacity Assignment Problem

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

(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 larger than the average delay then the problem is not feasible.

(b) SetLow: When this procedure is invoked 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.
3.4.1 AddFast Procedure

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 is 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.

Method

BEGIN

Repeat

Find the packet class \(k\) which satisfies the following equation:

\[
\max_k \left\{ \frac{Z_k}{B_k} \right\}
\]

Find the link \(j\) which carries \(k\) class packets and satisfies the following:

\[
\max_j \left\{ \frac{\lambda_k (T_k - T_k^*)}{D_j^* - D_j} \right\}
\]

Link \(j := C_j^*\)

Until \((Z_k \leq B_k\) is true for all \(k\))

END Procedure AddFast
3.4.2 DropFast Procedure

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.

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.

Method

BEGIN

Repeat

Find the packet class k which satisfies the following equation:

\[ \text{MIN}_k \left\{ \frac{Z_k}{B_k} \right\} \]

Find the link j which carries k class packets and satisfies the following:

\[ \text{MIN}_j \left\{ \frac{\lambda_k \left( T_k - T_j \right)}{D_j - D_j^*} \right\} \]

Link j := C_j

Until \((Z_k^* \leq B_k)\) holds for all k

END Procedure DropFast
3.4.3 Exc Procedure

This procedure 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.

\[ C_i := C_i^- \]
\[ C_j := C_j^- \]

This reassignment must not violate any delay constraint and must satisfy,

\[ D_i + D_j > D_i^- + D_j^- \]

The pseudo-code of this procedure is as follows.

**Procedure Exc**

**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.

**Method**

**BEGIN**

For \( i = 1 \) to maxlinks Do
    For \( j = 1 \) to maxlinks Do
        If \( (i \neq j) \) AND \( (D_i + D_j > D_i^- + D_j^-) \)
            Then \( C_i := C_i^- \)
                \( C_j := C_j^- \)
        End-If
    End-For
End-For

**END Procedure Exc**
Chapter 3: The Capacity Assignment Problem

To allow the concatenation of the heuristics described above the algorithm provides two interfaces, ResetHigh and ResetLow, which are described 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, that is, \( C^- \).

(b) **ResetLow**: In this procedure the capacity of each link is decreased to the next lower one, that is, \( C^+ \).
3.4.4 The Marayuma-Tang Algorithm

After performing several experiments using these heuristics on a number of different problems it was determined that a solution given by one heuristic can often be improved by running other heuristics consecutively. Many algorithms comprised of a composite of the heuristics were tried and it was found that most solutions gave very good results. The MT-CA algorithm (shown in Section 3.4.2) is the best such composite algorithm. It yielded the best overall performance based on both the optimality of the solution as well as the computational efficiency.

Input: The network characteristics and packet types.
Output: The lowest cost network capacity assignment vector.

Method

BEGIN-MAIN //The Marayuma/Tang Algorithm
    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 //The Marayuma/Tang Algorithm
3.4.5 Experimental Results

Marayuma and Tang provided cost figures for specific problem solutions but did not present a way by which their results could be verified or the quality of their routines measured. Levi and Ersoy [LE94] later showed that the MT-CA solution was quite slow and that it incurred enormous overheads. Another problem with the MT-CA algorithm is that the heuristic is deterministic and therefore produces the same result every time. While this maintains the consistency of a single solution, it is not possible to produce alternatives to the initial solution. This means that if the initial solution obtained after a single run is not the optimal one there is no way to migrate to the optimal solution by this method.

Table 3.4.1 shows results obtained in five consecutive program executions. In each test the final cost obtained and the execution time are provided. It is clear that the performance of the algorithm decreases as the size and complexity of the network increases. The cost values obtained are substantially closer to the optimal value for the less complicated networks. For larger networks the difference between the optimal value and the value obtained by the algorithm tends to increase proportionally with the size of the network. The execution time of the algorithm also increases proportionally with the size and complexity of the network.

\[\text{1 Since we are dealing with an NP-Complete problem there is no single way of computing the optimal value. We refer here to the value obtained by our superior algorithms presented in Chapters 4 and 5.}\]
## Table 3.4.1 Results for MT-CA Algorithm tests using the networks described in Section 3.3.
3.5 The Levi-Ersoy Solution

To our knowledge the faster and more accurate scheme is the Levi-Ersoy solution to the CA problem (LE-CA) 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 on 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, then the neighbor solution is accepted unconditionally. If however the value of the objective function of the neighbor solution is worse than the previous solution then 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, $C = (C_1, C_2, C_3, ..., C_n, ..., C_m)$ 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.
3.5.1 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 capacity assignment vectors, $C_n$, to the currently accepted solution, $C_a$.

2. Calculate the cost difference, $\Delta D$, 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, $C_0$, are acceptable. This means that $t_0$ is chosen so that $P_0 = \exp(-\Delta D_{\text{mean}}/t_0) \cong 1$, for all possible feasible neighbors. $\Delta D_{\text{mean}}$ is the average increase in cost and is calculated by looking at $W$ random neighbors of $C_0$. Therefore $t_0$ becomes $\Delta D_{\text{mean}}/\ln(1/P_0)$. 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.

\[
t_k = t_0 \times e^{-\alpha k}
\]

where, \( \alpha \) 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 follows.
3.5.2 The Levi-Ersoy Algorithm

Input: The network characteristics and packet types.
Output: The lowest cost network capacity assignment vector.

Method

BEGIN-MAIN //The Levi/Ersoy Algorithm

//Initialization Section
k := 0
P₀ := 0.99
max-iterations := MAX{30, 2*number-of-links}
Choose the initial feasible capacity assignment vector C₀ at random
For max-iterations Do
    Create a neighbor of Cᵣ and call it Cᵣ
    If (Cᵣ is feasible) AND (the cost of Cᵣ is larger than Cᵣ)
        Then ΔD_mean := AVE(cost of Cᵣ - cost of Cᵣ)
    End-If
End-For

nᵣ := ∆D_mean/ln(1/P₀)
//Annealing Section
Cₙ := Cᵣ
Repeat
    n := 0
    Repeat
        n := n+1
        Create a neighbor of Cₙ and call it Cₙ
        If (Cₙ is feasible)
            Then If (the cost of Cₙ(Dₙ) is lower then the cost if Cₙ(Dₙ))
                Then Cₙ := Cₙ
            End-If
        Else If EXP((Dₙ - Dₙₙ)/nᵣ) > RANDOM[0, 1]
            Then Cₙ := Cₙₙ
        End-If
    End-If
Until (the number of acceptances of Cₙ as Cₙ is 5) OR (n = 2*number of links)
k := k+1
nᵣ := t₀ * e⁻ⁿᵣ
Until (Dₙ is the same for 3 consecutive value of k)

END-MAIN //The Levi/Ersoy Algorithm
3.5.3 Experimental Results

Table 3.5.1 shows the results for five consecutive executions of the LE-CA algorithm. It is clear from these results that better results can be obtained from the LE-CA algorithm as compared with the MT-CA algorithm. For example, let us examine the results obtained for Network #4 for which the MT-CA algorithm produces a Final Cost value of $53,765.90. Over the five tests the LE-CA algorithm produces four Final Cost values that are superior to this value, the lowest being $45,709.60, and only one that is higher. Also, the MT-CA algorithm takes 2.14 seconds to execute while the LE-CA algorithm takes 1.93 seconds in the worst case and 1.21 seconds in the best case. A better cost value is obtained for every test network although the results are different every time the algorithm is executed. Additionally, the execution times are substantially better for the LE-CA algorithm, which in nearly every case runs 50% faster than the MT-CA algorithm. Note that the size of the network has little or no effect on the quality of the cost values obtained. In the best cases the cost values are much closer to the optimal value than those obtained using the MT-CA algorithm. Table 3.5.2 shows the best results obtained over a longer range of tests which confirms the fact that not only does the LE-CA algorithm produce cost values that are much closer to the optimal value but also runs much faster than the MT-CA algorithm. These results are consistent for every network tested.
## Table 3.5.1 Average results for LE-CA Algorithm tests using the networks described in Section 3.3.

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

## Table 3.5.2 Best Results for LE-CA Algorithm tests using the networks described in Section 3.3.

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>5735.04</td>
<td>7214.22</td>
<td>10295.70</td>
<td>45709.60</td>
<td>39838.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>1.21</td>
<td>1.10</td>
<td>1.93</td>
<td>4.01</td>
</tr>
</tbody>
</table>
Chapter 3: The Capacity Assignment Problem

It should be noted that the time that is required to find the initial feasible solution is not accounted for in the times given in the results. The time taken by the LE-CA algorithm to find the initial solution fluctuates widely depending on the random numbers generated that are used but tends to increase slightly as the size and complexity of the network increases.

3.6 Comparison

It is clear from the results that the LE-CA algorithm is, on average, faster than the MT-CA algorithm. The difference in time is due to the many checks and overhead computations that arise due to the nature of the MT-CA algorithm. Additionally, the execution time of the MT-CA algorithm increases at a higher rate than the LE-CA algorithm with both the size of the network and the number of classes.

The LE-CA algorithm, on average, produces equivalent or better cost results than the MT-CA algorithm, especially in the cases of more complex networks. Indeed, the reliability of the MT-CA algorithm decreases as the network becomes more complicated while the LE-CA algorithm always has the same high probability of returning a superior result. Also, since the LE-CA algorithm is probabilistic the same problem can be solved multiple times to find different results, and the best solution can be chosen. The MT-CA algorithm is deterministic, and therefore always returns the same result for the same input.
3.7 Conclusion

In this chapter we have described the CA problem and stated the basic assumptions and the delay formulae that are used for all the solutions. Two previous solutions to the problem have been presented. The first, due to Marayuma-Tang (MT-CA), is based on cost/delay heuristics and performs adequately in finding an acceptable solution but is generally unacceptable in terms of the time required as the size of the network increases. The accuracy of the algorithm also decreases as the networks become larger and more complex. The second method, due to Levi-Ersoy (LE-CA), is based on the concept of simulated annealing. This algorithm generally provides a solution with a lower cost and a substantial improvement in execution time when compared to the MT-CA algorithm. Also, the accuracy of the solution produced by the LE-CA algorithm is less affected by the size and complexity of the network while the reliability of the MT-CA algorithm decreases.

In the following chapter a better solution to the CA problem that uses learning automata is presented and produces results superior to both of the schemes discussed in this chapter.
CHAPTER 4

THE CONTINUOUS AUTOMATA SOLUTION TO THE CA PROBLEM

4.1 Introduction

In the previous chapter the CA problem was defined and two solutions to the problem were presented. The first, due to Marayuma/Tang (MT), produced adequate results for small networks but its performance, measured in terms of time and cost, degraded substantially and quickly as the size and complexity of the network increases. The second solution, due to Levi/Ersoy (LE), is faster and produces better cost results than the MT solution, but still did not find optimal solutions, especially for larger, and more complex networks.

In this chapter, a new solution is presented that uses learning automata to achieve cost results that are closer to the optimal value, and is faster than either of the previous solutions. The solution uses a continuous learning scheme that achieves superior cost results, and is faster than the MT or LE solutions.
4.2 Introduction to 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, some of these are listed below:

- parameter optimization
- statistical decision making
- telephone routing
- pattern recognition
- game playing
- natural language processing
- modeling biological learning systems.

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 Narendra 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, \ldots, \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 $\alpha_i$, 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 $\alpha_i$. 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 diagrammatically below.

\[
\{c_1, ..., c_r\} \\
\text{Random Environment} \\
\alpha = \{\alpha_1, ..., \alpha_r\} \\
\beta = \{0, 1\} \\
\text{Learning Automaton}
\]

Figure 4.2.1: The Automaton - Environment Feedback Loop.
4.2.1 Random Environments

Suppose the action $\alpha_i$, chosen by the LA at the $n^{th}$ time instant, serves as input to the RE. Upon receiving the input, the RE gives its response $\beta(n) \in \{0, 1\}$. The RE penalizes the LA, $\beta(n) = 1$, with probability $c_n$ where,

$$c_i(n) = \Pr(\beta(n) = 1 \mid \alpha(n) = \alpha_i) \quad 1 \leq i \leq r.$$

Therefore, the characteristics of the RE are specified by the set of penalty probabilities $\{c_1(n), ..., c_r(n)\}$. If the $c_i$'s are time invariant then the RE is said to be stationary, otherwise the RE is non-stationary.

4.2.2 Learning Criteria

With no a priori information, the automaton chooses all the actions with equal probability. Therefore, the initial expected penalty ($M$), at time instant $n = 0$, is given as:

$$M_0 = (c_1 + c_2 + \ldots + c_r) / r.$$

An LA is said to learn expediently if, as time approaches infinity, the expected penalty is less than $M_0$. Let the expected penalty at time ‘$n$’ be $E[M(n)]$. The LA is said to be optimal if $E[M(n)]$ converges to the minimum penalty probability as time approaches infinity. It is $\varepsilon$-optimal if for any arbitrary $\varepsilon > 0$, in the limit, $E[M(n)] < c_m + \varepsilon$, where $c_m$ is the minimum penalty probability.
4.3 Classification of Learning Automata

The internal decision process of an LA is modeled using a transition matrix that describes the changes of the internal state of the LA on receiving responses form the RE. Similarly, the action chosen by the LA is a function of its internal state. This function is determined by an output matrix. Based on the structure of the transition and output matrices LA can be classified into the following two categories.

1. Fixed Structure Stochastic Automata (FSSA)

4.3.1 Fixed Structure Stochastic Automata

Fixed Structure Stochastic Automata (FSSA) exhibit transition and output matrices that are time invariant. The output matrix of an FSSA is usually deterministic but the transition matrix can either be deterministic (for example, the Tsetlin and Krinsky automata) or stochastic (for example, the Krylov automaton).

An FSSA can be formally defined as a quintuple \((\alpha, \phi, \beta, F, G)\), where,

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

(ii) \(\phi = \{\phi_1, ..., \phi_k\}\) is the set of states, with \(k \geq r\).

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

(iv) \(F\) is the transition matrix. This is a map from \(\phi \times \beta\) to \(\phi\) and defines the transition of the internal state of the LA on receiving an input.

(v) \(G\) is the output matrix. This is a map from \(\phi \times \alpha\) and determines the action taken by the LA if it is in a certain state.
Chapter 4: The Continuous Automata Solution to the CA Problem

The actual process of learning in the FSSA is accomplished by the following process. The LA is offered a set of \( r \) actions \( \{\alpha_1, ..., \alpha_r\} \), where for example, each \( \alpha_i \) has \( N \) states \( \{\phi_1, ..., \phi_N\} \) associated with it, \( 1 \leq i \leq r \). With no loss of generality, we can assume that \( \phi_1 \) is the most internal state and \( \phi_N \) is the boundary state of \( \alpha_i \). If at time 'n' the LA chooses action \( \alpha_i \) and is in state \( \phi_j \), where \( 1 \leq j \leq N \), and it receives a reward from the RE \((\beta(n) = 0)\), then the LA interprets the reward as an indication that \( \alpha_i \) may be the optimal action. Consequently, the LA goes deeper into \( \alpha_i \) by moving toward the most internal state, that is, from \( \phi_j \) to \( \phi_k \), where \( 1 \leq j, \, k \leq N \) and \( k \leq j \). Alternatively, if a penalty is received \((\beta(n) = 1)\), the LA assumes that \( \alpha_i \) may not be the optimal action and starts leaving it by moving toward the boundary state of \( \alpha_i \), that is from \( \phi_j \) to \( \phi_k \), where \( 1 \leq j, \, k \leq N \) and \( k > j \). If \( \alpha_i \) is the optimal action the LA receives rewards more frequently than if any other action were chosen, and therefore, the LA ultimately circulates within the state near \( \phi_1 \), the most internal state of \( \alpha_i \). However, if \( \alpha_i \) is not the optimal action then the LA receives penalties more frequently than if the optimal action was chosen, and the LA eventually reaches \( \phi_N \), the boundary state of \( \alpha_i \), and moves to another action, \( \alpha_p \). This procedure repeats until the LA arrives at the optimal action.

FSSA are easy to implement but they have a major disadvantage that makes them unsuitable to the CA problem. If an LA is currently choosing an action it will always choose the same action in the next time instant, even if it receives a penalty from the RE. The only exception occurs once the LA has reached a boundary state of the action. In this case, the LA will leave the current action and choose a new action. This characteristic is particularly undesirable when the number of states of an action is large because the LA will take a long time to leave an action, even if it is an incorrect one.
4.3.2 Variable Structure Stochastic Automata

Variable Structure Stochastic Automata (VSSA) exhibit transition and output matrices that are time invariant. 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 \(\alpha_i\) at time ‘\(n\)’,

\[
0 \leq p_i(n) \leq 1, \text{ and}
\]

\[
\sum_{i=1}^{r} p_i(n) = 1.
\]

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. This feature overcomes the previously mentioned problem with FSSA, and generally makes VSSA converge much faster than FSSA.

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, meaning that the penalty probabilities are time varying, since 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 \((\alpha, p, \beta, T)\), 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 \(\alpha_i\) 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’ represents 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.

In this thesis, VSSA are used to accomplish the learning process. The rules of some action probability updating schemes are shown below.

(i) **Reward-Penalty** - The probabilities are modified regardless of whether the response received was a reward or a penalty. This usually leads to an ergodic scheme.

(ii) **Reward-Inaction** - The probabilities are modified only when the action chosen is rewarded, and leads to absorbing schemes.

(iii) **Inaction-Penalty** - The probabilities are modified only when a penalty response is obtained. This leads to various ergodic schemes.

**Example 4.3.1**

Consider a VSSA that has 2 possible actions, \((\alpha_1, \alpha_2)\) with linear reward and penalty multiplying factors \(\lambda_r, \lambda_p\) where \(\lambda_r = 0.9, \lambda_p = 0.8\). If both actions are equally likely at first, then the initial probability vector is \([0.5, 0.5]^T\). The updating rules for the L_{RP} scheme are as follows.
\textbf{Chapter 4: The Continuous Automata Solution to the CA Problem}

\begin{align*}
    p_{1}(n+1) &= 1 - \lambda_{r}p_{2}(n) \quad \text{if } \alpha_{1} \text{ is chosen and } \beta = 0 \\
    p_{1}(n+1) &= \lambda_{p}p_{1}(n) \quad \text{if } \alpha_{1} \text{ is chosen and } \beta = 1 \\
    p_{2}(n+1) &= \lambda_{r}p_{1}(n) \quad \text{if } \alpha_{2} \text{ is chosen and } \beta = 0 \\
    p_{2}(n+1) &= 1 - \lambda_{p}p_{2}(n) \quad \text{if } \alpha_{2} \text{ is chosen and } \beta = 1
\end{align*}

As stated above, both rewards and penalties are processed in this scheme. If \(\alpha_{1}\) is chosen and receives a penalty then the probability of choosing this action on the next iteration, \(p_{1}(n+1)\), must be reduced. This is accomplished by multiplying the present probability by the parameter \(\lambda_{p}\). However, if \(\alpha_{1}\) is chosen and receives a reward the probability of choosing this action on the next iteration must be increased. This, in turn, is accomplished by subtracting the value given by \(\lambda_{r}p_{2}(n)\) from unity. Similarly, if \(\alpha_{2}\) is chosen and receives a penalty then the probability of choosing this action on the next iteration, \(p_{2}(n+1)\), must be reduced. However, unlike the case for \(\alpha_{1}\), this is accomplished by subtracting the value \(\lambda_{p}p_{2}(n)\) from unity. Also, if \(\alpha_{2}\) is chosen and receives a reward then the probability of choosing this action on the next iteration must be increased which is accomplished by multiplying the present probability for \(\alpha_{1}\), \(p_{1}(n)\), by the parameter \(\lambda_{r}\).

In the present case if the current probability vector is \([0.5, 0.5]^{T}\) there are four possible next states.

- Case 1: LA chooses \(\alpha_{1}\), RE responds with \(\beta = 0\) \(\Rightarrow\) \(p(1)=[0.55, 0.45]^{T}\)
- Case 2: LA chooses \(\alpha_{1}\), RE responds with \(\beta = 1\) \(\Rightarrow\) \(p(1)=[0.4, 0.6]^{T}\)
- Case 3: LA chooses \(\alpha_{2}\), RE responds with \(\beta = 0\) \(\Rightarrow\) \(p(1)=[0.45, 0.55]^{T}\)
- Case 4: LA chooses \(\alpha_{2}\), RE responds with \(\beta = 1\) \(\Rightarrow\) \(p(1)=[0.6, 0.4]^{T}\)

This type of VSSA uses a simple \textbf{Linear Reward-Penalty} (LRP) scheme. In this case although we are trying to achieve convergence to a unit vector that would indicate the optimal action (where convergence to \([1, 0]\) would correspond to \(\alpha_{1}\) being the optimal action, and
convergence to [0, 1] would correspond to $\alpha_2$ being the optimal action) it can be shown that the system converges in distribution. Indeed, since this type of system processes both rewards and penalties it does not converge to a unit vector. Consequently, as we shall see in the case of the CA problem, it is not very suitable in finding the optimal solution for stationary environments.

Fortunately, absorbing schemes will converge to a unit vector. One such scheme is Linear Reward-Inaction ($L_{RI}$). While in the case of the $L_{RP}$ scheme both rewards and penalties are processed, for the $L_{RI}$ scheme only rewards are processed. A simple $L_{RI}$ scheme is shown in the example below.

**Example 4.3.2**

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.

\[
\begin{align*}
    p_i(n+1) &= 1 - \sum_{j \neq i} \lambda_j p_j(n) & \text{if} & & \alpha_i \text{ is chosen and } \beta = 0 \\
    p_i(n+1) &= p_i(n) & \text{if} & & \alpha_i \text{ is chosen and } \beta = 1 \\
    p_j(n+1) &= \lambda_j p_j(n) & \text{if} & & \alpha_i \text{ is chosen and } \beta = 0 \\
    p_j(n+1) &= p_j(n) & \text{if} & & \alpha_i \text{ is chosen and } \beta = 1
\end{align*}
\]

As stated above, only rewards are processed in this scheme. Therefore, if $\alpha_i$ is chosen and receives a reward then 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 $\alpha_j$, for all $j \neq i$, on the next iteration are reduced by setting $p_j(n+1)$ to $\lambda_j p_j(n)$ for all $j \neq i$. Next, the probability of choosing $\alpha_i$ 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. There are no modifications for penalties.
Chapter 4: The Continuous Automata Solution to the CA Problem

By way of example, if \( r = 2 \), with a linear reward multiplying factor \( \lambda_r = 0.9 \), and an initial probability vector \( [0.5, 0.5]^T \), there are four possible next states.

Case 1: LA chooses \( \alpha_1 \), RE responds with \( \beta = 0 \) \( \Rightarrow \) \( p(1) = [0.55, 0.45]^T \)

Case 2: LA chooses \( \alpha_1 \), RE responds with \( \beta = 1 \) \( \Rightarrow \) \( p(1) = [0.5, 0.5]^T \)

Case 3: LA chooses \( \alpha_2 \), RE responds with \( \beta = 0 \) \( \Rightarrow \) \( p(1) = [0.45, 0.55]^T \)

Case 4: LA chooses \( \alpha_2 \), RE responds with \( \beta = 1 \) \( \Rightarrow \) \( p(1) = [0.5, 0.5]^T \)

It can be shown that as this process continues, a unit vector will eventually be reached since the chain is absorbing. Indeed, the LRI scheme can be shown to be \( \varepsilon \)-optimal - it converges to the optimal action with a probability as close to unity as desired. For this reason it is ideal for solving the CA problem.

VSSA have many powerful properties. Indeed, they form the basis for Reinforcement Learning Schemes used in neural networks and adaptive algorithms [NT89].
4.4 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.

\[(C_1, C_2, ..., C_n, ..., C_a)\]

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 be **remain unchanged**

\(D_{ij}\) 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.
Chapter 4: The Continuous Automata Solution to the CA Problem

We now present the various initial settings for the probability vector. Indeed, by the virtue of the various values for \( C_n \), there are three possible settings for the initial probability vector \((I_n, S_n, D_n)\) given below as \(\text{Init 1} \), \(\text{Init 2} \) and \(\text{Init 3} \) 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,
\[
I_n = 1/2, \quad S_n = 1/2, \quad D_n = 0,
\]

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,
\[
I_n = 0, \quad S_n = 1/2, \quad D_n = 1/2,
\]

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

**Init 3:** This is the scenario where the capacity 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,
\[
I_n = 1/3, \quad S_n = 1/3, \quad D_n = 1/3 \quad \text{for } 0 < j < n
\]

It should be noted that since we will be using an absorbing scheme there should be no initial preference given to any one strategy since this might result in the automaton converging prematurely. As a result, each possible strategy is for a given link is equally likely at the start of the algorithm.

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 and chooses links at random
trying to find the lowest possible cost by raising and lowering the capacities of the link using the $L_{R1}$ learning strategy. At each step of this process a different capacity assignment vector is examined. 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, $\lambda_{R1}$, is the reward parameter to be used when a feasible solution is reached, and the second, $\lambda_{R2}$, 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
Chapter 4: The Continuous Automata Solution to the CA Problem

the optimal probability vector decreases with the parameters. The closer they are to unity, the slower the algorithms are. We clarify the operation of the scheme by the following example.

Example 4.4.1

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. In what follows let $t$ be the time index.

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

\[
I_{ij}(t+1) = \lambda_{R1} \cdot I_{ij}(t)
\]

\[
S_{ij}(t+1) = \lambda_{R1} \cdot S_{ij}(t)
\]

\[
D_{ij}(t+1) = 1 - (I_{ij}(t+1) + S_{ij}(t+1)).
\]

(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:

\[
I_{ij}(t+1) = \lambda_{R2} \cdot I_{ij}(t)
\]

\[
D_{ij}(t+1) = \lambda_{R2} \cdot D_{ij}(t)
\]

\[
S_{ij}(t+1) = 1 - (I_{ij}(t+1) + D_{ij}(t+1)).
\]

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, $C_n$, 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.
4.4.1 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 $\lambda_{R1}, \lambda_{R2}$.

Output: 
The lowest cost network capacity assignment vector.

Method

START-MAIN

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) $C_i = \text{RAND}(1, \text{maxcap})$ 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)
   Repeat
      i = RAND(1, maxlinks)
      Action = RAND(Increase$_{ij}$, Stay$_{ij}$, Decrease$_{ij}$)
      If (Action = Increase$_{ij}$) Then $C_i = C_i^+$ End-if
      Else If (Action = Decrease$_{ij}$) Then $C_i = C_i^-$ End-if
   End-if
Until (all links are set)
current-cost = calculate-network-cost()
For ($i=1$ to maxlinks)
   j = $C_i$
   If (network is feasible)
      Then If (Action = Increase$_{ij}$) Then Raise($D_{ij}, \lambda_{R1}$) End-If
      If (Action = Stay$_{ij}$) Then Raise($S_{ij}, \lambda_{R1}$) End-If
      If (Action = Decrease$_{ij}$) Then Raise($D_{ij}, \lambda_{R1}$) End-If
   Else Reset all links to best-cost capacities.
End-If
Chapter 4: The Continuous Automata Solution to the CA Problem

If (network is feasible) AND (current-cost < best-cost)
Then If (Action = Increase) Then Raise(D_{ij}, \lambda_{R2}) End-If
    If (Action = Stay) Then Raise(S_{ij}, \lambda_{R2}) End-If
    If (Action = Decrease) Then Raise(S_{ij}, \lambda_{R2}) End-If
    For (i=1 to maxlinks) best-C_i = C_i End-For
    best-cost = current-cost
End-If
End-While
END-MAIN

Procedure Raise

Input:
(i) Action_{ij} : either - Increase, Stay, Decrease - performed on link i with current capacity j.
(ii) \lambda_R, which is the learning scheme modification parameter. \lambda_R is set to \lambda_{R1} or \lambda_{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 \times D_{ij}
    S_{ij} = \lambda_R \times S_{ij}
    I_{ij} = 1 - (D_{ij} + S_{ij})
Else If (Action = Stay)
    Then I_{ij} = \lambda_R \times I_{ij}
    D_{ij} = \lambda_R \times D_{ij}
    S_{ij} = 1 - (I_{ij} + D_{ij})
End-If
Else If (Action = Decrease)
    Then I_{ij} = \lambda_R \times I_{ij}
    S_{ij} = \lambda_R \times S_{ij}
    D_{ij} = 1 - (I_{ij} + S_{ij})
End-If
End-If

END Procedure Raise
4.5 Experimental 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 with selected values for $\lambda_{R1}$ and $\lambda_{R2}$ which ranged from 0.8 to 0.95. Table 4.5.7 contains the best results obtained from the algorithm over the entire testing phase. A comparison with the MT and LE solutions concludes the section.

The result of each test is measured in terms of three parameters, Final Cost, Time, and Iterations. The Final Cost and Time parameters are used for comparison with the MT and LE algorithms while the Iterations parameter will be used for comparison with the discretized version of the new algorithm which is presented in the following chapter and which proves to be faster while maintaining the quality of the Final Cost parameter. For example, in Table 4.5.1 for Test #1, $\lambda_{R1}$ was set to 0.8 and $\lambda_{R2}$ was set to 0.8. For this test the Final Cost obtained was $5,074.68 and the algorithm took a time of 0.05 seconds to complete 40 Iterations.

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.
Figure 4.5.1: The Decrease in Cost with the Reward Parameters for the CASCA Algorithm
Figure 4.2.1: The increase in execution time with the reward parameters for the CASCA algorithm.
**Chapter 4: The Continuous Automata Solution to the CA Problem**

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>5074.68</td>
<td>14491.60</td>
<td>12306.60</td>
<td>72868.80</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.33</td>
<td>0.22</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>40</td>
<td>175</td>
<td>62</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>5062.68</td>
<td>8766.54</td>
<td>15113.50</td>
<td>54385.80</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.11</td>
<td>0.44</td>
<td>1.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>64</td>
<td>59</td>
<td>166</td>
<td>249</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>5068.68</td>
<td>10316.50</td>
<td>15942.30</td>
<td>60639.50</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.22</td>
<td>0.22</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>50</td>
<td>109</td>
<td>75</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6988.94</td>
<td>24585.50</td>
<td>61297.20</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.11</td>
<td>0.61</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>78</td>
<td>67</td>
<td>208</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4919.68</td>
<td>8843.10</td>
<td>14016.30</td>
<td>52466.20</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.22</td>
<td>0.28</td>
<td>0.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>31</td>
<td>127</td>
<td>75</td>
<td>79</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5.1: Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are $\lambda_{R1} = 0.8$, $\lambda_{R2} = 0.8$. 
### Table 4.5.2 Results for the CASCA Solution to the CA Problem

The networks that are used are described in Section 3.3 and the parameters are $\lambda_{R1} = 0.9$, $\lambda_{R2} = 0.8$.  

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($$$)</td>
<td>4919.68</td>
<td>7186.38</td>
<td>13741.80</td>
<td>49865.80</td>
<td>48521.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.16</td>
<td>0.61</td>
<td>0.66</td>
<td>4.61</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>47</td>
<td>98</td>
<td>190</td>
<td>128</td>
<td>449</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($$$)</td>
<td>4992.16</td>
<td>7216.54</td>
<td>9934.63</td>
<td>45720.10</td>
<td>66753.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.28</td>
<td>0.43</td>
<td>1.21</td>
<td>2.20</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>81</td>
<td>159</td>
<td>125</td>
<td>216</td>
<td>213</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($$$)</td>
<td>4919.68</td>
<td>8736.38</td>
<td>10528.40</td>
<td>46547.00</td>
<td>51397.90</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.06</td>
<td>0.33</td>
<td>0.44</td>
<td>0.93</td>
<td>2.41</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>63</td>
<td>131</td>
<td>128</td>
<td>168</td>
<td>241</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($$$)</td>
<td>4919.68</td>
<td>8764.22</td>
<td>11213.60</td>
<td>44452.80</td>
<td>81755.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.16</td>
<td>0.55</td>
<td>0.50</td>
<td>1.54</td>
<td>5.32</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>103</td>
<td>239</td>
<td>167</td>
<td>291</td>
<td>514</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($$$)</td>
<td>5735.04</td>
<td>7186.38</td>
<td>11793.40</td>
<td>48323.50</td>
<td>51093.10</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.22</td>
<td>0.66</td>
<td>0.65</td>
<td>2.47</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>87</td>
<td>109</td>
<td>196</td>
<td>112</td>
<td>241</td>
</tr>
<tr>
<td>Test</td>
<td>Category</td>
<td>Net 1</td>
<td>Net 2</td>
<td>Net 3</td>
<td>Net 4</td>
<td>Net 5</td>
</tr>
<tr>
<td>------</td>
<td>------------</td>
<td>--------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>8843.10</td>
<td>12961.60</td>
<td>49876.90</td>
<td>40406.90</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.33</td>
<td>0.66</td>
<td>1.16</td>
<td>1.65</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>76</td>
<td>171</td>
<td>249</td>
<td>214</td>
<td>155</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>5020.00</td>
<td>10144.60</td>
<td>10006.60</td>
<td>49221.60</td>
<td>48688.90</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.72</td>
<td>0.66</td>
<td>1.31</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>69</td>
<td>358</td>
<td>191</td>
<td>242</td>
<td>139</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4919.68</td>
<td>7265.26</td>
<td>11744.70</td>
<td>48376.00</td>
<td>37809.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.22</td>
<td>0.60</td>
<td>1.37</td>
<td>3.13</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>74</td>
<td>105</td>
<td>199</td>
<td>245</td>
<td>301</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>5161.20</td>
<td>6940.22</td>
<td>10779.20</td>
<td>45709.60</td>
<td>37771.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.06</td>
<td>0.33</td>
<td>0.66</td>
<td>1.38</td>
<td>2.14</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>80</td>
<td>171</td>
<td>218</td>
<td>265</td>
<td>204</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4919.68</td>
<td>7448.10</td>
<td>10194.70</td>
<td>43189.50</td>
<td>37723.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.33</td>
<td>0.83</td>
<td>1.05</td>
<td>2.37</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>83</td>
<td>162</td>
<td>291</td>
<td>184</td>
<td>223</td>
</tr>
</tbody>
</table>

Table 4.5.3 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are $\lambda_{R1} = 0.9$, $\lambda_{R2} = 0.9$. 
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>10015.80</td>
<td>44138.60</td>
<td>37543.90</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.55</td>
<td>1.10</td>
<td>2.36</td>
<td>2.58</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>148</td>
<td>255</td>
<td>324</td>
<td>444</td>
<td>272</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>10872.90</td>
<td>45803.80</td>
<td>44855.20</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.32</td>
<td>0.77</td>
<td>0.88</td>
<td>1.97</td>
<td>2.25</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>257</td>
<td>427</td>
<td>2.91</td>
<td>366</td>
<td>210</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7265.26</td>
<td>10728.20</td>
<td>49729.70</td>
<td>37744.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.44</td>
<td>1.38</td>
<td>1.92</td>
<td>4.12</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>126</td>
<td>221</td>
<td>457</td>
<td>349</td>
<td>394</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6940.22</td>
<td>10514.70</td>
<td>44526.00</td>
<td>37437.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.28</td>
<td>0.82</td>
<td>0.93</td>
<td>1.81</td>
<td>3.90</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>209</td>
<td>383</td>
<td>326</td>
<td>327</td>
<td>385</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>9909.11</td>
<td>46034.20</td>
<td>38072.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.17</td>
<td>0.55</td>
<td>1.37</td>
<td>2.31</td>
<td>3.46</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>148</td>
<td>263</td>
<td>434</td>
<td>433</td>
<td>329</td>
</tr>
</tbody>
</table>

Table 4.5.4 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are $\lambda_{R1} = 0.95, \lambda_{R2} = 0.9.$
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7016.78</td>
<td>10458.80</td>
<td>41888.40</td>
<td>39303.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.87</td>
<td>1.37</td>
<td>2.58</td>
<td>2.08</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>160</td>
<td>422</td>
<td>429</td>
<td>467</td>
<td>194</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>9939.27</td>
<td>44138.60</td>
<td>40866.20</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.55</td>
<td>1.16</td>
<td>2.80</td>
<td>2.14</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>163</td>
<td>244</td>
<td>389</td>
<td>510</td>
<td>203</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>10725.90</td>
<td>40348.90</td>
<td>37862.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.66</td>
<td>1.81</td>
<td>3.40</td>
<td>3.46</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>138</td>
<td>300</td>
<td>629</td>
<td>637</td>
<td>348</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7186.38</td>
<td>10479.70</td>
<td>45385.00</td>
<td>37629.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.28</td>
<td>0.55</td>
<td>1.10</td>
<td>3.08</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>214</td>
<td>285</td>
<td>348</td>
<td>565</td>
<td>176</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6988.94</td>
<td>10707.30</td>
<td>41888.40</td>
<td>38957.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.28</td>
<td>0.72</td>
<td>1.10</td>
<td>2.80</td>
<td>1.87</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>190</td>
<td>325</td>
<td>384</td>
<td>520</td>
<td>180</td>
</tr>
</tbody>
</table>

Table 4.5.5 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are $\lambda_{R1} = 0.95$, $\lambda_{R2} = 0.95$.  

Table 4.5.6 Results for the CASCA Solution to the CA Problem. The networks that are used are described in Section 3.3 and the parameters are $\lambda_{R1} = 0.99$, $\lambda_{R2} = 0.99$. 

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7265.26</td>
<td>9909.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.93</td>
<td>3.25</td>
<td>9.00</td>
<td>18.78</td>
<td>12.14</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>759</td>
<td>1613</td>
<td>3117</td>
<td>3696</td>
<td>1257</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214</td>
<td>10136.70</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.32</td>
<td>3.79</td>
<td>7.69</td>
<td>19.61</td>
<td>12.47</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>1076</td>
<td>1803</td>
<td>3658</td>
<td>3800</td>
<td>1299</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6937.90</td>
<td>9909.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.88</td>
<td>4.29</td>
<td>6.92</td>
<td>14.06</td>
<td>12.46</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>770</td>
<td>2178</td>
<td>2345</td>
<td>2714</td>
<td>1288</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>10505.20</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.10</td>
<td>3.79</td>
<td>8.13</td>
<td>22.74</td>
<td>11.15</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>860</td>
<td>1850</td>
<td>2784</td>
<td>4297</td>
<td>1188</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7186.38</td>
<td>9909.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.95</td>
<td>3.24</td>
<td>7.08</td>
<td>14.66</td>
<td>11.09</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>808</td>
<td>1615</td>
<td>2424</td>
<td>2774</td>
<td>11^4</td>
</tr>
</tbody>
</table>
Chapter 4: The Continuous Automata Solution to the CA Problem

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6937.90</td>
<td>9909.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.39</td>
<td>0.70</td>
<td>1.33</td>
<td>4.12</td>
</tr>
</tbody>
</table>

Table 4.5.7 Best Results for the CASCA Solution to the CA Problem using the networks described in Section 3.3.

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. 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. If we examine the results for the set of five tests where \( \lambda_{R1} \) is set to 0.9 and \( \lambda_{R2} \) 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 is 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 \( \lambda_{R1} \) is set to 0.95 and \( \lambda_{R2} \) is set to 0.95 the CASCA algorithm always finds a superior cost value, even though the execution times are larger than those produced by the MT or LE solutions.
Chapter 4: The Continuous Automata Solution to the CA Problem

It is clear from the results that as the reward values get closer to unity the accuracy of each final cost value improves but the execution times and the amount of iterations increase proportionally. This means that the algorithm can be optimized for speed (decrease the parameters $\lambda_{R1}$, $\lambda_{R2}$), accuracy (increase the parameters $\lambda_{R1}$, $\lambda_{R2}$) or some combination of the two that the user finds appropriate for their particular requirements. Also, the value of $\lambda_{R2}$ should always be set lower than, or equal to, $\lambda_{R1}$ since this is only invoked when a lower cost solution is found and is not used as much as $\lambda_{R1}$, 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.

4.6 Conclusion

In this chapter we have presented a new solution to the CA problem that uses Learning Automata (LA). Section 4.2 provides an introduction to the elementary concepts of LA which aims to learn the optimal action offered by the Random Environment (RE) with which it interacts. LA can be classified into two distinct groups, namely, Fixed Structure Stochastic Automata (FSSA) and Variable Structure Stochastic Automata (VSSA), each of which were briefly discussed. In the new solution to the CA problem we have used VSSA since they prove to be more flexible and converge faster than FSSA.

Section 4.3 presents the Continuous Automata Solution to CA (CASCA) solution to the CA problem and describes the various structures and mechanisms that are used by the algorithm to generate the final cost solution. The CASCA algorithm is formally given in Section 4.4 and a sample of results obtained with various parameter values are given in Section 4.5. These results prove that not only does the CASCA solution produce superior cost results when compared with the MT and LE solutions, but is also significantly faster than either of the previous algorithms.


CHAPTER 5

THE DISCRETE AUTOMATA SOLUTION TO THE CA PROBLEM

5.1 Introduction

In the previous chapter a new solution to the CA problem was presented that used a continuous learning automata scheme. This solution produced superior results in terms of cost and time when compared with solutions by Marayuma/Tang [MT77] and Levi/Ersoy [LE94]. In this chapter, we present a discretized automaton version of the algorithm that improves the execution times of the continuous version while maintaining the essential quality of the final cost solutions.

5.2 Discretized Learning Automata

In the previous chapter it was shown that while FSSA are easy to implement they have a major disadvantage in that if the LA is currently choosing an action it will almost always (that is, except at a boundary state) choose the same action in the next time instant, regardless of the response given by the RE. VSSA provide an alternative approach in that they choose an action based on a distribution specified by an action probability vector. This gives the VSSA the flexibility of choosing two different actions at two consecutive time instants and allows easier analysis of the properties of the LA. However, VSSA are also limited in that the probability of choosing an action can be any real number in the interval [0, 1] which, as we shall see, results in a rate of convergence which can be improved.

97
Chapter 5: The Discrete Automata Solution to the CA Problem

Thathachar and Oommen [TO79] 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 unity asymptotically - in increasingly smaller steps.

Another benefit of discretizing the probability of choosing an action is that it reduces the requirements on the system’s random number generator. This is important since VSSA use a random number generator in their implementation. In theory, it is assumed that any real value in [0, 1] can be obtained from the machine, but in practice, only a finite number of these values are obtainable. Discrete VSSA also use integer value addition and subtraction, as opposed to real number multiplication and subtraction in the continuous versions, to keep track of the action probability vector which usually results in an increased number of iterations before the LA converges. However, the time (measured in clock cycles) taken by the microprocessor to accomplish each iteration is substantially reduced since addition/subtraction is usually quicker than multiplication. The total memory required is also reduced since the storage requirements of integers is usually less than that required for floating point numbers.
In a sense, discrete VSSA are a hybrid of FSSA and VSSA in that they consist of finite sets like FSSA, but are characterized as VSSA due to the fact that they are characterized by a probability vector that evolves with time.
5.3 The Discrete Automata Solution to CA

We now propose a discretized LA which can be used to solve the CA problem. The Discrete Automata Solution to CA (DASCA) algorithm is faster than either the MT-CA and 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 in the previous chapter.

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:

\[(C_1, C_2, ..., C_n),\]

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 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.

\[(\tau_{ij}, \sigma_{ij}, \delta_{ij})\]

where, \(\tau_{ij}\) is the discretized Increase parameter of link \(i\) with current capacity \(j\),

\(\sigma_{ij}\) the discretized Stay parameter of link \(i\) with current capacity \(j\), and

\(\delta_{ij}\) 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.

\[I_{ij} = \tau_{ij} / \text{total-steps}\]
Chapter 5: The Discrete Automata Solution to the CA Problem

\[ S_\eta = \sigma_\eta / \text{total-steps} \]

\[ D_\eta = \delta_\eta / \text{total-steps} \]

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 \(\sigma\), 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 illustrated in the examples below.

Example 5.3.1

Let us examine a discretized VSSA that has 2 possible actions, \(\{\alpha_1, \alpha_2\}\). Assume that \([0, 1]\) is subdivided into a total number of steps, total-steps. If both actions are equally likely at first, then the initial convergence vector is \([x_1, x_2]^T = [\text{total-steps}/2, \text{total-steps}/2]^T\). The updating rules for the DL_RP scheme are as follows.

\[
x_1(n+1) = x_1(n) + 1 \quad \text{if} \quad \alpha_1 \text{ is chosen and } \beta = 0
\]

\[
x_1(n+1) = x_1(n) - 1 \quad \text{if} \quad \alpha_1 \text{ is chosen and } \beta = 1
\]

\[
x_2(n+1) = x_2(n) + 1 \quad \text{if} \quad \alpha_2 \text{ is chosen and } \beta = 0
\]

\[
x_2(n+1) = x_2(n) - 1 \quad \text{if} \quad \alpha_2 \text{ is chosen and } \beta = 1
\]

Note that \(p_i(n) = x_i(n) / \text{total-steps}\) for all \(n\) and \(i = 1, 2\).

We know that for a DL_RP scheme both rewards and penalties are processed. If \(\alpha_1\) is chosen and receives a penalty then the probability of choosing this action on the next iteration, \(p_1(n+1)\), must be reduced, and \(p_2(n+1)\) must be increased. This is accomplished by subtracting a single step from the current \(x_1(n)\) and adding it to \(x_2(n)\). However, if \(\alpha_1\) is chosen and the LA receives a
reward, the probability of choosing this action on the next iteration must be increased and the probability of choosing \( \alpha_2 \) must be decreased. This, in turn, is accomplished by subtracting a single step from the current \( x_2(n) \) and adding it to \( x_1(n) \). Similar reasoning can be used for \( \alpha_2 \).

The \( \{p_i(n)\} \) is directly obtained from \( \{x_i(n)\} \).

In this example we will assume that there are 50 total steps and the current convergence vector is \( [25, 25]^T \). This means that there are four possible next states.

Case 1: LA chooses \( \alpha_1 \), RE responds with \( \beta = 0 \) \( \Rightarrow \) \( x(1) = [26, 24]^T \)

Case 2: LA chooses \( \alpha_1 \), RE responds with \( \beta = 1 \) \( \Rightarrow \) \( x(1) = [24, 26]^T \)

Case 3: LA chooses \( \alpha_2 \), RE responds with \( \beta = 0 \) \( \Rightarrow \) \( x(1) = [24, 26]^T \)

Case 4: LA chooses \( \alpha_2 \), RE responds with \( \beta = 1 \) \( \Rightarrow \) \( x(1) = [26, 24]^T \)

This type of VSSA uses a simple **Discrete Linear Reward-Penalty** (DLRP) scheme. Once again, we are trying to converge to a unit vector (where convergence to \( [50, 0] \) would correspond to \( \alpha_1 \) being the optimal action, and convergence to \( [0, 50] \) would correspond to \( \alpha_2 \) being the optimal action. As we have shown in the case of the LRP scheme, the DLRP scheme does not converge to a unit vector. As a result, this scheme is not very suitable in finding the optimal solution for stationary environments, as exists in the CA problem.

Fortunately, discretized absorbing schemes will converge to a unit vector. One such scheme is **Discrete Linear Reward-Inaction** (DLRI). While in the case of the DLRP scheme both rewards and penalties are processed, for the DLRI scheme only rewards are processed. A simple DLRI scheme is shown in the example below.
Chapter 5: The Discrete Automata Solution to the CA Problem

Example 5.3.2

We now consider a discretized VSSA that has \( r \) possible actions, \( \{\alpha_1, \ldots, \alpha_r\} \) with penalty probabilities \( \{c_1, \ldots, c_r\} \). Assume there are a total number of steps, total-steps. If these actions are equally likely at first, then the initial convergence vector is \( x(0) = [\text{total-steps}/r, \ldots, \text{total-steps}/r]^T \).

The updating rules for the DLKI scheme are as follows.

\[
\begin{align*}
    x_i(n+1) &= \text{total-steps} - \sum_{j \neq i} x_j(n+1) & \text{if} & \alpha_i \text{ is chosen and } \beta = 0 \\
    x_i(n+1) &= x_i(n) & \text{if} & \alpha_i \text{ is chosen and } \beta = 1 \\
    x_i(n+1) &= x_i(n) - 1 & \text{if} & \alpha_i \text{ is chosen and } \beta = 0 \\
    x_i(n+1) &= x_i(n) & \text{if} & \alpha_j \text{ is chosen and } \beta = 1
\end{align*}
\]

We know that only rewards are processed in this scheme. Therefore, if \( \alpha_i \) is chosen and the LA receives a reward, then the probability of choosing this action on the next iteration, \( x_i(n-1) \), must be increased. This is accomplished in two steps. First, the probability of choosing any other action \( \alpha_j \), for all \( j \neq i \), on the next iteration are reduced by setting \( x_j(n+1) \) to \( x_j(n) - 1 \) for all \( j \neq i \). Next, the probability of choosing \( \alpha_i \) on the next iteration, \( x_i(n+1) \), is increased by subtracting the sum of all \( x_j(n+1) \) for \( j \neq i \), from total-steps, the total number of steps. Finally, \( \{p(n)\} \) is obtained from \( \{x_i(n)\} \) by dividing by total-steps. There are no modifications for penalties.

As an example, if \( r = 2 \), with a total number of steps of 50, and an initial convergence vector \( [25, 25]^T \), there are four possible next states.

Case 1: LA chooses \( \alpha_1 \), RE responds with \( \beta = 0 \) \( \Rightarrow \) \( x(1) = [26, 24]^T \)

Case 2: LA chooses \( \alpha_1 \), RE responds with \( \beta = 1 \) \( \Rightarrow \) \( x(1) = [25, 25]^T \)

Case 3: LA chooses \( \alpha_2 \), RE responds with \( \beta = 0 \) \( \Rightarrow \) \( x(1) = [24, 26]^T \)

Case 4: LA chooses \( \alpha_2 \), RE responds with \( \beta = 1 \) \( \Rightarrow \) \( x(1) = [25, 25]^T \)
Chapter 5: The Discrete Automata Solution to the CA Problem

Since the chain is absorbing it can be shown that this process continues until a unit probability vector is reached. Again, as with the LRI scheme, the DLRI scheme can be shown to be $\varepsilon$-optimal and, as a result, is ideal for solving the CA problem.

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 Init 1, Init 2 and Init 3 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,

$\ell_o = \text{total-steps}/2$, $\sigma_o = \text{total-steps}/2$, $\delta_o = 0$,

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,

$\ell_n = 0$, $\sigma_n = \text{total-steps}/2$, $\delta_n = \text{total-steps}/2$

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,

$\ell_j = \text{total-steps}/3$, $\sigma_j = \text{total-steps}/3$, $\delta_j = \text{total-steps}/3$ for $0 < j < n$

It should be noted that since we will be using an absorbing scheme there should be no initial preference given to any one strategy since this might result in the automaton converging prematurely. As a result, each possible strategy is for a given link is equally likely at the start of the algorithm.
Chapter 5: The Discrete Automata Solution to the CA Problem

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 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 $\delta_{ij}$ entry in the convergence vector,
(b) if the capacity stayed the same we raise the $\sigma_{ij}$ entry in the convergence vector. and
(c) if the capacity was decreased we raise the $\delta_{ij}$ 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 $\delta_{ij}$ entry in the convergence vector,
(b) if the capacity stayed the same we raise the $\sigma_{ij}$ entry in the convergence vector. and
(c) if the capacity was decreased we raise the $\sigma_{ij}$ 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, $t_{ij}$, of the convergence vector at any point in time.

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. The operation of this scheme is illustrated in the following example.

**Example 5.3.3**

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. In what follows let \( t \) be the time index.

(i) Since the solution vector is feasible, we shall raise the \( \delta_q \) entry in the convergence vector by performing the following steps:

\[
\begin{align*}
\nu_q(t+1) &= \nu_q(t) - 1 \\
\sigma_q(t+1) &= \sigma_q(t) - 1 \\
\delta_q(t+1) &= \text{total-steps} - (\nu_q(t+1) + \sigma_q(t+1)).
\end{align*}
\]

(ii) Now, since the solution vector results in a lower network cost it implies that the stay parameter, \( \sigma_{ij} \), should be raised.

\[
\begin{align*}
\nu_q(t+1) &= \nu_q(t) - 1 \\
\delta_q(t+1) &= \delta_q(t) - 1 \\
\sigma_q(t+1) &= \text{total-steps} - (\nu_q(t+1) + \delta_q(t+1)).
\end{align*}
\]

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 have \( \sigma \), or stay, parameters that are close enough to the total number of steps.

As with the CASCA algorithm, if the test for feasibility fails then the link capacities are reset to their last lowest cost values which enables the algorithm to examine all possible configurations. The DASCA algorithm is formally given below.
Chapter 5: The Discrete Automata Solution to the CA Problem

5.3.1 The DASCA 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) total-steps - the total number of steps on the convergence scale.

Output: 
The lowest cost network capacity assignment vector.

Method

START-MAIN

For (i=1 to maxlinks) 
    For (j=1 to maxcaps) 
        If (link_i = left-boundary-state) Then \[ t_{ij} = \text{total-steps}/2, \sigma_{ij} = \text{total-steps}/2, \delta_{ij} = 0 \] 
        End-If 
        If (link_i = right-boundary-state) Then \[ t_{ij} = 0, \sigma_{ij} = \text{total-steps}/2, \delta_{ij} = \text{total-steps}/2 \] 
        End-If 
        If (link_i = internal-state) 
            Then \[ t_{ij} = \text{total-steps}/3, \sigma_{ij} = \text{total-steps}/3, \delta_{ij} = \text{total-steps}/3 \] 
        End-If 
    End-For 
End-For

Repeat For (i=1 to maxlinks) \[ C_i = \text{RAND}(1, \text{maxcap}) \] 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) ) 
    Repeat i = \text{RAND}(1, \text{maxlinks}) 
        Action = \text{RAND}(\text{Increase}_i, \text{Stay}_i, \text{Decrease}_i) 
        If (Action = \text{Increase}_i) Then \[ C_i = C_i^+ \] 
        Else If (Action = \text{Decrease}_i) Then \[ C_i = C_i^- \] End-if 
    End-if 
    Until (all links are set) 
current-cost = calculate-network-cost() 
For (i=1 to maxlinks) 
    j = C_i 
    If (network is feasible) 
        Then If (Action = \text{Increase}_i) Then \text{Raise}(\delta_{ij}, \text{total-steps}) End-If 
            If (Action = \text{Stay}_i) Then \text{Raise}(\sigma_{ij}, \text{total-steps}) End-If 
            If (Action = \text{Decrease}_i) Then \text{Raise}(\delta_{ij}, \text{total-steps}) End-If 
        Else Reset all links to best-cost capacities. 
        End-If 
End-If
Chapter 5: The Discrete Automata Solution to the CA Problem

If (network is feasible) AND (current-cost < best-cost)
    Then If (Action = Increase) Then Raise(δ_j, total-steps) End-If
        If (Action = Stay) Then Raise(σ_j, total-steps) End-If
        If (Action = Decrease) Then Raise(σ_j, total-steps) End-If
    For (i=1 to maxlinks) best-C_i = C_i End-For
    best-cost = current-cost
End-If
End-While

END-MAIN

Procedure Raise

Input: (i) Action_{ij}, either - Increase, Stay, Decrease - that was performed on the capacity link i with current capacity j.
        (ii) total-steps - the discrete learning parameter - which is the learning scheme modification parameter.
        (iii) The current convergence vector for link i with current capacity j, (t_{ij}, σ_{ij}, δ_{ij}).

Output: The modified convergence vector for link i with current capacity j, (t_{ij}, σ_{ij}, δ_{ij}).

Method

BEGIN

If (Action = Increase)
    Then δ_{ij} = δ_{ij} - 1
        σ_{ij} = σ_{ij} - 1
        t_{ij} = total-steps - (δ_{ij} + σ_{ij})
Else If (Action = Stay)
    Then t_{ij} = t_{ij} - 1
        δ_{ij} = δ_{ij} - 1
        σ_{ij} = total-steps - (t_{ij} + δ_{ij})
End-If
Else If (Action = Decrease)
    Then t_{ij} = t_{ij} - 1
        σ_{ij} = σ_{ij} - 1
        δ_{ij} = total-steps - (t_{ij} + σ_{ij})
End-If
End-If

END Procedure Raise
5.4 Experimental Results

In order to demonstrate that the new DASCA algorithm was faster than the CASCA algorithm, described in the previous chapter, and still 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 with selected values for the resolution, total-steps, which ranged from 20 to 500. Table 5.4.7 contains the best results obtained from the algorithm over the entire testing phase. A comparison with the CASCA algorithm as well as the MT and LE solutions concludes the section.

The result of each test is measured in terms of three parameters, Final Cost, Time, and Iterations. The Final Cost and Time parameters are used for comparison with the MT and LE algorithms while the Iterations parameter is used for comparison with the continuous version of the algorithm presented in the previous chapter. For example, in Table 5.4.1 for Test #1, the total number of steps was set to 20. For this test the Final Cost obtained was $7,055.60 and the algorithm took a time of 0.06 seconds to complete 34 Iterations. A comparison of the various algorithms follows after the tables.

As mentioned earlier, 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.
Figure 5.4.1: The Decrease in Cost with the Reward Parameters for the DASCA Algorithm
Figure 5.4.2: The increase in execution time with the reward parameters for the DASCA algorithm.
### Table 5.4.1 Results for the DASCA Solution to the CA Problem with 20 steps. The networks used are described in Section 3.3.

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>7055.60</td>
<td>10116.80</td>
<td>17161.00</td>
<td>54096.10</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.06</td>
<td>0.17</td>
<td>0.55</td>
<td>1.21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>34</td>
<td>66</td>
<td>176</td>
<td>238</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>5304.88</td>
<td>6940.22</td>
<td>16177.50</td>
<td>61681.60</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.22</td>
<td>0.17</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>25</td>
<td>131</td>
<td>56</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>5062.68</td>
<td>8921.54</td>
<td>18100.70</td>
<td>62585.90</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.16</td>
<td>0.27</td>
<td>0.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>50</td>
<td>55</td>
<td>86</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>5223.68</td>
<td>8292.78</td>
<td>11002.20</td>
<td>61920.40</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.05</td>
<td>0.22</td>
<td>0.39</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>37</td>
<td>86</td>
<td>117</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>6703.16</td>
<td>10462.40</td>
<td>18435.40</td>
<td>49405.10</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.06</td>
<td>0.11</td>
<td>0.22</td>
<td>1.10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>18</td>
<td>51</td>
<td>73</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>Category</td>
<td>Net 1</td>
<td>Net 2</td>
<td>Net 3</td>
<td>Net 4</td>
<td>Net 5</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>14689.90</td>
<td>10593.50</td>
<td>47378.80</td>
<td>80205.10</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.28</td>
<td>0.55</td>
<td>1.09</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>69</td>
<td>122</td>
<td>180</td>
<td>205</td>
<td>86</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>5508.60</td>
<td>7341.38</td>
<td>15152.90</td>
<td>49645.90</td>
<td>80750.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.17</td>
<td>0.66</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>79</td>
<td>79</td>
<td>198</td>
<td>178</td>
<td>91</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>10314.20</td>
<td>14762.90</td>
<td>53353.90</td>
<td>70918.90</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.06</td>
<td>0.27</td>
<td>0.28</td>
<td>0.60</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>65</td>
<td>151</td>
<td>91</td>
<td>120</td>
<td>91</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>5149.20</td>
<td>8736.38</td>
<td>13189.20</td>
<td>49014.80</td>
<td>77385.50</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.16</td>
<td>0.55</td>
<td>0.61</td>
<td>1.31</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>72</td>
<td>75</td>
<td>204</td>
<td>112</td>
<td>123</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>5020.00</td>
<td>9067.38</td>
<td>17232.40</td>
<td>44515.60</td>
<td>77460.10</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.06</td>
<td>0.33</td>
<td>0.44</td>
<td>1.10</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>64</td>
<td>146</td>
<td>159</td>
<td>218</td>
<td>82</td>
</tr>
</tbody>
</table>

Table 5.4.2 Results for the DASCA Solution to the CA Problem with 40 steps. The networks used are described in Section 3.3.
### Table 5.4.3 Results for the DASCA Solution to the CA Problem with 60 steps. The networks used are described in Section 3.3.

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7186.38</td>
<td>12217.90</td>
<td>45385.00</td>
<td>46352.30</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.38</td>
<td>0.60</td>
<td>1.81</td>
<td>1.92</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>99</td>
<td>170</td>
<td>202</td>
<td>356</td>
<td>177</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7186.38</td>
<td>14283.40</td>
<td>49106.40</td>
<td>55589.30</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.16</td>
<td>0.44</td>
<td>0.66</td>
<td>0.83</td>
<td>1.49</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>125</td>
<td>191</td>
<td>243</td>
<td>170</td>
<td>134</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4919.68</td>
<td>14236.10</td>
<td>10291.70</td>
<td>51165.20</td>
<td>39905.60</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.28</td>
<td>0.72</td>
<td>1.43</td>
<td>1.97</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>81</td>
<td>125</td>
<td>251</td>
<td>283</td>
<td>178</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4992.16</td>
<td>7448.10</td>
<td>13979.80</td>
<td>45856.20</td>
<td>55761.90</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.11</td>
<td>0.38</td>
<td>0.66</td>
<td>1.37</td>
<td>2.14</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>98</td>
<td>196</td>
<td>226</td>
<td>274</td>
<td>203</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4919.68</td>
<td>8815.26</td>
<td>10637.60</td>
<td>45877.10</td>
<td>61552.01</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.17</td>
<td>0.44</td>
<td>0.99</td>
<td>1.10</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>118</td>
<td>201</td>
<td>324</td>
<td>229</td>
<td>169</td>
</tr>
</tbody>
</table>
## Table 5.4.4 Results for the DASCA Solution to the CA Problem with 80 steps. The
networks used are described in Section 3.3.
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7216.54</td>
<td>10132.10</td>
<td>41888.40</td>
<td>37781.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.50</td>
<td>1.48</td>
<td>3.07</td>
<td>2.69</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>188</td>
<td>215</td>
<td>399</td>
<td>596</td>
<td>254</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7265.26</td>
<td>10733.40</td>
<td>46752.60</td>
<td>38646.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.66</td>
<td>1.10</td>
<td>2.04</td>
<td>3.13</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>152</td>
<td>312</td>
<td>330</td>
<td>400</td>
<td>293</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7186.38</td>
<td>10507.50</td>
<td>44515.60</td>
<td>38870.10</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.66</td>
<td>1.53</td>
<td>1.97</td>
<td>4.18</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>171</td>
<td>297</td>
<td>506</td>
<td>392</td>
<td>386</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4919.68</td>
<td>7186.38</td>
<td>11489.30</td>
<td>45720.10</td>
<td>37750.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.66</td>
<td>1.54</td>
<td>2.64</td>
<td>2.75</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>134</td>
<td>306</td>
<td>519</td>
<td>516</td>
<td>257</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7186.38</td>
<td>10359.00</td>
<td>41888.40</td>
<td>47751.70</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.55</td>
<td>1.71</td>
<td>2.97</td>
<td>2.36</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>170</td>
<td>294</td>
<td>637</td>
<td>574</td>
<td>217</td>
</tr>
</tbody>
</table>

Table 5.4.5 Results for the DASCA Solution to the CA Problem with 100 steps. The networks used are described in Section 3.3.
### Chapter 5: The Discrete Automata Solution to the CA Problem

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>10636.00</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.93</td>
<td>3.90</td>
<td>6.05</td>
<td>12.25</td>
<td>16.97</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>753</td>
<td>2100</td>
<td>2151</td>
<td>2417</td>
<td>1674</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7186.38</td>
<td>10479.70</td>
<td>45385.00</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.99</td>
<td>4.01</td>
<td>6.48</td>
<td>13.73</td>
<td>15.22</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>747</td>
<td>1922</td>
<td>2141</td>
<td>2730</td>
<td>1484</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6937.90</td>
<td>9967.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.94</td>
<td>5.61</td>
<td>8.51</td>
<td>14.06</td>
<td>15.27</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>766</td>
<td>3033</td>
<td>3034</td>
<td>2795</td>
<td>1524</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6968.06</td>
<td>9909.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.87</td>
<td>5.72</td>
<td>9.17</td>
<td>12.69</td>
<td>15.16</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>712</td>
<td>3072</td>
<td>3318</td>
<td>2512</td>
<td>1524</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>7214.22</td>
<td>10489.00</td>
<td>45385.00</td>
<td>37307.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.99</td>
<td>3.13</td>
<td>11.70</td>
<td>12.85</td>
<td>16.25</td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>809</td>
<td>1514</td>
<td>4189</td>
<td>2543</td>
<td>1671</td>
</tr>
</tbody>
</table>

Table 5.4.6 Results for the DASCA Solution to the CA Problem with 500 steps. The networks used are described in Section 3.3.
Table 5.4.7 Best Results for the DASCA Solution to the CA Problem using the networks described in Section 3.3.

The results displayed in the tables demonstrate that the DASCA solution to the CA problem produces superior results when compared with both the MT and LE solutions and also proves to be faster than the CASCA algorithm. 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 DASCA algorithm also has consistently lower execution times in most cases when compared to either of the previous solutions as well as the CASCA algorithm.

If we examine the best results obtained for the DASCA algorithm the results clearly demonstrate that the discrete scheme is faster than the continuous scheme. 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 $37,307.40 which took 4.12 seconds. The DASCA algorithm produces the same final cost value of $37,307.40, but takes only 2.51 seconds.
Chapter 5: The Discrete Automata Solution to the CA Problem

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 for his particular requirements.

5.5 Conclusion

In this chapter we have presented a solution to the CA problem that uses discrete Learning Automata (LA). Section 5.2 introduced discretized LA and discussed the advantages that these schemes have over the continuous scheme. Section 5.3 presented the Discrete Automata (DASCA) solution to the CA problem and described the various structures and mechanisms that are used by the algorithm to generate the final cost solution. The DASCA algorithm is formally given in Section 5.4 and a sample of results obtained with various parameter values are given in Section 5.5. These results prove that not only is the DASCA algorithm faster than the CASCA solution, but also maintains the CASCA's superior cost results when compared with the MT and LE solutions.
CHAPTER 6

THE CA PROBLEM WITH PRIORITY ASSIGNMENTS

6.1 Introduction

In general, a substantial reduction in network cost can be achieved by distinguishing delay requirements among different classes of packets. Each class of packet, can be given a different level of service based on some criteria, for example delay requirements, according to a certain priority scheme. These priorities are usually given as input to the system which then computes the best capacity for the links in order to satisfy the specified scheme. A further reduction in cost may be achieved if the priorities can be assigned by the algorithm, rather than be predetermined, during the course of its execution. This enables the algorithm to search for the best overall combination of link capacities and packet priorities and, in so doing, achieve a improved lower total network cost. This is called the Capacity Assignment with Priority Assignments (CAPA) problem.
6.2 Assumptions, Delay Formulae and Experimental Test Bench

We shall use the same assumptions and delay formulae as we used for the original CA problem as stated in Chapter 3. We shall also use the same test bench we used for the original CA problem as stated in Chapter 3, with the following addition. For all networks, each packet class will have average path lengths assigned according to the table below. This data was used by Marayuma and Tang in their solution to the problem.

<table>
<thead>
<tr>
<th>PACKET CLASS</th>
<th>AVERAGE PATH LENGTH (Km)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.621</td>
</tr>
<tr>
<td>2</td>
<td>3.455</td>
</tr>
<tr>
<td>3</td>
<td>5.395</td>
</tr>
<tr>
<td>4</td>
<td>7.143</td>
</tr>
<tr>
<td>5</td>
<td>9.743</td>
</tr>
</tbody>
</table>

Table 6.2.1 Average Packet Path Lengths used for packets traversing the networks described in Section 3.3
6.3 Problem Complexity

Unless a specific priority assignment for the packet classes is given, the priority assignment itself can become part of the network design problem. There is no guaranteed method for determining an optimal priority assignment without exhaustively solving the capacity assignment problem for all possible priority assignments. This is explained further below.

If it is assumed that \( n \) is the number of packet classes that use the network and \( N(n) \) is the total number of different priority assignments on \( n \) packet classes, then it is possible to state the following.

A brief insight by Marayuma/Tang [MT77] into the complexity of the CAPA problem is not out of place. Let \( \binom{n}{r} \) denote the number of different ways to partition \( n \) different items into \( r \) non-empty groups. This number is often referred to as the Stirling number which satisfies the following recurrence equation.

\[
\binom{n}{r} = \binom{n-1}{r-1} + r \binom{n-1}{r}
\]

where \( \binom{1}{1} = \binom{n}{n} = 1 \)

For each partition there are \( r! \) ways to order groups and to assign numbers which correspond to distinct priority levels. Therefore, there are \( r! \binom{n}{r} \) different ways to map \( n \) packet classes into \( r \) priority levels. The total number of different priority assignments \( N(n) \) on \( n \) packet classes can thus be given as the following.
\[ N(n) = \sum_{r=1}^{n} r! \left( \begin{array}{c} n \\ r \end{array} \right) \]

The different number of priority assignments \( N(n) \) grows very fast as the number of packet classes, \( n \), increases. For example, \( N(2) = 3 \), \( N(3) = 14 \), \( N(4) = 75 \), \( N(5) = 541 \), \( N(6) = 4683 \), and so on. As it is clearly not practical to solve this problem using their solution to the CA problem \( N(n) \) times, Marayuma-Tang tried to find a way to reduce the number of priority assignments examined. They produced a heuristic procedure that consists of two steps. The first step is to order the packet classes according to values of priority preference, and the second is to find a good partition for such a sequence.
6.4 The Marayuma-Tang Solution to the CAPA Problem

Marayuma and Tang [MT77] extended their investigation of the CA problem, discussed in Chapter 3, to include the problem of priority assignment on different classes of packets in order to achieve further reduction in network cost. Specifically, they addressed the problem of mapping classes of packets into priority levels. Although it is possible to consider a different priority assignment on classes of packet at each node, such priority assignments are not desirable from the systems point of view and are therefore not considered in the solution. The Marayuma/Tang algorithm (MT-CAPA) assigns sub-optimal priorities on classes of packets based on parameters such as delay requirement, path length, packet rate, and packet length.

6.4.1 Priority Preference Among Classes - The H4 Heuristic

The Marayuma/Tang solution to the problem is based on the combination of several heuristic routines of their solution to the CA problem along with a new heuristic for determining an initial priority preference among a set of packet classes.

The method begins with the idea to develop a figure of merit that can be used to order the packet classes into a sequence of non-increasing priority ‘preference’ values. Such a sequence can be denoted by $K = \{k_1, k_2, \ldots, k_n, k_{n-1}, \ldots, k_1\}$ where $k_i$ denotes a packet class whose priority preference is the $i^{th}$ among $n$ packet classes. The priority preference sequence implies that it is preferable to assign a higher or equal priority to a packet class $k_i$ than to packet class $k_j$ where $1 \leq i \leq n$, denoted by the following expression:
Chapter 6: The CA Problem with Priority Assignments

\[ P_{ty}(k_i) \geq P_{ty}(k_j) \quad \text{if} \quad 1 \leq i < j \leq n \]

Let the set of parameters that defines a packet class \( k \) be denoted as,

\[ (B_k, l_k, \mu_k, \gamma_k) \]

where, \( B_k \) is the average delay bound,

\( l_k \) is the average path length, and

\( 1/\mu_k \) is the average packet length

\( \gamma_k \) is the average rate of packet class \( k \) entering the network

These four parameters are considered in the figure of merit to assign the priority preference values. If there exists two packet classes, \( k \) and \( h \), the following heuristics are intuitively reasonable.

**H1:** if \( B_k < B_h \) then \( P_{ty}(k) \geq P_{ty}(h) \).

Thus, if packet class \( k \) has a lower average delay bound than packet class \( h \) then the priority of packet class \( k \) should be greater than, or equal to, the priority of packet class \( h \).

**H2:** if \( l_k > l_h \) then \( P_{ty}(k) \geq P_{ty}(h) \).

In this case if packet class \( k \) has a greater average path length than packet class \( h \) then the priority of packet class \( k \) should be greater than, or equal to, the priority of packet class \( h \).

To accommodate cases where both delay bounds and average path lengths are different, the following two heuristics are derived.

**H3:** if \( B_k/l_k < B_h/l_h \) then \( P_{ty}(k) \geq P_{ty}(h) \),

**H4:** if \( (B_k - Z_k)/l_k < (B_h - Z_h)/l_h \) then \( P_{ty}(k) \geq P_{ty}(h) \).

Note that \( Z_k \) and \( Z_h \) are the average delays of packet classes \( k \) and \( h \) computed under
equal priority assignment.

In examining the performance between H3 and H4 multiple examples were considered
where parameters such as delay bounds and average path lengths were the same among packet
classes but with differing average packet lengths or traffic rates. It was observed that H4 gives
performance better than, or equal to, H3. The heuristic H4 is given below.

Procedure H4

Input: Packet class characteristics.
Output: Packet class priority preference sequence.

Method

BEGIN-MAIN

For (k=0 to num-packets)
  For (h=0 to num-packets)
    If ((B_k - Z_k)/l_k < (B_h - Z_h)/l_h)
      Then Pty(k) ≥ Pty(h)
  End-If
End-For
End-For

END-MAIN

//Procedure H4

6.4.2 Determination Of The Number Of Priority Classes

Once a priority preference scheme has been established using heuristic H4 the next step
is to determine the optimal number of priority partitions. It is clear that an ordered sequence of n
elements has $2^{n-1}$ different partitions. It is impractical to examine all of these partitions to find the
optimal priority assignment so a sequential partition strategy was established which examines at most \( n(n - 1)/2 + 1 \) different priority assignments. This means that the algorithm solves up to \( n(n - 1)/2 + 1 \) problems using the Marayuma/Tang solution to the CA problem in order to find a good priority assignment on \( n \) packet classes. This strategy is described below.

Let \( K \) be the priority preference sequence of \( n \) classes of packets where \( K = (k_1, k_2, \ldots, k_n) \) such that \( P_{ty}(k_i) \geq P_{ty}(k_j) \) for all \( 1 \leq i < j \leq n \). At the initial stage \( K^{(1)} = K \) and \( D^{(1)} \) is equal to the network cost with equal priority assignment. At the \( r^{th} \) stage \( K^{(r)} = \{K_1^{(r)}, \ldots, K_j^{(r)}, \ldots, K_r^{(r)}\} \) where \( D^{(r)} \) is equal to the network cost when the packet classes in \( K^{(r)} \) are assigned to priority level \( j \) for all \( j \), and \( D^{(1)} > D^{(2)} > \ldots > D^{(r)} \).

We now specify how \( K^{(r-1)} \) can be computed from \( K^{(r)} \). If the minimum network cost \( D^{(r-1)} \) can be achieved by partitioning \( K_j^{(r)} \) into two sub-partitions, \( K_{j_1}^{(r)} \) and \( K_{j_2}^{(r)} \) and if \( D^{(r-1)} < D^{(r)} \), \( K^{(r-1)} \) is assigned according to the following rules:

\[
K_i^{(r-1)} = K_i^{(r)}, \quad \text{for } 1 \leq i \leq j-1 \\
K_j^{(r-1)} = K_{j_1}^{(r)}, \\
K_{j-1}^{(r-1)} = K_{j_2}^{(r)}, \text{ and} \\
K_{j+1}^{(r-1)} = K_{j}^{(r)}, \quad \text{for } j+1 \leq i \leq n
\]

Otherwise the procedure terminates with the best partition as \( K^{(r)} \) with \( r \) priority levels and with cost \( D^{(r)} \). This is the philosophy behind the MT-CAP algorithm which is given formally below.
6.4.3 The MT-CAPA Algorithm

**Input:** The network characteristics and packet types.

**Output:** The lowest cost network capacity assignment vector and best priority level configuration.

**Assumption:** Original MT-CA algorithm available and invoked by MT-CA(priority-level-partition).

**Method**

```
START-MAIN //Priority Assignment - Marayuma/Tang Solution

r = 1
D = MT-CA(equal-lowest-priority-assignment)
D = D(0)
K = H(1)
K = K(0)
stop = False
While (stop = False)
    r = r + 1;
    If (r > n) Then stop = True
    Else Find new partition of K(r)
    D(r) = MT-CA(K(r))
    If (D > D(r)) Then stop = True
    Else D = D(r)
    K = K(r)
End-If
End-If
End-While

END-MAIN //Priority Assignment - Marayuma/Tang Solution
```
6.4.4 Experimental Results

The MT-CAP algorithm was tested using the same technique that was used for the original MT-CA algorithm. The algorithm was executed five consecutive times for each network. The results obtained are given in the table below.

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4731.68</td>
<td>6275.70</td>
<td>9346.07</td>
<td>25983.10</td>
<td>37491.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>183.01</td>
<td>409.36</td>
<td>1816.71</td>
<td>388.62</td>
<td>4197.74</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4731.68</td>
<td>6275.70</td>
<td>9346.07</td>
<td>25983.10</td>
<td>37491.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>184.14</td>
<td>411.32</td>
<td>1820.81</td>
<td>390.63</td>
<td>4199.84</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4731.68</td>
<td>6275.70</td>
<td>9346.07</td>
<td>25983.10</td>
<td>37491.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>184.78</td>
<td>411.09</td>
<td>1812.59</td>
<td>388.51</td>
<td>4199.03</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4731.68</td>
<td>6275.70</td>
<td>9346.07</td>
<td>25983.10</td>
<td>37491.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>183.99</td>
<td>411.48</td>
<td>1815.69</td>
<td>392.57</td>
<td>4192.77</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4731.68</td>
<td>6275.70</td>
<td>9346.07</td>
<td>25983.10</td>
<td>37491.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>183.38</td>
<td>409.09</td>
<td>1814.53</td>
<td>399.56</td>
<td>4197.89</td>
</tr>
</tbody>
</table>

Table 6.4.4.1 Results for the MT-CAPA Algorithm when used to solve the CAPA problem.

The networks used are described in Section 3.3 and Section 6.2.
These results prove that the MT-CAPA algorithm generally produces superior final cost results when compared with the original MT-CA algorithm. Consider the tests for Network #5. The MT-CA algorithm produced a final cost of $43,341.90 whereas the MT-CAPA algorithm produced a final cost of $41,342.40. However, the MT-CAPA algorithm does not always find the lowest possible final cost which is illustrated by the results for Network #2. The MT-CAPA algorithm produced a final cost of $15,786.10 whereas the MT-CA algorithm produced a final cost of $12,686.10. This is a direct result of the nature of the testing process in which several different priority level configurations are examined, as opposed to just one, each of which is a new CA problem. Since all possible priority configurations cannot be tested, some lower cost configurations, that originate from higher cost configurations, may not be found. Naturally, the MT-CAPA algorithm has longer execution times than the MT-CA for all cases since, as before, due to the nature of the testing process many different CA problems have to be solved consecutively. For example, the MT-CAPA algorithm takes 4192.77 seconds when Network #5 is used whereas the MT-CA algorithm takes only 2.14 seconds.

In the following sections we will present two new algorithms that solve the CAPA problem using Learning Automata in a similar fashion to the methods presented in Chapter 4 and Chapter 5. These algorithms not only produce superior results when compared with the MT-CAPA algorithm but also have substantially lower execution times.
6.5 The Continuous Automata Solution to the CAPA Problem

If we examine the MT-CAPA solution algorithm, it is clear that it is built around the original MT-CA algorithm solution for the CA problem. The additional framework is used for deciding on the number of priority levels that is used for a given test and at what point the algorithm is stopped while the actual test solutions are found using the MT-CA algorithm. In the same manner, the Continuous Automata Solution to CA with Priority Assignment (CASCAPA) solution, that uses continuous LA, will be based on the original CASCA solution to the CA problem. An additional computational layer is added in the CASCAPA algorithm which is instrumental in deciding on the number of priority levels that is used in a given test and the point at which the algorithm is terminated. However, every solution is found using the original CASCA algorithm.

A new vector is needed that is analogous to the probability vector associated with the link capacity assignment vector. The new vector is a priority level probability vector that contains an entry for each of the different possible priority levels. For example, if the maximum number of priority levels for a given network is 3, then the priority level probability vector is composed of three quantities:

(i) The probability that the optimal number of priority levels is one. This means that packets can only have a priority level of 1.

(ii) The probability that the optimal number of priority levels is two. This means that packets can have a priority level of 1 or 2.

(iii) The probability that the optimal number of priority levels is three. This means
that packets can have a priority level of 1, 2, or 3.

For \( r \) priority levels, the initial priority preference vector is given as \([1/r, \ldots, 1/r]\). In the example of three priority levels the initial priority preference vector is given as \([0.333, 0.333, 0.333]\). The sum of the probability quantities of the vector must always be unity and therefore the optimal number of priority levels is defined as the entry that has the highest value. This means that if the highest value in the vector indicates that the optimal number of priority levels is two then packets can either be assigned a priority of 1 or 2 in the solution. The maximum number of priority levels is obviously limited to being lower than, or equal to, the maximum number of packet classes.

We must also keep track of the best link capacity assignment for each number of priority levels. Consequently, if there are three priority levels we must maintain optimal link capacity assignments for three priority level configurations, as explained in the discussion above.

The next question we address is that of determining the degree by which the probability vectors are modified. These are done in terms of two user defined quantities \( \lambda_{R3} \) and \( \lambda_{R4} \). The first, \( \lambda_{R3} \), is the reward parameter to be used when a feasible solution is reached with the best number of priority levels, and the second, \( \lambda_{R4} \), is used when the solution also has a lower cost or has an equivalent or lower cost with an increase in the number of priority levels. As in the case of the CASCA algorithm, the closer these values are to unity, the more accurate the solution with the rate of convergence to the optimal probability vector decreasing with the parameters. Similarly, the closer the parameters are to unity, the slower the algorithms are.
Chapter 6: The CA Problem with Priority Assignments

Assume the priority level probability vector is given as \((p_1, ..., p_n, ..., p_n)\) where \(n\) is the maximum number of priority levels. The vector is modified in two cases.

Case 1: The new solution is feasible and additionally, the cost is lower than the previous best cost, or the cost is lower than, or equal to, the previous best cost and the number of priority levels has been increased. If the present number of priority levels is \(i\), then,

(i) \(p_j(n+1) = p(n)_j \cdot \lambda_{R4}\) where \(j \neq i\), and,

(ii) \(p_i(n+1) = 1 - \sum_{j \neq i} p_j(n+1)\).

Case 2: The solution is feasible and the number of priority levels is equal to the best number of priority levels. In this case,

(i) \(p_j(n+1) = p_j(n) \cdot \lambda_{R3}\) where \(j \neq i\), and,

(ii) \(p_i(n+1) = 1 - \sum_{j \neq i} p_j(n+1)\)

The operation of this scheme is illustrated in the example below.
Chapter 6: The CA Problem with Priority Assignments

Example 6.5.1

Let us assume that there are three possible priority levels for the packets. This means that the initial priority preference vector at time n, \( [p_1(n), p_2(n), p_3(n)] \), is given as \([0.333, 0.333, 0.333]\).

Assume that we have reward parameters of \( \lambda_{R3} = 0.9 \) and \( \lambda_{R4} = 0.9 \) and that the solution under investigation, which has three priority levels, is feasible, has a lower cost and has the best number of priority levels. The new priority preference vector will be produced as follows:

\[
\begin{align*}
\text{Step 1:} & \quad p_1(n+1) = 0.333 \times 0.9 = 0.2997, \\
p_2(n+1) = 0.333 \times 0.9 = 0.2997, \\
p_3(n+1) = 1.0 - (0.2997 + 0.2997) = 0.4006. \\
\text{Step 2:} & \quad p_1(n+1) = 0.2997 \times 0.9 = 0.26973, \\
p_2(n+1) = 0.2997 \times 0.9 = 0.26973, \\
p_3(n+1) = 1.0 - (0.2997 + 0.2997) = 0.46054.
\end{align*}
\]

The new priority preference vector is given as \([0.26973, 0.26973, 0.46054]\).

The CASCAPA algorithm is presented formally in the following section.
6.5.1 The CASCAPA 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 $\lambda_{R1}$, $\lambda_{R2}$, $\lambda_{R3}$, $\lambda_{R4}$.

Output: The lowest cost capacity assignment vector and best priority level configuration.

Assumption: The original CASCA algorithm is available and is invoked by CASCA($\lambda_{R1}$, $\lambda_{R2}$).

Method

START-MAIN //Priority Assignment Using LA - Continuous Version

Repeat  
(i) For (i=1 to maxlinks) $C_i = \text{RAND}(1, \text{maxcap})$ End-For
Until (network is feasible)
best-$C_i = C_i$
While ( (num-inters < max-itors) AND (no priority-prob > req-priority-accuracy) )  
(i) For (i=1 to maxpriorities) num-levels = MAX(Pr[i-priority-levels]) End-For
(ii) For (i=1 to maxpackets) packet.priority = RAND(1, num-levels) End-For
(iii) For (i=1 to maxlinks) $C_i = \text{MAX}(Pr[\text{Stay}], \text{capacity for num-levels})$ End-For
(iv) For (i=1 to maxlinks) $C_i = \text{best-$C_i$, found using CASCA($\lambda_{R1}$, $\lambda_{R2}$)}$ End-For
current-cost = lowest-cost found using CASCA($\lambda_{R1}$, $\lambda_{R2}$)  
If ( (network is feasible)
AND ( (current-cost < best-cost)  
OR ((current-cost \leq best-cost) AND (num-levels > best-num-levels))))
Then Raise-Priority(num-levels, $\lambda_{R4}$)
best-cost = current-cost
best-num-levels = num-levels
For (i=1 to maxlinks) best-$C_i = C_i$ End-For
End-If
If (network is feasible) AND (num-levels = best-num-levels)
Then Raise-Priority(num-levels, $\lambda_{R3}$)
End-If
End-While

END-MAIN //Priority Assignment Using LA - Continuous Version
Procedure Raise-Priority

Input: (i) num-levels
      (ii) $\lambda_R$, the corresponding reward parameter.
Output: The modified priority level probability vector for link i.

Method

BEGIN-MAIN

  total = 0
  For (i=0 to total-prior-levels)
    If (i = num-levels)
      Then prior-level$_i$ = prior-level$_i$ * $\lambda_R$
          total = total + prior-level$_i$
    End-If
  End-For
  prior-level$_{num-levels}$ = 1 - total

END-MAIN //Procedure Raise-Priority
6.5.2 Experimental Results

An extensive range of tests were performed on the suite of same networks in order to ensure that the CASCAPA algorithms performance surpassed that of the MT-CAPA algorithm. The results for the CASCAPA algorithm are presented in a similar format as we used for the CASCA results with a set of tables consisting of the Final Cost values and Execution Times. The Iterations field has been eliminated for this series of tests. The main difference is that the two new parameters, \( \lambda_3 \) and \( \lambda_4 \), that are used by the priority level probability vector have been added which ranged in value from 0.7 to 0.9. Therefore, in Table 6.5.2.1 we have \( \lambda_1 = 0.8 \) and \( \lambda_2 = 0.8 \) which are used for the probability vector in the CASCA algorithm, and the new parameters, namely \( \lambda_3 = 0.8, \lambda_4 = 0.7 \), which are used by the priority level probability vector in the CASCAPA. The section concludes with a comparison of the MT-CAPA and CASCAPA algorithms.
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6343.14</td>
<td>9379.07</td>
<td>9545.07</td>
<td>51873.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.10</td>
<td>1.43</td>
<td>1.92</td>
<td>25.70</td>
<td>20.21</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6788.30</td>
<td>9638.11</td>
<td>22203.70</td>
<td>47765.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.82</td>
<td>1.26</td>
<td>2.20</td>
<td>17.42</td>
<td>18.84</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6324.70</td>
<td>12254.90</td>
<td>12802.70</td>
<td>66883.30</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.99</td>
<td>4.61</td>
<td>2.97</td>
<td>10.82</td>
<td>7.14</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4762.68</td>
<td>9723.701</td>
<td>10498.40</td>
<td>16927.50</td>
<td>48736.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.33</td>
<td>26</td>
<td>2.52</td>
<td>41.31</td>
<td>36.36</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4756.68</td>
<td>6463.70</td>
<td>9346.07</td>
<td>10648.70</td>
<td>52166.20</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.49</td>
<td>2.58</td>
<td>2.41</td>
<td>31.20</td>
<td>13.84</td>
</tr>
</tbody>
</table>

Table 6.5.2.1 Results for CASCAPA Solution the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.8$, $\lambda_{R2} = 0.8$, $\lambda_{R3} = 0.8$, $\lambda_{R4} = 0.7$. 
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9813.15</td>
<td>53837.10</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.26</td>
<td>3.90</td>
<td>3.57</td>
<td>43.17</td>
<td>17.96</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6373.30</td>
<td>9863.71</td>
<td>10321.00</td>
<td>54352.30</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.99</td>
<td>2.69</td>
<td>2.75</td>
<td>35.15</td>
<td>20.38</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>10662.70</td>
<td>11238.60</td>
<td>47839.50</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.82</td>
<td>5.55</td>
<td>3.51</td>
<td>12.53</td>
<td>26.03</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4731.68</td>
<td>6670.14</td>
<td>9358.07</td>
<td>11504.60</td>
<td>38197.70</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.37</td>
<td>5.33</td>
<td>10.60</td>
<td>8.07</td>
<td>34.94</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4743.68</td>
<td>6145.70</td>
<td>9371.07</td>
<td>9216.07</td>
<td>44093.60</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.71</td>
<td>4.34</td>
<td>8.62</td>
<td>12.85</td>
<td>36.53</td>
</tr>
</tbody>
</table>

Table 6.5.2.2 Results for CASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.9$ $\lambda_{R2} = 0.8$ $\lambda_{R3} = 0.8$ $\lambda_{R4} = 0.7$. 
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9641.07</td>
<td>9385.67</td>
<td>39115.10</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.36</td>
<td>4.56</td>
<td>5.61</td>
<td>23.29</td>
<td>49.54</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9371.07</td>
<td>9216.70</td>
<td>38775.90</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.48</td>
<td>2.80</td>
<td>7.14</td>
<td>15.77</td>
<td>13.24</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6287.70</td>
<td>9216.07</td>
<td>9216.70</td>
<td>39375.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.71</td>
<td>3.40</td>
<td>3.02</td>
<td>13.29</td>
<td>35.43</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.70</td>
<td>40813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.43</td>
<td>3.57</td>
<td>3.35</td>
<td>6.92</td>
<td>27.69</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9702.07</td>
<td>37219.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.02</td>
<td>3.08</td>
<td>5.17</td>
<td>32.79</td>
<td>92.33</td>
</tr>
</tbody>
</table>

Table 6.5.2.3 Results for CASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.9$ $\lambda_{R2} = 0.9$ $\lambda_{R3} = 0.8$ $\lambda_{R4} = 0.7$. 
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9617.23</td>
<td>37104.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>4.99</td>
<td>7.41</td>
<td>9.50</td>
<td>35.20</td>
<td>44.21</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.708</td>
<td>9216.07</td>
<td>9647.39</td>
<td>36974.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.46</td>
<td>85</td>
<td>11.31</td>
<td>32.96</td>
<td>34.43</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9369.07</td>
<td>37327.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>4.77</td>
<td>11.26</td>
<td>10.33</td>
<td>51.19</td>
<td>46.30</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9615.67</td>
<td>37097.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.42</td>
<td>5.99</td>
<td>8.89</td>
<td>14.45</td>
<td>26.92</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9371.07</td>
<td>37777.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.59</td>
<td>7.42</td>
<td>14.39</td>
<td>82.28</td>
<td>37.51</td>
</tr>
</tbody>
</table>

Table 6.5.2.4 Results for CASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95$ $\lambda_{R2} = 0.9$ $\lambda_{R3} = 0.8$ $\lambda_{R4} = 0.7$. 
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9363.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.64</td>
<td>5.05</td>
<td>14.01</td>
<td>49.37</td>
<td>55.20</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.69</td>
<td>4.83</td>
<td>10.88</td>
<td>30.21</td>
<td>176.53</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9957.71</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.02</td>
<td>5.11</td>
<td>15.77</td>
<td>48.33</td>
<td>62.50</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.59</td>
<td>6.37</td>
<td>16.65</td>
<td>27.69</td>
<td>69.59</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.52</td>
<td>6.81</td>
<td>8.74</td>
<td>50.14</td>
<td>43.11</td>
</tr>
</tbody>
</table>

Table 6.5.2.5 Results for CASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95$ $\lambda_{R2} = 0.95$ $\lambda_{R3} = 0.8$ $\lambda_{R4} = 0.7$. 
### Table 6.5.2.6 Results for CASCAPA Solution for the CAPA Problem

The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.9$ $\lambda_{R2} = 0.8$ $\lambda_{R3} = 0.9$ $\lambda_{R4} = 0.8$.

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6463.70</td>
<td>9388.07</td>
<td>9216.07</td>
<td>40802.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>5.00</td>
<td>5.28</td>
<td>8.89</td>
<td>41.85</td>
<td>64.43</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9510.07</td>
<td>9747.83</td>
<td>39813.20</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.81</td>
<td>4.23</td>
<td>8.46</td>
<td>30.10</td>
<td>110.18</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9439.03</td>
<td>9216.07</td>
<td>41067.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.40</td>
<td>11.53</td>
<td>9.72</td>
<td>22.19</td>
<td>120.24</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9938.27</td>
<td>39223.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.80</td>
<td>7.08</td>
<td>12.14</td>
<td>47.40</td>
<td>85.24</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6287.70</td>
<td>9216.07</td>
<td>9539.07</td>
<td>39813.80</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>6.65</td>
<td>6.59</td>
<td>12.08</td>
<td>55.44</td>
<td>98.05</td>
</tr>
<tr>
<td>Test</td>
<td>Category</td>
<td>Net 1</td>
<td>Net 2</td>
<td>Net 3</td>
<td>Net 4</td>
<td>Net 5</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>-------</td>
<td>-------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td>Cost ($</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9358.07</td>
<td>9358.07</td>
<td>36968.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.57</td>
<td>14.83</td>
<td>10.55</td>
<td>58.77</td>
<td>35.81</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36989.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>5.99</td>
<td>8.84</td>
<td>11.70</td>
<td>86.89</td>
<td>124.41</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37240.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.53</td>
<td>11.80</td>
<td>17.74</td>
<td>18.23</td>
<td>72.50</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37474.00</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.53</td>
<td>13.52</td>
<td>12.64</td>
<td>69.59</td>
<td>75.96</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>39591.10</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>5.49</td>
<td>21.59</td>
<td>11.97</td>
<td>32.57</td>
<td>82.28</td>
</tr>
</tbody>
</table>

Table 6.5.2.7 Results for CASCAPA Solution for the CA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are \( \lambda_{R1} = 0.9 \), \( \lambda_{R2} = 0.9 \), \( \lambda_{R3} = 0.9 \), \( \lambda_{R4} = 0.8 \).
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36974.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>9.17</td>
<td>12.91</td>
<td>18.73</td>
<td>105.02</td>
<td>77.72</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>7.52</td>
<td>20.54</td>
<td>27.68</td>
<td>46.90</td>
<td>141.18</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>8.52</td>
<td>10.21</td>
<td>36.91</td>
<td>72.50</td>
<td>133.69</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>4.72</td>
<td>16.20</td>
<td>26.75</td>
<td>128.25</td>
<td>72.06</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>11.59</td>
<td>13.24</td>
<td>39.71</td>
<td>61.90</td>
<td>78.76</td>
</tr>
</tbody>
</table>

Table 6.5.2.8 Results for CASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95$, $\lambda_{R2} = 0.9$, $\lambda_{R3} = 0.9$, $\lambda_{R4} = 0.8$. 
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>9.55</td>
<td>26.96</td>
<td>51.03</td>
<td>79.75</td>
<td>178.95</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>8.57</td>
<td>48.12</td>
<td>16.04</td>
<td>98.48</td>
<td>176.09</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>8.18</td>
<td>26.48</td>
<td>31.36</td>
<td>67.39</td>
<td>115.51</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>6.21</td>
<td>17.74</td>
<td>41.08</td>
<td>45.91</td>
<td>105.45</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>4.94</td>
<td>21.75</td>
<td>29.00</td>
<td>64.37</td>
<td>56.63</td>
</tr>
</tbody>
</table>

Table 6.5.2.9 Results for CASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are $\lambda_{R1} = 0.95$, $\lambda_{R2} = 0.95$, $\lambda_{R3} = 0.9$, $\lambda_{R4} = 0.8$. 
The results displayed in the tables demonstrate the overwhelming superiority of the CASCAPA algorithm over the MT-CAPA algorithm. In nearly every test the CASCAPA algorithm produces a Final Cost result that is superior to that produced by the MT-CAPA irregardless of the level of the parameters. For example, even with $\lambda_{R1} = 0.9$, $\lambda_{R2} = 0.8$, $\lambda_{R3} = 0.8$ and $\lambda_{R4} = 0.7$ we obtain at least one superior cost result for each network tested, the only exception being network #5. However, as expected, consistently superior results are obtained the closer the parameters are to unity. So, for example, with $\lambda_{R1} = 0.95$, $\lambda_{R2} = 0.95$, $\lambda_{R3} = 0.9$ and $\lambda_{R4} = 0.8$ the CASCAPA algorithm produces a superior cost result for every test. Another startling result is that the execution times of the CASCAPA are vastly superior to those obtained for the MT-CAPA algorithm. For each of the networks tested there is a set of parameter settings that will produce superior cost results in a substantially shorter time interval. For example, in the case of Network #1 the MT-CAPA algorithm produces a Final Cost of $4,731.68$ in $183.01$ seconds in its best test, while the CASCAPA algorithm produces a Final Cost of $4,601.68$ in $9.55$ seconds in its worst case. Also, this difference in execution times increases with the size of the network. Notice that the execution times for Network #5 range from $4192.77$ seconds to $4199.84$ seconds while the execution times for the CASCAPA algorithm range from $26.92$ seconds to $178.95$ seconds to produce a superior result. These results are easily explained since, although the CAPA problem requires that the algorithm solve the CA problem many times, we know that the original CASCA algorithm gives superior cost and execution time results when compared with the MT-CA algorithm. This also accounts for the fact that the execution times are increased when compared with the CASCA algorithm.
Chapter 6: The CA Problem with Priority Assignments

As with the CASCA algorithm, when the reward values get closer to unity the accuracy of each final cost value improves but the execution times increase. Indeed, the CASCAPA algorithm exhibits all of the properties discussed for the CASCA algorithm which means that it can be optimized for speed, accuracy or some combination of the two. Also, as with the parameters $\lambda_{R1}$ and $\lambda_{R2}$, the value of $\lambda_{R4}$ should always be set lower than, or equal to, $\lambda_{R3}$ since this is only invoked when a lower cost solution is found and is not used as much as $\lambda_{R4}$, which is invoked when any feasible solution is found. This means that the algorithm can make larger jumps for the priority levels once it has found the optimal number of priority levels.
6.6 The Discrete Automata Solution to the CAPA Problem

In this section we present a solution to the CAPA problem that uses a discretized LA scheme to assign the number of priority levels. In the same manner that the CASCAPA algorithm is based on the original CASCA solution, the Discretized Automata Solution to Capacity Assignment with Priorities (DASCAPA) algorithm is based on the original DASCA algorithm. As discussed earlier, an additional computational layer is added to the DASCA algorithm that is used to decide the number of priority levels that are used in a given test as well as the point at which the algorithm is stopped. The actual solutions are found using the original DASCA algorithm and as a result produce equivalent results but have faster execution times when compared with the CASCAPA algorithm.

A new vector called the priority level convergence vector, that uses integers to keep track of the automaton, will replace the priority level probability vector. This vector will contain an entry for each of the possible combinations of priority levels. For example, if the maximum number of priority levels for a given network is 3, then the priority level convergence vector is composed of three quantities:

(i) The probability that the optimal number of priority levels is one. This means that packets can only have a priority level of 1.

(ii) The probability that the optimal number of priority levels is two. This means that packets can have a priority level of 1 or 2.

(iii) The probability that the optimal number of priority levels is three. This means that packets can have a priority level of 1, 2, or 3.
Chapter 6: The CA Problem with Priority Assignments

For \( r \) priority levels, the initial priority preference vector is given as \([\text{total-steps}/r, ..., \text{total-steps}/r]\). If there are three priority levels with 30 total steps, then the initial priority preference vector is given as \([10, 10, 10]\). The total of the entries in the convergence vector must always be equal to the total number of steps with the optimal priority level being given by the entry that has the value closest to the convergence value, that is, the total number of steps. The degree by which the convergence vector is modified remains fixed for any resolution, total-steps, where the entries are always decreased by unity or increased by subtracting the total of the decreased entries from the total number of steps. Therefore, as in the case of the DASCA algorithm, the higher the total number of steps the more accurate the solution will be. However, the rate of convergence to the optimal value will be slower and this will increase the execution times of the algorithm.

Once again, we must also keep track of the best link capacity assignment for each number of priority levels. So, if there are three priority levels then we must maintain optimal link capacity assignments for three priority level configurations, as explained above.

Assume the priority level convergence vector is given as \((x_1, ..., x_n, ..., x_n)\) where \( n \) is the maximum number of priority levels. The vector is modified in two cases.

Case 1: The new solution is feasible and additionally, the cost is lower than the previous best cost, or the cost is lower than, or equal to, the previous best cost and the number of priority levels has been increased. If the present number of priority levels is \( i \), then.

\[
\begin{align*}
(i) & \quad x_j(n+1) = x(n)_j - 1 & \text{where } j \neq i, \text{ and,} \\
(ii) & \quad x_i(n+1) = \text{total-steps} - \sum_{j=1}^{n} x_j(n+1).
\end{align*}
\]
Case 2: The solution is feasible and the number of priority levels is equal to the best number of priority levels.

(i) $x_j(n+1) = x_j(n) - 1$ where $j \neq i$, and,

(ii) $x_i(n+1) = \text{total-steps} - \sum_{j \neq i} x_j(n+1)$

The operation of this scheme is illustrated in the example below.

**Example 6.6.1**

Say that there are three possible priority levels for the packets with 30 total steps. This means that the initial priority preference convergence vector at time $n$, $[x_1(n), x_2(n), x_3(n)]$, is given as $[10, 10, 10]$. Let us also assume that the solution under investigation, which has three priority levels, is feasible, has a lower cost and has the best number of priority levels. The new priority preference convergence vector will be given as:

Step 1: $x_i(n+1) = 10 - 1 = 9,$

$x_2(n+1) = 10 - 1 = 9,$

$x_3(n+1) = 30 - (9 + 9) = 30 - 18 = 12.$

Step 2: $x_i(n+1) = 9 - 1 = 8,$

$x_2(n+1) = 9 - 1 = 8,$

$x_3(n+1) = 30 - (8 + 8) = 30 - 16 = 14.$

The new priority preference convergence vector is given as $[8, 8, 14]$ and the corresponding priority probability vector is $[8/30, 8/30, 14/30]$.

The DASCAPA algorithm is presented formally in the following section.
6.6.1 The DASCAPA 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 automaton must reach before we accept it as having converged.
(iv) Parameters total-steps₁, total-steps₂.

Output:
The lowest cost capacity assignment vector and best priority level configuration.

Assumption:
The original DASCA algorithm is available and is invoked by DASCA(total-steps₁).

Method

START-MAIN //Priority Assignment Using LA - Discrete Version

Repeat
For (i=1 to maxlinks) Cᵢ = RAND(1, maxcap) End-For
Until (network is feasible)
best-Cᵢ = Cᵢ
While ( (num-inters < max-itors) AND (no priority-entry > req-priority-accuracy) )
  For (i=1 to maxpriorities) num-levels = MAX(Pr[i-priority-levels]) End-For
  For (i=1 to maxpackets) packet.priority = RAND(1, num-levels) End-For
  For (i=1 to maxlinks) Cᵢ = MAX(Pr[Stayᵢ] capacity for num-levels) End-For
  For (i=1 to maxlinks) Cᵢ = best-Cᵢ found using DASCA(total-steps₁) End-For
  current-cost = lowest-cost found using DASCA(total-steps₁)
  If ( (network is feasible)
    AND ( (current-cost < best-cost)
    OR ( (current-cost ≤ best-cost) AND (num-levels > best-num-levels)) )
  Then Raise-Priority(num-levels, total-steps₂)
    best-cost = current-cost
    best-num-levels = num-levels
    For (i=1 to maxlinks) best-Cᵢ = Cᵢ End-For
  End-If
  If (network is feasible) AND (num-levels = best-num-levels)
    Then Raise-Priority(num-levels, total-steps₂)
  End-If
End-While

END-MAIN //Priority Assignment Using LA - Discrete Version
Procedure Raise-Priority

Input: (i) num-levels  
       (ii) total-steps  

Output: The modified priority level convergence vector for link i.

Method

BEGIN-MAIN
  total = 0  
  For (i=0 to total-prior-levels)  
    If (i = num-levels)  
      Then prior-level_i = prior-level_i - 1  
          total = total + prior-level_i  
    End-If  
  End-For  
  prior-level_{num-levels} = total-steps - total

END-MAIN \Procedure Raise
6.6.2 Experimental Results

The DASCAPA algorithm was tested extensively using the experimental test bench described in Chapter 3 and the results are presented in the same format used above for the CASCAPA algorithm. The parameters are now specified in terms of total number of steps for the convergence vector in the original DASCA, given as num-steps₁, and the priority level convergence vector in the DASCAPA algorithm, given as num-steps₂. A comparison of the CASCAPA and DASCAPA algorithms concludes the section.
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4907.68</td>
<td>6947.30</td>
<td>12111.40</td>
<td>21065.20</td>
<td>84785.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.39</td>
<td>1.43</td>
<td>11.26</td>
<td>4.23</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>8835.04</td>
<td>8664.02</td>
<td>18666.10</td>
<td>36291.10</td>
<td>97840.70</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.17</td>
<td>0.77</td>
<td>0.98</td>
<td>0.55</td>
<td>26.26</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4777.68</td>
<td>19040.20</td>
<td>11873.30</td>
<td>18880.10</td>
<td>83219.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.44</td>
<td>1.54</td>
<td>0.44</td>
<td>2.47</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>5080.68</td>
<td>13746.10</td>
<td>13309.00</td>
<td>19582.90</td>
<td>93156.30</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.33</td>
<td>0.71</td>
<td>0.49</td>
<td>1.37</td>
<td>2.64</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>6734.00</td>
<td>14355.70</td>
<td>12181.40</td>
<td>19399.30</td>
<td>72908.30</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.22</td>
<td>0.60</td>
<td>0.55</td>
<td>0.38</td>
<td>2.64</td>
</tr>
</tbody>
</table>

Table 6.6.2.1 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1$ = 20, num-steps$_2$ = 20.
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6573.18</td>
<td>9677.07</td>
<td>9585.51</td>
<td>37679.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.49</td>
<td>0.88</td>
<td>1.20</td>
<td>1.60</td>
<td>6.10</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6308.70</td>
<td>9587.59</td>
<td>9346.07</td>
<td>37833.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.33</td>
<td>0.93</td>
<td>2.91</td>
<td>1.54</td>
<td>13.73</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4743.68</td>
<td>6145.70</td>
<td>9369.07</td>
<td>10311.60</td>
<td>38654.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.60</td>
<td>2.97</td>
<td>3.95</td>
<td>1.27</td>
<td>7.25</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9790.55</td>
<td>38355.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.38</td>
<td>0.61</td>
<td>0.93</td>
<td>1.71</td>
<td>4.17</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>9513.50</td>
<td>9805.35</td>
<td>10463.30</td>
<td>37556.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.28</td>
<td>1.43</td>
<td>1.04</td>
<td>4.06</td>
<td>9.56</td>
</tr>
</tbody>
</table>

Table 6.6.2.2 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps, = 40, num-steps, = 20.
Table 6.6.2.3 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1 = 60$, num-steps$_2 = 20$. 

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6953.78</td>
<td>9363.07</td>
<td>9216.07</td>
<td>37104.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.65</td>
<td>1.49</td>
<td>6.64</td>
<td>1.54</td>
<td>11.09</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9730.11</td>
<td>9710.71</td>
<td>37240.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.88</td>
<td>3.96</td>
<td>2.14</td>
<td>1.37</td>
<td>8.68</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6536.18</td>
<td>9216.07</td>
<td>9527.07</td>
<td>36943.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.60</td>
<td>1.43</td>
<td>1.27</td>
<td>2.63</td>
<td>7.86</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37435.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.53</td>
<td>1.60</td>
<td>4.34</td>
<td>1.37</td>
<td>57.50</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6629.70</td>
<td>9346.07</td>
<td>9216.07</td>
<td>37262.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.82</td>
<td>1.71</td>
<td>1.15</td>
<td>1.70</td>
<td>7.53</td>
</tr>
<tr>
<td>Test</td>
<td>Category</td>
<td>Net 1</td>
<td>Net 2</td>
<td>Net 3</td>
<td>Net 4</td>
<td>Net 5</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37258.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.42</td>
<td>2.80</td>
<td>5.11</td>
<td>2.97</td>
<td>12.02</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9388.07</td>
<td>9216.07</td>
<td>37123.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.83</td>
<td>2.26</td>
<td>2.31</td>
<td>5.05</td>
<td>13.02</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9729.51</td>
<td>37098.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.72</td>
<td>3.73</td>
<td>5.38</td>
<td>3.24</td>
<td>12.80</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6308.70</td>
<td>9216.07</td>
<td>10439.20</td>
<td>37630.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.55</td>
<td>1.54</td>
<td>2.74</td>
<td>4.72</td>
<td>10.11</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9753.67</td>
<td>37357.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>0.56</td>
<td>3.73</td>
<td>2.91</td>
<td>9.61</td>
<td>9.07</td>
</tr>
</tbody>
</table>

Table 6.6.2.4 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps\(_1\) = 80, num-steps\(_2\) = 20.
## Table 6.6.2.5 Results for DASCAPA Solution to the CAPA Problem

The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps, = 100, num-steps, = 20.
### Table 6.6.2.6 Results for DASCAPA Solution to the CAPA Problem

The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps, = 40, num-steps, = 40.

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9369.07</td>
<td>37395.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.13</td>
<td>4.45</td>
<td>2.58</td>
<td>39.50</td>
<td>31.14</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6315.30</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37562.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.14</td>
<td>2.09</td>
<td>5.11</td>
<td>15.93</td>
<td>43.06</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9395.07</td>
<td>9379.07</td>
<td>37444.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.20</td>
<td>2.75</td>
<td>9.40</td>
<td>64.04</td>
<td>44.33</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37424.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.64</td>
<td>3.63</td>
<td>5.50</td>
<td>33.78</td>
<td>40.86</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37138.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>4.06</td>
<td>2.31</td>
<td>2.04</td>
<td>20.54</td>
<td>75.96</td>
</tr>
<tr>
<td>Test</td>
<td>Category</td>
<td>Net 1</td>
<td>Net 2</td>
<td>Net 3</td>
<td>Net 4</td>
<td>Net 5</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>--------</td>
<td>--------</td>
<td>---------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6308.70</td>
<td>9216.07</td>
<td>9371.07</td>
<td>37073.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>4.72</td>
<td>5.00</td>
<td>2.91</td>
<td>30.16</td>
<td>118.86</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9363.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.25</td>
<td>12.52</td>
<td>7.80</td>
<td>8.84</td>
<td>83.76</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9380.07</td>
<td>37097.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.15</td>
<td>4.83</td>
<td>5.93</td>
<td>14.23</td>
<td>55.36</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37107.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>8.73</td>
<td>4.72</td>
<td>6.09</td>
<td>4.17</td>
<td>57.86</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37129.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.86</td>
<td>8.12</td>
<td>9.39</td>
<td>34.88</td>
<td>66.79</td>
</tr>
</tbody>
</table>

Table 6.6.2.7 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1$ = 60, num-steps$_2$ = 40.
<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37407.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>12.20</td>
<td>5.38</td>
<td>10.76</td>
<td>27.07</td>
<td>139.73</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37215.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.76</td>
<td>5.99</td>
<td>10.05</td>
<td>23.40</td>
<td>165.71</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37302.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>8.68</td>
<td>8.35</td>
<td>10.00</td>
<td>8.41</td>
<td>167.90</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>37110.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>1.87</td>
<td>5.83</td>
<td>14.99</td>
<td>38.29</td>
<td>93.60</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36989.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>5.99</td>
<td>5.18</td>
<td>4.94</td>
<td>15.93</td>
<td>69.09</td>
</tr>
</tbody>
</table>

Table 6.6.2.8 Results for DASCAPA Solution to the CAPA Problem. The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps$_1$ = 80, num-steps$_2$ = 40.
### Table 6.6.2.9 Results for DASCAPA Solution to the CAPA Problem

The networks used are described in Section 3.3 and Section 6.2 and the parameters are num-steps\(_1 = 100\), num-steps\(_2 = 40\).

<table>
<thead>
<tr>
<th>Test</th>
<th>Category</th>
<th>Net 1</th>
<th>Net 2</th>
<th>Net 3</th>
<th>Net 4</th>
<th>Net 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>2.97</td>
<td>6.92</td>
<td>18.46</td>
<td>30.92</td>
<td>70.75</td>
</tr>
<tr>
<td>2</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.40</td>
<td>5.93</td>
<td>11.04</td>
<td>12.58</td>
<td>124.46</td>
</tr>
<tr>
<td>3</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.84</td>
<td>14.83</td>
<td>14.89</td>
<td>21.15</td>
<td>145.67</td>
</tr>
<tr>
<td>4</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>3.90</td>
<td>19.12</td>
<td>15.66</td>
<td>27.46</td>
<td>117.27</td>
</tr>
<tr>
<td>5</td>
<td>Cost ($)</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td></td>
<td>Time (sec)</td>
<td>7.58</td>
<td>10.17</td>
<td>6.92</td>
<td>30.53</td>
<td>153.73</td>
</tr>
</tbody>
</table>
Chapter 6: The CA Problem with Priority Assignments

The results presented in the tables above clearly show that the DASCAPA algorithm is faster than the CASCAPA algorithm while maintaining the superior performance level that distinguishes the LA algorithms from the MT-CAPA solution.

The results displayed in the tables demonstrate that the DASCAPA solution produces superior results when compared with the MT-CAPA algorithm and also converges faster than the CASCAPA algorithm. For example, the best results obtained by the CASCAPA algorithm for Network #3 is a final cost of $9,216.07 which took 3.02 seconds while the DASCAPA algorithm produces the same final cost value of $9,216.07 but takes only 0.93 seconds. Once again, this improvement becomes more pronounced as the size of the network increases. For example, the best results obtained by the CASCAPA algorithm for Network #5 is a final cost of $36,813.40 which took 43.11 seconds while the DASCA algorithm produces the same final cost value of $36,813.40, but takes only 18.13 seconds.

Another property that becomes evident from the results is that as the total number of steps get larger, the accuracy of each final cost value improves but the execution times are increased. Therefore, like all of the LA algorithms, the DASCAPA algorithm can be optimized for speed, accuracy or some combination of the two that is appropriate for the particular requirements application. It should also be noted that the DASCAPA algorithm is slower than its DASCA counterpart because it must solve the original CA problem for many different priority level configurations.
6.7 Conclusion

In this chapter we have introduced a new problem that is directly related to the Capacity Assignment (CA) problem, namely the Capacity Assignment with Priority Assignment (CAPA) problem. In this problem the object is to find the lowest cost capacity assignment vector that coincides with the best possible number of priority levels, both of which are set by the solution algorithms. The various assumptions, delay formulae and the experimental test-bench that are used are the same as those used for the original CA problem. Section 6.3 discusses the complexity of this problem and Section 6.4 presented an algorithm developed by Marayuma and Tang in which they expanded their original solution to the CA problem (the MT-CA algorithm presented in Chapter 3) to include priority assignments, namely, the MT-CAPA algorithm. Naturally, this solution produces better cost results than the MT-CA algorithm but has large execution times, especially for large networks.

Section 6.5 introduced the Continuous Automata Solution to CAPA (CASCAPA) algorithm that uses the CASCA algorithm, discussed in Chapter 4, to solve the CAPA problem. The results of numerous experiments demonstrate that this algorithm produced superior cost results and also had substantially lower execution times, when compared with the MT-CAPA algorithm. Next, a discretized version of the algorithm was presented in Section 6.5, namely the Discrete Automata Solution to CAPA (DASCAPA) algorithm. This algorithm produces equivalent cost results when compared with the CASCAPA algorithm but also has lower execution times. The LA algorithms that manipulate priority levels produce better results than the original solutions but have higher execution times.
7.1 SUMMARY

In this thesis 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 that traverse the network. We have also examined the Capacity Assignment Problem with Priorities (CLAP) in which the priorities of the packet classes can be chosen in such a manner to minimize the capacity requirements of the links in the network and, in so doing, further reduce the cost of the network.

The first reported solution to the CA problem is due to Marayuma and Tang (MT) and uses a set of heuristic procedures to find the lowest cost link capacity configuration. This algorithm proves to be only adequate in finding the low cost solution and 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 and always produces a superior low cost capacity assignment, and has much faster execution times, when compared with the MT algorithm.

The third method presented, which constitutes the research component of the thesis, uses Learning Automata (LA) to provide a solution to the CA problem. LA are stochastic algorithms that provide a method for machine learning when the system interacts with a random environment. Learning is achieved by the automaton choosing an action and then processing the response given by the random environment which it utilizes to choose a new action. Eventually, a good automaton will choose one action more than any other and this is hopefully the optimal
action. LA are classified into two groups - Fixed Structure Stochastic Automata (FSSA) and Variable Structure Stochastic Automata (VSSA). We have argued that FSSA are unsuitable for this application and so, in this thesis, VSSA have been used to provide the solution. Some of the LA studied in the literature include Linear Reward-Penalty ($L_{RP}$), Linear Reward-Inaction ($L_{RI}$) and Linear Inaction-Penalty ($L_{IP}$). For example, in the $L_{RI}$ scheme the automaton updates the action probability vector only if the action chosen by the automaton was rewarded. These schemes can be classified as Absorbing or Non-Absorbing. Absorbing schemes (such as $L_{RI}$) are used in stationary environments since they converge to a single optimal action while non-absorbing schemes (such as $L_{RP}$, $L_{IP}$) do not converge to a single optimal action and are generally used in non-stationary environments.

The CA problem provides a stationary environment since there is only one optimal lowest cost capacity assignment, and so an absorbing scheme such as the $L_{RI}$ scheme is used in our algorithms.

LA can also be classified as continuous or discretized which characterize how they modify the action probability vector. A continuous scheme moves toward the optimal action in progressively smaller steps in a continuous space, whereas a discretized scheme will approach the optimal value in discrete jumps typically of the same size. Discretized schemes are generally faster than continuous schemes while maintaining the desired quality of the results produced. In this thesis we evaluate both continuous and a discretized schemes in providing solutions for the CA problem.

The Continuous Automata Solution to CA (CASCA) 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. The Discrete Automata Solution to CA (DASCA) algorithm produces equivalent capacity assignments when compared with the CASCA algorithm.
but also proves to be even faster. A summary of the best results obtained by all four algorithms are shown in the table 7.1.1.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>NET 1</th>
<th>NET 2</th>
<th>NET 3</th>
<th>NET 4</th>
<th>NET 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-CA</td>
<td>5735.04</td>
<td>12686.10</td>
<td>11669.30</td>
<td>53765.90</td>
<td>43341.90</td>
</tr>
<tr>
<td>0.22</td>
<td>0.77</td>
<td>1.86</td>
<td>2.08</td>
<td>67.45</td>
<td></td>
</tr>
<tr>
<td>LE-CA</td>
<td>5735.04</td>
<td>7214.22</td>
<td>10295.70</td>
<td>45709.60</td>
<td>39838.40</td>
</tr>
<tr>
<td>0.22</td>
<td>1.21</td>
<td>1.10</td>
<td>1.93</td>
<td>4.01</td>
<td></td>
</tr>
<tr>
<td>CASCA</td>
<td>4907.68</td>
<td>6937.90</td>
<td>9909.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td>0.05</td>
<td>0.39</td>
<td>0.70</td>
<td>1.33</td>
<td>4.12</td>
<td></td>
</tr>
<tr>
<td>DASCA</td>
<td>4907.68</td>
<td>6937.90</td>
<td>9909.11</td>
<td>40348.90</td>
<td>37307.40</td>
</tr>
<tr>
<td>0.05</td>
<td>0.33</td>
<td>0.51</td>
<td>1.32</td>
<td>2.51</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1.1 Best Results for CA Problem for all the algorithms studied.

The networks that were used are described in Section 3.3.

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>NET 1</th>
<th>NET 2</th>
<th>NET 3</th>
<th>NET 4</th>
<th>NET 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MT-CAPA</td>
<td>4731.68</td>
<td>6275.70</td>
<td>9346.07</td>
<td>25983.10</td>
<td>37491.80</td>
</tr>
<tr>
<td>183.01</td>
<td>409.09</td>
<td>1812.59</td>
<td>388.62</td>
<td>4192.77</td>
<td></td>
</tr>
<tr>
<td>CASCAPA</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td>0.82</td>
<td>2.80</td>
<td>3.02</td>
<td>6.92</td>
<td>38.54</td>
<td></td>
</tr>
<tr>
<td>DASCAPA</td>
<td>4601.68</td>
<td>6145.70</td>
<td>9216.07</td>
<td>9216.07</td>
<td>36813.40</td>
</tr>
<tr>
<td>0.28</td>
<td>0.61</td>
<td>0.93</td>
<td>1.37</td>
<td>27.61</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.1.2 Best Results for CAPA problem for all the algorithms studied.

The networks that were used are described in Section 3.3 and Section 6.2.
Chapter 7: Conclusions

The overall cost of the network can be reduced further by allowing the algorithm to manipulate the priorities of the packet classes to find the best match of packet priority levels and link capacity assignments. This is called the Capacity Assignment with Priority Assignment (CAPA) problem. Marayuma and Tang extended their investigation of the CA problem to develop an algorithm that included this feature. Their solution, called the MT-CAPA algorithm, produced a substantially better low cost value than the original MT-CA algorithm, in which the priorities are fixed, but naturally, greatly increased the execution times. We have also extended the LA algorithms discussed above to address the CAPA problem. The Continuous Automata Solution to CAPA (CASCAPA) algorithm produces a superior low cost value when compared with the MT-CAPA algorithm and is vastly superior in terms of execution times. As with the CA problem, the Discrete Automata Solution to CAPA (DASCAPA) algorithm maintains the quality of the results of the CASCAPA algorithm but generally reduces the execution times. The best results obtained by the algorithms are shown in table 7.1.2.
7.2 FUTURE WORK

As with any field of research there are several areas that may be considered for modification and/or improvement. These considerations would relate to both of the problems we have discussed since the CAP problem is essentially a modified version of the original CA problem.

There are a number of factors that could be considered for modification. The nodal and propagation delays could be added to the delay considerations which would increase the complexity of the evaluation of average delay. The routing for each link could be made a decision variable in addition to the capacity assignments which would allow for more flexibility and accuracy in the final solution. Similarly, the topology of the network could also be made a decision variable. However, adding this feature as well as simultaneous decisions for both routing and capacity assignment will make the problem much more complicated. The final results could be verified by a network simulation process in order to assure the accuracy of the modeling assumptions. Indeed, some of the assumptions that are made for the purposes of modeling (such as the Kleinrock Independence Assumption) are not realistic but are necessary to reduce the mathematical complexity of the problem. The cost structure of the links could be modeled to be more realistic and future research could incorporate costs based on traffic intensity and/or time and day considerations that could be built into the cost function. Another interesting idea would be to try to solve the CA problem using another area of artificial intelligence known as genetic algorithms.
REFERENCES


References


References


