INFORMATION THEORETIC MODELS
OF STORAGE AND MEMORY

by

Susan Aileen Hall

This report was based on the unaltered thesis of Susan Aileen Hall, submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering at the Massachusetts Institute of Technology. The research was conducted at the MIT, Laboratory for Information and Decision Systems with support provided by the Air Force Office of Scientific Research under Contract AFOSR-80-0029 (MIT OSP Nos. 91076 and 92567).

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology
Cambridge, MA 02139
INFORMATION THEORETIC MODELS
OF STORAGE AND MEMORY

by

SUSAN AILEEN HALL
A.B., Oberlin College
(1976)

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1982

© Massachusetts Institute of Technology 1982

Signature of Author

Department of Electrical Engineering and Computer Science

Certified by

Alexander H. Levis
Thesis Supervisor

Accepted by

Chairman, Departmental Committee on Graduate Students
INFORMATION THEORETIC MODELS
OF STORAGE AND MEMORY

by

SUSAN AILEEN HALL

Submitted to the Department of Electrical Engineering
and Computer Science on August 20, 1982 in partial fulfillment of the
requirements for the Degree of Master of Science in
Electrical Engineering

ABSTRACT

The functions of storage and memory as they occur in a rate-constrained
decisionmaking system are modeled in an information theoretic framework.
Two types of storage, and their roles in the decisionmaking process, are
investigated. Buffer storage, used in processing statistically dependent
inputs simultaneously, is modeled according to the input the system receives.
Memory is subdivided into two categories: permanent memory is modeled as
information which may be accessed by the algorithms but cannot be revised by
them; temporary memory is modeled as information that the algorithms store
for use in future iterations. Tradeoffs between improved performance,
increased activity, and longer system response time, are examined. An
example of a decisionmaking system performing two tasks by switching be-
tween sets of data stored in memory, is analyzed.

Thesis Supervisor: Alexander H. Levis
Title: Senior Research Scientist
ACKNOWLEDGEMENT

There are many people to whom I would like to say thank you. First, I am deeply grateful to Alex Levis for his help in introducing me to the world of research. I could not have asked for a more supportive advisor. Thanks also to Kevin Boettcher and Gloria Chyen, both of whose ideas contributed heavily to mine. I would also like to extend a special thank you to Al Drake for all his kindness and encouragement throughout my years here at MIT.

I could not have finished this thesis without the help of Frantiska Frolik's and Pat Dumont's typing skills and Art Giordani's beautiful figures. Thanks also to Lisa Babine for cheering up this lab and for letting me use her typewriter.

There are two more special thank you's that I have to include here. First, I have to thank David Bass for his love and support throughout my thesis trials and tribulations. And finally, I would like to thank my parents and two sisters for being a wonderful family to me.

This research was conducted at the M.I.T. Laboratory for Information and Decision Systems with support provided by the Air Force Office of Scientific Research under Contract AFOSR-80-0229 (M.I.T. OSP Nos. 91076 and 92567).
# TABLE OF CONTENTS

1. INTRODUCTION  
   1.1 Overview  
   1.2 The Thesis in Outline  

2. ON INFORMATION THEORETIC MODELS OF DECISIONMAKING  
   2.1 Background  
   2.2 Entropy and Transmission  
   2.3 Memory and Performance  
   2.4 Constraints  

3. PARTITION LAWS OF INFORMATION RATES  
   3.1 Rates  
   3.2 Partition Laws  
   3.3 Information Theoretic Constraints  

4. BUFFER STORAGE  
   4.1 Assumptions on Sources  
   4.2 The Basic Buffer Storage Model  
   4.3 Shift Register Buffers  
   4.4 Fixed-Length String Buffers  
   4.5 Variable-Length String Buffers  
   4.6 Comparison of the Three Buffer Models  

5. BUFFER STORAGE: A COIN-TOSSING EXAMPLE  
   5.1 Introduction and Assumptions  
   5.2 Analysis: Cost and Activity  
   5.3 Results  

Page  
7  
7  
9  
10  
10  
12  
14  
15  
17  
17  
18  
20  
22  
22  
23  
26  
33  
39  
48  
52  
52  
54  
60
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>PERMANENT AND TEMPORARY MEMORY</td>
<td>63</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>63</td>
</tr>
<tr>
<td>6.2</td>
<td>Permanent Memory</td>
<td>65</td>
</tr>
<tr>
<td>6.3</td>
<td>Temporary Memory</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>PERMANENT MEMORY: THE DUAL-TASK PROBLEM</td>
<td>75</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction and Assumptions</td>
<td>75</td>
</tr>
<tr>
<td>7.2</td>
<td>Information Theoretic Analysis</td>
<td>78</td>
</tr>
<tr>
<td>7.3</td>
<td>Effect of Task Division on Performance</td>
<td>84</td>
</tr>
<tr>
<td>8</td>
<td>CONCLUSIONS AND FUTURE RESEARCH</td>
<td>87</td>
</tr>
<tr>
<td>8.1</td>
<td>Conclusions</td>
<td>87</td>
</tr>
<tr>
<td>8.2</td>
<td>Future Research</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td><strong>APPENDIX A: DERIVATION OF SEVERAL INFORMATION THEORETIC EXPRESSIONS</strong></td>
<td></td>
</tr>
<tr>
<td>A.1</td>
<td>Derivation of Partition Law of Information Rates</td>
<td>91</td>
</tr>
<tr>
<td>A.2</td>
<td>Derivation of Activity Expressions</td>
<td>92</td>
</tr>
<tr>
<td>A.3</td>
<td>Derivation of Dual-Task Coordination Expression</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td><strong>APPENDIX B: DERIVATION OF MARKOV PROCESS QUANTITIES</strong></td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>REFERENCES</td>
<td>103</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Model of Decisionmaking Process with Performance Evaluation Mechanism</td>
<td>12</td>
</tr>
<tr>
<td>4.1</td>
<td>Basic Buffer Model</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Markov Representation of Source</td>
<td>27</td>
</tr>
<tr>
<td>4.3</td>
<td>Shift Register Buffer Model</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Markov Process Representation of Periodic Source</td>
<td>34</td>
</tr>
<tr>
<td>4.5</td>
<td>Buffer Model for Fixed-Length Independent Strings</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Markov Process Representation of Variable-Length String Source</td>
<td>40</td>
</tr>
<tr>
<td>4.7</td>
<td>Buffer Model for Variable-Length Independent Strings</td>
<td>42</td>
</tr>
<tr>
<td>4.8</td>
<td>Markov Process Model for Values of P</td>
<td>43</td>
</tr>
<tr>
<td>4.9</td>
<td>Markov Process Model for Buffer Variable $b_1$</td>
<td>45</td>
</tr>
<tr>
<td>4.10</td>
<td>Markov Process Model for Buffer Variable $b_2$</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>Model of System with Buffer for Coin-Tossing Example</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>Model of SA Subsystem with Memory</td>
<td>65</td>
</tr>
<tr>
<td>6.2</td>
<td>Example of SA Subsystem with Permanent Memory</td>
<td>66</td>
</tr>
<tr>
<td>7.1</td>
<td>Model of Decisionmaking System with Permanent Memory for Dual-Task Problem</td>
<td>77</td>
</tr>
<tr>
<td>7.2</td>
<td>Representation of G vs. J for Binary Variation of Pure Strategies</td>
<td>81</td>
</tr>
<tr>
<td>7.3</td>
<td>Activity vs. Performance and Task Division</td>
<td>84</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

The process of decisionmaking, as it occurs both in human decision-makers and in machines, has been studied extensively in recent work, in part to try to determine how groups of decisionmakers, i.e. organizations, can process information more efficiently. With the advent of extremely sophisticated computer/communications networks, enormous amounts of information are being transferred among organizations daily; it is crucial therefore that these organizations be structured in such a way that they can process efficiently only the information that contributes to their overall organizational objectives. For example, Stabile [1], [2] found that partitioning the input to a single-echelon organization, such that each member receives only the information that he is particularly well-suited to process, may improve the performance of the organization.

This need for organizational efficiency has been felt by many types of organizations, including the military. Lawson [3], among others, has noted the need for a study of how the military should distribute information in the command control process, where this process is defined as that "by which a properly designated commander exercises authority and direction over assigned forces in the accomplishment of his mission." This type of study is fairly tractable in the context of a military organization for several reasons. First, the organizational objective (mission) is well-defined. Second, the tasks which must be accomplished in order to achieve this objective become well-defined as soon as the commander exercises his authority and direction. Because of this, the human decisionmakers who are part of the organization may be characterized specifically in terms of their skills and limitations in addressing these various tasks. Also, a tactical organization generally operates in a high data-rate environment, which is well-suited to evaluating decisionmaking systems at the limits of their processing capabilities. However, in order to be able to analyze
organizational performance in any detail, it is first necessary to understand thoroughly the components of that organization, namely individual decisionmaking systems.

In recent work on the design of military organizations, several models of decisionmaking have been developed in the framework of information theory (see chapter 2), but none of these has included explicit models of storage and memory. The belief that it is essential to include the effect of memory on the performance of a system, has motivated the development of the models of storage and memory, and their role in the decisionmaking process, presented here. At this point, it is emphasized that the models presented in this thesis are meant to apply to the decisionmaking process, rather than specifically to either human decisionmakers or machines. In fact, it is believed that these models may be applied to both men and machines: this may be very useful in the analysis of organizations which are made up of both. In particular, the decisionmaking process is assumed to be in the steady state; i.e. the structure of the decisionmaking algorithms is fixed. These algorithms may have been developed by a human decisionmaker during a learning or training period, or they may have been programmed directly into a machine; but these processes are not considered here.

Two basic types of storage are modeled in this thesis. The first of these stores information from the environment temporarily, to allow the system to process statistically dependent inputs simultaneously. The second type of storage provides a more permanent repository of information, to which the algorithms have access during the decisionmaking process.

Included in the model of decisionmaking considered here are two constraints. The first of these is an upper bound on the system's rate of information-processing, where this constraint is modeled quantitatively in information theoretic terms. Note that this constraint may be applied to a system with or without memory. With the addition of memory to the decisionmaking model, a second constraint may have to be applied. A system that stores information and waits until it has received all the data per-
taining to one task before performing that task, may introduce a delay: i.e. 
the system's response time may be increased from what it was in the system 
without memory. In some organizations, particularly in the military, this 
delay may not be tolerable, and the organization designer may choose to 
 impose an upper bound on the amount of delay that any given system can 
 introduce.

1.2 THE THESIS IN OUTLINE

This thesis has been organized into eight chapters as follows.
Chapter 2 presents a brief history of past decisionmaking models, followed 
by an introduction to information theory and an overview of some of the 
performance and constraint considerations for a decisionmaking system 
with memory. Chapter 3 provides the quantitative framework for the thesis, 
with a discussion of rates of information and partition laws applying to 
these rates. Chapter 4 presents the models developed for the first type 
of storage described previously, called buffer storage. Included in this 
chapter are the analyses of three different buffer storage models, and a 
discussion of performance and constraint considerations. In chapter 5, 
a quantitative example of a decisionmaking system with buffer storage is 
presented to demonstrate the tradeoffs that may occur between performance, 
activity, and delay considerations. Chapter 6 presents the models for the 
second type of storage described previously: this type of storage is 
called memory, for the purposes of this thesis, and subdivided into 
permanent and temporary memory. Analyses of both types are given, and 
then in chapter 7, an application of a decisionmaking system with permanent 
memory is presented. Specifically, the problem of having to switch be-
tween two sets of information stored in permanent memory, depending on 
which of two tasks arrives at the system, is examined; and the perfor-
mance of this system is compared to that of the same system when only 
one task must be performed. Chapter 8 includes the conclusions of the 
thesis, as well as some suggestions for further research in the area.
CHAPTER 2
ON INFORMATION THEORETIC MODELS OF DECISIONMAKING

2.1 BACKGROUND

Information Theory or Communication Theory, developed by Shannon [4] and others in the late 1940's, has been applied to many areas outside of communications, most notably in some simple models of the human decisionmaker [5]. But, as Laming [6] observed, the human decision-maker does not act like a memoryless communications channel, and, in fact, the purpose of most decisionmaking systems is quite other than to reproduce faithfully at the output what was given to the system as input. In accordance with this observation, a two-stage information theoretic model of the decisionmaking process has been developed (Boettcher [7] and Boettcher and Levis [8]), which more realistically includes internal variables and algorithms between the input and the output. However, that model is memoryless; that is, it is unable to recognize any statistical dependence that might exist in the input to the decisionmaking system. This is a simplifying but very limiting assumption: certainly a military organization receives many inputs related to the same tactical situation, and many of these are statistically dependent on one another. Sen and Drenick [9] recognized the need for adding memory to models of decisionmaking systems. They modeled the human decisionmaker as an adaptive channel, i.e. a channel whose output may depend on present and past inputs. With this addition of memory, they achieved results which, in some experimental situations, reflect observed behavior. However, they have made no attempt to model explicitly the various types of memory that may be found in a decisionmaking system.

This thesis, which does model memory explicitly, is most closely related to the work of Boettcher and Levis [8], so at this time a brief overview of that decisionmaking model will be given.
In classical decision theory, the decisionmaker has full knowledge of all possible actions he may take, and of the consequences of each of these actions; he also has a cost ordering on these consequences; and finally he has an unlimited amount of time in which to make his decision. However, in complex decisionmaking situations when a limited amount of time is available for the decisionmaking process, the decisionmaker may be better modeled as being boundedly rational, i.e. constrained in his abilities to formulate actions and foresee consequences. Rather than always being able to make the optimal decision, a decisionmaker with bounded rationality may satisfice, that is, may seek to satisfy some set of minimal criteria in making a decision [10].

Decisionmaking under bounded rationality is modeled as a two-stage process. The first is the situation assessment stage (SA) in which one of U algorithms is selected via the variable u (which is assumed to be statistically independent of the input), to evaluate the input and "hypothesize about its origin." The output of the SA stage, z, could be an estimate of the actual signal given the observed signal, or some other statistic of the input, or even the entire input itself. The variable z is then given to the response selection stage (RS), and one of V algorithms is chosen via the variable v, to process the evaluated input into an appropriate response. Both sets of algorithms are assumed to be deterministic, so that, given an input x, and the values of u and v, the output y may be exactly determined. Bounded rationality is modeled by requiring that the total rate of activity of the system, where total rate of activity is a well-defined information theoretic quantity, be less than some maximum value, which is specific to a given decisionmaking system.

The performance of the decisionmaker is evaluated as shown in Figure 2.1. The actual input $x'$ is corrupted by noise, $n$, so that the system receives $x = x' + n$, a noisy version of the input. Note that this noise could range from representing actual interference with a message sent to the decisionmaker along standard communications channels, to representing the decisionmaker's inability to observe perfectly, or obtain perfect information pertaining to, his environment. The
mapping \( L(x') \) yields \( y' \), which is defined as the ideal response to the actual input \( x' \); then \( y' \) is compared to the output of the system, \( y \). The performance measure of the system is \( J \), the expectation of \( d(y, y') \), where the latter is the cost of deciding \( y \) when \( y' \) is the desired response. In the context of this model, then, a satisficing decisionmaker must choose a decision strategy, two probability distributions on \( u \) and \( v \), that results in \( J \leq \bar{J} \), where \( \bar{J} \) is the maximum cost that can be tolerated.

The model description is developed in the analytic context of information theory. A brief introduction to the two important quantities in information theory, entropy and transmission, will now be presented.

2.2 ENTROPY AND TRANSMISSION

As already mentioned, information theory was first developed as an application in communications theory. But, as Khinchin [11] showed, information theory is a valid mathematical theory in its own right, in the area of probability theory, and it is useful for applications in many disciplines, including the analysis of information-processing systems.
There are two primary quantities in information theory. The first of these is entropy: given a variable $x$, which is an element of the alphabet $X$, and occurs with probability $p(x)$, the entropy of $x$, $H(x)$, is defined to be

$$H(x) = - \sum_x p(x) \log p(x)$$  \hspace{2cm} (2.1)$$

and is measured in bits when the base of the logarithm is two. Entropy is also known as the average information or uncertainty in $x$, where information does not refer to the content of the variable $x$, but rather to the average amount by which knowledge of $x$ reduces the uncertainty about it. As Garner [12] describes it, "...information is a function not of what the stimulus is, but rather of what it might have been, of its alternatives." The other quantity of interest in information theory is average mutual information or transmission: given two variables $x$ and $y$, elements of the alphabets $X$ and $Y$, and given $p(x)$, $p(y)$, and $p(x|y)$ (the conditional probability of $x$, given the value of $y$), the transmission between $x$ and $y$, $T(x:y)$, is defined to be

$$T(x:y) = H(x) - H_y(x)$$  \hspace{2cm} (2.2)$$

where

$$H_y(x) = - \sum_y p(y) \sum_x p(x|y) \log p(x|y)$$  \hspace{2cm} (2.3)$$

is the conditional uncertainty in the variable $x$, given full knowledge of the value of the variable $y$. Transmission measures the relatedness or constraint holding between two variables. It can be interpreted as the amount by which knowledge of $y$ reduces the uncertainty in $x$, or vice versa, as it is a symmetric quantity in $x$ and $y$. Entropy and transmission are particularly useful for measuring the uncertainty and relatedness of quantities which are not numerically defined.

McGill [13] extended this basic two-variable input-output theory
to N dimensions. Noticing that Eq. (2.2) can also be written as

$$T(x:y) = H(x) + H(y) - H(x,y)$$  \hspace{1cm} (2.4)

McGill's extension to N dimensions,

$$T(x_1:x_2:\ldots:x_N) \equiv \sum_{i=1}^{N} H(x_i) - H(x_1,x_2,\ldots,x_N)$$  \hspace{1cm} (2.5)

seems quite natural. N-dimensional mutual information measures the total constraint holding between all N variables of a system (constraints between pairs, triples, etc.). The beauty of this measure for systems analysis, as Ashby [14] and, more recently, Conant [15] have pointed out, is that it may be expressed as the sum of simpler quantities. For example, a system may be decomposed into subsystems, and the N-dimensional mutual information of this system is the sum of two quantities: the transmissions of the individual subsystems, and the transmission between subsystems. With N = 4, this might be expressed:

$$T(x_1:x_2:x_3:x_4) = T(x_1:x_2) + T(x_3:x_4) + T(x_1,x_2:x_3,x_4)$$  \hspace{1cm} (2.6)

2.3 MEMORY AND PERFORMANCE

As stated in the introduction, this thesis models storage of two basic functional types. The first type is storage designed to retain inputs temporarily so that a given input may be processed simultaneously with those previous inputs upon which it is statistically dependent. The second type is memory designed to store information on a more permanent basis to be available to the decisionmaking algorithms to access as needed in the decisionmaking process. By considering these two types of storage, it is hoped that the decisionmaking model will become more dynamic. However, it should be emphasized that the dynamics that are being added here do not include the concept of learning, i.e. neither the set of algorithms nor the structure of any one of these algorithms is allowed to evolve over time. Although the decisionmaking system modeled here is operating in time and is considering present events in the light of past ones, it is also operating in the steady
state, i.e. the decisionmaker is well-trained.

Evaluation of the performance of a decisionmaking system with memory will be similar to that for the memoryless decisionmaking system described in Section 2.1, but with some notable differences. Of course, as long as the inputs to the system are statistically independent, the same measure may be used, whether the system has any internal memory subsystems or not. But once dependence is introduced, it is no longer sufficient. The ideal response to a given input may depend also on previous inputs; the new mapping may be \( L[x'(t)|x'(t-1), x'(t-2), \ldots, x'(t-N+1)] \) if the source generates inputs which are statistically dependent on the previous \( N-1 \) inputs; or it may be \( L[x'(t), x'(t-1), \ldots, x'(t-N+1)] \) if the source generates strings of \( N \) letters, which are mutually dependent but independent of all other inputs; and if only one response is demanded per string. The performance of the system may improve with the addition of memory, but this will also depend on the constraints on the system.

2.4 CONSTRAINTS

One of the primary constraints to be considered here is the constraint of bounded rationality, or the maximum information-processing rate which seems to exist in human decisionmakers and may also be applied to machines. When the input rate is low enough to ensure that the rate of information-processing stays below the maximum, then simple information-processing tasks (such as identifying which of several stimuli are presented) are performed without error [5]. As the input rate increases, the information-processing rate also increases, until the maximum is reached. If the input rate increases further still, the condition of overload is said to exist. An overloaded decisionmaker cannot keep up with the fast rate of input; thus, errors appear in the output. However, the way in which the decisionmaker attempts to reduce his information-processing load is not statistically predictable and may take many forms, such as ignoring entire inputs, filtering out some features of the input, giving incorrect or less accurate responses, etc. [16]. Information overload is certainly worthy of study, but it is little understood and beyond the scope of this thesis.
Therefore, a constraint to be uniformly applied here is that the rate of information processing of a system be less than a given maximum.

A similar constraint is encountered in communications theory, the capacity constraint on a communications channel. This constraint defines in information theoretic terms the maximum transmission between input and output that a particular channel can provide. This constraint alone, however, is not adequate to describe the limitations of a decisionmaking system. Some decisionmaking tasks may require very little actual input-output transmission (for example, any yes-no decision has an input-output transmission of, at most, one bit of information), but may nonetheless require a great deal of internal processing in order to arrive at a decision [15]. In this case, the need for a constraint on all of the information-processing tasks being performed becomes apparent.

A decisionmaking system may also be constrained by the amount of delay that can be tolerated between the arrival of a symbol at the system and the response to that symbol. If the source generates statistically independent inputs, then each one is immediately processed, and the issue of delay need not be addressed. The introduction of dependence creates the option of storing inputs, thereby introducing a time delay, until enough information has been received to make a good decision, i.e. to achieve a high level of performance. The trade-off between delay and performance is a classic C³ problem. In a tactical environment, there may be a high cost associated with a late response, but the cost may be even higher for a bad decision. This issue of delay is addressed in conjunction with buffer storage.

In the next chapter, the total rate of activity of a decisionmaking system with internal variables will be defined in terms of information theoretic quantities associated with those variables.
CHAPTER 3
PARTITION LAWS OF INFORMATION RATES

As already stated, a decisionmaking system does much more than to transmit information from input to output. Conant [15] defines the total information theoretic activity of a system to be the sum of the entropies of all the individual variables in the system, without considering constraints between variables. This is not unreasonable, since it gives a measure of how much information-processing each variable is contributing to the whole. The notion of total activity is especially valuable because it can be decomposed, using Conant's Partition Law of Information Rates, into quantities which seem to correspond to what can actually happen to information as it is processed by a system. First, however, rates of information must be defined.

3.1 RATES

Because this thesis is concerned with memory, and sequential inputs which are dependent on each other, the entropy rate, $\overline{H}(x)$, which describes the average entropy of $x$ per unit time, will be used. The entropy rate of $x$ is defined to be:

$$\overline{H}(x) \equiv \lim_{m \to \infty} \frac{1}{m} \sum_{t=0}^{m-1} H(x(t), x(t+1), \ldots, x(t+m-1))$$

For example, let a dial be set to either 0 or 1, independently and with equal probability, at noon each day; and let $x$ be defined to be the reading on the dial at any given hour. Then the entropy of $x$ is one bit, since at any given hour, the dial is equally likely to read 0 or 1. But the entropy rate of $x$ is $\frac{1}{24}$ bits per hour, since there is one bit of uncertainty at noon every day, and the reading on the dial for the 23 hours following is completely determined by the noon reading. Transmission rates, $T(x:y)$, are defined exactly like transmissions, but using entropy rates in the definition rather than entropies. Fortunately,
the calculation of the entropy rates for most of the sources to be
discussed here take on simpler forms than that given in Eq. (3.1).
These will be noted as each of these sources is discussed.

3.2 PARTITION LAWS

Conant's Partition Law of Information Rates (PLIR) [15] is defined
for a system with \( N - 1 \) internal variables, \( w_1 \) through \( w_{N-1} \), and an
output variable, \( y \), also called \( w_N \). The PLIR states

\[
\sum_{i=1}^{N} H(w_i) = T(x:y) + \overline{T}(x:w_1, w_2, \ldots, w_{N-1}) + \overline{T}(w_1:w_2: \ldots :w_{N-1}:y)
\]

\[
+ H_x(w_1, w_2, \ldots, w_{N-1}, y) \tag{3.2}
\]

and is easily derived using information theoretic identities (see Ap-
pendix A,1). As noted in the introduction to this chapter, the left-
hand side of Eq. (3.2) refers to the total rate of activity of the
system, also designated \( F \). Each of the quantities on the right-hand
side of Eq. (3.2) has its own interpretation; these will now be
discussed in turn.

The first term, \( \overline{T}(x:y) \), is called the throughput rate of the system
and designated \( F_t \). It measures the amount by which the output of the
system is related to the input. This is the quantity that a communi-
cations channel tries to maximize. The second term,

\[
\overline{T}(x:w_1, w_2, \ldots, w_{N-1}) = \overline{T}(x:w_1, w_2, \ldots, w_{N-1}, y) - \overline{T}(x:y) \tag{3.3}
\]

is called the blockage rate of the system and designated \( F_b \). As the
above expansion demonstrates, blockage may be thought of as the amount
of information in the input to the system that is not included in the
output. The third term, \( \overline{T}(w_1:w_2: \ldots :w_{N-1}:y) \), is called the coordination
rate of the system and designated \( F_c \). It is just the \( N \)-dimensional
transmission of the system, the amount by which all of the internal
variables in the system constrain each other. As noted before, if the
system may be broken up into independent subsystems, then this coordination
term may be expressed as a simple sum of the coordinates of the individual subsystems. The last term, \( \bar{H}_x(y, w_1, w_2, \ldots, w_{N-1}) \), is called the noise rate of the system and designated \( F_n \). It represents the uncertainty that remains in the system variables when the input is completely known. Here, noise should not be construed to be necessarily undesirable as it is in communications theory: it may also be thought of as internally-generated information, information supplied by the system to supplement the input and facilitate the decisionmaking process. The PLIR may be abbreviated:

\[
F = F_t + F_b + F_c + F_n \tag{3.4}
\]

Another interesting and useful decomposition is that of \( \bar{H}(x) \):

\[
\bar{H}(x) = \bar{H}(x:y) + \bar{H}_y(x) = \bar{T}(x:y) + \bar{T}_y(x) - \bar{H}_{w_1, w_2, \ldots, w_{N-1}}(x) + \bar{H}_{w_1, w_2, \ldots, w_{N-1}}(x)
\]

\[
= \bar{T}(x:y) + \bar{T}_y(x:w_1, w_2, \ldots, w_{N-1}) + \bar{H}_{w_1, w_2, \ldots, w_{N-1}}(x) \tag{3.5}
\]

The first and second terms of this decomposition have already been defined as throughput and blockage. The third term is called the rejection rate and designated \( F_r \): it is the uncertainty remaining in the input with complete knowledge of the system variables and output. It may be thought of as information in the input which the system does not recognize, information reflected at the boundary of the system. An analogy between Eq. (3.5) and electromagnetic radiation passing through a medium might be made: in Eq. (3.5), the information in the input to a system may be transmitted through it, blocked by it, or rejected at the boundary of the system; likewise, electromagnetic radiation passing through a medium may be transmitted through the medium, absorbed by the medium, or reflected at the surface of the medium. Equation (3.5) may be abbreviated:

\[
\bar{H}(x) = F_t + F_b + F_r \tag{3.6}
\]
In this work, no mechanism has been provided by which rejection can occur. In fact, this is a process which has yet to be modeled, i.e. the process by which a system only allows certain information to cross its boundaries. Because this has not yet been done, it is assumed here that the rejection rate of the system is identically zero. Note that the expression for the blockage of the system becomes particularly simple when the assumption of zero rejection is applied:

\[ F_b = \overline{H}_y(x) \]  

(3.7)

In fact, this quantity, the uncertainty in the input of the system, given the output, is known as the channel equivocation in communications theory. Also note that the uncertainty in the input is now confined to throughput and blockage, i.e.:

\[ \overline{H}(x) = F_t + F_b \]  

(3.8)

3.3 INFORMATION THEORETIC CONSTRAINTS

Now that the information theoretic relations to be used in this thesis have been defined, the constraints to be applied to the decision-making model with memory may be expressed quantitatively.

The bounded rationality constraint is expressed by postulating the existence of a maximum rate of information-processing, or a maximum rate of total activity, \( F_{\text{max}} \), at which a given decisionmaking system can operate without overload. Then, any strategy that is chosen must have a total rate of activity that is less than this maximum. Note that the addition of memory to the decisionmaking model increases the total number of variables in the system and may, therefore, restrict the strategies that may be used to those with lower activity. However, note also that executing a task with memory may result in a better performance than that achievable in a system without memory. As Boettcher shows [7], if \( J \), the performance measure, is chosen to be the probability of error, then both \( J \) and the total activity of the system may be found as functions of the decision strategy, and hence,
parametrically, as functions of each other. In this case, then, if the bounded rationality constraint eliminates some of the possible decision strategies, it may be seen directly how this affects the performance of the decisionmaking system.

The only other information theoretic constraints to be considered here are implicit constraints on particular memory subsystems, which act more as functional definitions than as constraints. For example, the purpose of a buffer is to store inputs so that dependent inputs may be processed together. Clearly, this particular subsystem should not reject any information, it should not block any information, nor should it add any information to the input: i.e. $F_r = F_b = F_n = 0$. In fact, the purpose of a buffer is not to do any information processing per se, but rather to reformat the input in such a way that it may be better processed by the rest of the system. Note that if these constraints are satisfied, all of the information in the input is transmitted (see Eq. (3.8)):

$$F_t = \overline{H}(x)$$

(3.9)

On the other hand, if permanent memory devices have any uncertainty in their values, they may add extra information or noise to the system, i.e. $F_n \neq 0$. The desirable information-processing activities of any memory subsystem are clearly a function of the type of subsystem being considered and its intended usage.

In the next chapter, buffer storage devices will be modeled for three classes of sources, where classification is determined by the statistical structure of the input that the source generates. These models will be analyzed to determine the amount of activity they add to a decisionmaking system, and then an example will be given to demonstrate the improvement in performance that the addition of buffer storage may allow.
CHAPTER 4
BUFFER STORAGE

This thesis considers two types of information storage that a decision-making system may have, where type is classified according to function rather than structure. The first type is storage whose function it is to allow a given input to be processed together with all previous inputs upon which it is statistically dependent. It is called buffer storage here, and it is located structurally between the source and the system, where it can reformat the input that the source generates. The second type of storage is that whose function it is to provide the decisionmaking algorithms with previously stored information to be used in the decisionmaking process. For the purposes of this thesis, this second type of storage is called memory, and it acts as a repository of information or reference data for the algorithms. Memory is further subdivided into two types: that in which information is stored indefinitely, called permanent memory; and that which contains information that may be appended and revised by the decisionmaking algorithms, called temporary memory.

This chapter examines buffer storage. First, a general introduction is given, in which both the input received by the decisionmaking system, and the basic buffer storage model are described. Then, several specific storage models, appropriate to input having various statistical structures, are analyzed in an information theoretic context to determine the additional activity incurred by each one. Finally, a comparison of the buffer models is made and a discussion of performance and delay considerations is presented.

4.1 ASSUMPTIONS ON SOURCES

Several assumptions on the input to the decisionmaking system must be made, if information theory is to be used to make a quantitative analysis of the system. Specifically, the input is assumed to be generated by a
discrete stationary ergodic source. (Periodic sources of period $L$ may be included in this definition if sequences of $L$ symbols are considered to be super symbols, which constitute stationary input.) The assumptions of stationarity and ergodicity are certainly restrictive, and some real systems will not have these characteristics. However, many systems may be approximately stationary over a given length of time, and many may also be considered to be ergodic over a given period, if the state of the source at some initial time is specified. Therefore, the insights to be gained from an information theoretic analysis may actually be applicable to a wide range of systems.

It is also assumed that $T$, the interarrival time for individual inputs, is constant. This may be an intrinsic characteristic of the source itself; or it may be that $T$ is probabilistic but has a bounded range, so that the input rate can be regulated by a device located between the source and the decisionmaking system. If this latter is the case, then the combination of the source and the regulator is viewed as a super source, which generates an input every $T$ time units.

Other assumptions which apply to specific models are introduced as they are needed.

4.2 THE BASIC BUFFER STORAGE MODEL

An input generated by an arbitrary discrete stationary ergodic source may be statistically dependent on an infinite number of previous inputs, in which case it is futile to attempt to process it together with all previous inputs.

Fortunately, though, as pointed out by Gallager [17], "most well-behaved stationary sources can at least be approximated by" sources which generate inputs that depend on only a finite number of previous inputs. (These are called Markov sources and are discussed in more detail in the next section.)
Consequently, if any input from such a source depends on, at most, \( N-1 \) previous inputs, then it is only necessary to store \( N \) pieces of data at any one time.

The general model of these \( N \) storage locations consists of \( N \) buffer variables, \( b_1 \) through \( b_N \). The data are stored in these locations according to a set of rules which may be different for sources with different statistical structures. The output of the buffer, \( q \), is an \( N \)-vector which is sent on to the rest of the decisionmaking system for processing.

Figure 4.1. Basic Buffer Model

In the next sections, three specific buffer models will be analyzed. Each model is designed to deal with one of three classes of input, where classification is made according to string length. A string is defined here as a set of sequential inputs which may be statistically dependent on each
other, but are statistically independent of inputs generated by the source at all other times. For example, a source might generate every third input by an independent, equally-likely choice between 0 and 1, and then repeat that value as the next two inputs; it would be described as a source with fixed string length 3. It is also possible for a source to generate strings whose lengths are finite (all possible strings are of length \( \leq N \), say) but variable: a source which generated either a 0 or the sequence 11 by an independent choice is one example. A general Markov source, then, with every input dependent on the previous N-1 inputs, may be described as a source with infinite string length.

At this point, some notion will be given of the types of sources that can be modeled according to the classifications just specified. That is, one may choose to model some real-life sources of information as Markov sources (infinite string length) and some as sources which generate finite-length strings. A discussion of this modeling process follows.

In general, the modeling of a source is influenced as much by the modeler's perspective and needs as by the actual statistical nature of the source. For one thing, events which are "almost" independent, such as events having a very low correlation coefficient, may be approximated in some cases as being independent. For instance, although there may be a small amount of correlation between the large scale movements of aircraft at two distant bases in different countries, these two sets of movements may be considered to be independent. Second, whether two events are considered to be dependent or independent may be a function of the observer's state of knowledge. Consider the example just given: if intelligence has indicated that there is a secret military alliance between the two nations, the observer may want to evaluate the aircrafts' movements as dependent events: that is, the observer may use information about one set of movements to draw conclusions as to the meaning of the other.
set. Finally, it may be the case that although some dependence exists between two events, that dependence is not relevant to the processing that the decisionmaking system is to carry out, and may therefore be disregarded. For example, from a decoder's point of view, a Morse code source may be thought of as generating finite, variable-length independent strings: even though sequential letters are actually correlated, the decoder can carry out its task by decoding one letter at a time, independently.

These observations have been made to stress that, in modeling these sources, it is not the actual physical nature of the source that one attempts to capture: rather, the state of knowledge or environment of the decisionmaking system, and the function of the system itself, must be taken into consideration in the modeling process. Morse code is a specific example of a source that might be modeled as a finite, variable-length string source. An example of a fixed-length string source might be a random number generator that generates 3-digit numbers: again, although there is a definite correlation between sequential strings, they may be considered to be independent. A source which generates the location and speed of a given aircraft once every minute could be modeled as a Markov or infinite-length string source: the information generated each minute is dependent at least on the previous minute's data. Buffers designed to handle this last type of input will be discussed in the next section.

4.3 SHIFT REGISTER BUFFERS

4.3.1 Markov Input

As already stated, most discrete stationary ergodic sources can be approximated by Markov sources. A discrete-state discrete trial Markov process is defined by a set of \( M \) states \( \{1, 2, \ldots, M\} \), and a matrix of transition probabilities from one state to another, with transitions occurring at specified instants. The process is defined to have the Markov property if and only if the transition probabilities depend only on the present
state of the process [18]. This definition appears to be more limiting than it really is. As long as a propitious choice of what to call a state is made, sources whose inputs depend on many past inputs may be described as Markov sources. For instance, say a source generates a 1 if both of the previous two inputs have been 0, generates a 0 if both of the previous two inputs have been 1, and makes an equally likely choice between 0 and 1 if the previous two inputs have been different. Then the source may be represented by the following Markov process state-transition diagram, where the circles represent states and the directed lines between them represent transition probabilities. Here, a state is defined as the two most recent inputs (with the newest input on the right).

![Markov Representation of Source](image)

**Figure 4.2. Markov Representation of Source**

For a general stationary ergodic Markov source, a state consists of the previous \( N-1 \) inputs, and \( \overline{H}(x) \) takes on a particularly simple form:

\[
\overline{H}(x) = \sum_{i=1}^{M} \pi_i H(x | s = i)
\]  

(4.1)
That is, the uncertainty of the source is just the weighted sum of the uncertainties of the next symbol, given the source is presently in state \( i \). Here the weighting, \( \pi_i \), is the steady-state probability that the source will be in state \( i \) after many trials. Note that

\[
\sum_{i=1}^{M} \pi_i = 1
\]  

(4.2)

In order to allow a decision-making system to process an input from such a source along with all previous inputs upon which it is statistically dependent, a shift register buffer is necessary.

4.3.2 The Shift Register Model

A shift register retains the \( N \) most recent inputs and sends these out for processing every time unit. After an \( \mathbf{N} \)-vector is sent out, the oldest piece of information is discarded, and the other \( N-1 \) are shifted one buffer variable down, so that the next input may be placed in buffer variable \( b_1 \).

![Diagram of Decision Making System]

Figure 4.3. Shift Register Buffer Model
Specifically, at time \( t \), with data arriving once every second,

\[
b_1(t) = x(t)
\]

\[
b_i(t) = b_{i-1}(t-1); \quad i = 2, 3, ..., N
\]

\[
g(t) = b(t) = [b_1(t)b_2(t)...b_N(t)]
\]  \( (4.3) \)

The addition of the shift register to the decisionmaking system will necessarily increase the total activity of the system, because of the \( N+1 \) extra variables, \( b_1, b_2, ..., b_N \), and \( g \). The type and amount of additional activity can be determined as follows.

Looking at this buffer as a separate subsystem of the overall decision-making system, with input \( x \), internal variables \( b_1 \) through \( b_N \) and output \( g \), the throughput, blockage, noise, and coordination rates may be calculated.

The throughput rate

\[
\overline{T}(x; g) = \overline{H}(g) - \overline{H}_x(g)
\]  \( (4.4) \)

is just \( \overline{H}(g) \), because knowledge of \( x \) provides complete knowledge of \( g \), and so \( \overline{H}_x(g) \) is zero. But \( \overline{H}(g) \) is equal to \( \overline{H}(x) \), because \( g \) takes on exactly the same values as \( x \) (albeit at different times, which does not make any difference when calculating entropy rates). So the throughput rate, \( F_t \), is just \( \overline{H}(x) \). Assuming that none of the input is rejected, the blockage can be found using Eq. (3.8):

\[
F_b = \overline{H}(x) - F_t
\]  \( (4.5) \)
With \( F_t = \overline{H}(x) \), this rate is zero. The noise rate, \( \overline{H}_x(b_1, b_2, \ldots, b_N, \mathbf{q}) \), is also zero because knowledge of \( x \) provides complete knowledge of the values of \( b_1 \) through \( b_N \) and \( \mathbf{q} \). So far, all is as expected. This ideal buffer does not generate information (noise), nor does it block any of the input (blockage), but it allows all of the information in \( x \) to be transmitted (throughput), albeit in a different format than that of the original input.

The coordination rate,

\[
F_C = \sum_{i=1}^{N} \overline{H}(b_i) + \overline{H}(\mathbf{q}) - \overline{H}(b_1, b_2, \ldots, b_N, \mathbf{q}) \tag{4.6}
\]

demonstrates the additional activity that the reformatting requires. Consider the last term of Eq. (4.6):

\[
\overline{H}(b_1, b_2, \ldots, b_N, \mathbf{q}) = \overline{H}(b_1, b_2, \ldots, b_N) + \overline{H}(b_1 b_2 \ldots b_N, \mathbf{q}) \tag{4.7}
\]

The vector \( \mathbf{q} \) is completely determined by the buffer variables, so the last term of this expression is zero, and the rate of uncertainty in the buffer variables is just \( \overline{H}(x) \). So

\[
F_C = \sum_{i=1}^{N} \overline{H}(b_i) + \overline{H}(\mathbf{q}) - \overline{H}(x) \tag{4.8}
\]

But, as already stated, \( \overline{H}(\mathbf{q}) = \overline{H}(x) \), so

\[
F_C = \sum_{i=1}^{N} \overline{H}(b_i) \tag{4.9}
\]

That is, the coordination rate of this shift register buffer is just the sum of the uncertainty rates for all of the buffer variables, \( b_1 \) through \( b_N \).
Since each buffer variable takes on all the values that \( x \) takes on, albeit at different times, then

\[
\overline{H}(b_i) = \overline{H}(x) \quad i=1,2,...N \tag{4.10}
\]

and the coordination of this buffer subsystem is

\[
F_c = N \overline{H}(x) \tag{4.11}
\]

As a check on these calculations, it is useful to recall the definition of the left hand side of Eq. (3.4). The total rate of activity \( F \) is the sum of the entropy rates of all the individual variables in the system:

\[
F = \sum_{i=1}^{N} \overline{H}(b_i) + \overline{H}(q) = (N+1) \overline{H}(x) \tag{4.12}
\]

This total should be equal to \( F_t + F_b + F_n + F_c \). As already found, the shift register has zero blockage and noise, its throughput is \( \overline{H}(x) \), and its coordination is \( N \overline{H}(x) \), so the equality is verified.

Looking at the decisionmaking system as a whole, it is easily seen that all of the activity of the shift register is included in the coordination term for the overall system. The buffer does not add any noise or blockage to the overall system. Also, because the output of the buffer, \( q \), is not one of the system output variables, the throughput of the shift register subsystem is not included in the calculation of total system throughput [15]. To calculate the addition to the coordination rate, recall the decomposition rule introduced in section 3.2, and apply it to
the coordination of a system $S$ made up of a buffer $B$ and a set of decision-making algorithms $\text{DAs}$:

$$F_c(S) = F_c(B) + F_c(\text{DAs}) + \bar{T}(B;\text{DAs}) \quad (4.13)$$

The buffer contributes to the first and last terms of this expression. The first term, the internal coordination of the buffer, has already been found to be $N \bar{H}(x)$. The last term, the coordination between the shift register and the rest of the system, accounts for the remaining $\bar{H}(x)$ in the total activity of the buffer subsystem.

4.3.3 Comments

The calculation of the total activity of the shift register subsystem may now be used in the determination of the total activity of the decision-making system, so that the bounded rationality constraint, requiring that this activity be less than or equal to the given maximum, may be applied. Note that if $N$ or $\bar{H}(x)$ is large, then this constraint may require that only decisionmaking algorithms with lower amounts of activity be used. Also note that the shift register buffer provides that every individual input to the decisionmaking system be processed $N$ times. This creates a great deal of redundancy in the decisionmaking process, which may or may not be desired.

The shift register is indeed the appropriate model for a buffer which allows an input from a general Markov source to be processed together with all previous inputs upon which it is statistically dependent. When a source can be modeled as generating strings of finite length, string buffers, which operate differently from shift registers, may be used. These buffers still have the desired property of processing dependent inputs together, and they have the added advantage of
usually requiring less overall activity than the shift register buffer. String buffers unfortunately increase the average amount of time between the arrival of an input at the decisionmaking system and the appearance of the corresponding output, i.e. they introduce a delay. The two string buffer models, designed to handle both sources which generate fixed-length strings and those which generate variable-length strings, will now be examined in some detail.

4.4 FIXED-LENGTH STRING BUFFERS

4.4.1 Periodic Sources

A source which generates strings of fixed length N may be represented as a periodic Markov source of period N. Its state includes the values of the previous N-1 inputs, but also contains information as to the relative position of the most recent input within the string. That is, with a symbol being generated once every second, the symbol occurring at time \( t = KN+i \) is the \( i \)-th symbol in the \( K \)-th string of length N, so that \( i \) represents the relative position of the symbol within the \( K \)-th string. For a periodic Markov source, then, the previous N-1 inputs, and the value of \( i \) for the most recent of these are sufficient information to define a state. For instance, consider the example given in section 4.2: a source with \( N \) equal to 3 generates every third input by an independent equally likely choice between 0 and 1, and then repeats that value for the next two inputs. Let \( XY_i \) represent the state of this source, where \( X \) is the second most recent input, \( Y \) is the most recent input, and \( i \) is the most recent input's location in the string as defined above. Then this source may be represented by the Markov process state transition diagram in Figure 4.4.

Unfortunately, periodic sources are neither stationary nor ergodic; fortunately, in spite of this, it is still possible to define \( H(x) \) for a periodic Markov source:
Figure 4.4. Markov Process Representation of Periodic Source

\[ \overline{H}(x) = \sum_{i=1}^{M} l(i) H(x|s=i) \]  

(4.14)

Here, \( l(i) \) is roughly the percentage of times the system is in state \( i \) over many trials. More exactly,

\[ l(i) = \lim_{L \to \infty} \frac{1}{L} \sum_{L=1}^{L} \text{Prob}[S(L) = i] \]  

(4.15)

where \( S(L) \) is the state of the system after the \( L \)th trial. For example, for the source just described, the system is equally likely to be in states \( 00_1, 10_1, 01_1, \) or \( 11_1 \), after the generation of the first letter of the string. But the first string letter is only generated every third time unit, so \( l(i) \) for these 4 states is equal to \( \frac{1}{3} \frac{1}{4} = \frac{1}{12} \). Similarly, \( l(i) \) for the remaining 4 states is equal to \( \frac{1}{3} \frac{1}{2} = \frac{1}{6} \). Now, \( H(x|s) \) is zero for any of the six states with subscripts 1 or 2; given that the system is in any one of these states, there is no uncertainty as to what the next input will be. So \( \overline{H}(x) \) for the source is: 

-34-
\[
\bar{H}(x) = \frac{1}{6} H(x|s=00) + \frac{1}{6} H(x|s=11)
\]
\[
= \frac{1}{6} (1) + \frac{1}{6} (1) = \frac{1}{3} \text{ bits/time unit}
\] 

This is exactly what one would expect: this source exhibits one bit of uncertainty every third time unit, so its average uncertainty should be \(\frac{1}{3}\).

4.4.2 Fixed-length String Buffer Model

For a periodic source like the ones just described, it is only necessary to process every N inputs together in order to guarantee that each input is being processed along with all other inputs upon which it is statistically dependent. The following buffer model achieves that goal.

![Diagram of a buffer model for fixed-length independent strings]

Figure 4.5. Buffer Model for Fixed-Length Independent Strings

The buffer is represented by N storage variables, denoted \(b_1\) through \(b_N\). An additional variable, \(P\), which may take the values 1 through N, determines which of the buffer variables is connected to the input at any given time. The output of the buffer (and the input to the rest of the decision-making system) is a concatenation of N input letters, represented by the N-vector \(q\). \(P\) is deterministic with respect to time, and inputs are assumed to arrive once every second: that is, when \(t=1\), \(P=1\); when \(t=2\), \(P=2\); and in gen-
eral, when \( t=K, \ P = [(K-1) \mod N] + 1 \). So, \( P \) switches upon the arrival of each new input. The buffer operates by sequentially filling up with input letters, and, when full, every \( N \) time units, sending its entire contents in the form of the vector \( q \) on to the rest of the decisionmaking process. Specifically, \( b_i \) through \( b_N \) and \( q \) take values at time \( t \) as follows:

\[
\begin{align*}
b_i(t) &= x(t) & \text{if } P = i \\
b_i(t) &= b_i(t-1) & \text{if } P \neq i \\
q(t) &= (b_1 b_2 ... b_N) & \text{if } P = N \\
q(t) &= q(t-1) & \text{if } P \neq N
\end{align*}
\]  (4.17)

As in the case of the shift register buffer, this buffer may be viewed as a separate subsystem with input \( x \), internal variables \( P \) and \( b_1 \) through \( b_N \), and output \( q \). And, as in the case of the shift register buffer, it is found that the blockage and noise rates are zero and the throughput rate is just \( \overline{H}(x) \), satisfying the implicit constraints that define a buffer subsystem.

The coordination rate is given by:

\[
F_c = \overline{H}(P) + \sum_{i=1}^{N} \overline{H}(b_i) + \overline{H}(q) - \overline{H}(P, b_1, b_2, ..., b_N, q) \]  (4.18)

Because \( P \) is deterministic in time, \( \overline{H}(P) \) is zero. To find the value of the last term, consider the expansion

\[
\overline{H}(P, b_1, b_2, ..., b_N, q) = \overline{H}(P) + \overline{H}_P(b_1, b_2, ..., b_N) + \overline{H}_{Pb_1b_2...b_N}(q) \]  (4.19)

Again, \( \overline{H}(P) \) is zero, the uncertainty in \( q \) given the values of the buffer variables is zero, and the uncertainty in the buffer variables themselves is just \( \overline{H}(x) \). So Eq. (4.18) becomes

\[
F_c = \sum_{i=1}^{N} \overline{H}(b_i) + \overline{H}(q) - \overline{H}(x) \]  (4.20)
And, as in the shift register case, with $\overline{H}(q)$ equal to $\overline{H}(x)$ this reduces to

$$F_c = \sum_{i=1}^{N} \overline{H}(b_i)$$  \hspace{1cm} (4.21)

Unlike the shift register case, $\overline{H}(b_i)$ may now take on a different value for each value of $i$. In fact, the $i$-th buffer variable may only take on the values that $x$ takes on at times $t = KN+i$, for $K = 0,1,\ldots$, and $i = 1,2,\ldots,N$. Because the strings are generated independently, and because buffer variable $b_i$ changes only once every $N$ time units, then

$$\overline{H}(b_i) = \frac{1}{N} H(x_i)$$  \hspace{1cm} (4.22)

where

$$H(x_i) = \sum_{x_i} p(x_i) \log p(x_i)$$  \hspace{1cm} (4.23)

and $p(x_i)$ is the marginal probability distribution for the $i$-th letter of any string. Note that, because the source is defined to be periodic with period $N$, the $i$-th letters of all strings of length $N$ are not only independent, but are also identically distributed.

Therefore, the coordination term becomes

$$F_c = \frac{1}{N} \sum_{i=1}^{N} H(x_i)$$  \hspace{1cm} (4.24)

and the total activity that this buffer adds to the decisionmaking system is

$$F = F_t + F_b + F_n + F_c$$

$$= \overline{H}(x) + 0 + 0 + \frac{1}{N} \sum_{i=1}^{N} H(x_i)$$  \hspace{1cm} (4.25)
Here, as in the case of the shift register buffer, all of this additional activity contributes to the coordination term of the overall system. The internal coordination of the buffer subsystem is given by \( \frac{1}{N} \sum_{i=1}^{N} H(x_i) \), and the coordination between the buffer and the rest of the decisionmaking system is \( \overline{H}(x) \).

4.4.3 Comments

The fixed-length string buffer is an appropriate model when the information to be processed consists of independent tasks, each of which requires only one decision (though the task consists of \( N \) sequential inputs). Because the entire task is not performed until all of the inputs of which it consists have arrived, an average delay of at least \( \frac{N-1}{2} \) time units is added to the time between the arrival of a symbol at the system and the first appearance of output to which it contributes. This delay may or may not be tolerable. In fact, it is possible that the system may be constrained to respond to a given symbol in an amount of time less than \( \frac{N-1}{2} \), so that this buffer model may not be usable at all. This is a particularly important consideration in a tactical environment, where a delayed response can be very costly. So there exists a tradeoff here: string buffer memory may improve performance, but it may also increase overall system response time.

Note that if \( H(x_i) \) for the fixed-length string buffer and \( \overline{H}(x) \) for the shift register buffer are of approximately the same order of magnitude, then the shift register buffer may add a good deal more activity to the system than the string buffer. This observation does not even take into account the fact that the system with the shift register buffer must process an \( N \)-vector every time unit while the string buffer system need only process an \( N \)-vector every \( N \) time units. With algorithms of comparable activity levels per task, then, the shift register buffer system would
have a much higher demand on it than the string buffer system, and might have to sacrifice performance to satisfy its bounded rationality constraint.

If a source generates strings which are not of constant length, then a variable-length string buffer may be called for. The next section provides an examination of these buffers.

4.5 VARIABLE-LENGTH STRING BUFFERS

4.5.1 Variable-length String Sources

When a source generates strings of fixed length $N$, it is trivial to recognize the end of one string and the beginning of another: a new string begins after every $N$-th input has been generated. The task of recognizing the end of one string and the beginning of another becomes more difficult when the strings are of variable length. There are two basic solutions to this problem. The input may have been prefix-condition coded, where no codeword in such a code is the prefix of any other codeword [17]. In this case, the system which is receiving the input must have a complete listing of all codewords to be employed, so that it can recognize a codeword when its transmission has been completed. This method may require a great deal of processing and will not be considered here. The second option is that a neutral symbol, which is not an element of the alphabet for any input symbol, be employed to indicate the end of a string. For example, Morse code, the classic example of a variable-length string source, employs a space to indicate the end of a string. Here, it will be assumed that a variable-length string source terminates a string with the symbol *. It is also assumed that all strings are finite, of length less than or equal to $N$. Note that, with the addition of the termination symbol, *, the effective length of each string is increased by one.

-39-
The variable-length string source may be represented as a Markov source, where a state is defined to be the previous \( N \) inputs. For example, say a source with \( N \) equal to 2 is equally likely to generate the strings 0, 1, 10, and 11; then with the addition of the termination symbol, a typical sequence of inputs might be 0*10*11*1*. The possible states of this process (the two most recent inputs, with the most recent on the right) are then 0*, 1*, *0, *1, 10, and 11, and the state-transition diagram for the source is given in Figure 4.6:

![State Transition Diagram](image)

**Figure 4.6.** Markov Process Representation of Variable-Length String Source

These sources are both stationary and ergodic, so \( H(x) \) may be found by using Eq. (4.1). Following through with the previous example, the steady-state probabilities are found to be (see Appendix B):
\[ \pi_i = 0.1 \quad i = *0, 10, 11 \]
\[ \pi_i = 0.2 \quad i = 0*, 1* \]
\[ \pi_i = 0.3 \quad i = *1 \]  
(4.26)

Since \( H(x) \) is zero if the present state of the system is \(*0, 10, \) or \(11, \) the entropy calculation reduces to:

\[
\bar{H}(x) = 0.2 \left[ \frac{1}{4} \log_2 4 + \frac{3}{4} \log_2 \frac{4}{3} \right] + 0.2 \left[ \frac{1}{4} \log_2 4 + \frac{3}{4} \log_2 \frac{4}{3} \right] \\
+ 0.3 \left[ \left( \frac{1}{3} \log_2 3 \right) \right]
= 0.8 \text{ bits/time unit} 
\]  
(4.27)

This is what one would expect, as the following reasoning demonstrates. The entropy rate of a source which generates strings of symbols independently and according to the same distribution may be found by calculating the uncertainty in one such generation and then dividing by the average length of time between generations. Now consider the above example: every time a new string is about to begin, there is an equally likely choice among four possibilities, providing 2 bits of uncertainty. Since half of the strings are of length 2 (\(0*, 1*\)) and half of length 3 (\(10*, 11*\)) the average string length is 2.5, so the entropy rate of this source is, as found before,

\[ \bar{H}(x) = \frac{2}{2.5} = 0.8 \text{ bits/time unit} \]  
(4.28)

4.5.2 Variable-length String Buffer Model

The model to be used for the variable-length string buffer is similar to that used for the fixed-length string buffer and is depicted in Figure 4.7. This buffer is represented by \(N+1\) storage variables and the variable \(P\) which may take the values 1 through \(N+1\). The output of the buffer,
\( q \) is assumed to be only an \( N \)-vector. Since the buffer variable \( b_{N+1} \) exists solely to record the end of an \( N \)-symbol string, it will never contain actual information: therefore only the contents of the first \( N \) buffer variables are sent out for processing. Specifically the variables of the buffer take on values at time \( t \) as follows:

\[
\begin{align*}
P(t+1) &= P(t) + 1 \quad \text{if } x(t) \neq \ast \\
P(t+1) &= 1 \quad \text{if } x(t) = \ast \\
b_i(t) &= x(t) \quad \text{if } P = i \\
b_i(t) &= b_i(t-1) \quad \text{if } i > i \\
b_i(t) &= \ast \quad \text{if } P < i \\
q(t) &= q(t-1) \quad \text{if } x(t) \neq \ast \\
q(t) &= (b_1b_2...b_N) \quad \text{if } x(t) = \ast 
\end{align*}
\]

(4.29)

This buffer may again be regarded as a separate subsystem, and the information theoretic rates of throughput, noise, blockage, and coordination may be calculated. As in the previous two examples, this buffer does not block any information \( (F_B = 0) \) nor does it add any internally-generated information to the input \( (F_n = 0) \) but it transmits all of the information in the input to the rest of the system \( (F_t = H(x)) \). For this more complicated buffer,
the calculation of the coordination expression becomes more involved:

\[
F_c = \overline{H}(P) + \sum_{i=1}^{N+1} \overline{H}(b_i) + \overline{H}(q) - \overline{H}(P, b_1, \ldots, b_N, b_{N+1}, q)
\]  

(4.30)

Expand the last term:

\[
\overline{H}(P, b_1, \ldots, b_N, b_{N+1}, q) = \overline{H}(b_1, \ldots, b_N, b_{N+1}) + \overline{H}(b_1 \ldots b_N b_{N+1}, P(q))
\]  

(4.31)

The uncertainty in the buffer variables is just \(\overline{H}(x)\). The variable \(P\) is no longer deterministic, but given knowledge of the values of the buffer variables, both \(P\) and \(q\) are completely determined, so the last two terms of Eq. (4.31) are zero. With \(\overline{H}(q)\) equal to \(\overline{H}(x)\), \(F_c\) reduces to:

\[
F_c = \overline{H}(P) + \sum_{i=1}^{N+1} \overline{H}(b_i)
\]  

(4.32)

Since \(b_{N+1}\) will always contain the value \(*\), \(\overline{H}(b_{N+1}) = 0\), giving:

\[
F_c = \overline{H}(P) + \sum_{i=1}^{N} \overline{H}(b_i)
\]  

(4.33)

The value of \(\overline{H}(P)\) will be very dependent on the structure of the input. If the probabilities of all string lengths are known, then the values of \(P\) can be modeled as a Markov process, and hence \(\overline{H}(P)\) may be found. For example, assume that all string lengths 1 through N are equally likely to occur. Then the Markov state-transition diagram for the values of \(P\) is given below:

![Markov Process Model for Values of P](image)

Figure 4.8. Markov Process Model for Values of P
In order to understand how the transition probabilities for this process are found, consider what may happen if the process is presently in state $N-1$. At this point, the three equally likely possibilities are that the string being generated is of length $N-2$, in which case a * has just been generated and 1 will be the next value $P$ attains (probability $1/3$); or the string is of length $N-1$ or $N$, in which case $N$ will be the next value $P$ attains (probability $2/3$). For each state, $\pi_i$ and $H(P|s=i)$ must be found. Application of standard techniques for finding the steady-state probabilities of a Markov process yields

$$\pi_1 = \frac{2}{N+3}$$

$$\pi_i = \frac{2(N-i+2)}{N(N+3)}; \quad i = 2, 3, ..., N, N+1 \quad (4.34)$$

If $P$ is equal to 1 or $N+1$, there is no uncertainty as to its next value, so $H(P|s=1)$ and $H(P|s=N+1)$ are both zero. Also, it may be found that

$$H(P|s=i) = \frac{1}{N-i+2} \log_2(N-i+2) + \frac{N-i+1}{N-i+2} \log_2\left(\frac{N-i+2}{N-i+1}\right) \quad (4.35)$$

So,

$$\overline{H}(P) = \sum_{i=2}^{N} \frac{2(N-i+2)}{N(N+3)} \left[ \frac{1}{N-i+2} \log_2(N-i+2) + \frac{N-i+1}{N-i+2} \log_2\left(\frac{N-i+2}{N-i+1}\right) \right]$$

$$= \frac{2}{N+3} \log_2 N \ \text{bits/time unit} \quad (4.36)$$

For complete derivations of the above quantities, see Appendix B. For the example given in section 4.5.1, in which string lengths of 1 or 2 are equally likely,

$$\overline{H}(P) = \frac{2}{N+3} \log_2 N$$

$$= \frac{2}{5} \log_2 2$$

$$= 0.4 \ \text{bits/time unit} \quad (4.37)$$
The uncertainty rates in the buffer variables may also be found by modeling the values that a given buffer variable may take as a Markov process. A state for this process is defined by the most recent value of the variable and the most recent value of P. For instance, consider the example given in section 4.5.1. A state for the Markov process model of $b_1$ is given by $X_Y$, where $X$ represents the most recent value of $b_1$ and $Y$ represents the most recent value of $P$. Then the following Markov process state-transition diagram represents the process by which $b_1$ takes values:

![Markov Process Model for Buffer Variable $b_1$](image)

Notice that a 1 stored in $b_1$ may remain in storage for two time units (if the string is $1^*$) or three (if the string is $11^*$ or $10^*$), but a zero will only remain stored in $b_1$ for two time units (if the string is $0^*$). Using this model, it is easy to establish that (see Appendix B)

$$\bar{H}(b_1) = 0.368 \quad (4.38)$$
For this example, the values that $b_2$ may take can also be modeled as a Markov process, with a state again represented by $x_y$, the most recent value stored in $b_2$, $x$, and the most recent value of $P$, $y$. The following is the Markov process state-transition diagram for the values of $b_2$:

![Markov Process Diagram](image)

Figure 4.10. Markov Process Model for Buffer Variable $b_2$

For this model, $\overline{H}(b_2)$ is found to be 0.6 bits per time unit (for details see Appendix B).

For this complete example then, it has been found that

$$F_c = \overline{H}(P) + \sum_{i=1}^{N} \overline{H}(b_i)$$

$$= 0.4 + 0.368 + 0.6 = 1.368 \text{ bits/time unit} \quad (4.39)$$

The coordination term may be calculated, in a manner similar to that carried out above, for any specific source.
The extra activity that this buffer $B$ adds to the overall system contributes solely to the system's coordination. This extra activity is given by:

\[
F_B = (F_t + F_b + F_n + F_c)B
\]

\[
= \bar{H}(x) + 0 + 0 + [\bar{H}(P) + \sum_{i=1}^{N} \bar{H}(b_i)]
\]

\[
= 0.8 + 1.368
\]

\[
= 2.168 \text{ bits/time unit}
\]  

(4.40)

And again, the last two terms are the internal coordination of the buffer, and $\bar{H}(x)$ is the coordination between the buffer and the rest of the system.

4.5.3 Comments

The variable-length string buffer is the appropriate buffer model when the source generates independent strings of variable length. Here also, as in the fixed-length string case, a delay is introduced into the system. For the variable-length case, the average additional delay is given by:

\[
\text{Average delay} = \sum_{i=1}^{N} \frac{i+1}{2} [\text{prob(string is of length } i)] 
\]  

(4.41)

Again, whether this type of buffer can be used may depend on the maximum response time that the system demands.

There is also an implicit assumption made in connection with the variable-length string buffer: it is assumed that the decisionmaking system can process strings of length $k$ in less than $k+1$ time units (the extra time unit is due to the termination symbol *): that is, the system must be able to process short strings faster than long ones. This is not an unreasonable assumption for some algorithms. For
instance, an algorithm which computes the average value of the components of the string probably could process a string of length 2 faster than a string of length 10. However, this result is not general.

4.6 COMPARISON OF THE THREE BUFFER MODELS

The fixed-length and variable-length string buffers are very similar, and in fact, there may be situations in which a variable-length string buffer is called for, but a fixed-length string buffer may be substituted and possibly improve the performance of the system. This substitution may be beneficial when the average length of the variable-length string is close to N, and the variance of the length is small, i.e. most strings are close to length N. Then null symbols may be added to any strings which are not of length N, in order to make all strings the same length. Consider the example analyzed extensively in the previous section: the variable-length string source is equally likely to generate the strings 0, 1, 10, or 11. With the addition of the termination symbol * to each of these, the effective average string length of this source is 2.5 symbols per string. There is another way of representing these four strings, however, and that is to make them all 2 symbols long by the addition of a null symbol "-" to the strings which are only one symbol long. Thus, the new source sends the four strings 0-, 1-, 10, and 11; and a fixed-length string buffer (N=2) may be used in conjunction with this modified source.

There are certain advantages gained by the use of the fixed-length string buffer in a case like this. First, there may be a slight increase in the activity of the buffer variables: in fact, in this case \( \overline{H}(b_1) \) increases from a value of 0.368 to 0.406, and \( \overline{H}(b_2) \) from 0.6 to 0.75 (see Appendix B). However, there is now no uncertainty in P, whereas P for the variable-length string buffer has an average uncertainty of 0.4 bits per second. Therefore, the coordination rate of the buffer has
decreased by 0.212 bits per second. Because of the new shorter average string length, the throughput of the buffer, given by $H(x)$, increases for the fixed-length case to 1.0 bits per second (from 0.8). Even with this increase, the fixed-length string buffer subsystem adds less total activity to the rest of the system than the variable-length string buffer, 2.156 and 2.168 bits per second respectively. This small difference may not appear to be very critical, but there are two points to consider in this regard: first, this example is particularly simple so the numbers involved are small; second, the decisionmaking systems being considered here are set in a $C^3$ environment and therefore are operating at or near to their bounded rationality limit, so that any savings in activity may allow new, perhaps better decision strategies into the feasible set, thereby improving performance.

A second reason that a fixed-length string buffer may be more desirable in this case is that it is a simpler device to implement. It requires one less storage unit than the variable-length buffer, and rather than having to recognize the occurrence of a * to know when to begin storing a new string, the fixed-length string buffer cycles back in a regular manner, every N time units.

There is yet a third reason that the switch to a fixed-length string buffer may be desirable in this situation: the switch may reduce the average delay between the arrival of a symbol at the system and the first appearance of output to which it contributes. Again consider the example of the previous section. With a variable-length buffer, the symbols 0 and 1 of the strings 0* and 1* experience a one second delay before they enter the decisionmaking process. The first symbols of the strings 10* and 11* experience a two second delay, and the second symbols a one second delay, so these two-symbol strings both experience an average delay of 1.5 seconds. Since one-symbol and two-symbol strings are equally likely, the average delay is 1.25 seconds. With the fixed-length string buffer, the one-symbol strings have the same amount of delay because they essentially still have a termination symbol (the null symbol
"-". However, the two-symbol strings do not, so their average delay is reduced by one second to 0.5 seconds, and the overall average delay is 0.75 seconds, a reduction of 0.5 seconds. Again, this may be critical, particularly in a $C^3$ environment, where delay can be very costly. Note that a variable-length source, with a large variance in the length of its strings, or a small (compared to $N$) average string length does not benefit in terms of delay by filling out all of its strings with null symbols and using a fixed-length string buffer. In fact, in many such cases, the average delay will increase with the use of a fixed-length buffer.

If a system can tolerate only minimal delay, then it may be necessary to use a shift register buffer, even if the input consists of independent strings. Recall that, every time unit, the shift register buffer sends out the most recent $N$ inputs for processing, thereby introducing no additional delay but a great deal of redundancy, which may result in a high level of activity, perhaps restricting the set of feasible decision strategies.

The performance of a system with buffer storage must be defined by considering the ideal response of the system not only to the present actual input but also to the previous actual inputs upon which it is statistically dependent; i.e., for systems using any of the three buffers models discussed in this chapter,

$$y'_t \equiv L[x'(t), x'(t-1), \ldots, x'(t-N+1)] \quad (4.42)$$

The cost function $d$ may also include information about any delay the buffer introduces. So if $y'_t$, the system response to the received input $[x(t), x(t-1), \ldots x(t-N+1)]$, actually occurs at time $t + \Delta t$, then the cost function $d$ may also be dependent on $\Delta t$, i.e.

$$d = d(y'_t, y'_t, \Delta t) \quad (4.43)$$
On the other hand, delay may be included as another constraint on the system, i.e. it may be required that

\[ \Delta t \leq D_{\text{max}} \]  
(4.44)

Some examples will now be given of the types of decisionmaking systems which can be modeled as having these three types of buffers. Machines, such as computers, often have fixed-length algorithms: that is, some machine algorithms may be able to process only strings of one length N. Therefore, while they may accommodate either a shift register or a fixed-length string buffer, they may not be able to accommodate a variable-length string buffer. On the other hand, many human decisionmaking tasks may be represented with a variable-length string buffer. For instance, a doctor who diagnoses many people in one day may be modeled as receiving variable-length independent tasks, since the amounts of information received from various patients will differ widely. Because of the delay considerations inherent in the military environment, and because of the Markovian nature of the information received there, it appears that military commanders are best modeled as decisionmaking systems with shift register buffers. As soon as new information is received, it, along with information received a certain amount of time into the past, (the past 24 hours, for example) are processed to arrive at the appropriate response to the new information. Yet this is not quite the correct picture of this situation. If new information is received every hour, it is probably not retained in storage in the same form it was received in: it may be that only a statistic of the information is retained; or the information may even be combined with data from the past and then stored. Clearly, a model of a different type of storage than buffer storage is necessary for this situation: the algorithms must be able to operate on the received information and then store it away for use in future decision-making. This type of storage will be explored in Chapter 6.

In the next chapter, a quantitative example of a buffer storage system will be given, with an examination of performance criteria.
CHAPTER 5
BUFFER STORAGE: A COIN-TOSSING EXAMPLE

5.1 INTRODUCTION AND ASSUMPTIONS

The example to be presented in this chapter is a relatively simple one, in order to allow the information theoretic analyses to be somewhat straightforward. However, this simple example does illustrate some rather general results; the range of problems to which these results might apply is discussed at the end of the chapter. The example is formulated as a hypothesis-testing problem as follows.

New coins (quarters, say) are received by a quality-control center, and a decision must be made as to whether the coins are fair, or biased towards heads, by tossing each one a certain number of times. If it is decided that a coin is biased, it is discarded. With the probability of heads represented by \( p \), one of two hypotheses, \( H_0 \) and \( H_1 \), must be chosen for each coin:

\[
H_0: \quad p = 0.5 \\
H_1: \quad p = 0.6
\] (5.1)

It is known that the coins are nine times more likely to be fair than to be biased towards heads, i.e.

\[
\text{prob}(H_0) = 0.9 \\
\text{prob}(H_1) = 0.1
\] (5.2)

There are two types of errors that may be made in making the decision as to whether the coin is fair. A type I error is deciding the coin is biased when it is actually fair; and a type II error is deciding the coin is fair when it is actually biased. The cost of letting a bad coin into circulation
is assumed to be nine times as much as the cost of discarding a good coin.

\[
\text{Cost of type I error} = \$0.25 \\
\text{Cost of type II error} = \$2.25
\] (5.3)

The performance of the decisionmaking system is measured by the expected value of the cost of testing a single coin. The lower the expected cost is, the better the performance of the system. If \( \alpha \) is defined as the probability of making a type I error, and \( \beta \) the probability of a type II error,

\[
\begin{align*}
\alpha &= \text{prob}\ (\text{decide } H_1 | H_0 \text{ correct}) \\
\beta &= \text{prob}\ (\text{decide } H_0 | H_1 \text{ correct})
\end{align*}
\] (5.4)

then the expected value of the cost \( C \) is given by

\[
E(C) = (0.25) P(H_0) (\alpha) + (2.25) P(H_1) (\beta)
= 0.225 (\alpha + \beta)
\] (5.5)

The system designer would like to minimize the expected cost of each test, subject to constraints, by choosing one of several similar systems. The basic model for all of these systems is shown in Figure 5.1.

![Diagram of Decisionmaking System with Buffer](image-url)

**Figure 5.1.** Model of System with Buffer for Coin-Tossing Example
For each coin, the system receives N values of x, where x is 0 if a toss of the coin comes up tails, and 1 if it comes up heads. Rather than make a decision as to the coin's fairness based on one toss, this system stores N dependent pieces of information about the coin in a fixed-length string buffer, and then sends these out in the form of an N-vector, $\mathbf{q}$, to the decisionmaking algorithm. The single situation assessment (SA) algorithm, f, adds the values of the components of $\mathbf{q}$ to produce z, i.e. $z$ is the number of heads in N tosses:

$$z = \sum_{i=1}^{N} q_i$$  \hspace{1cm} (5.6)

The response selection (RS) algorithm, h, compares z to some threshold, determined by the performance criterion. If z is less than the threshold, $H_0$ is accepted and the coin goes into circulation; otherwise $H_0$ is rejected and the coin is discarded. The choice that the designer has is in what the value of N should be. Two cases will be analyzed here, that of N = 2 and that of N = 10. For each case, the minimum expected cost will be found (by the selection of the correct threshold value in h), and the total rate of activity of the decisionmaking system will be calculated.

5.2 ANALYSIS: COST AND ACTIVITY

First consider the case of N = 2. In order to minimize the expected cost due to error, it is necessary to minimize the quantity $\alpha + \beta$ (see Eq. (5.5)). The probability distributions for z, the total number of heads in two tosses, under both hypotheses, are given below:

| z | $p(z|H_0)$ | $p(z|H_1)$ |
|---|------------|------------|
| 0 | 0.25       | 0.16       |
| 1 | 0.50       | 0.48       |
| 2 | 0.25       | 0.36       |  \hspace{1cm} (5.7)
To determine the threshold value, \( T \), that minimizes \( \alpha + \beta \), consider the following:

\[
\begin{array}{|c|c|c|c|}
\hline
T & \alpha = \text{prob(}z \geq T\mid H_0\text{)} & \beta = \text{prob(}z < T\mid H_1\text{)} & \alpha + \beta \\
\hline
0 & 1.0 & 0.0 & 1.0 \\
1 & 0.75 & 0.16 & 0.91 \\
2 & 0.25 & 0.64 & 0.89 \\
3 & 0.0 & 1.0 & 1.0 \\
\hline
\end{array}
\]

From this, it is clear that the threshold value should be chosen to be 2, in which case the expected cost is given by

\[
E(C) = 0.225 \ (\alpha + \beta) \\
= 0.225 \ (0.89) \\
= 0.20025 \ \text{dollars/coin} \quad (5.9)
\]

For a fixed-length string buffer, the total activity of the buffer is given by Eq. (4.25):

\[
F_{\text{buffer}} = \bar{H}(x) + \sum_{i=1}^{N} \bar{H}(b_i) \\
= \bar{H}(x) + (0.5) \sum_{i=1}^{N} \bar{H}(x_i) \quad (5.10)
\]

The entropy of the source, \( \bar{H}(x) \), is most easily found as follows. Every two time units (assumed to be seconds from here on) when a new coin arrives, strings of length 2 are generated independently, with the following probabilities:

\[
p(00) = p(00\mid \text{fair}) \ p(\text{fair}) + p(00\mid \text{biased}) \ p(\text{biased}) \\
= (0.25)(0.9) + (0.16)(0.1) \\
= 0.241 \\
p(01) = 0.249 \\
p(10) = 0.249 \\
p(11) = 0.261
\quad (5.11)
Therefore, the average uncertainty in $x$ is one-half the uncertainty generated every two seconds, or:

$$
\overline{H}(x) = -0.5\left[(0.241)\log_2(0.241) + (0.498)\log_2(0.249) + (0.261)\log_2(0.261)\right] \\
= 0.999707 \text{ bits/sec.} \quad (5.12)
$$

The marginal distributions for the $i$-th letter of a string are the same for all values of $i$, specifically,

$$
p(x_i=0) = p(0|\text{fair}) \cdot p(\text{fair}) + p(0|\text{biased}) \cdot p(\text{biased}) \\
= (0.5)(0.9) + (0.4)(0.1) \\
= 0.49 \\
p(x_i=1) = 0.51 \\
\{i=1,2\} \quad (5.13)
$$

so $H(x_i)$ is found to be 0.999711, and the total rate of activity of the buffer is found to be

$$F_{\text{buffer}} = 1.99418 \text{ bits/sec.} \quad (5.14)$$

In order to find the rate of activity for the rest of the system, the internal variables of the two algorithms must be defined. The SA algorithm contains the variable $z$ which is defined to be the sum of $q_1$ and $q_2$. The RS algorithm contains one variable $w_1$ which stores the threshold value (2 in this case), one comparison variable $w_2$ that compares $w_1$ to $z$ and takes the value 1 if $z \geq w_1$ and 0 otherwise, and the output variable $y$, which takes the value "discard" or "circulate." Specifically,

$$z = q_1 + q_2$$

$$w_1 = 2$$

$$w_2 = \begin{cases} 
1 & \text{if } z \geq w_1 \\
0 & \text{if } z < w_1 
\end{cases}$$

$$y = \begin{cases} 
\text{discard} & \text{if } w_2 = 1 \\
\text{circulate} & \text{if } w_2 = 0 
\end{cases} \quad (5.15)$$
The total rate of activity of the rest of the system may be found as the sum of the rates of uncertainty of all the individual variables of the algorithms, i.e.,

\[ F_{\text{algorithms}} = \overline{H}(z) + \overline{H}(w_1) + \overline{H}(w_2) + \overline{H}(y) \]  

(5.16)

Since \( w_1 \) is constant, \( \overline{H}(w_1) \) is zero. The other three variables take on values independently every two seconds, so their entropy rates are one-half of the entropies calculated using these probability distributions:

\[
\begin{align*}
    p(z=0) &= 0.241 & p(w_2 = >) &= 0.261 & p(y=\text{discard}) &= 0.261 \\
    p(z=1) &= 0.498 & p(w_2 = <) &= 0.739 & p(y=\text{circulate}) &= 0.739 \\
    p(z=2) &= 0.261 &
\end{align*}
\]

(5.17)

The entropy rates are found to be

\[
\begin{align*}
    \overline{H}(z) &= 0.7507 \text{ bits/sec.} \\
    \overline{H}(w_2) &= \overline{H}(y) = 0.4141 \text{ bits/sec.} \\
    F_{\text{algorithms}} &= 1.57896 \text{ bits/sec.}
\end{align*}
\]

(5.18)

and the activity of the overall system is then

\[
\begin{align*}
    F &= F_{\text{buffer}} + F_{\text{algorithms}} \\
    &= 1.99418 + 1.57896 \\
    &= 3.57314 \text{ bits/sec.}
\end{align*}
\]

(5.19)

Now consider the case of \( N = 10 \). Note that, with coins arriving once every 2 seconds, the coin must be tossed five times as fast, so that input \( x \) now arrives at the rate of one every 0.2 seconds. The minimum
The expected cost is again obtained by minimizing $C + \beta$. With $N = 10$ the probability distributions for $z$ are:

$$
\begin{align*}
 p(z = z_0 | H_0) &= \binom{10}{z_0} (0.5)^{10-z_0} (0.5)^{z_0} \\
 p(z = z_0 | H_1) &= \binom{10}{z_0} (0.6)^{z_0} (0.4)^{10-z_0}
\end{align*}
$$

$z_0 = 0, 1, \ldots, 10$

and the threshold value for this case is 6, i.e.

$$
\begin{align*}
 z > 6: & \text{ decide } H_1 \\
 z < 6: & \text{ decide } H_0
\end{align*}
$$

(5.21)

giving an expected cost of

$$
E(c) = 0.225(C + \beta)
= 0.225[\text{prob } (z > 6 | H_0) + \text{prob } (z < 6 | H_1)]
= 0.1674 \text{ dollars/coin}
$$

(5.22)

It is again necessary to find the activity of the buffer, and therefore first to find $\overline{H}(x)$, which may be computed as follows:

$$
\overline{H}(x) = -(0.5) \sum_{k=0}^{10} \binom{10}{k} \left[ (0.9)^{(0.5)^{10-k}} + (0.1)^{(0.6)^{10-k}} \right]
\cdot \log_2 \left[ (0.9)^{(0.5)^{10-k}} + (0.1)^{(0.6)^{10-k}} \right]
$$

(5.23)

The buffer variables take on values according to the same distribution and at the same rate as the $N=2$ case, so

$$
\overline{H}(b_i) = (0.5)H(x_i)
= (0.5)(0.999711)
= 0.49986 \text{ bits/sec.}
$$

(5.24)
and from Eq. (5.10),

\[
F_{\text{buffer}} = 4.9983 + (10)(0.49986) \\
= 9.99686 \text{ bits/sec.} \tag{5.25}
\]

The RS algorithm has the same structure as in the N=2 case, but the threshold variable now has the value 6. With only binary operations allowed, the SA algorithm requires many more variables in order to add together all 10 components of \( q \). Specifically, the algorithm variables are defined as follows, where this implementation is the most activity-efficient [19]:

\[
\begin{align*}
\text{SA} & \\
\quad w_1 &= q_{(2i-1)} + q_{2i} \quad i=1,2,3,4,5 \\
\quad w_6 &= w_1 + w_2 \\
\quad w_7 &= w_3 + w_4 \\
\quad w_8 &= w_5 + w_6 \\
\quad z &= w_7 + w_8
\end{align*}
\]

\[
\begin{align*}
\text{RS} & \\
\quad w_9 &= 6 \\
\quad w_{10} &= \begin{cases} 
1 & \text{if } z > w_9 \\
0 & \text{if } z < w_9
\end{cases} \\
\quad y &= \begin{cases} 
\text{discard} & \text{if } w_{10} = 1 \\
\text{circulate} & \text{if } w_{10} = 0
\end{cases}
\end{align*}
\tag{5.26}
\]

The variables \( w_1 \) through \( w_5 \) have the same rate of uncertainty as \( z \) in the N=2 case:

\[
\overline{H}(w_i) = 0.7507 \quad i=1,2,\ldots,5 \tag{5.27}
\]
Variables $w_6$ and $w_7$ are both the sum of four components of $g$, $w_8$ is the sum of six components, and $z$ is the sum of ten components. Their entropy rates are found as follows:

$$\bar{H}(a) = -(0.5) \sum_{k=0}^{n} \binom{n}{k} (0.9)^{n-k} (0.5)^k (0.4)^{(n-k)}$$

\[\cdot \log_2 \binom{n}{k} (0.9)^{n-k} (0.5)^k (0.6)^{(n-k)}\]

for $a = w_6, w_7$ & $n = 4$
$a = w_8$ & $n = 6$
$a = z$ & $n = 10$

$$\bar{H}(w_6) = \bar{H}(w_7) = 1.01859 \quad \bar{H}(w_8) = 0$$

$$\bar{H}(w_9) = 1.17268 \quad \bar{H}(w_{10}) = \bar{H}(y) = 0.486226$$

$$\bar{H}(z) = 1.36452$$

$$F_{\text{algorithms}} = 9.3003 \text{ bits/sec.} \quad (5.28)$$

and the activity of the entire system is now

$$F = F_{\text{buffer}} + F_{\text{algorithms}}$$

$$= 9.99686 + 9.3003$$

$$= 19.2972 \text{ bits/sec.} \quad (5.29)$$

5.3 RESULTS

Now compare the results of the two analyses with $N=2$ and $N=10$. First, the system with the larger buffer gives a lower expected cost per coin tested, i.e. its performance is higher. Although this is not a general result for any experiment, it is true in many situations that the more information about a single task (i.e. dependent information) that can be
processed together, the better the performance of the system. In fact, for this example performance improves monotonically as the amount of memory in the system, i.e. N, increases [20]. Note that if each coin is tossed ten times and the system with N=2 is used, its performance remains exactly the same: even though a lot of information about a single coin is now available, the system simply does not have enough memory to take advantage of it. Second, the system with the larger buffer has a much higher rate of activity than the system with N equal to 2. For this example, increasing the value of N always results in more activity (because of the extra variables added to the buffer and the algorithms), i.e. the activity rate increases monotonically with N. If the system designer has available for use systems with any value of N, then he will minimize the expected cost of the system subject to the maximum rate of activity constraint (assuming that this constraint has the same value for all systems), by choosing the system with the largest value of N that still satisfies this constraint.

But consider the designer’s options if he must choose between the two systems analyzed here. If the rate of activity of the system with N=10, given in Eq. (5.29), is less than the maximum allowable, then this system will certainly be chosen, since it gives the lower expected cost. However, if 19.3 bits per second exceeds the limit of the system, then the designer has at least two options open to him. He may decide to accept a poorer performance and use the system with N=2. Or he may decide to use more than one system with N=10 in an organizational design known as alternate processing [2]; i.e. each of K systems receives a coin to test every 2K seconds, as opposed to every 2 seconds. For example, if the rate of activity constraint of the system with N=10 is 10 bits per second, then the organization designer may decide to use two of these systems in parallel. Coins are then sent for testing to each of these systems alternately, so that each can now operate at a rate of activity or tempo that is exactly half of that given in Eq. (5.29) and just within the limit.
This solution may, unfortunately, introduce another problem, that of delay. In both of the original systems with N=2 and N=10, a coin arrives at the system, and within two seconds a decision is made as to whether that coin will be circulated or not. However, in the alternate processing organization with two systems, a coin arrives at a given system and a decision as to the quality of the coin may not be made for four seconds. This may not seem crucial, particularly since decisions on individual coins are still occurring at the rate of one every two seconds; but consider that when the process is just starting up, the first decision is not made until four seconds have passed, and then consider the implications this may have if it is aircraft that are being tested or cars coming off of an assembly line rather than coins that are being tossed. In fact, the testing of aircraft or cars is just the type or problem that this coin-tossing example is meant to illustrate. The same kinds of trade-offs must be made between performance (a heavy cost is exacted if a faulty vehicle passes inspection), activity constraints (even with many people working on one vehicle, there is a limit to how fast the testing can be done, if a certain performance level is to be achieved), and delay (United doesn't want to wait forever to get its first 757 and GM wants to get its cars out as quickly as possible).

In the next chapter, a different type of storage will be modeled. This model incorporates memory into the decisionmaking algorithms themselves, as described briefly at the end of Chapter 4.
CHAPTER 6
PERMANENT AND TEMPORARY MEMORY

6.1 INTRODUCTION

The second type of storage that may exist in a decisionmaking system is called, for the purposes of this thesis, memory. Memory consists of both permanent and temporary stores of information which may be drawn upon by the situation assessment and the response selection algorithms during the decisionmaking process. Permanent memory contains values which are constant; that is, although the information stored in permanent memory may be accessed by the algorithms, it may not be revised or appended by them. On the other hand, temporary storage contains values which may be revised by the algorithms; for example, a discrete Kalman filter algorithm would include temporary storage of the best estimate of the present state of the process, to be used in the next iteration of the algorithm. Permanent memory therefore has the effect of allowing the algorithms to access constant values or unchanging information. Temporary memory has the effect of adding memory to the algorithms themselves; with temporary memory available, the algorithms can remember values from one iteration to the next.

The division of memory into permanent and temporary bears a strong resemblance to the division of memory that is made in the cognitive sciences, into long-term and short-term memory [21]. (Actually, a third type of memory, called sensory memory, is also hypothesized by psychologists. Information from the environment is stored in sensory memory before it undergoes any processing; sensory memory might therefore be compared to the buffer storage model presented in Chapter 4.) The model of permanent memory presented here is similar to long-term memory, in that information is stored in both indefinitely and is accessible by information processing mechanisms. The model of temporary memory is similar to short-term memory, in that information stored in both may be revised and accessed by the information-processing mechanisms. There are also some notable differences between the two
sets of models. First, new information is being added continuously to long-term memory; the permanent memory model provides no mechanism for this addition. Second, information may be lost from long-term memory; the permanent memory model does not have a forgetting mechanism. Third, short-term memory seems to have a definite capacity of about seven "chunks" of information; no such explicit capacity constraint has been applied to the temporary memory model. These differences are noted to indicate that, although similarities exist between the model of memory presented here and that found in the cognitive sciences, permanent and temporary memory are not intended to be models of long-term and short-term memory per se.

Permanent and temporary memory may be accessed by both the situation assessment and the response selection algorithms. However, in this chapter, consideration will be limited to the first set, the situation assessment algorithms. The relationships derived for these algorithms are the same as they would be for the response selection algorithms, since the two halves of the decisionmaking process are structurally identical.

It is quite possible that a decisionmaking system may contain both buffer storage and permanent and temporary memory units. However, in order to simplify the analysis of memory, the assumption is made that the decisionmaking system contains no buffer storage, although dependence may exist between sequential inputs. At the end of this thesis, a discussion of decisionmaking systems with both buffer storage and memory will be presented.

The models for temporary and permanent memory are essentially the same, and the basic model for both is shown in Figure 6.1.
Figure 6.1. Model of SA Subsystem with Memory

The memory unit consists of M variables, \( d_1 \) through \( d_M \), as well as an input M-vector, \( D_I \), and an output M-vector \( D_O \). Note that because permanent memory may not be revised, its model will not contain the input vector \( D_I \).

6.2 PERMANENT MEMORY

The purpose of permanent memory is to allow the decisionmaking algorithms access to fixed information during the decisionmaking process. Note that the term permanent here only applies to the period for which this decisionmaking model is defined; that is, the information is permanent once the system enters the steady-state or can be defined as well-trained.

At first glance, it might seem that the addition of permanent
memory to a decisionmaking system might have no effect at all on the
total information theoretic rate of activity of the system: if the
values of \( d_k \) for \( k = 1,2,\ldots,M \) do not change over time, then

\[
\overline{H(d_k)} = 0 \quad k = 1,2,\ldots,M
\]  \( \text{(6.1)} \)

Since total activity is just the sum of the entropies of the individual
variables in the system, it appears that the addition of \( M \) deterministic
variables to a system should have no effect on its total activity.
However, the problem is actually more complex. In order to demonstrate
the types of changes that occur when permanent memory is added to the
model, a particularly simple example will be analyzed.

It is assumed that the permanent memory unit consists of one
variable, \( d_1 \), which may be accessed by one SA algorithm, \( f_1 \), as
shown in Figure 6.2.

![Diagram](Image)

**Figure 6.2.** Example of SA Subsystem with Permanent Memory
Algorithm \( f_1 \) provides the average value of the two components of a vector input. The specific values that the algorithm's internal variables may take will be described shortly, and a discussion of how the algorithm would be structured if the permanent memory unit \( d_1 \) were not available will be presented. First, however, a brief discussion of active and inactive algorithm variables is necessary [7].

Whenever a specific algorithm is accessed by the decisionmaking system, the variables of that algorithm are defined to be active, and take values according to some probability distribution which is a function of the input. When that algorithm is not accessed, its variables are defined to be inactive: that is, they assume some fixed value which they may not assume when they are active. For example, if variable \( s \) were equally likely to assume the values 5 and 6 when algorithm \( 1 \) was accessed, its probability distribution would be:

\[
\text{prob}(s=k) = \begin{cases} 
\frac{1}{2} \text{[prob}(u=1)] & k = 5, 6 \\
\text{prob}(u \neq 1) & k = \Box 
\end{cases} \quad (6.2)
\]

where \( \Box \) represents the fixed inactive value.

Now consider algorithm \( f_1 \), which provides the mean value of the two components of a vector input. There are two similar ways of implementing this algorithm. The first does not access permanent memory, although there is some implicit memory in the algorithm itself:

\[
\begin{align*}
  w_1^1 & = 2 \\
  w_2^1 & = x_1^1 + x_2 \\
  w_3^1 & = \frac{w_2^1}{w_1^1} \\
  z & = w_3^1 \\
\end{align*}
\]

algorithm \( f_{1A} \) \hfill (6.3)

The second does access permanent memory:
\[ d_1 = 2 \] \quad \text{defined outside of algorithm } f_{1B} 

\begin{align*}
w_1^1 &= d_1 \\
w_2^1 &= x_1 + x_2 \\
w_3^1 &= \frac{w_2^1}{w_1^1} \\
z &= w_3^1
\end{align*}

(6.4)

In the second example, all of the variables in algorithm \( f_{1B} \) \((w_1^1, w_2^1, w_3^1)\) are inactive when the algorithm is not accessed. In the first example, only the variables \( w_2^1 \) and \( w_3^1 \) are inactive when the algorithm is not accessed: \( w_1^1 \) must retain the value of 2 throughout, since no means have been provided to reinitialize its value each time the algorithm is accessed.

Continuing with this example, it is now possible to compare the levels of activity of the system with the permanent memory unit and that without. First consider realization \( f_{1A} \). The throughput, blockage, and noise rates of the SA subsystem are given by (see Appendix A.2):

\[
\begin{align*}
F_t &= \overline{H}(z) - \overline{H}(z) \\
F_b &= \overline{T_z} \left( x; w_1^1, w_2, \ldots, w_{U}^1 \right) \\
F_n &= \overline{H}(u)
\end{align*}
\]

(6.5)

With inputs arriving once every second, the coordination rate is found as follows:

\[
F_c = \sum_{i=1}^{U} \sum_{j=1}^{U} \overline{H(w_i^j)} + \overline{H(u)} + \overline{H(z)} - \overline{H(u,w,z)}
\]

(6.6)

Here, \( w_i^j \) represents the j-th variable of algorithm \( i \); and \( W \) represents the entire set of \( w_i^j \) in the SA subsystem, i.e.
\[ W = \{ w_1^1, w_2^1, \ldots, w_{a_1}^1, w_1^2, \ldots, w_{a_2}^U \} \] (6.7)

Finally, \( a_i \) is the number of internal variables of algorithm \( i \) which are active or inactive according to the value of \( u \); e.g., \( a_{1A} \) is equal to 2. Equation (6.6) may be reduced to (see Appendix A.2):

\[ F_c = \sum_{i=1}^{U} a_i H(p(u)=i) + \sum_{i=1}^{U} a_i \sum_{j=1}^{\alpha_i} \overline{H_u(w_j^i)} - \overline{H_u(W)} + \overline{H(z)} \] (6.8)

The symbol \( H \) denotes the binary entropy of its argument, given by:

\[ H(p) = -p \log_2 p - (1-p) \log_2 (1-p), \quad 0 \leq p \leq 1 \] (6.9)

Using algorithm \( f_{1B} \) in place of \( f_{1A} \), the same quantities may be calculated. The rates of throughput, blockage, and noise are not affected by the small structural difference in algorithm \( f_{1B} \); the rate of coordination does change. Consider Eq. (6.8): \( a_{1B} \) is now equal to 3, because \( w_1^1 \) is now active when \( u = 1 \) and inactive otherwise. Therefore, the first term of Eq. (6.8) is increased by the amount \( H(p(u=1)) \). The second and third terms remain the same: even though there is now some uncertainty associated with the value of \( w_1^{1B} \), knowledge of the value of \( u \) resolves that uncertainty, so that

\[ \overline{H_u(w_1^{1B})} = 0 \] (6.10)

Similarly, \( \overline{H_u(W)} \) is unchanged. Only the structure of the algorithm has been changed, so the output remains the same, and the last term of Eq. (6.8) is unchanged. Therefore, the addition of one unit of permanent memory to the SA subsystem provides a total increase in activity of \( H(p(u=1)) \). Generalizing this result, if algorithm \( i \) directly accesses \( \beta_i \) values from permanent memory, and no other changes are made in the algorithms, then the additional activity of the system, \( \Delta F \), is given by

\[ \Delta F = \sum_{i=1}^{U} \beta_i H(p(u=i)) \] (6.11)
Since permanent memory increases the overall system activity without creating any change in the throughput of the system, one might ask why permanent memory would be a desirable feature. First, instead of having to store fixed information that all of the algorithms require within every algorithm, that information can be stored in one place. While the amount of activity of the system increases, the total amount of storage required for the fixed information decreases substantially. Because there may be a cost associated with this storage, it is conceivable that the addition of permanent memory could improve the overall performance of the system. Second, permanent memory makes individual algorithms much more flexible. By switching between sets of data stored in permanent memory, an algorithm may be substantially changed: although its structure will remain the same, any number of parameters may be altered, which may significantly affect the output of the algorithm. In the next chapter, an example is presented in which switching between two sets of stored data in the execution of two different tasks is analyzed and the effect this has on performance is examined.

6.3 TEMPORARY MEMORY

Temporary memory provides a storage place for information derived by the decisionmaking algorithms. This information may be retrieved by the algorithms at any iteration subsequent to storage, for use in the decisionmaking process. A general analysis of temporary storage may not be made, because the amount of information theoretic activity associated with temporary memory may be heavily dependent on the particular information that is stored, and on the number of iterations it remains in storage before it is accessed and/or replaced. A specific example will be analyzed here to demonstrate the type of result one might expect to receive; then a discussion of the implications of the addition of temporary memory to a system will be presented.

In this example, all of the situation assessment algorithms have access to a temporary storage unit D, which is made up of M storage variables, \( d_1 \) through \( d_M \) (see Figure 6.1). It is assumed that the
algorithms are deterministic; but the values of the set of internal variables $W$ are determined not only by the values of $u$ and the input $x$, but also by the present values of the storage variables. After the value of $z$ has been determined at each iteration, the information in temporary memory is replaced by new information which is completely determined by the present values of the internal variables of the system. Specifically, at time $t$, with inputs arriving once every second,

$$
\begin{align*}
&W(t) \quad \text{determined by } u(t), z(t), \text{ and } D(t-1) \\
&z(t) \quad \text{determined by } W(t) \\
&D(t) \quad \text{determined by } W(t) \text{ and } z(t)
\end{align*}
$$

(6.12)

Note that, since $W(t)$ determines $z(t)$, the last expression of Eq. (6.12) can be stated

$$
D(t) \quad \text{determined by } W(t)
$$

(6.13)

As in the case of permanent memory, the throughput, blockage, noise, and coordination of the situation assessment subsystem with temporary memory may be found.

The throughput and blockage have the same form as in the permanent memory example:

$$
\begin{align*}
F_t &= \overline{H}(z) - \overline{H}_x(z) \\
F_b &= \overline{T}_z(x;W)
\end{align*}
$$

(6.14)

Note that sequential values of $z$ are related not only by dependence that exists in the input, but also through the values stored in $D$. Even if sequential inputs were statistically independent, sequential values of $z$ could be dependent. The internally-generated information is given by

$$
F_n = \overline{H}_x(u,W,z,D)
$$

$$
= \overline{H}_x(u) + \overline{H}_{xu}(D) + \overline{H}_{xuD}(W) + \overline{H}_{xuDW}(z)
$$

(6.15)
The values of \(x, u,\) and \(D\) determine the values of \(W,\) and the values of \(W\) determine the value of \(z,\) so the last two terms of Eq. (6.11) are equal to zero. Also, the value of \(u\) is assumed to be independent of the input, so the first term is just \(\overline{H}(u)\). Finally, the second term is also equal to zero which may be seen as follows:

\[
\overline{H}_{xu}(D) = \lim_{m \to \infty} \frac{1}{m} \left\{ \overline{H}_{x} [D(t)] + \overline{H}_{xu} [D(t+1)] + \ldots + \overline{H}_{xuD} [D(t+m-1)] \right\}
\]

(6.16)

Here, the vectors \(x, u,\) and \(D\) represent the values of the variables \(x\) and \(u\) from time \(t\) until time \(t+m-1,\) and the values of the variable \(D\) from time \(t\) until time \(t+m-2.\) The values of \(x(t), u(t),\) and \(D(t)\) determine the values of \(W(t);\) and the values of \(W(t)\) determine the values of \(D(t+1);\) therefore, all but the first term of Eq. (6.16) are equal to zero. In the limit, then

\[
\overline{H}_{xu}(d) = 0
\]

(6.17)

and

\[
F_n = \overline{H}(u)
\]

(6.18)

Finally, the rate of coordination of the subsystem is given by

\[
F_c = \sum_{i=1}^{U} \sum_{j=1}^{L} \overline{H}(w_i^j) + \overline{H}(u) + \sum_{k=1}^{M} \overline{H}(d_k) + \overline{H}(z) - \overline{H}(W,u,D,z)
\]

(6.19)

The last term of the above expression decomposes as follows:

\[
\overline{H}(W,u,D,z) = \overline{H}(W) + \overline{H}_w(u) + \overline{H}_u(z) + \overline{H}_{uw}(D)
\]

(6.20)

The last two terms of Eq. (6.20) are equal to zero, because the values of \(W\) determine the value of \(z\) and the values of \(D.\) The second term of Eq. (6.20) is also equal to zero, since knowledge of the values of all the internal variables provides exact information as to which algorithm is being used, i.e., to the value of \(u.\) This reduces Eq.(6.19) to:
\[
F_c = \sum_{i=1}^{U} \sum_{j=1}^{\alpha_i} \bar{H}(w^i_j) + \bar{H}(u) + \sum_{k=1}^{M} \bar{H}(d_k) + \bar{H}(z) - \bar{H}(W)
\]  

(6.21)

Because of the introduction of the temporary memory unit, there may now be dependence either between sequential values of the \(w^i_j\), if algorithm \(i\) is accessed twice in a row, or between sequential values of the internal variables of algorithms \(i\) and \(k\) if use of algorithm \(i\) is followed by use of algorithm \(k\). Therefore, further decomposition of Eq. (6.21) is somewhat meaningless; however, the total coordination term may be expressed in a more significant form. Recall the decomposition rule for \(N\)-dimensional transmission first presented in section 2.2.

Considering all of the situation assessment algorithms (including the variables \(u, z\), and all of the \(w^i_j\), but not \(D\)) to be subsystem \(A\), and considering the temporary storage unit to be subsystem \(D\), the coordination term may be expressed:

\[
F_c = F_c(A) + F_c(D) + \bar{T}(A:D) \\
= [\sum_{i,j} \bar{H}(w^i_j) + \bar{H}(u) + \bar{H}(z) - \bar{H}(W,u,z)] + [\sum_k \bar{H}(d_k) - \bar{H}(D)] \\
+ [\bar{H}(W,u,z) + \bar{H}(D) - \bar{H}(W,u,z,D)]
\]  

(6.22)

To see what the coordination between the temporary storage unit and the algorithm subsystem is, expand the last bracketed expression above:

\[
\bar{T}(A:D) = [\bar{H}(W,u,z) + \bar{H}(D) - \bar{H}(W,u,z) - \bar{H}_{Wuz}(D)]
\]  

(6.23)

The uncertainty in \(D\), given the values of \(W\) and \(z\) is zero, so the last term of Eq. (6.23) is zero, giving

\[
\bar{T}(A:D) = \bar{H}(D)
\]  

(6.24)

for the coordination between subsystems. After substitution of Eq. (6.24) into Eq. (6.22), a comparison may be made with Eq. (6.21) since the two methods of finding total coordination should yield the
same results. This is found to be the case, upon noting that

$$\overline{H}(W,u,z) = \overline{H}(W) + \overline{H}_W(u) + \overline{H}_{wu}(z)$$  \hspace{1cm} (6.25)

and that the last two terms of Eq. (6.25) are zero.

Temporary memory may be very useful for particular types of processing. Consider the issues presented in section 4.3. It was noted there that a shift register buffer is useful for Markovian input, but that this buffer provides a great deal of redundancy, since every symbol is actually processed N times. Rather than processing every input along with those previous ones upon which it is statistically dependent, it may be sufficient to retain a statistic of the previous N-1 inputs in temporary memory, and access this statistic when processing the N\textsuperscript{th} input. This could substantially reduce the total rate of activity of the system. Note that in the example presented in this section, the temporary memory unit is actually behaving as a single-unit shift register. That is, each second, the information in the shift register is sent out to be used with the input in the decisionmaking process; then that information is replaced by a new set of information to be used in the next iteration of the process. Temporary memory can also be implemented as a multi-unit shift register. The stored information is again used by the algorithms every second; then, one new piece of information is stored, and the oldest piece discarded. There are, of course, many possible and useful implementations of temporary memory. In fact, temporary memory seems to be used frequently in tactical encounters, when a commander will decide on a course of action, based not only on what the enemy has just done, but also on the enemy's movements and responses in the recent past.

In the next chapter, a specific example of a decisionmaking system with permanent memory will be presented.
CHAPTER 7
PERMANENT MEMORY: THE DUAL-TASK PROBLEM

7.1 INTRODUCTION AND ASSUMPTIONS

It has been observed that if a person must execute two tasks by switching between them, his level of performance may be different than when he is allowed to confine himself to one task [22], [23], even if the arrival rate for individual tasks is the same for both cases. If there is some synergy between the two tasks—that is, if the two tasks are related and executing one actually helps the execution of the other—then performance may improve. If, on the other hand, the two tasks are dissimilar or simply do not reinforce each other, performance may decline from what it was in the single-task case. It is this latter phenomenon that will be explored by this chapter. The dual-task problem will be defined in the context of the decisionmaking system with memory modeled here, and then analyzed to determine the effect that executing two non-synergistic tasks has on system performance. Finally, some suggestions will be made as to what experiments might be carried out as quantitative tests of the model's predictions.

Even in the context of the proposed model, there are numerous possible ways in which the dual-task problem might be modeled. For example, if the two tasks to be performed are assumed to be so different from each other that they demand different sets of algorithms, then a pre-processor may be required for the system. The pre-processor determines which type of task each input is, and then allows access to a set of decisionmaking algorithms appropriate to that task. Of course, the activity of the pre-processor increases overall system activity and may, therefore, lower performance [24]. On the other hand, if the two tasks to be performed are assumed to be similar but non-synergistic, they may be able to use the same basic set of algorithms, as long as these algorithms are adaptable to each task through two different sets of parameter values stored in permanent memory. Notice that there is
an implicit need for a pre-processor in this problem as well, since the
algorithms must have some way of knowing which type of input has been
received in order to determine which set of values stored in memory to
access. Overall activity is increased in this formulation as well, by
the necessity of switching between sets of information. An example of
this second problem is a hotel switchboard operator who has to process
both incoming and outgoing calls; although the tasks require the same
basic action, they differ with respect to the information required to
execute the tasks. The formulation of the first problem with the pre-
processor is important, but it will not be addressed in this thesis.
Attention will be focused here on the formulation of the dual-task
problem as approached by a decisionmaking system with two alternate
sets of algorithm parameter values stored in permanent memory.

In order to simplify this problem, several assumptions will be
made. First, to circumvent the need for a pre-processor, it is assumed
that there are two separate inputs to the system, \( x_A \) and \( x_B \), which are
members of disjoint alphabets, \( X_A \) and \( X_B \). Only one of these inputs is
active at any given time: if \( x_A \) is active, task A must be performed, and
if \( x_B \) is active, task B must be performed. Inputs arrive at the system
once every second, and there is a probability \( TD \) (representing the task
division) that \( x_A \) will be active at any given time. Note that \( TD \) is a
fixed quantity for any given dual-task problem.

Inputs are assumed to be statistically independent, to model their
non-synergistic nature. Therefore, in the results which follow, entropies
rather than entropy rates will be used, as their use lends an interpre-
tation to the results that would not otherwise be clear. Recall that
if \( x \) is generated independently every \( T \) seconds, then

\[
\overline{H}(x) = \frac{1}{T} H(x)
\]  

(7.1)

The letter \( G \) is used to denote activities, in place of the letter \( F \) for
rates of activity. The units for \( G \) are bits per symbol (as opposed to
bits per second for \( F \)). Note that for the problem with synergy between

-76-
tasks, the assumption of dependence between sequential inputs would probably be made.

The basic model for the problem is shown in Figure 7.1. The variable

\[ u \], which acts independently of the input \( x \), controls which of two situation assessment (SA) algorithms, \( f_1 \) and \( f_2 \), will be accessed. The decision strategy for a system such as this may then be defined by the probability that \( u \) is equal to 1; the symbol \( \delta \) will represent this probability. Notice that there is no response selection subsystem in this example: it is assumed that the purpose of both tasks is merely to assess the situation, so that \( z \) is the output of the decisionmaking system. The variables \( s_1 \) and \( s_2 \) are represented as switches external to the algorithms only so that their function may be understood. Figure 7.1 does not explicitly depict the mechanism by which \( s_1 \) and \( s_2 \) take their values, but they are dependent on the value of \( u \) and on the values of \( x_A \) and \( x_B \).
Specifically, they take values as follows:

\[
    s_i = \begin{cases} 
        A & \text{if } u = i, \ x_A \neq \emptyset, \ x_B = \emptyset \\
        B & \text{if } u = i, \ x_A = \emptyset, \ x_B \neq \emptyset \\
        \emptyset & \text{if } u \neq i
    \end{cases} \quad i = 1, 2
\]

(7.2)

In addition to \( s_1 \) and \( s_2 \), algorithms \( f_1 \) and \( f_2 \) contain \( a_1 \) and \( a_2 \) internal variables. Finally, \( D_A \) and \( D_B \) are the two sets of information or data needed by the algorithms to process input from \( X_A \) and \( X_B \), respectively. It is assumed that both algorithms use all of the information in \( D_A \) when performing task A and all of \( D_B \) for task B.

7.2 INFORMATION THEORETIC ANALYSIS

In order to see if a change in performance level occurs between the single-task and dual-task situations, a measure of this performance is required. Here, the measure that is used is the probability of error in the decisionmaking process, represented by \( J \). In terms of the quantities defined in section 2.1 and depicted in Figure 2.1 (with the output of the system now equal to \( z \)),

\[
    d(z, z') = \begin{cases} 
        1 & \text{if } z \neq z' \\
        0 & \text{if } z = z'
    \end{cases}
\]

(7.3)

and therefore \( J \), the expectation of \( d(z, z') \) is

\[
    J = \langle 1 \rangle \{ \text{prob}(z \neq z') \} + \langle 0 \rangle \{ \text{prob}(z = z') \}
    = \text{prob}(z \neq z')
\]

(7.4)

Because two tasks are being performed, \( J_A \) is defined as the probability of error in executing a type A task; and \( J_B \) as the probability of error in a type B task. More precisely,

\[
    J_i = \text{prob}(z = z'| x \in X_i) \quad i = A, B
\]

(7.5)

Note that these quantities are independent of the task division TD.
the probability that \( x \in X_A \). Note also that they will be dependent in general on the decision strategy \( \delta \), the probability that \( u \) is equal to 1. In fact, if it is known how the system performs when pure strategies are employed (either \( u \) is always 1, and algorithm \( f_1 \) is always used, or \( u \) is always 2, and algorithm \( f_2 \) is always used), the performance of the system under the mixed strategy \( \delta \) (algorithm \( f_1 \) is used with probability \( \delta \)) is simply a convex combination of the performances using pure strategies \([8]\), i.e.

\[
J_i(\delta) = \delta \langle J_i | u = 1 \rangle + (1-\delta) \langle J_i | u = 2 \rangle \quad i = A, B; \quad 0 \leq \delta \leq 1
\]

(7.6)

Note that with this definition of the performance criterion, it is also possible to define an overall performance criterion for the system. The performance of the system may be measured by the probability of making any kind of error:

\[
J(\delta) = (TD)J_A(\delta) + (1-TD)J_B(\delta)
\]

(7.7)

If errors on one task are more detrimental than errors on the other, then each of the terms on the right-hand side of Eq. (7.7) may be weighted appropriately to define the overall system performance measure.

The activity of the system will change both as a function of the decision strategy \( \delta \), and as a function of the task division \( TD \). In fact, \( G \), the total activity of the system, is convex both in \( \delta \) (with a fixed \( TD \)) and in \( TD \) (with \( \delta \) fixed). The convexity of \( G \) in \( \delta \) has already been shown \([8]\), but will be reviewed in the context of finding \( G \) for the single-task case. Then \( G \) will be found for the dual-task case, and the convexity of \( G \) in \( TD \) will be demonstrated.

First assume that only task \( A \) is being performed. Note that under this assumption, the need for variables \( s_1 \) and \( s_2 \) disappears; the algorithms may be directly connected to database \( D_A \). In this case, it has been shown \([8]\) (see Appendix A.2) that the levels of activity for a
decisionmaking system with two SA algorithms \( f_1 \) and \( f_2 \), containing \( \alpha_1 \) and \( \alpha_2 \) internal variables, respectively, and a decision strategy \( \delta \), are given by:

\[
\begin{align*}
G_t + G_b &= H(x) \\
G_n &= H(u) = H(\delta) \\
G_c &= (\alpha_1 + \alpha_2)H(\delta) + H(z) + \sum_{i=1}^{2} p(u=i)g_c^i
\end{align*}
\] (7.8)

The definition of the quantity \( H \) is given in Eq. (6.9); \( g_c^1 \) and \( g_c^2 \) are defined to be the internal coordinations of algorithms \( f_1 \) and \( f_2 \) respectively, where internal coordination is defined as

\[
g_c^i = \frac{1}{\text{prob}(u=i)} \left[ \alpha_i \sum_{j=1}^{\alpha_i} H(w_j^i | u=i) - H(W_i^i | u=i) \right]
\] (7.9)

and \( W_1^i \) represents all of the variables of algorithm \( i \), i.e.:

\[
W_i = (w_1^i, w_2^i, \ldots, w_{\alpha_i}^i)
\] (7.10)

The total activity of the system is then the sum of the quantities given in Eq. (7.8):

\[
G = H(x) + H(z) + (\alpha_1 + \alpha_2 + 1)H(\delta) + \sum_{i=1}^{2} p(u=i)g_c^i
\] (7.11)

Note that all of the above quantities are conditional on task \( A \) being performed: e.g., \( g_c^i \) could be written \( (g_c^i|X \in X_A) \). As already stated, it can be shown \[8\] that \( G \) is convex in the decision strategy, i.e.:

\[
G(\delta) \geq (\delta)(G|u=1) + (1-\delta)(G|u=2)
\] (7.12)

Therefore, using Eqs. (7.6) and (7.12), \( G \) may be found parametrically as a function of \( J \) for the single-task problem, as shown in Figure 7.2.

The dual-task problem requires the variables \( s_1 \) and \( s_2 \) to be included in the model. The activity levels for the system which must switch be-
between tasks can now be found. It will still be the case that

\[ G_t + G_b = H(x) \]
\[ G_n = H(u) = H(\delta) \]  
(7.13)

Let \( d_k \) represent a single variable of the permanent memory unit, let \( D \) represent all the permanent memory (both \( D_A \) and \( D_B \)), and let \( W \) represent all of the internal variables of both algorithms \( f_1 \) and \( f_2 \), then the coordination of the system performing two tasks is given by:

\[ G_c = \sum_{i=1}^{2} \sum_{j=1}^{2} H(w_j^i) + H(s_1) + H(s_2) + H(u) + H(z) + \sum_{k} H(d_k) \]
\[ - H(w, s_1, s_2, u, z, D) \]  
(7.14)

After much manipulation (see Appendix A.3 for details), Eq. (7.14) may be reduced to

\[ G_c = (\alpha_1 + \alpha_2 + 2)H(\delta) + \sum_{i=1}^{2} p(u=i) [(TD) (g_i^c|X_A) + (1-TD) (g_i^c|X_B)] \]
\[ + \sum_{i=1}^{2} \sum_{j=1}^{2} T(w_j^i; s_i) u=i) + H(z) \]  
(7.15)
and the total activity for the system performing two tasks is given by:

\[
G = H(x) + H(z) + (\alpha_1 + \alpha_2 + 3)H(\delta) \\
+ \sum_{i=1}^{2} p(u=i)[(TD)(g_i^A|x_1 x_A) + (1-TD)(g_i^B|x_1 x_B)] \\
+ \sum_{i=1}^{2} \sum_{j=1}^{\alpha_i} T(w_j^i; s_i | u=i)
\]

(7.16)

Note that there are now two additional \(H(\delta)\) terms: these are due to the presence of the two additional system variables, \(s_1\) and \(s_2\). The internal coordination term is now a convex combination of the internal coordinations found when only task A or task B is performed. Finally, consider the last term of Eq. (7.16), which does not even appear in Eq. (7.11):

\[
T(w_j^i; s_i | u=i) = H(s_i | u=i) - H^{w_j^i}(s_i | u=i)
\]

(7.17)

This may be interpreted as the amount of information transmitted between \(s_i\) and \(w_j^i\), given that algorithm \(i\) is being used; i.e., it is the extent to which variable \(w_j^i\) reflects which task is being performed. Since

\[
H(s_i | u=i) = p(u=i)H(TD)
\]

(7.18)

then

\[
0 \leq T(w_j^i; s_i | u=i) \leq p(u=i)H(TD)
\]

(7.19)

It will now be shown that for a fixed value of \(\delta\), \(0 \leq \delta \leq 1\), \(G\) is convex in the task division, i.e.

\[
G(TD) \geq (TD)(G|x_1 x_A) + (1-TD)(G|x_1 x_B) \\
0 \leq TD \leq 1
\]

(7.20)

First note that the right-hand side (RHS) of Eq. (7.20) may be found using Eq. (7.11):

\[
RHS = (TD)[H^A(x) + H^A(z) + (\alpha_1 + \alpha_2 + 1)H(\delta) + \sum_{i=1}^{2} p(u=i)(g_i^A|x_1 x_A)] \\
+ (1-TD)[H^B(x) + H^B(z) + (\alpha_1 + \alpha_2 + 1)H(\delta) + \sum_{i=1}^{2} p(u=i)(g_i^B|x_1 x_B)]
\]

(7.21)
Here, $H^A(x)$ and $H^A(z)$ are the entropies of $x$ and $z$ which occur in the single-task case of $x \in X_A$

$$H^A(x) = - \sum_{x \in X_A} p(x|x \in X_A) \log_2 p(x|x \in X_A)$$

$$H^A(z) = - \sum_{z \in Z_A} p(z|z \in Z_A) \log_2 p(z|z \in Z_A)$$ (7.22)

and $Z_A$ is the set of possible outputs in the single-task case of $x \in X_A$.

The quantities $H^B(x)$ and $H^B(z)$ are defined similarly. The probability distributions for $x$ and $z$ in the dual-task case are a convex combination of those for the single-task cases, i.e.

$$p(x) = (TD)p(x|x \in X_A) + (1-TD)p(x|x \in X_B)$$

$$p(z) = (TD)p(z|z \in Z_A) + (1-TD)p(z|z \in Z_B)$$ 0 $\leq$ TD $\leq$ 1 (7.23)

When a probability distribution is the convex combination of two others, as in Eq. (7.23), the following result holds [17]:

$$H(x) \geq (TD)H^A(x) + (1-TD)H^B(x)$$

$$H(z) \geq (TD)H^A(z) + (1-TD)H^B(z)$$ 0 $\leq$ TD $\leq$ 1 (7.24)

Rewrite Eq. (7.21) as follows:

$$RHS = [(TD)H^A(x) + (1-TD)H^B(x)] + [(TD)H^A(z) + (1-TD)H^B(z)]$$

$+ \left[ \sum_{\alpha_1 + \alpha_2 + 1} H(\delta) \right] + \sum_{i=1}^{2} p(u=i)[TD(g^i_C|x \in X_A) + (1-TD)(g^i_C|x \in X_B)]$$ (7.25)

Now compare Eq. (7.25) to Eq. (7.16), using the results of Eq. (7.24), the fact that $H(\delta) \geq 0$, and the fact that transmissions must also be non-negative, to find that Eq. (7.20) does indeed hold, and $G$ is convex in the task division. In fact, if a mixed strategy is being used ($0 < \delta < 1$), or if any of the internal variables of an algorithm in use reflects which task is being performed, i.e.,
\[ T(w^i_j; s^i_\{u=i\}) > 0 \]  

then the inequality of Eq. (7.20) will be strict.

7.3 EFFECT OF TASK DIVISION ON PERFORMANCE

To see the effects that this result may have on performance, consider a particularly simple example. It is assumed that the single-task, activity versus performance curves are identical for task A and task B (this implies that \( J_A \) and \( J_B \) are the same functions of \( \delta \); see Figure 7.3a).

Now consider the evolution of the G versus \( J_A \) curve as TD changes from 0 to 1. It is meaningless to define \( J_A \) for the single-task case in which task B is always performed (TD = 0), but for very small values of TD, \( J_A \) is defined as in Eq. (7.5). To find the G versus \( J_A \) curve for TD \( \approx 0 \), consider Eq. (7.16). Since \( H(TD) \approx 0 \) for TD \( \approx 0 \), its last term is small (see Eq. (7.19)). The rest of Eq. (7.16) reduces to Eq. (7.27) shown on the next page.

![Figure 7.3. Activity vs. Performance and Task Division](image-url)
\[ G(TD \approx 0) \approx \mathbb{E}(G|X \in X_B) + 2H(\delta) \] (7.27)

In other words, the \( G \) versus \( J_A \) curve will be the same as either single-task curve, with the quantity \( 2H(\delta) \) added on due to the addition of variables \( s_1 \) and \( s_2 \) (see Figure 7.3b). As TD increases, \( G \) will continue to increase to some point (because of its convexity in TD), dependent on the value of the last term of Eq. (7.16) and the values of \( H(x) \) and \( H(z) \).

For TD equal to 0.5, the \( G \) versus \( J_A \) curve might look something like that shown in Figure 7.3c. Finally, \( G \) will decrease until TD\(\approx1 \) and \( G \) versus \( J_A \) is again as shown in Figure 7.3b. For a fixed value of \( \delta \) then, say \( \delta = 0.2 \), the activity versus task division curve might look something like that shown in Figure 7.3d. Note that the maximum activity need not occur at TD\(=0.5 \).

Now consider what happens to performance if the maximum total activity constraint is given by the value marked \( G_{\max} \) in Figure 7.3, i.e., the system is required to perform at an activity level \( G \leq G_{\max} \). In the two single-task cases, the system is unconstrained and may use any strategy \( 0 \leq \delta \leq 1 \). However, for this example, when both tasks arrive with equal probability (TD\(=0.5 \)), the set of feasible strategies is greatly reduced, and the allowable performances are limited to very poor ones. Also, the particular strategy of \( \delta = 0.2 \) may only be used for task divisions close to 0 or 1 (see Figure 7.3d).

This simple example illustrates some rather general results. The convexity of \( G \) in the task division implies that the rate of activity of the system will be greater in the dual-task case than in at least one of the single-task cases. Note that if the activity levels of the two single-task cases are very disparate, then the opportunity to switch between a very activity-intensive task and a very easy one may actually reduce the activity from what it is in the case that only the difficult task is being performed. When the activity levels for the two single-task cases are comparable, though, as in the preceding example, the level of activity for the dual-task case is greater than that for either of the single-task cases. This increase in activity arises from three
basic sources, which may be seen by an examination of Eq. (7.16). First, the variable \( x \) and in most cases also the variable \( z \) will have a larger uncertainty associated with them because of their larger alphabets in the dual-task situation. Second, the dual-task problem requires that the system have some means of switching between the sets of data stored in permanent memory; the variables \( s_1 \) and \( s_2 \) provide that mechanism but also increase the uncertainty of the system. Third, the rest of the internal variables may, because of access to different values stored in memory, take on a wider range of values when both tasks must be performed than when only one is performed. If the system performing either task alone is operating near to its maximum allowable rate, then requiring the system to switch between the two tasks has the effects of both eliminating the more active decision strategies from the feasible set, and, in the case that the more active strategies also result in better performance, lowering the performance of the system.

In order to test experimentally this prediction of lower performance in the dual-task problem as defined here, several criteria must be met. First, the two tasks must be similar enough that the same set of algorithms may be used for both tasks; however, they should be independent enough such that execution of one task does not aid in the execution of the other. Second, it should be necessary to switch between tasks, i.e. two different tasks may not be performed simultaneously. Third, individual tasks should arrive at the same rate in the dual-task test as in the single-task test. Finally, this rate of presentation should be near to the activity limit of the system, since it is hypothesized that it is this constraint that leads to a degradation of system performance.

In the next chapter, the main points of this thesis will be summarized, and some promising areas for future research will be suggested.
CHAPTER 8
CONCLUSIONS AND FUTURE RESEARCH

8.1 CONCLUSIONS

In this thesis, models of two types of memory, as they occur in a decisionmaking system, have been presented. Buffer storage, which allows the system to process sequential statistically dependent inputs simultaneously, has been modeled in three different ways, according to the class of input that the system receives. Shift register buffers provide the storage rule necessary to process input from a general Markov source. They add a great deal of activity to the system, however, and also result in redundant processing. Fixed-length string buffers are a good model for the type of storage found in machines, whose algorithms often are not adaptable to inputs having varying string lengths. Fixed-length string buffers, by eliminating the redundancy introduced by shift register buffers, do not increase the overall rate of activity of the system as much as shift registers. However, fixed-length string buffers require that the system postpone its decisionmaking, until all of the information pertinent to one event has been received, thereby increasing the system response time, i.e. introducing a delay. Variable-length string buffers provide a good model for some types of storage used by human decisionmakers. They also introduce a delay, though in many cases not as much as the fixed-length string buffers. However, because of the uncertainty associated with the length of a string, variable-length string buffers generally increase the overall rate of activity of the decision-making system more than do fixed-length string buffers.

The analysis of the decisionmaking system with memory is completed with the addition of models of permanent and temporary memory. Permanent memory provides the decisionmaking algorithms with access to information which does not change over time. It also gives the algorithms more
flexibility by allowing them to switch between sets of data, depending on the input that arrives at the system. Temporary memory allows the algorithms access to information which the algorithms themselves have stored at some previous iteration, i.e. it allows them to remember, at a later time, information that they generated. It is satisfying to note that the models of memory developed in the cognitive sciences, sensory memory and long- and short-term memory, parallel the models presented here of buffer storage, and permanent and temporary memory.

The example of a decisionmaking system gaining access to one of two sets of data stored in permanent memory, depending on which of two non-reinforcing tasks arrives at the system, has been analyzed extensively. The analysis leads to the conclusion that, under certain assumptions, the performance of this system will be poorer than that of the same system receiving only one task. Poorer performance is due to the higher activity rate that results both from the addition of the variables which switch between the two sets of data, and from the higher uncertainty rates of the internal variables in the dual-task case: under the assumption that the system is close to its bounded rationality constraint, this higher rate of activity can eliminate some desirable strategies and lower the performance of the system.

8.2 FUTURE RESEARCH

In connection with the example just described, at least two areas have yet to be explored. The first area is an experimental program, such as that described at the end of Chapter 7, to test the analytical conclusions reached in that chapter. The value of this type of experiment is that it would be one of the first quantitative tests of this decision-making model. By careful adjustment of the various parameters of the experiment, real insight could be gained as to which aspects of the model appear to be valid and which need some modification.
The second area is the development of the model of the pre-processor as another illustration of the dual-task problem: when the two tasks are so dissimilar that they require access to different sets of algorithms, then a pre-processor may be used to determine which set a given input will have access to. Actually, the pre-processor model has much wider applicability than simply to the dual-task problem. For example, there should be some mechanism by which a system can decide to ignore input which either does not need processing or is irrelevant to the task at hand. A pre-processor, which determines the set of algorithms appropriate to any input, could also act as a rejection mechanism, by sending inputs which do not require processing to a null algorithm.

Several other areas remain open for future research. First, the comparison of fixed-length and variable-length string buffers raises the issue of algorithms that deal with variable-length strings. The basic structure of an algorithm which can operate in conjunction with a variable-length string buffer, and the information theoretic rate of activity of such an algorithm, are important areas of investigation. In some cases, it may be possible to use an algorithm designed for fixed-length strings, filling in the unused variables with null values as necessary, but this method is valid only for a specific class of algorithms. Another possible model would require information as to how long a particular string was, and then use that information to give the algorithm the appropriate dimension. This algorithm design is an important issue which has yet to be addressed in an information theoretic framework.

A second issue of importance, noted by Drenick [25], that inevitably rises with the use of information theory is that of coding. By employing optimal coding techniques on the input to a communications channel, it is possible to reduce the average code-word length (and thereby make for more efficient transmission) to within an arbitrarily small fraction of the entropy of the input. This would seem to imply that, in some cases, an encoder at the front of a decisionmaking system, (which could consist of

-89-
anything from an actual hardware device to a staff of people) might increase its information-processing capacity. This is an interesting notion which should be investigated in the future.

In this thesis, the task of giving the decisionmaking model more dynamics and making it able to deal with events related in time has been started. A larger issue must eventually be explored and that is the issue of how the model itself changes over time, i.e. the issue of learning. The present decisionmaking model probably contains enough dynamics to model a member of a C³ system as he acts for the duration of one maneuver. But often the period of interest for a C³ system may be as long as years, and during that period, its members learn new ways of handling both old and new inputs; i.e. the set of algorithms for any given decisionmaker, and the structure of those algorithms, may change. The way in which this learning takes place, and the activity associated with it, is a large area of endeavor which deserves attention in the future.
APPENDIX A
DERIVATION OF SEVERAL INFORMATION THEORETIC EXPRESSIONS

A.1 DERIVATION OF PARTITION LAW OF INFORMATION RATES

The Partition Law of Information Rates (PLIR) [15] states that for a system with input $x$, $N-1$ internal variables $w_1$ through $w_{N-1}$, and an output variable $y$, also known as $w_N$,

\[
\sum_{i=1}^{N} \overline{H}(w_i) = \overline{T}(x:y) + \overline{T}(x:w_1,w_2,\ldots,w_{N-1}) + \overline{T}(w_1:w_2:\ldots:w_{N-1}:y) \\
+ \overline{H}(w_1,w_2,\ldots,w_{N-1},y) \\
= F_t + F_b + F_c + F_n \tag{A.1}
\]

Expand each term on the right-hand side of Eq. (A.1):

\[
F_t = \overline{H}(x) - \overline{H}_y(x) \\
F_b = \overline{H}_y(x) - \overline{H}_{y_1,y_2,\ldots,y_{N-1}}(x) \\
F_c = \sum_{i=1}^{N} \overline{H}(w_i) - \overline{H}(w_1,w_2,\ldots,w_{N-1},y) \\
F_n = \overline{H}_{x}(w_1,w_2,\ldots,w_{N-1},y) \tag{A.2}
\]

Rearranging and adding these four expressions gives for Eq. (A.1):

\[
\sum_{i=1}^{N} \overline{H}(w_i) = \sum_{i=1}^{N} \overline{H}(w_i) - [\overline{H}_{y_1,y_2,\ldots,y_{N-1}}(x) + \overline{H}(w_1,w_2,\ldots,w_{N-1},y)] \\
+ [\overline{H}(x) + \overline{H}_{x}(w_1,w_2,\ldots,w_{N-1},y)] \\
= \sum_{i=1}^{N} \overline{H}(w_i) - [\overline{H}(x,w_1,w_2,\ldots,w_{N-1},y)] + [\overline{H}(x,w_1,w_2,\ldots,w_{N-1},y)] \\
= \sum_{i=1}^{N} \overline{H}(w_i) \tag{A.3}
\]

which was to be shown.
A.2 DERIVATION OF ACTIVITY EXPRESSIONS

These results are adaptations of an almost identical result found by Boettcher and Levis [8]. Equations (6.5) and (6.8) state that for a situation assessment subsystem with \( U \) algorithms, of which the \( i \)-th contains \( \alpha_i \) internal active/nonactive variables, and with \( W \) defined by Eq. (6.7),

\[
F_t = \bar{H}(z) - \bar{H}_x(z)
\]

\[
F_b = \bar{T}_z(x;W)
\]

\[
F_n = \bar{H}(u)
\]

and

\[
F_c = \sum_{i=1}^U \sum_{j=1}^\alpha_i \bar{H}(w_i^j) + \bar{H}(u) + \bar{H}(z) - \bar{H}(u,W,z)
\]

\[
= \sum_{i=1}^U \sum_{j=1}^\alpha_i \bar{H}_u(w_i^j) - \bar{H}_u(W) + \bar{H}(z)
\]

(A.4)

First consider Eq. (A.4). The first two expressions follow by definition (see section A.1). The noise expression is found thus:

\[
F_n = \bar{H}_x(u,W,z)
\]

\[
= \bar{H}_x(u) + \bar{H}_{xu}(W) + \bar{H}_{xuw}(z)
\]

\[
= \bar{H}_x(u)
\]

\[
= \bar{H}(u)
\]

(A.6)

Here the third equality follows because the algorithms are deterministic (\( x \) and \( u \) determine the values of the internal variables); and the values of the internal variables determine \( z \). The fourth equality is due to the independence of \( x \) and \( u \).

Now consider the first half of Eq. (A.5) and expand its last term:

\[
\bar{H}(u,W,z) = \bar{H}(u) + \bar{H}_u(W) + \bar{H}_{uw}(z)
\]

(A.7)
Again, the uncertainty in \( z \), given the values of the internal variables, is zero, so the first half of Eq. (A.5) becomes:

\[
F_c = \sum_{i=1}^{U} \sum_{j=1}^{L} \alpha_i \bar{H}(w_j^i) + \bar{H}(z) - \bar{H}_u(W)
\]

\[
= \sum_{i=1}^{U} \sum_{j=1}^{L} \alpha_i \bar{H}(w_j^i) + \bar{H}(z) - \bar{H}_u(W) - \sum_{i=1}^{U} \sum_{j=1}^{L} \bar{H}_u(w_j^i) + \sum_{i=1}^{U} \sum_{j=1}^{L} \bar{H}_u(w_j^i)
\]

\[
= \sum_{i,j} \left[ \bar{H}(w_j^i) - \bar{H}_u(w_j^i) \right] + \bar{H}(z) - \bar{H}_u(W) + \sum_{i,j} \bar{H}_u(w_j^i) \tag{A.8}
\]

Consider one of the terms in brackets:

\[
\bar{H}(w_j^i) - \bar{H}_u(w_j^i) = \bar{T}(u; w_j^i)
\]

\[
= \bar{H}(u) - \bar{H}_u(w_j^i) \tag{A.9}
\]

Now, \( u \) is chosen independently with every input, and only information about the present value of \( w_j^i \) may reduce the uncertainty in \( u \). Therefore, assuming inputs arrive once every second, the last term of Eq. (A.9) becomes:

\[
\bar{H}_u(w_j^i) = \lim_{m \to \infty} \frac{1}{m} \left\{ \bar{H}_{w_j^i}(u(1)) + \bar{H}_{w_j^i}(u(2)) + \ldots + \bar{H}_{w_j^i}(u(m)) \right\}
\]

\[
= \lim_{m \to \infty} \frac{1}{m} \left\{ \bar{H}_{w_j^i}(1) + \bar{H}_{w_j^i}(2) + \ldots + \bar{H}_{w_j^i}(m) \right\}
\]

\[
= \bar{H}_{w_j^i}(u(i)) \text{; and similarly,} \quad \bar{H}(u) = \bar{H}_u(W) \tag{A.10}
\]

Here, the vectors \( w_j^i \) and \( u \) represent the values of \( w_j^i \) from time 1 to time \( m \), and those of \( u \) from time 1 to time \( m-1 \). Note that, with inputs arriving once every \( \tau \) seconds rather than once every second,

\[
\bar{H}_{w_j^i}(u) = \frac{1}{\tau} \bar{H}_{w_j^i}(u) \quad \text{and} \quad \bar{H}(u) = \frac{1}{\tau} \bar{H}(u) \tag{A.11}
\]

So each term in brackets at the end of Eq. (A.8) becomes:
\[ H(u) - H_{w_j}^{i}(u) = - \sum_{u} p(u) \log_2 P(u) - \sum_{u} p(w_j^i) \sum_{u} p(u|w_j^i) \log_2 P(u|w_j^i) \]

\[ \text{(A.12)} \]

But if \( w_j^i \) is active, there is no uncertainty in \( u \), i.e., \( \text{prob}(u=i|w_j^i \neq \emptyset) \) is one, and its log is zero. Also, \( \text{prob}(w_j^i = \emptyset) \) is just \( \text{prob}(u=i) \), so Eq. (A.12) becomes:

\[ H(u) - H_{w_j}^{i}(u) = - \sum_{u} p(u) \log_2 P(u) + p(u \neq i) \sum_{u \neq i} p(u|u \neq i) \log_2 P(u|u \neq i) \]
\[ \text{Eq. (A.9)} \]
\[ = - \sum_{u} p(u) \log_2 P(u) + p(u \neq i) \frac{p(u)}{u \neq i} \log_2 \frac{p(u)}{u \neq i} \]
\[ \text{Eq. (A.10)} \]
\[ = - \sum_{u} p(u) \log_2 P(u) - \sum_{u=i} p(u) \log_2 P(u \neq i) \]
\[ \text{Eq. (A.11)} \]
\[ = H[p(u=i)] \]

\[ \text{(A.13)} \]

Note that this expression is independent of \( j \); therefore, for a given value of \( i \), Eq. (A.9) yields the same result for all \( j \), of which there are \( \alpha_1 \). So Eq. (A.8) becomes:

\[ F_c = \sum_{i=1}^{U} \alpha_i [H(p(u=i)] + \sum_{i=1}^{U} \sum_{j=1}^{\alpha_i} H_{w_j^i}(w_j^i) \]
\[ \text{(A.14)} \]
\[ = H(u) - H(v) + H(z) \]

which was to be shown.

Now consider Eq. (7.8), which is stated under the same assumptions as Eqs. (6.5) and (6.8), with the additional specification that \( U \) is equal to 2:

\[ G_t + G_b = H(x) \]
\[ G_n = H(u) = H(\delta) \]
\[ G_c = (\alpha_1 + \alpha_2)H(\delta) + H(z) + \sum_{i=1}^{2} p(u=i) g_i^c \]

\[ \text{(A.15)} \]

The first of these equations is derived exactly like Eq. (3.8) under the assumption of no rejection, and using entropies instead of entropy rates. The second is found exactly like Eq. (A.6). Since \( u \) may only take the values 1 or 2, and since \( p(u=1) \) is \( \delta \),

\[ -94 - \]
\[
H(u) = -\log_2 \delta - (1-\delta)\log_2 (1-\delta)
\]
\[
= H(\delta)
\]  

(A.16)

Now consider the coordination term:

\[
G_c = \sum_{i=1}^{2} \sum_{j=1}^{1} \frac{\alpha_i}{\sum_j H(w^i_j)} + H(u) + H(z) - H(W,u,z)
\]
\[
= \sum_{i=1}^{2} \frac{\alpha_i}{\sum_j H(w^i_j)} + H(z) - H_u(W)
\]
\[
= \sum_i \left[ H(w^i_j) - H_u(w^i_j) \right] + H(z) - H_u(W) + \sum_i \left[ \sum_j H(w^i_j) - H_u(w^i_j) \right]
\]
\[
= \sum_i \left[ H(w^i_j) - H_u(w^i_j) \right] + H(z) - H_u(W) + \sum_i \sum_j H(w^i_j)
\]
\[
= (\alpha_1 + \alpha_2) H(\delta) + H(z) + \sum_i \sum_j \left[ \sum H(w^i_j) - H_u(w^i_j) \right]
\]  

(A.17)

The first equality is just the definition of coordination. The second equality follows by using Eq. (A.7). The fourth equality is found by using Eq. (A.13). To find the fifth equality, first note that \(H(\delta)\) is equal to \(H(1-\delta)\). Second, because the values that the two algorithms take are independent,

\[
H_u(W) = H_u(w^1, w^2)
\]
\[
= H_u(w^1) + H_u(w^2)
\]
\[
= H_u(w^1) + H_u(w^2)
\]  

(A.18)

where \(w^i\) is defined in Eq. (7.10). Now consider one of the terms in brackets at the end of Eq. (A.17). If \(u\) is anything other than \(i\), then all of the variables of algorithm \(i\) are inactive, so there is no uncertainty in them, and

\[
\frac{\alpha_i}{\sum_j H_u(w^i_j)} - H_u(w^i) = \frac{\alpha_i}{\sum_j H(w^i_j | u=i)} - H(w^i | u=i)
\]  

(A.19)

Noting the definition for internal coordination given in Eq. (7.9),

-95-
Eq. (A.17) becomes:

\[ G_c = (\alpha_1 + \alpha_2)H(\delta) + H(z) + \sum_{i=1}^{2} p(u=i) q_c^i \]  

(A.20)

which was to be shown.

**A.3 DERIVATION OF DUAL-TASK COORDINATION EXPRESSION**

For a system with two situation assessment algorithms, having \( \alpha_1 \) and \( \alpha_2 \) internal variables respectively; two switching variables, \( s_1 \) and \( s_2 \); and two sets of data, \( D_A \) and \( D_B \) (collectively called \( D \)), the coordination of the system is given by Eqs. (7.14) and (7.15). Because there is no uncertainty in \( d_k \), Eq. (7.14) becomes:

\[ G_c = \sum_{i,j} H(w_i^j) + H(s_1) + H(s_2) + H(u) + H(z) - H(w, s_1, s_2, u, z, D) \]  

(A.21)

Expand the last term of Eq. (A.21):

\[ H(w, s_1, s_2, u, z, D) = H(u) + H(s_1) + H(s_2) + H(s_1 s_2) + H(s_1 w) + H(s_2 w) + H(s_1 s_2 w) + H(s_1 s_2 w z) \]  

(A.22)

There is no uncertainty in \( D \), and \( z \) is determined by \( w \), so the last two terms of Eq. (A.22) are zero. Also, because the \( w_i^j \) are independent and \( w_i^j \) is independent of \( s_j \) for \( i \neq j \),

\[ H_{s_1 s_2} (w_1, w_2) = H_{s_1 s_2} (w_1) + H_{s_1 s_2} (w_2) \]

\[ = H_{s_1} (w_1) + H_{s_2} (w_2) \]  

(A.23)

Finally, \( s_1 \) gives no information about \( s_2 \) that \( u \) does not give, so

\[ H(w, s_1, s_2, u, z, D) = H(u) + \sum_{i=1}^{2} [H(s_i) + H_{s_i} (w_i)] \]  

(A.24)

and
\[ G_c = \sum_{i,j} H(w^i_j) + \sum_{i=1}^{2} [H(s^i_i) - H_u(s^i_i) - H_{us^i_i}] + H(z) \]
\[ = \sum_{i,j} [H(w^i_j) - H_u(w^i_j)] + \sum_{i=1}^{2} [H(s^i_i) - H_u(s^i_i)] \]
\[ + \sum_{i,j} [H_u(w^i_j) - H_{us^i_i}(w^i_j)] + \sum_{i,j} [H_{us^i_i}(w^i_j) - H_{us^i_i}(w^i_j)] + H(z) \]
\[ (A.25) \]

Each term in the first and second set of brackets contributes an \( H(\delta) \) term, by the reasoning given in the derivation of Eq. (A.13). Since there is no uncertainty in \( w^i_j \) when \( u \neq i \),

\[ H_u(w^i_j) - H_{us^i_i}(w^i_j) = H(w^i_j|u=i) - H(w^i_j|u=i,s^i_i) \]
\[ = T(w^i_j;s^i_i|u=i) \]
\[ (A.26) \]

By similar reasoning,

\[ \sum_j H_{us^i_i}(w^i_j) - H_{us^i_i}(w^i_j) = \sum_j H(w^i_j|u=i,s^i_i) - H(w^i_j|u=i,s^i_i) \]
\[ = \sum_{\lambda=A,B} p(s^i_i=\lambda|u=i) \sum_j H(w^i_j|u=i,s^i_i=\lambda) - H(w^i_j|u=i,s^i_i=\lambda) \]
\[ (A.27) \]

Notice that information as to the value of \( s^i_i \) is equivalent to knowledge of which type of input has arrived, i.e.,

\[ p(s^i_i=A|u=i) = p(xeX_A) \]
\[ = TD \]
\[ (A.28) \]

Also notice that \( H(w^i_j|u=i,s^i_i=A) \) and \( H(w^i_j|u=i,s^i_i=A) \) are the quantities used to calculate \( g^i_c \) in the single-task case of \( xeX_A \). Therefore, Eq. (A.27) can be rewritten

\[ \sum_j H_{us^i_i}(w^i_j) - H_{us^i_i}(w^i_j) = p(u=i)[(TD)(g^i_c|xeX_A) + (1-TD)(g^i_c|xeX_B)] \]
\[ (A.29) \]

and finally, combining Eqs. (A.25), (A.26), and (A.29) gives:
\[ G_c = (\alpha_1 + 2\alpha_2 + 2)H(\delta) + \sum_{i=1}^{2} p(u=i) \{ (TD)(g_c^i|x \in X_A^i) + (1-TD)(g_c^i|x \in X_B^i) \} \]

\[ + \sum_{i=1}^{2} \sum_{j=1}^{\lambda} T(w_i^j:s_i^j|u=i) + H(z) \]  

(A.30)

which was to be shown.
APPENDIX B
DERIVATION OF MARKOV PROCESS QUANTITIES

The derivation of steady-state probabilities for a Markov Process is found by solving simultaneously the set of equations given by

\[ \begin{align*}
\pi P &= \pi \\
\sum_{i=1}^{M} \pi_i &= 1
\end{align*} \] (B.1)

where \( \pi \) is the \( M \)-vector of steady-state probabilities, and \( P \) is the \( M \times M \) matrix of transition probabilities, i.e., \( p_{ij} \) is the probability of moving from state \( i \) to state \( j \) on a given trial. The \((M+1)\)th equation is necessary, because only \( M-1 \) of the other \( M \) equations are linearly independent.

For the Markov Process depicted in Figure 4.6, then, \( P \) is given by

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 0 & 0 & 1/4 & 3/4 & 0 & 0 \\
2 & 0 & 0 & 1/4 & 3/4 & 0 & 0 \\
3 & 1 & 0 & 0 & 0 & 0 & 0 \\
4 & 1/3 & 0 & 0 & 1/3 & 1/3 & 0 \\
5 & 1 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\] (B.2)

where the states are numbered to simplify the notation. The steady-state probabilities are found by solving the following set of equations simultaneously:

\[
\begin{align*}
\pi_1 &= \pi_3 + \pi_5 \\
\pi_2 &= 1/3 \pi_4 + \pi_6 \\
\pi_3 &= 1/4(\pi_1 + \pi_2) \\
\pi_4 &= 3/4(\pi_1 + \pi_2) \\
\pi_5 &= 1/3 \pi_4 \\
\pi_6 &= 1/3 \pi_4 \\
\sum_{i=1}^{6} \pi_i &= 1
\end{align*} \] (B.3)
to get the result given in Eq. (4.26).

For the Markov Process shown in Figure 4.8, symmetry considerations allow the construction of \( P \):

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & \ldots & i & i+1 & \ldots & N & N+1 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & \frac{1}{N} & 0 & \frac{N-1}{N} & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & \frac{1}{N-1} & 0 & 0 & \frac{N-2}{N-1} & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
i-1 & \frac{1}{N-i+3} & 0 & 0 & 0 & \frac{N-i+2}{N-i+3} & 0 & 0 & 0 & 0 \\
i & \frac{1}{N-i+2} & 0 & 0 & 0 & 0 & \frac{N-i+1}{N-i+2} & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
N-1 & \frac{1}{N} & 0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 0 \\
N & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 \\
N+1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

(B.4)

To find the steady-state probabilities, write the set of equations given in Eq. (B.1). It may be found that,

\[
\pi_i = \frac{N-i+2}{N} \pi_l \quad \text{for } i = 2, 3, \ldots, N+1
\]

(B.5)

Using this equation and the second half of Eq. (B.1), it is found that,

\[
\pi_l = \frac{2}{N+3} \\
\pi_i = \frac{2(N-i+2)}{N(N+3)} \quad \text{for } i = 2, 3, \ldots, N+1
\]

(B.6)

which is the result given in Eq. (4.34). Given the system is in state \( i \), where \( 2 \leq i \leq N \), it may either return to state 1 with probability \( 1/(N-i+2) \) or go on to state \( i+1 \) with probability \( \frac{N-i+1}{N-i+2} \). Therefore, the uncertainty in the next value of \( P \), given it is presently equal to \( i \), is given by

\[
H(P|s=i) = \frac{1}{N-i+2} \log_2 (N-i+2) + \frac{N-i+1}{N-i+2} \log_2 \frac{N-i+2}{N-i+1} \quad \text{for } 2 \leq i \leq N
\]

(B.7)
Note that

\[ H(P|s=1) = H(P|s=N+1) = 0 \quad (B.8) \]

From Eq. (4.1),

\[
\bar{H}(P) = \frac{N+1}{1} \sum_{i=1}^{N} H(P|s=i)
\]

\[
= \frac{N}{2} \sum_{i=2}^{N} \left\{ \frac{1}{N-i+2} \log_2(N-i+2) + \frac{N-i+1}{N-i+2} \log_2 N-i+1 \right\}
\]

\[
= \frac{2}{N(N+3)} \sum_{i=2}^{N} \log_2(N-i+2) \cdot (N-i+2) - \log_2(N-i+1) \cdot (N-i+1)
\]

\[
= \frac{2}{N(N+3)} \log_2 \frac{N}{(N-1)} \cdot \frac{(N-1)}{(N-2)} \cdot \frac{(N-2)}{(N-3)} \cdots \frac{3}{2} \cdot \frac{2}{1}
\]

\[
= \frac{2}{N+3} \log_2 N \quad (B.9)
\]

the result given in Eq. (4.36).

The values of \( \bar{H}(b_1) \) and \( \bar{H}(b_2) \) are found by applying Eq. (4.1) to the steady-state probabilities found by using Eq. (B.1).

For the source described in section 4.6, the four strings 0-, 1-, 10, and 11 are equally likely to be generated. Thus, the \( P \) matrix for \( b_1 \) is given by the following where the notation for a state is the same used for Figure 4.9:

\[
\begin{array}{c|cccc}
 & 0_1 & 0_2 & 1_1 & 1_2 \\
\hline
0_1 & 0 & 1 & 0 & 0 \\
0_2 & 1/4 & 0 & 3/4 & 0 \\
1_1 & 0 & 0 & 0 & 1 \\
1_2 & 1/4 & 0 & 3/4 & 0 \\
\end{array}
\quad (B.10)
\]

That for \( b_2 \) is given by the following where the notation for a state is the same used for Figure 4.10:

-101-
\[
\begin{array}{ccccccc}
\text{-1} & -2 & 0_1 & 0_2 & 1_1 & 1_2 \\
\hline
-1 & 0 & 1 & 0 & 0 & 0 & 0 \\
-2 & 1/2 & 0 & 1/4 & 0 & 1/4 & 0 \\
0_1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0_2 & 1/2 & 0 & 1/4 & 0 & 1/4 & 0 \\
1_1 & 0 & 0 & 0 & 0 & 0 & 1 \\
1_2 & 1/2 & 0 & 1/4 & 0 & 1/4 & 0 \\
\end{array}
\]

(B.11)

Using the matrices given in (B.10) and (B.11), along with Eqs. (4.1) and (B.1), the values of \( \bar{H}(b_1) \) and \( \bar{H}(b_2) \) given in section 4.6 may be found.
REFERENCES


