Optimal Hidden Markov Models

Bill Frederick McKee

BEng(Hons), MSc

A thesis submitted to the University of Plymouth
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

[illegible]

School of Electronic, Communication and Electrical Engineering
Faculty of Technology

September 1999

Optimal Hidden Markov Models

Bill Frederick McKee

Abstract

In contrast with training algorithms such as Baum-Welch, which produce solutions that are a local optimum of the objective function, this thesis describes the attempt to develop a training algorithm which delivers the global optimum Discrete Hidden Markov Model for a given training sequence.

A total of four different methods of attack upon the problem are presented. First, after building the necessary analytical tools, the thesis presents a direct, calculus-based assault featuring Matrix Derivatives. Next, the dual analytic approach known as Geometric Programming is examined and then adapted to the task. After that, a hill-climbing formula is developed and applied.

These first three methods reveal a number of interesting and useful insights into the problem. However, it is the fourth method which produces an algorithm that is then used for direct comparison with the Baum-Welch algorithm: examples of global optima are collected, examined for common features and patterns, and then a rule is induced.

The resulting rule is implemented in 'C' and tested against a battery of Baum-Welch based programs. In the limited range of tests carried out to date, the models produced by the new algorithm yield optima which have not been surpassed by (and are typically much better than) the Baum-Welch models. However, far more analysis and testing is required and in its current form the algorithm is not fast enough for realistic application.

Contents

Acknowledgement

I wish to thank my friends at the School of Electronic, Communication and Electrical Engineering, and especially my supervisor, Ian Barlow, for his wise and faithful guidance.

I also wish to thank the people around me who have given so much support and encouragement through the years. These include my mother, Mrs. Elsie McKee; my office colleagues, Tom Allsop and Alastair Hood; and my house-mates, Steve Roberts and Geraldine O'Driscoll.

Author's Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

The work presented in this thesis is solely that of the author.

Signed Bill F. McKee

Date 24 FEBRUARY 2000

1. Introduction

1.1. Overview

Since their development more than 30 years ago, Hidden Markov Models (HMMs) have remained one of the most successful tools for carrying out Automatic Speech Recognition [1-6]. They provide a mathematically rigorous approach to developing robust statistical speech models during the training phase, and then classifying unknown utterances at the recognition phase [7,8].

Training consists of finding a set of model parameters which maximize the probability of reproducing the training data. With this probability as the objective function, and the model parameters as the domain variables, training algorithms (such as the classic Baum-Welch) operate by hill-climbing from an arbitrary starting location in the domain to a local maximum of the objective.

Because the probability function is particularly non-linear in the model parameters, little effort has been made in the past to search for the set of parameters which is globally optimal. The lack of literature suggests that authors may indeed have given up on the possibility, believing that "given any finite observation sequence as training data, we cannot optimally train the model" [9]. (See also [10,11])

The aim of the current research has been to challenge that view, by attempting to develop a procedure by which optimal Hidden Markov Models can be identified.

This goal is, admittedly, very ambitious, so to keep the problem more manageable work has focused on the simplest possible case, in which a Discrete HMM is optimized for a single observation sequence. The following chapters survey some of the in-roads which have been made in this undertaking.

1.2. HMMs and Speech Recognition

Roughly speaking, Automatic Speech Recognition (ASR) is the attempt to have computers identify human utterances. One visualizes a 'phonetic typewriter' style of solution, where a human speaks into a microphone and the computer displays the correct string of words on a screen.

This is an old subject of research [12], but the progress to date is still very limited. The difficulties can be summarized in terms of the two 'dimensions' shown below (Fig 1.1) . Where the characteristics of only a single speaker are involved, and the words are separated by clear gaps, recognition systems can achieve high levels of success. But as the number of speakers or the fluency of the speech increases, recognition accuracy falls off dramatically.

Historically, there have been two main approaches to ASR (see Fig 1.2). In the cognitive or Knowledge-Based approach [13], one attempts to elicit and then automate the rules used by knowledgeable experts. However, successfully capturing all of the complex interrelationships of speech redundancies in one comprehensive structural model has proved extremely difficult [14].

Alternatively, in the information-theoretic or Template Matching approach [15], a collection of prototypical patterns are collected into a reference memory. Then a sample utterance is compared against the complete set of patterns, and a classification is made based upon the 'best' match. This brute force approach is limited by the size of the reference memory and the complexity of the comparison algorithm.

Recently, Neural Network recognizers have come along [16] which bridge the gap between the older approaches. The neural network performs a type of matching based upon features that it 'learns' by itself from the training data.

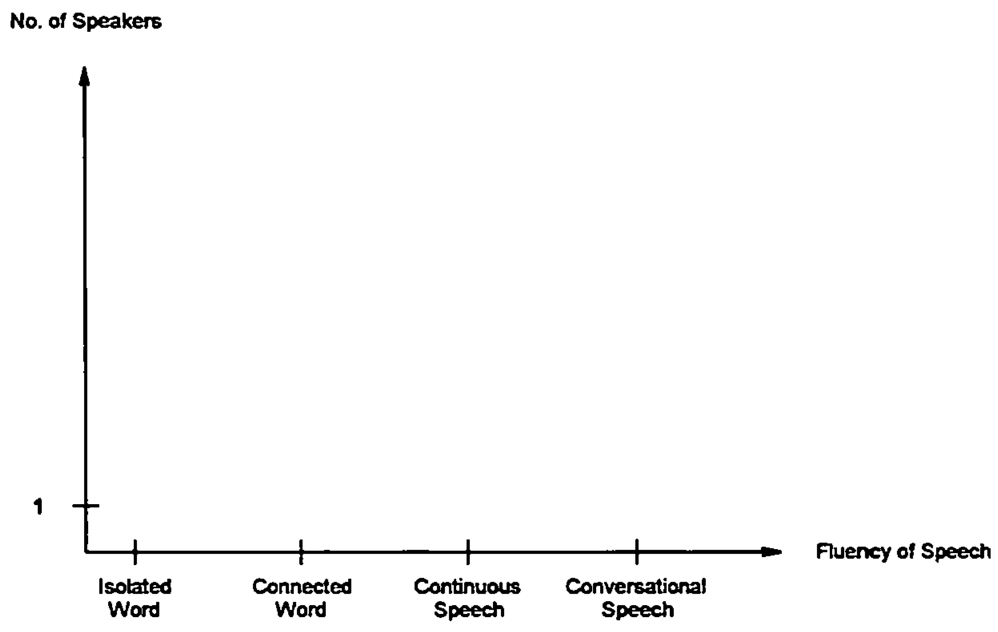


Fig 1.1 'Dimensions' of Difficulty

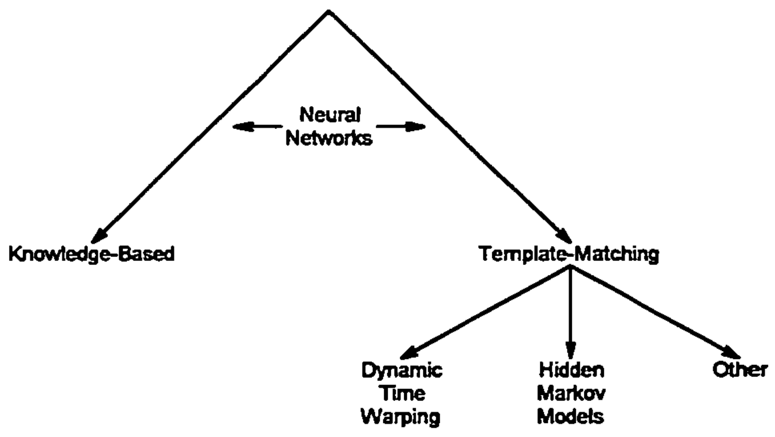


Fig 1.2 ASR Approaches

Hidden Markov Models (HMMs) belong to the template matching approach, but are generally superior to the other members of that family. Within template matching, the usual procedure is for the training data to be clustered into classes, after which the centroid of each class is determined. Template matchers then retain the collection of centroids, which forms the reference memory, but discard the other available information, such as the distribution of the clusters about their centroids. On the other hand, HMMs are able to incorporate this extra information and, consequently, tend to achieve better results. In fact, HMMs currently offer the best performance available for isolated word/single speaker recognition [17] and form the basis of almost all successful commercial and laboratory recognition systems.

1.3. HMM Fundamentals

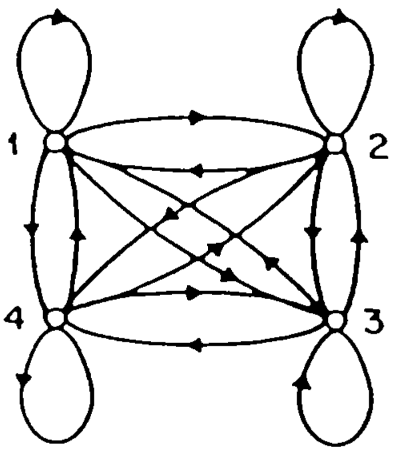
A Hidden Markov Model can be regarded as a special type of finite state machine, with two important differences (Fig 1.3). Unlike the standard finite state machine, the output from states and the transition between states do not depend upon external inputs. Instead, the transition between states is a random process which satisfies the (first order) Markov property, whereby the choice of state at time t depends only upon the previous state. Secondly, the sequence of states through which the model passes is not directly observable (hence the term 'hidden'). However at each discrete time t, upon entering a state, the model emits a signal which is observable. The choice of signal to emit is also made randomly, with the decision governed by a different random process for each state.

Thus, a Hidden Markov Model represents a doubly-embedded stochastic process, with an underlying stochastic process which is not observable, but which can be sensed via another set of stochastic processes that produce a sequence of observations [18].

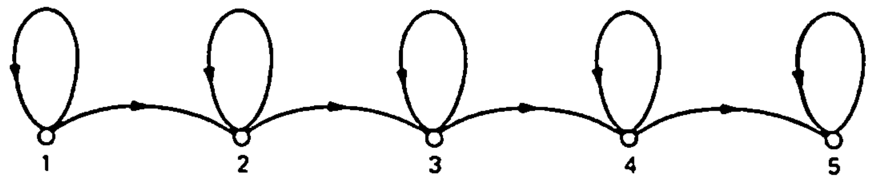
In using an HMM to model a body of data, one assumes that the process which generated the data was itself an HMM, and then attempts to identify the parameters of that underlying HMM (i.e. the parameters of its component probability distributions). According to the usual criterion, the best model (set of parameters) for a body of data is the one which exhibits the greatest likelihood of having generated that data.

Thus while standard template matching techniques reduce each class of training data to a single average, or prototype, with HMMs each class of training data is instead used to produce a stochastic model of the process which was responsible for generating the training data.

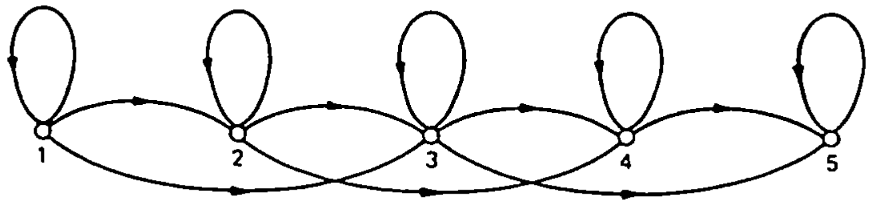
To carry out speech recognition, a library of HMMs is constructed, with a model for each item (word, syllable, phoneme, etc.) in the vocabulary. A test pattern comprising a sequence of observations is then compared against the reference HMMs by computing



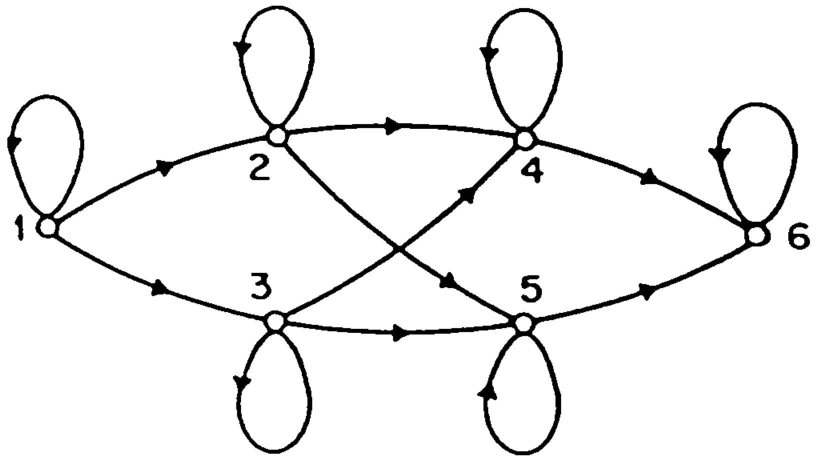
Ergodic 4-State Model



Simple Bakis Model



Bakis Model with Jump States



Cross-Coupled Model

Fig 1.3 Example HMMs

the probability that each HMM might have generated that sequence. The HMM which displays the highest probability of having produced the test pattern is deemed the nearest match, and the pattern is so classified.

1.4. Types of HMMs

As per the above discussion, the elements which make up a Hidden Markov Model are

(1) N, the number of states in the model.

(2) an $N \times N$ matrix of transition probabilities, A, where element a_{ij} gives the probability of moving from state i to state j

(3) an $N \times 1$ initial state distribution vector, π, where element π_i gives the probability that the model begins in state i

(4) a set of observation probability distributions, one for each state, which give the probability of generating a given observation from within state j.

Note that the actual connections between model states in the state diagram are realized by the A matrix. For example, when element $a_{ij} = 0$ this indicates that no transition is possible (i.e. no path exists) from state i to state j, while a non-zero a_{ij} implies that a connection does exist.

Starting from this basic specification, numerous variations of HMM are possible reflecting, for example, the topology (i.e. pattern of connections) of the state diagram, the form of the observation probability densities, the modelling of state durations, etc. Here we focus on the form of the observation probability densities.

Perhaps the simplest such variation is the discrete density HMM (DHMM), in which the set of possible outputs are restricted to a finite set, or alphabet. Under this assumption, the state output distribution functions are then discrete probability densities, and it becomes possible to replace the collection of N output distributions with a single $N \times M$ matrix, B, where M is the length of the alphabet, and where b_{jk} gives the probability of generating the k^{th} alphabet entry while in state j.

However, while there are physical systems where a discrete alphabet of observations is entirely appropriate, other systems (such as the speech signals under discussion) wouldn't normally be restricted to a finite set of outputs. For example, an observation might consist of a D-element vector of LPC coefficients derived from a frame of speech samples. Then the components of the observation would be continuous-valued (within the precision bounds of the hardware involved), and the set of possible outputs would constitute a continuous D-dimensional space.

In order to approximate a continuous system of this sort by a discrete HMM, two vector quantization steps are necessary. First, during model building, the continuous vector space has to be replaced by the finite alphabet, or 'codebook', mentioned earlier. Typically, the training data might be clustered, and the centroids of the clusters chosen to make up the codebook. Second, during model use, a preliminary pattern matching operation occurs, whereby each continuous-valued observation is compared with the codebook entries and replaced by the index of its nearest match.

But in general, when the continuous-valued observations become quantized, the modelling of the physical system may become degraded [19]. Consequently, an alternative to the discrete HMM is the continuous-density HMM (CDHMM), in which the observations are not quantized but instead remain continuous-valued. This means that, for the D-element observations described above, each of the N state observation probability distributions must take the form of a continuous density function defined on the D-dimensional vector space.

To model these continuous density functions, a result by Kolmogorov [20] is commonly invoked which states that a continuous function can be approximated to an arbitrary accuracy by a sum of suitably sized and positioned 'bumps'. Any elliptically symmetric function (i.e. contours are ellipses) constitutes a suitable bump, and D-dimensional multivariate Gaussian shapes are typically used for the purpose. A sum of M of these shapes is referred to as a 'mixture'. Each Gaussian in the mixture can be described in terms of a D-element means vector and a $D \times D$ covariance matrix. Finally, a vertical scaling factor is required for each Gaussian to ensure that the total volume enclosed by the mixture equals 1 (as per a probability density).

In summary, one observes a trade-off at work between the two different types of HMM. The CDHMM offers a more precise model of the physical system, leading (hopefully) to greater recognition accuracy. This is offset by greater complexity, plus a larger number of parameters to be estimated from the finite body of training data.

Over the course of time, many other types of HMM have been introduced [21,22,23] which attempt to refine the balance point of this trade-off and/or provide additional features. But today almost all applications aimed at speech recognition employ some form of continuous HMM.

1.5. HMM Calculations

Regardless of the type of HMM under consideration, the literature identifies three basic problems that must be solved in order for the Hidden Markov Model to be useful in real-world applications [24]. Let the vector $O = o_1\, o_2 \ldots o_T$ represent a sequence of observations made over T consecutive time steps, and let $Q = q_1\, q_2 \ldots q_T$ represent a sequence of states. Then the three problems are :

Explanation	Given the observation sequence O and model $M = (A, B, \pi)$, compute the sequence of states, Q, most likely to have taken place as the observations were generated
Classification	Given the observation sequence O and model M, compute $P(O \mid M)$, the probability of the observation sequence given the model. The model for which $P(O \mid M)$ is largest is generally taken as the best 'match' for the sequence of observations.
Training	Given the observation sequence O, compute the model parameters $M = (A, B, \pi)$ which maximize $P(O \mid M)$

The third of these tasks is by far the most challenging, and several solution methods have been suggested, although the favorite for most applications seems to be the classic Baum-Welch algorithm. This estimates the state transition probabilities by calculating

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \tag{1.1}$$

and similarly for the other parameters, depending upon the form of the model (discrete or continuous) .

The Baum-Welch algorithm owes its popularity to several advantages :

(a) It is mathematically sound, and is guaranteed to arrive at a solution to the task.

(b) It is reasonably efficient. By employing a device (the doubling-up procedure) from the mathematical toolbox known as Dynamic Programming [25], it is able to streamline an otherwise prohibitive amount of computation to a manageable level.

(c) It is readily adapted to handle multiple sequences of observations, which are typically necessary in order to provide sufficient data to make reliable estimates of all the model parameters.

On the other hand, the Baum-Welch algorithm displays one potential disadvantage : namely, it may not produce the 'best' possible model for the given training data. To carry out the model-building calculations, the algorithm must be given an initial set of parameter values, which are then 're-estimated' with each iteration of the formulas. In this way, the algorithm performs a type of hill-climb from the initial set of parameter values to the final guaranteed solution. However, the final solution is only a local optimum of the objective function $P(O \mid M)$, which may or may not be the global optimum depending upon whether the initial parameter values were located on the highest hill. In general, unless something of the 'landscape' of the objective function is known, there can be no way of being certain that the global optimum has been achieved.

1.6. Goal of the Research

As an alternative to the standard Baum-Welch algorithm, the goal of the current research is to develop a training procedure which is guaranteed to produce the global optimum to the objective function for the given training data.

This is no easy task. The problem of finding the global optimum has received little or no attention in the speech recognition literature in the past 30 years, presumably on account of its difficulty. Accordingly, the scope of this research project was limited initially to the case of finding the global optimum of $P(O \mid M)$ for a discrete HMM and a single observation sequence as training data.

1.7. Value of this Analysis

Given that discrete HMMs are generally viewed as yesterday's technology, and that almost all real applications employ continuous HMMs with multiple training sequences per model, is there any point to an analysis of discrete HMMs with a single observation sequence?

We believe so, for several reasons :

First is the very practical reason that devising an algorithm for the global optimum HMM is a difficult problem to solve. As suggested in the previous section, it makes sense to simplify the problem as much as possible in order to maximize the chances of finding a solution.

A second reason is the possibility that insights gained during this analysis will shed light on the general situation involving continuous HMMs and multiple sequences. From this simple 'fruit fly' case, we may be able to make inferences about the more complicated cases.

A third reason is the prospect of extending the solution in ways which turn it into a useful product. For example, the Global-Optimum algorithm (could it be found) might be extended for multiple training sequences in much the same way that the Baum-Welch algorithm has been extended for multiple training sequences. Baum-Welch accomplishes this by calculating the numerator and denominator values of equation (1.1) for each individual training sequence, using the same set of model parameters (i.e. the current re-estimation) across all of the training sequences. Those numerators and denominators are then accumulated separately, after which the division step is performed as normal, and the process is repeated until convergence of the model parameters occurs. Imagine now that the Global-Optimum algorithm has been employed to produce the optimal set of parameters for each individual training sequence. Then the numerator and denominator values of equation (1.1) are calculated for each training sequence, using the same forward-backward calculations employed by Baum-Welch--except that this time, the model parameters used with a sequence are the optimal set for that sequence. Finally, the numerators and denominators are accumulated, after which the division step is performed. In this way, multiple sequences can be used to produce an 'average' model, via a process which resembles a single iteration of Baum-Welch *.

A final reason for carrying out the analysis is that this is a worthwhile problem in its own right. The lessons to be learned transcend HMM technology to include pattern recognition in general. For example, a global optimum HMM which has been derived for a sequence provides a very useful description of the amount and type of 'pattern' which may be present in that sequence. In addition, it is interesting to note that Discrete HMMs share certain similarities with quantum mechanics and with Superstring Theory (see Section 3.5.), so that progress within the context of Hidden Markov modelling might help to illuminate those disciplines also.

* Making the assumption that different observation sequences are conditionally independent given the model, then

$$P(O^{(1)}\, O^{(2)} \ldots O^{(K)} \,|\, M) = P(O^{(1)} \,|\, M) \times P(O^{(2)} \,|\, M) \times \ldots \times P(O^{(K)} \,|\, M)$$

Theoretically, this should provide a more satisfactory objective function to be maximized for multiple training sequences--although no known algorithm actually uses it.

1.8. Value of the Global Maximum

A similar question is, why bother with the global optimum, anyway?

Levinson et al [26] appeared to raise this question in their 1983 paper, in which a known Hidden Markov Model was employed to generate observation sequences which were subsequently used to train other HMMs . They implied that

(a) the original source model represents the 'best' model of the data, and that

(b) the failure of the secondary models to share the parameter values of the original did not degrade their performance significantly

Once again, we feel there are several reasons why the global optimum is worth finding and using :

The use of global optimum models should help at recognition time by increasing the resolving power between speech models. Recognition accuracy requires that the probability of the test sequence given the correct model should be high, while the probability of the test sequence given an incorrect model should be low. Because the global optimum models are more highly focused on their respective training data, they should demonstrate correspondingly greater powers of discrimination.

Secondly, if the underlying mechanisms which relate observation sequences to HMMs can be understood, it may be that the global optimum is actually the easiest (most computationally efficient) to find--in the way that the highest, most visible, peak in a landscape is usually the easiest to spot*. The result could be an algorithm which is more computationally efficient than Baum-Welch.

* As a second example, compare the effort in finding the longest path through a network with, say, finding the <u>second</u> longest path.

Finally, the global optimum model may be necessary in terms of resolving a certain theoretical issue connected with model training :

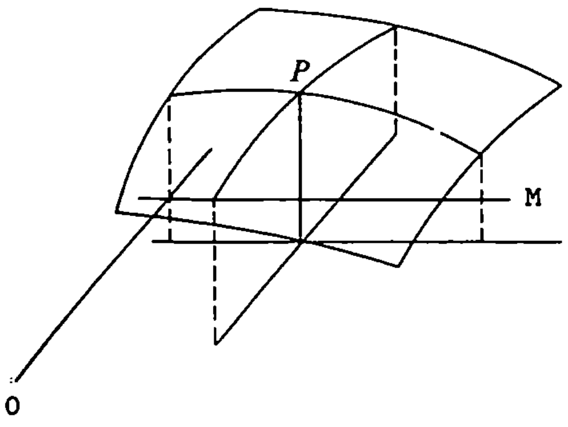
Model training has traditionally consisted of optimizing the objective function $P(O|M)$ = Probability($O|M$) , where $M = (A, B, \pi)$ is the given model, and where the otimization is performed over the space of possible models. This objective function is amenable to mathematical analysis, yet perhaps a more appropriate objective is $P(O,M)$ (i.e. the joint rather than the conditional probability function), for this reason--

When an HMM is 'kick-started' into action, it will generate a sequence of outputs [27]. The exact sequence will vary from trial to trial as per the stochastic nature of HMMs, and yet one expects that, for a well-trained model, the training data will be typical of the outputs in the sense that the mean of the population of outputs will equal the training data.

In other words, one wants a model M which satisfies two maximizations simultaneously--

it maximizes the likelihood of the training sequence over the space of models

it maximizes the likelihood of the training sequence over the space of training sequences



Clearly, this is accomplished by optimizing the joint probability distribution, $P(O,M)$, since

$$\frac{\partial P(O,M)}{\partial M} = 0 \qquad \text{handles the first condition}$$

$$\frac{\partial P(O,M)}{\partial O} = 0 \qquad \text{handles the second condition}$$

and the optimization of $P(O,M)$ performs both of these simultaneously.

Assuming this reasoning is correct, bear in mind also that since

$$P(O,M) = P(O|M) \times P(M)$$

then optimizing $P(O,M)$ requires both a large $P(O|M)$ and a large $P(M)$.

Global optimum models offer the largest possible $P(O|M)$. If, in addition, one argues that , being the most 'visible' (prominent), they naturally merit a larger likelihood of occurrence, $P(M)$, then the global optimum models also become the likely candidates for optimizing $P(O,M)$.

1.9. Organization of the Thesis

This thesis discusses a variety of different attempts to solve the global optimization problem. To date, none of these has been wholely successful. To arrive quickly at the most promising attempt, the reader should skip directly to Chapter 5 . On the other hand, the less developed attempts do make indirect contributions, and so are included here for completeness.

The first attempt was a straight-forward attack upon the problem using standard methods of calculus. The intention was to develop formulas for uncovering the complete set of optima and selecting the largest from among them.

The initial difficulty with this calculus approach is the very large number of partial differentiations which must be carried out *, and the equally large system of simultaneous equations which must be solved. Therefore, preliminary to this line of attack, mathematical tools were sought which could perform the differentiations as a block for all the elements of A simultaneously (and likewise for B and for π), and which could also assemble those derivatives into appropriate matrix equations ready for solution. Suitable tools were eventually found, and are collected together under the title of "Matrix Derivatives" in Appendix A . By and large, these tools were 'discovered' empirically without knowledge of existing source materials and, while many were later verified when sources came along, some have yet to be found in any other source. The exposition in Appendix A reflects this empirical development and makes little attempt at mathematical rigor, but merely presents the results as a primer for the mathematical operations carried out in other parts of the thesis.

* one each for the N^2 elements of A , the $N \times M$ elements of B , and the N elements of π

Armed with Matrix Derivatives, the way was clear to carry out the differentiations and attempt to solve the matrix equations. Chapter 2 presents some of the initial headway which was made with this analysis. Chapter 2 also establishes an upper bound on the solutions which occur within the interior of the solution space, given by

$$\prod_{k=1}^{M} \left(\frac{r_k}{T}\right)^{r_k}$$

where $$T = \sum r_k$$

and where r_k records the number of occurrences of codebook symbol k over the given training sequence.

However, a shortcoming of calculus-based optimization is that it tends to focus on interior solutions only, and to ignore endpoint extrema on the boundaries of the solution space. Chapter 3 opens with a demonstration that the global optimum solution must occur on a boundary of the solution space, where the different boundaries correspond to parameter sets in which one or more of the elements of A , B or π equal zero. Chapter 3 then goes on to show that these boundaries

are equivalent to fragments of the original objective function

include terms which reflect the state sequences the HMM might pass through as it generates the training sequence

have differing dimensions (vertex, edge, plane, etc) according to the number of non-zero elements of A , B and π

display a natural hierarchy (as a vertex can belong to an edge, an edge to a plane, etc)

The final contribution of Chapter 3 is a mechanical procedure for identifying the list of fragments which is relevant for a specific model size and training sequence.

Having established the need for optimizing the boundary fragments, and having learned how to identify those fragments, what follows is a succession of attempts to determine the global maximum of a given fragment.

The first of these is the Matrix Derivatives approach, now extended to deal with fragments.

The second attempt involves a relatively recent optimization technique known as Geometric Programming, which is naturally suited to objective functions of this form, and which turns out to be the 'dual' of the derivatives approach (as with 'primal-dual' methods in Linear Programming, for example).

The third attempt uses a new hill-climbing procedure. Unlike the previous calculus-based methods, hill-climbing cannot normally be relied upon to deliver a global optimum, unless the 'landscape' of the objective is reasonably well understood (the approximate location or, at least, the <u>number</u> of hills is known in advance). The intention here was to investigate certain formulas for nominating the initial starting point, to determine whether this new hill-climbing procedure, in conjunction with the 'right' starting point, would consistently yield the global optimum of the fragment.

All three of these approaches threw up intriguing insights worthy of further investigation, and all three were useful to what followed in the thesis. On the other hand, none of them could be developed sufficiently to achieve their stated goal of globally optimizing the fragment. Furthermore, their developments are quite 'dense', mathematically. For the latter two reasons, the thesis chapters which describe these approaches have been relegated to Appendices B , C and D , to be replaced by a summary in Chapter 4 .

The three methods of attack just described were all basically deductive in nature, and equally applicable to either the complete objective function or any of its fragments. The approach taken in Chapter 5 was the opposite of these : First, to collect examples of the global optimum solution for the complete objective function, and then attempt to <u>induce</u> the rules for finding them.

Based upon the body of examples, useful patterns did indeed appear. It seems that, in the great majority of cases, the global optimum solution is given by a single-term fragment, generally of lowest possible dimension. But even when the optimum is produced from a multi-term fragment, this fragment contains a 'singleton' at its nucleus together with terms which are related to it in a distinctive way--much as a crystal grows around a 'seed'.

Chapter 5 describes how these observations were converted into an algorithm for generating Hidden Markov Models, including the critical procedure for identifying the single-term 'seeds' from among the exponentially large number of candidate terms. It then describes a series of four tests which were carried out in order to help confirm that the new algorithm produces global optimum solutions.

Finally, Chapter 6 offers a summary plus recommendations for future work.

2. Interior Solutions

2.1. Analysis using Matrix Derivatives

Here we begin the search for Markov matrices A B & π which globally maximize the probability of reproducing the given training sequence, represented by $P(o_1\, o_2 .. o_T)$.

The general plan of attack used within this chapter is the standard calculus optimization strategy, involving differentiation, etc. However, in order to cope with the large number of independent variables involved, the approach taken here is to express the objective function and the necessary manipulations in terms of matrices. The reader is referred to Appendix A for a tutorial on the differentiation of matrices and their use in expressing the relevant constraints.

The objective is to

$$\text{Maximize} \quad \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \ldots A'\ \mathrm{Diag}(BD_2)\ A'\ \mathrm{Diag}(BD_1)\ \pi$$
$$- \lambda_1\,(\underset{1\text{x}N}{\pi'}\ \underset{N\text{x}1}{\mathbf{1}} - 1) - \underset{1\text{x}N}{\lambda_2'}\,(\underset{N\text{x}M}{B}\ \underset{M\text{x}1}{\mathbf{1}} - \underset{N\text{x}1}{\mathbf{1}}) - \underset{1\text{x}N}{\lambda_3'}\,(\underset{N\text{x}N}{A}\ \underset{N\text{x}1}{\mathbf{1}} - \underset{N\text{x}1}{\mathbf{1}})$$

The first term equals $P(o_1\, o_2 .. o_T)$ [28] and the remaining terms are Lagrange terms which express the constraints that π B & A respectively must equal Markov matrices.

Note that

> D_t is an $M \times 1$ column vector of zeros with a single 1 to mark the codebook symbol which occurred at time t
>
> Therefore BD_t extracts the k^{th} column of B, where symbol k occurred at time t

and Diag(BD_t) represents the diagonal matrix which contains BD_t along the major diagonal

Note also that

$\mathbf{1}$ is a column vector (of appropriate size) consisting of all 1's

the expression $X\mathbf{1} - \mathbf{1} = 0$

or $X\mathbf{1} = \mathbf{1}$

demonstrates that the rows of X each sum to 1

and the λ_i are appropriate Lagrange multipliers

To maximize $P(o_1 o_2 .. o_T)$ we

perform a partial differentiation of the full expression with respect to each unknown array,

set each result equal to zero,

and attempt to solve the simultaneous system of matrix equations for the unknowns

Differentiating--

w.r.t. π

$$\text{Diag}(BD_1)\ A\ \text{Diag}(BD_2)\ A \ldots A\ \text{Diag}(BD_T)\ \mathbf{1} \\ - \lambda_1 \mathbf{1} = \mathbf{0}$$

w.r.t. B

$$\begin{aligned} & \text{Diag}[\ \mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ ] \times \pi D_1' \\ + \quad & \text{Diag}[\ \mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ ] \times A'\ \text{Diag}(BD_1)\ \pi D_2' \\ \cdot \\ \cdot \\ + \quad & \text{Diag}[\ \mathbf{1}'\ ] \times A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi D_T' \\ & - \underset{N\times1}{\lambda_2} \underset{1\times M}{\mathbf{1}'} = \mathbf{0} \end{aligned}$$

w.r.t. A

$$\begin{aligned} & \text{Diag}(BD_1)\ \pi \times \mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2) \\ + \quad & \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi \times \mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_3) \\ \cdot \\ \cdot \\ + \quad & \text{Diag}(BD_{T-1})\ A' \ldots A'\ \text{Diag}(BD_1)\ \pi \times \mathbf{1}'\ \text{Diag}(BD_T) \\ & - \underset{N\times1}{\lambda_3} \underset{1\times N}{\mathbf{1}'} = \mathbf{0} \end{aligned}$$

w.r.t. λ_1

$$-(\pi' \mathbf{1} - 1) = 0 \qquad \text{or} \qquad \pi' \mathbf{1} = 1$$

w.r.t. λ_2

$$-(B\,\mathbf{1} - 1) = 0 \qquad \text{or} \qquad B\,\mathbf{1} = 1$$

w.r.t. λ_3

$$-(A\,\mathbf{1} - 1) = 0 \qquad \text{or} \qquad A\,\mathbf{1} = 1$$

We have 6 equations in the 6 unknowns π, B, A, λ_1, λ_2, λ_3

From here, the standard procedure [29] is to :

solve first for the Lagrange multipliers

then substitute their values to get a reduced system of equations in A B & π

First the π equation--

Transpose... then post-multiply by π

$$1' \text{ Diag}(BD_T)\ A' \ldots A' \text{ Diag}(BD_2)\ A' \text{ Diag}(BD_1)\ \pi = \lambda_1 \underbrace{1'\pi}_{\text{scalar 1}} = \lambda_1$$

The left hand side equals $\text{P}(o_1 o_2 .. o_T)$, which we are trying to maximize.

Therefore λ_1 equals the (sought for) maximum value of the objective function

Next the B equation--

Post-multiply by $\underset{\text{Mx1}}{\mathbf{1}}$

$$
\begin{aligned}
& \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\,]\ \pi D_1'\,\mathbf{1} \\
+\ & \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,]\ A'\ \text{Diag}(BD_1)\ \pi D_2'\,\mathbf{1} \\
\cdot\ & \\
\cdot\ & \\
+\ & \text{Diag}[\,\mathbf{1}'\,]\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi \underbrace{D_T'\,\mathbf{1}}_{\text{scalar 1}} = \lambda_2 \underbrace{\mathbf{1}'\,\mathbf{1}}_{\text{scalar M}}
\end{aligned}
$$

or

$$
\begin{aligned}
& \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\,]\ \pi \\
+\ & \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,]\ A'\ \text{Diag}(BD_1)\ \pi \\
\cdot\ & \\
\cdot\ & \\
+\ & \text{Diag}[\,\mathbf{1}'\,]\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi = M \underset{\text{Nx1}}{\lambda_2}
\end{aligned}
$$

Now pre-multiply by $\underset{\text{1xN}}{\mathbf{1}'}$

$$
\begin{aligned}
& \mathbf{1}'\ \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\,]\ \pi \\
+\ & \mathbf{1}'\ \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,]\ A'\ \text{Diag}(BD_1)\ \pi \\
\cdot\ & \\
\cdot\ & \\
+\ & \mathbf{1}'\ \text{Diag}[\,\mathbf{1}'\,]\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi = M\,\mathbf{1}'\,\lambda_2
\end{aligned}
$$

or

$$
\begin{array}{llllllll}
 & \mathbf{1}' & \mathrm{Diag}(BD_T) & A' \ldots A' & \mathrm{Diag}(BD_2) & A' & & \pi \\
+ & \mathbf{1}' & \mathrm{Diag}(BD_T) & A' \ldots A' & & A' & \mathrm{Diag}(BD_1) & \pi \\
\bullet & & & & & & & \\
\bullet & & & & & & & \\
+ & \mathbf{1}' & & A' \ldots A' & \mathrm{Diag}(BD_2) & A' & \mathrm{Diag}(BD_1) & \pi = M \underset{1\mathrm{xN}}{\mathbf{1}'} \underset{\mathrm{Nx1}}{\lambda_2}
\end{array}
$$

The right hand side is M times the sum of the elements of λ_2.

The left hand side is a series of objective functions with successive occurrences of $\mathrm{Diag}(BD_T)$ removed (or replaced by the identity matrix I). Missing out the proof, this is $\mathrm{P}(o_1\, o_2 .. o_T)$ with successive o_t removed

--i.e.

$$\mathrm{P}(o_2 .. o_T) + \mathrm{P}(o_1\, o_3 .. o_T) + \ldots + \mathrm{P}(o_1 .. o_{T-1}) = M\, \mathbf{1}'\, \lambda_2$$

Now the A equation--

Post-multiply by A'

$$\underbrace{\text{Diag}(BD_1)\ \pi}_{\text{Nx1}} \ \text{x}\ \underbrace{\mathbf{1}'\,\text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'}_{\text{1xN}}$$

$$+\ \underbrace{\text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi}_{\text{Nx1}} \ \text{x}\ \underbrace{\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,\text{Diag}(BD_3)\ A'}_{\text{1xN}}$$

$$\vdots$$

$$+\ \underbrace{\text{Diag}(BD_{T\text{-}1})\ A' \ldots A'\,\text{Diag}(BD_1)\ \pi}_{\text{Nx1}} \ \text{x}\ \underbrace{\mathbf{1}'\ \text{Diag}(BD_T)\ A'}_{\text{1xN}}$$

$$= \lambda_3\, \mathbf{1}'A' = \underset{\text{Nx1}}{\lambda_3}\ \underset{\text{1xN}}{\mathbf{1}'}$$

Ⓐ If two matrices are equal, then their traces are equal

Ⓑ The trace of a sum equals the sum of the traces

Ⓒ The trace of the outer product of two vectors equals their inner product :

$$\text{Tr}(\underset{\text{Nx1}}{X}\ \underset{\text{1xN}}{Y'}) = \underset{\text{1xN}}{Y'}\ \underset{\text{Nx1}}{X}$$

Consequently,

$$\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{x}\ \text{Diag}(BD_1)\ \pi$$

$$+\ \mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_3)\ A'\ \text{x}\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi$$

$$\vdots$$

$$+\ \mathbf{1}'\ \text{Diag}(BD_T)\ A'\ \text{x}\ \text{Diag}(BD_{T\text{-}1})\ A' \ldots A'\ \text{Diag}(BD_1)\ \pi$$

$$= \mathbf{1}'\,\lambda_3$$

The right hand side is the sum of the elements of λ_3 .

The left hand side is t - 1 copies of the objective function.

Thus

$$(t - 1)\ \mathrm{P}(o_1\, o_2 .. o_T) = \mathbf{1}'\, \lambda_3$$

or

$$(t - 1)\ \lambda_1 = \mathbf{1}'\, \lambda_3$$

2.2. Optimal Feasible Solutions

The analysis can (and will) be continued along these lines. But for now, returning to the π equation...

Let $L = \text{Diag}(BD_1)\ A\ \text{Diag}(BD_2)\ A \ldots A\ \text{Diag}(BD_T)\ \mathbf{1} = \{\text{ LHS }\}$

Then the π equation states that

$$L = \lambda_1 \mathbf{1} \tag{2.1}$$

which is a constant vector, whose value

$$\lambda_1 = \text{the max. value of the objective function } P(o_1 o_2 .. o_T)$$

Up till this point, the solution space has been defined as the set of (A, B, π) triples which are Markov.

Now, based on equation (2.1), a feasible solution might be defined as a triple for which

$$L = \text{Diag}(BD_1)\ A\ \text{Diag}(BD_2)\ A \ldots A\ \text{Diag}(BD_T)\ \mathbf{1}$$

produces a column vector consisting of identical values. If this L calculation doesn't yield a constant vector, the (A, B, π) triple must be ruled out as infeasible (i.e. not a stationary point).

Finally, an <u>optimal</u> feasible solution is a feasible solution for which the constant value produced is the largest one possible (because that constant equals the value of $P(o_1 o_2 .. o_T)$, which is to be maximized).

The remaining sections of this chapter will demonstrate that, within the interior of the solution space,

> an upper bound exists on the value of the feasible solutions (and, consequently, the stationary points), which is equal to
>
> $$\gamma = \prod_{k=1}^{M} \left(\frac{r_k}{T}\right)^{r_k}$$
>
> where $$T = \sum r_k$$
>
> and where the R vector is calculated as
>
> $$R = \sum_t D_t$$
>
> and records the number of occurrences of codebook symbol k over the given symbol sequence.

In other words, within the interior of the solution space, no (A, B, π) triple exists which possesses a constant vector L (feasible) which is greater than γ (optimal). For any triple, either the L vector is not constant or, if it is, that constant value is $\leq \gamma$.

(Technically, the proof is carried out over the interior of the π-space, which includes the interior of the (A, B, π) space as a subset.)

2.3. Proof of the Conjecture

2.3.1. Preliminary Problem

The proposed proof relies upon the solution of a related, but simpler, optimization problem :

Maximize the product $$x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$$

subject to the requirement that $$\sum_{k=1}^{n} x_k = 1$$

For the special case $n = 2$, this product can be written

$$x^p y^q = x^p (1-x)^q$$

Differentiating

$$p\, x^{p-1} (1-x)^q - q\, x^p (1-x)^{q-1} = 0$$

$$x^{p-1} (1-x)^{q-1} \, [p(1-x) - q x] = 0$$

$$p(1-x) - q x = 0 \qquad (\text{ for } 0 < x < 1)$$

$$(p+q)\, x = p$$

$$x = \frac{p}{p+q} \qquad \text{and} \qquad y = \frac{q}{p+q}$$

For the case when all of the $r_k = 1$, we invoke the Arithmetic-Geometric Mean inequality [30], which states that the arithmetic mean of a set of numbers is never less than the geometric mean :

for $\qquad x_k \geq 0 \qquad k = 1$ to n

$\qquad \alpha_k > 0$

and $\qquad \sum \alpha_k = 1$

then
$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \geq x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$$

with equality iff $\qquad x_1 = x_2 = \ldots = x_n$

Accordingly, let $\alpha_k = \frac{1}{n}$ and invoke the constraint $\sum x_k = 1$

Then
$$x_1^{\frac{1}{n}} x_2^{\frac{1}{n}} \cdots x_n^{\frac{1}{n}} \leq \frac{1}{n} \sum x_j = \frac{1}{n}$$

Raising both sides to the power n

$$x_1 x_2 \cdots x_n \leq \left(\frac{1}{n}\right)^n$$

The upper bound is achieved iff $x_1 = x_2 = \ldots = x_n$ which, with $\sum x_k = 1$,

implies $\qquad x_1 = x_2 = \ldots = x_n = \frac{1}{n}$

In summary, to maximize $\quad x_1 x_2 \cdots x_n \quad$ where $\quad \sum x_k = 1$

set each $x_k = \frac{1}{n}$, yielding the maximum value $\left(\frac{1}{n}\right)^n$

For the general case, we can apply the techniques of matrix differentiation :

Let $$e = x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$$

Then the task is to maximize

$$e - \lambda (X' \mathbf{1} - 1)$$

where $X = [x_1 \ x_2 \ \dots \ x_n]'$

and λ is the Lagrange multiplier

Differentiating with respect to X

$$\begin{bmatrix} r_1 x_1^{r_1 - 1} x_2^{r_2} \cdots x_n^{r_n} \\ \vdots \\ r_n x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n - 1} \end{bmatrix} - \lambda \mathbf{1} = 0$$

or

$$e \begin{bmatrix} \frac{r_1}{x_1} \\ \vdots \\ \frac{r_n}{x_n} \end{bmatrix} = \lambda \mathbf{1} \qquad (2.2)$$

Then

$$e X' \begin{bmatrix} \frac{r_1}{x_1} \\ \vdots \\ \frac{r_n}{x_n} \end{bmatrix} = \lambda X' \mathbf{1}$$

$$e \, T = \lambda \qquad \text{where } T = \sum r_k$$

Substituting for λ in equation (2.2),

$$e \begin{bmatrix} \frac{r_1}{x_1} \\ \vdots \\ \frac{r_n}{x_n} \end{bmatrix} = e\,T\,\mathbf{1}$$

$$e\,\mathrm{Diag}(X) \begin{bmatrix} \frac{r_1}{x_1} \\ \vdots \\ \frac{r_n}{x_n} \end{bmatrix} = e\,T\,\mathrm{Diag}(X)\,\mathbf{1}$$

$$e \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} = e\,T\,X$$

and

$$X = \frac{1}{T} \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix} = \frac{1}{T} R$$

So in general, to maximize $x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$

subject to $\sum x_k = 1$

where $\sum r_k = T$

the solution is $x_k = \frac{r_k}{T}$

which gives the maximum value $\prod_{k=1}^{n} \left(\frac{r_k}{T}\right)^{r_k}$

Note also that $r_k = 0$ is permissible, provided that 0^0 is defined to equal 1 .

2.3.2. Special Cases

To re-cap from Section 2.2., an optimal feasible solution is one for which the calculation

$$L = \text{Diag}(BD_1)\ A\ \text{Diag}(BD_2)\, A\ \ldots\ A\ \text{Diag}(BD_T)\ \mathbf{1}$$

produces a column vector of identical values which are the largest possible.

It is conjectured that, within the interior of the π-space (and, therefore, the interior of the solution space), no (A, B, π) triple exists which is both feasible (rows of L are equal) and produces an objective value $\lambda_1 > \gamma$.

This section provides some special cases for which the conjecture is easily proved.

As a first example, let $T = 3$

and make the substitutions

$$\begin{aligned} BD_1 &= X \\ BD_2 &= Y \\ BD_3 &= Z \end{aligned}$$

Now, consider $A = \mathrm{I}$ (the $N \times N$ identity matrix).

Then

$$L = \mathrm{Diag}(X)\ \mathrm{I}\ \mathrm{Diag}(Y)\ \mathrm{I}\ \mathrm{Diag}(Z)\ \mathbf{1}$$

$$= \begin{bmatrix} x_1 y_1 z_1 \\ \vdots \\ x_N y_N z_N \end{bmatrix}$$

The X Y & Z were drawn from the columns of B and may (or may not) include repetitions of columns. We can replace the terms $x_j y_j z_j$ by the equivalent formulation :

$$L = \begin{bmatrix} b_{11}^{r_1}\, b_{12}^{r_2} \cdots b_{1M}^{r_M} \\ \vdots \\ b_{N1}^{r_1}\, b_{N2}^{r_2} \cdots b_{NM}^{r_M} \end{bmatrix} \qquad \text{where some of the } r_k \text{ may equal } 0$$

Clearly, each element of L is of the form optimized in Section 2.3.1. They all share the same maximum value, which can be attained for all elements simultaneously by setting $b_{jk} = r_k / T$. In this way, the objective value equals

$$\gamma = \prod_{k=1}^{M} \left(\frac{r_k}{T}\right)^{r_k}$$

and it is clear no objective value could be higher. Therefore, the conjecture holds for the special case $A = \mathrm{I}$.

Consider next the class of triples where the columns of B are constant (rows of B are identical) --i.e.

$$B = \mathbf{1}\ [\,b_1\, b_2 \ldots b_M] \qquad \text{where } [\,b_1\, b_2 \ldots b_M]\ \mathbf{1} = 1$$

Make the substitutions

$$\begin{aligned} BD_1 &= X = x\,\mathbf{1} \\ BD_2 &= Y = y\,\mathbf{1} \\ BD_3 &= Z = z\,\mathbf{1} \end{aligned}$$

Then

$$\begin{aligned} L &= \mathrm{Diag}(X)\ \mathrm{I}\ \mathrm{Diag}(Y)\ \mathrm{I}\ \mathrm{Diag}(Z)\ \mathbf{1} \\ &= (x\mathrm{I})\ A\ (y\mathrm{I})\ A\ (z\mathrm{I})\ \mathbf{1} \\ &= x y z\ A^2\ \mathbf{1} \\ &= x y z\ \mathbf{1} \\ &= b_1^{r_1} b_2^{r_2} \ldots b_M^{r_M}\ \ 1 \end{aligned}$$

which, as before, is both feasible and optimal iff $b_k = r_k / T$, regardless of the value of A (provided only that it is Markov).

As a third special case, consider the class of triples where the rows of A are identical --i.e.

$$A = \mathbf{1}\ [\,a_1\, a_2 \ldots a_N\,] \qquad \text{where } [a_1\, a_2 \ldots a_N]\ \mathbf{1} = 1$$

After substituting as before,

$$L = \text{Diag}(X)\, A\, \text{Diag}(Y)\, A\, \text{Diag}(Z)\, \mathbf{1}$$

Note also that

$$L = \lambda_1\ \mathbf{1}$$

implies

$$A\, L = \lambda_1\ A\ \mathbf{1} = \lambda_1\, \mathbf{1} = L$$

so that

$$\begin{aligned}
L &= A\, \text{Diag}(X)\, A\, \text{Diag}(Y)\, A\, \text{Diag}(Z)\, \mathbf{1} \\
&= \mathbf{1}\ [a_1\, a_2 \ldots a_N]\, \text{Diag}(X)\ \mathbf{1}\ [a_1\, a_2 \ldots a_N]\, \text{Diag}(Y)\ \mathbf{1}\ [a_1\, a_2 \ldots a_N]\, \text{Diag}(Z)\ \mathbf{1} \\
&= \mathbf{1}\ [a_1\, a_2 \ldots a_N]\ X\ [a_1\, a_2 \ldots a_N]\ Y\ [a_1\, a_2 \ldots a_N]\ Z \\
&= \text{Avg}(X)\ \ \text{Avg}(Y)\ \ \text{Avg}(Z)\ \ \mathbf{1} \\
&= [\text{Avg}(B_{C1})]^{r_1}\ [\text{Avg}(B_{C2})]^{r_2} \ldots [\text{Avg}(B_{CM})]^{r_M}\ \mathbf{1} \qquad (2.3)
\end{aligned}$$

where $\text{Avg}(X)$ is the weighted average of the elements of column vector X performed by pre-multiplication with $[a_1\, a_2 \ldots a_N]$

and B_{Cj} is column j of the B-matrix

In this case L is automatically feasible (rows equal). To optimize its constant value, note that

$$\begin{aligned}
\mathrm{Avg}(B_{C1}) + \mathrm{Avg}(B_{C2}) + \ldots + \mathrm{Avg}(B_{CM}) &= \mathrm{Avg}(B_{C1} + B_{C2} + \ldots + B_{CM}) \\
&= \mathrm{Avg}(B\,\mathbf{1}) \\
&= [\ a_1\, a_2 \ldots a_N\ ]\ B\,\mathbf{1} \\
&= [\ a_1\, a_2 \ldots a_N\ ]\ \mathbf{1} \\
&= 1
\end{aligned}$$

so that the expression for the constant value of L in equation (2.3) again matches the form and conditions of Section 2.3.1. Consequently, its value can be no greater than γ, which can be attained provided the elements of B can be chosen so that

$$\mathrm{Avg}(B_{Ck}) = r_k / T$$

As a final example, consider the class of triples where A is upper triangular (the HMM is left-to-right).

Note that a product of upper triangular matrices is also upper triangular. Consequently, the calculation

$$L = \mathrm{Diag}(X)\,A\,\mathrm{Diag}(Y)\,A\,\mathrm{Diag}(Z)\,\mathbf{1}$$

produces the result (details omitted)

$$L = \begin{bmatrix} \vdots \\ x_N y_N z_N a_{NN}^2 \end{bmatrix}$$

Suffice it to say that the elements of L are quite complicated except for the final element, which is

$$\begin{aligned} x_N y_N z_N a_{NN}^2 &= x_N y_N z_N \quad \text{(since the rows of } A \text{ sum to 1)} \\ &= b_{N1}^{r_1} b_{N2}^{r_2} \cdots b_{NM}^{r_M} \end{aligned}$$

As previously, this bottom row of L is maximized by $b_{Nk} = r_k / T$

which produces the value $\displaystyle\prod_{k=1}^{M} \left(\frac{r_k}{T}\right)^{r_k} = \gamma$

Furthermore, the other rows of L can't exceed it and still remain feasible.

Therefore the class of solutions with an upper triangular A also obeys the conjecture.

2.3.3. The General Case

The previous section whittled away several special cases, each of which obeys the conjecture that no feasible solution can exhibit an objective value greater than γ.

What remains is the class of solutions for which the rows of A and B are completely general. Here, the optimization of

$$L = \mathrm{Diag}(X)\ A\ \mathrm{Diag}(Y)\ A\ \mathrm{Diag}(Z)\ \mathbf{1}$$

is a true generalization of the optimization problem

maximize the product	$x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$
subject to the requirement that	$\sum_{k=1}^{n} x_k = 1$

in two ways:

Ⓐ Non-constant vectors (X, Y, Z) are involved now, rather than scalars

Ⓑ The elements of Z aren't allowed to multiply the elements of Y, etc. without first being averaged by the rows of A

In other words, an intervening step takes place between the usual scalar multiplications. The application of A calculates N weighted averages of Z which, because Z isn't constant and A doesn't have identical rows, won't necessarily have equal values.

The argument for the general case follows along geometrical lines--

Since

$$A \underset{N\times 1}{\mathbf{1}} = 1 \underset{N\times 1}{\mathbf{1}}$$

it is apparent that every Markov matrix has the eigenvalue 1 and eigenvector **1** .

Somewhat less obvious is the fact that none of the other eigenvalues of a Markov matrix can exceed 1 in magnitude (the 'spectral radius' of every Markov matrix is 1 [31,32]) .

Assume that the A matrix can be diagonalized, --i.e.

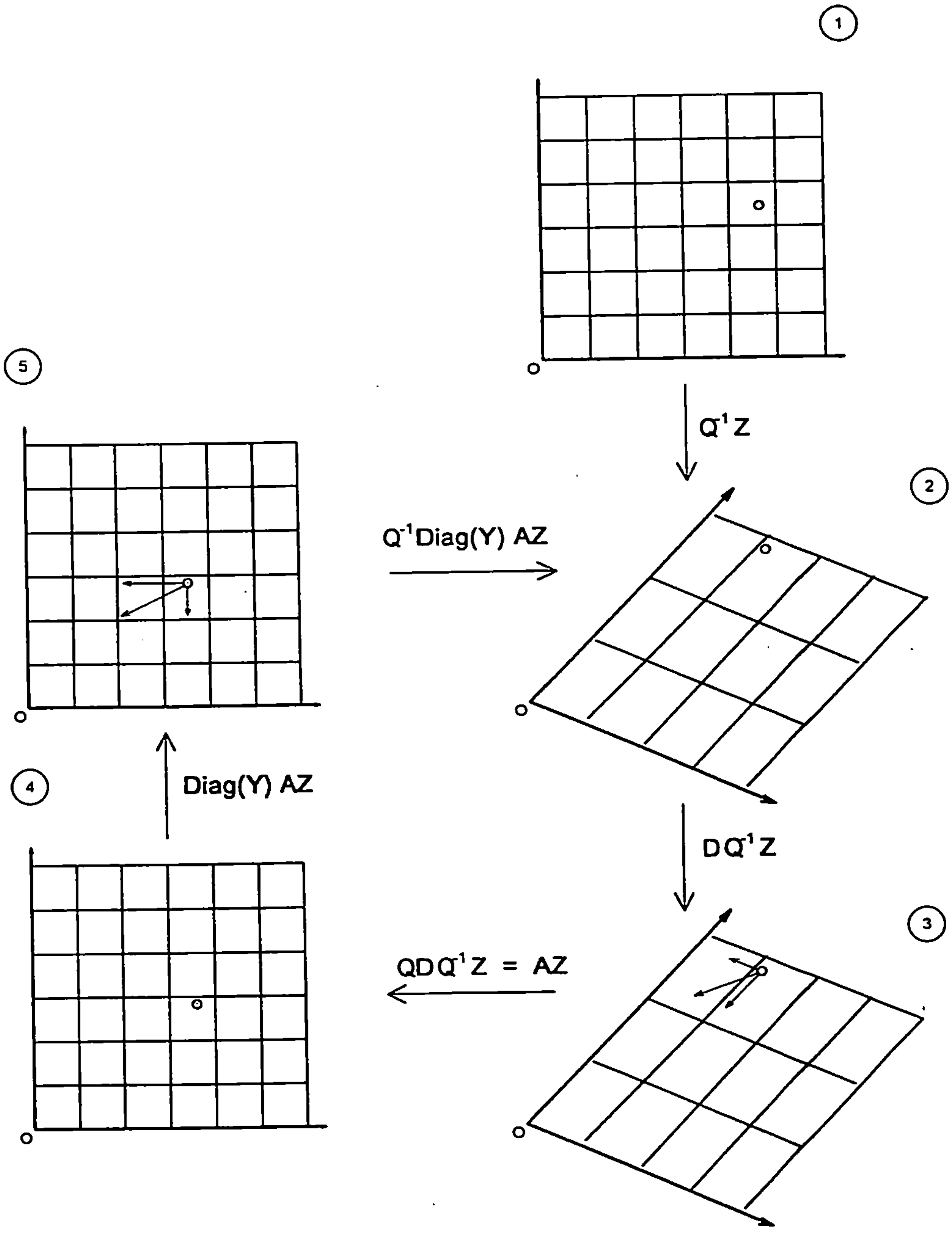
$$A = Q D Q^{-1}$$

Here D is the diagonal matrix consisting of A 's eigenvalues
and Q is the matrix of corresponding eigenvectors, arranged as column vectors

A can always be diagonalized if its eigenvalues are all distinct or, failing that, if the eigenvectors corresponding to repeated eigenvalues are linearly independent [33].
(For example, if an eigenvalue occurs with multiplicity 2 , can its corresponding eigenvectors span 2-Space?)

Writing
$$\begin{aligned} L &= \mathrm{Diag}(X)\, A\, \mathrm{Diag}(Y)\, A\, \mathrm{Diag}(Z)\, \mathbf{1} \\ &= \mathrm{Diag}(X)\, Q D Q^{-1}\, \mathrm{Diag}(Y)\, Q D Q^{-1} Z \end{aligned}$$

and with the aid of the diagram, we can visualize the evaluation of L proceeding from right to left :


1
$Q^{-1}Z$
2
$DQ^{-1}Z$
3
$QDQ^{-1}Z = AZ$
4
Diag(Y) AZ
5
Q^{-1}Diag(Y) AZ

$\textcircled{1} \rightarrow \textcircled{2}$ The multiplication $Q^{-1} Z$ doesn't move Z relative to the origin, but simply establishes a new coordinate system around them [34]. The new basis vectors are the eigenvectors of A, and the multiplication $Q^{-1} Z$ calculates the coordinates of Z within this new system.

$\textcircled{2} \rightarrow \textcircled{3}$ The application of D scales each coordinate of $Q^{-1} Z$ its respective diagonal entry in D which, because the eigenvalues of A are ≤ 1, always results in a contraction towards the origin.

$\textcircled{3} \rightarrow \textcircled{4}$ The application of Q simply re-establishes the original rectangular coordinate system, calculating the coordinates of $D Q^{-1} Z$ in this system.

$\textcircled{4} \rightarrow \textcircled{5}$ Y represents a selected column of the B matrix, whose elements are likewise ≤ 1. Multiplication by Y therefore produces a similar contraction towards the origin.

$\textcircled{5} \rightarrow \textcircled{2}$ We continue cycling around the lower loop, eventually halting at $\textcircled{5}$. With each transit of the loop, two contractions take place--first along the eigenvector basis vectors, and then along the rectangular basis vectors--the point migrating towards the origin with each contraction, until we arrive at the final set of coordinates, which is L.

Now compare

$$L \;=\; \mathrm{Diag}(X)\, Q\, D\, Q^{-1}\, \mathrm{Diag}(Y)\, Q\, D\, Q^{-1}\, Z$$

with the expression

$$\mathrm{Diag}(X)\, Q\;\; \mathrm{I}\, Q^{-1}\, \mathrm{Diag}(Y)\, Q\;\; \mathrm{I}\, Q^{-1}\, Z$$

where D has been replaced in each case by the identity matrix.

This alteration effectively removes half of the contractions, i.e. those occurring within the eigenvector coordinate system.

Therefore

$$\begin{aligned} L &= \mathrm{Diag}(X)\, Q\, D\, Q^{-1}\, \mathrm{Diag}(Y)\, Q\, D\, Q^{-1}\, Z \\ &\leq \mathrm{Diag}(X)\, Q\;\; \mathrm{I}\, Q^{-1}\, \mathrm{Diag}(Y)\, Q\;\; \mathrm{I}\, Q^{-1}\, Z \\ &= \mathrm{Diag}(X)\;\; \mathrm{I}\;\; \mathrm{Diag}(Y)\;\; \mathrm{I}\;\; Z \end{aligned}$$

But the first special case from Section 2.3.3. has already established that

$$\mathrm{Diag}(X)\;\; \mathrm{I}\;\; \mathrm{Diag}(Y)\;\; \mathrm{I}\;\; Z \qquad \leq \quad \gamma\, \mathbf{1}$$

Consequently, it is apparent that general (A, B, π) triples cannot exceed γ.

3. Boundary Solutions

3.1. Introduction

Chapter 2 presented the first applications of Matrix Derivatives for finding global optimum HMMs. The main result of the chapter was that, within the interior of the solution space, an upper bound exists on the value of the stationary points of $P(O)$, which is equal to

$$\gamma = \prod_{k=1}^{M} \left(\frac{r_k}{T}\right)^{r_k}$$

where $$T = \sum r_k$$

and where the r_k equal the number of occurrences of codebook symbol k over the given training sequence.

Furthermore, an assortment of (A, B, π) triples were displayed which attain that upper bound--among them, the class of triples where the rows of B are equal, with element $b_{jk} = r_k / T$, and the A and π are otherwise free to assume any values (provided only that they are Markov).

However, a limitation of calculus-based methods is that they focus on interior solutions and tend to overlook possible endpoint extrema at the boundaries of the solution space. In most cases where a specific training sequence is given, it is quite easy to find an endpoint solution which surpasses the upper bound of the interior. Clearly, a procedure to identify the global optimum must include a systematic search of the boundaries.

This chapter gives a simple proof that the global optimum HMM must indeed occur on a boundary of the solution space, followed by a description of what searching the boundaries entails.

3.2. Focus on the Boundary

The boundaries of the solution space are those triples (A, B, π) in which any one or more of the individual matrix elements equals zero.

It is simple to show that the global optimum must reside on a boundary, as follows :

From the π-equation and Section 2.2., the objective can be re-written as

$$\begin{aligned} P(O) &= L' \pi \\ &= 1' \operatorname{Diag}(BD_T) A' \ldots A' \operatorname{Diag}(BD_2) A' \operatorname{Diag}(BD_1) \pi \\ &= [f_1(A,B)\ f_2(A,B)\ \ldots\ f_N(A,B)]\ [\pi_1\ \pi_2 \ldots \pi_N]' \end{aligned}$$

where the $f_i(A, B)$ are functions of A and B only.

Since $\sum \pi_i = 1$, P can be viewed as a weighted average of the $f_i(A,B)$.

The greatest value which a weighted average can attain is that of its largest element.

Therefore, to maximize P it is sufficient to

(1) find the largest possible maximum among the $f_i(A,B)$, and
(2) assign a 1 to that position in π and 0's elsewhere.

In summary, the global maximum of P resides on the boundary, since all but one of the elements of π equal zero.

Furthermore, the task of optimizing P reduces (slightly) to finding the global maximum among the $f_i(A,B)$.

Finally, it is shown in Section 3.7. that the f_i are isomorphic to each other (identical to within a renaming of variables) and so share the same 'landscape' and the same global optimum values (though not at the same time). So without loss of generality we can take

$$\pi_i = \begin{cases} 1, & i = 1 \\ 0, & \text{otherwise} \end{cases}$$

and attempt to optimize $f_1(A,B)$.

3.3. Revised Objective

The previous section managed to reduce the problem to one of globally optimizing $f_1(A,B)$, a function of A and B only. To proceed with this new task, it is necessary to consider the nature of f_1 .

By expanding the equation

$$L = \text{Diag}(BD_1)\ A\ \text{Diag}(BD_2)\ A \dots A\ \text{Diag}(BD_T)\ \mathbf{1}$$

for the individual elements of L, one finds :

Given N states and T observed symbols, then $l_1 = f_1(A,B)$ is the sum of N^{T-1} terms, where each term corresponds to one of the possible state sequences (beginning from state 1) which the model could undergo while producing the outputs.

To focus on a concrete example, consider a simple 2-state model and the observation sequence $k\ k\ l\ k$, where those symbols simultaneously represent indices from a finite alphabet of outputs, as well as columns from the B-matrix.

Then there are $N^{T-1} = 2^{4-1} = 8$ possible state sequences

1--> 1--> 1--> 1
1--> 1--> 1--> 2
1--> 1--> 2--> 1
1--> 1--> 2--> 2
1--> 2--> 1--> 1
1--> 2--> 1--> 2
1--> 2--> 2--> 1
1--> 2--> 2--> 2

A term of f_1 is derived from each of those state sequences in two steps :

(1) Use consecutive pairs of states to form the subscripts of the factors from A

(2) Pair off the states with the indices $k\,k\,l\,k$ to form the subscripts of the factors from B

So in the present example

$$\begin{aligned}
f_1(A, B) &= a_{11}\, a_{11}\, a_{11}\, b_{1k}\, b_{1k}\, b_{1l}\, b_{1k} \\
&+ a_{11}\, a_{11}\, a_{12}\, b_{1k}\, b_{1k}\, b_{1l}\, b_{2k} \\
&+ a_{11}\, a_{12}\, a_{21}\, b_{1k}\, b_{1k}\, b_{2l}\, b_{1k} \\
&+ a_{11}\, a_{12}\, a_{22}\, b_{1k}\, b_{1k}\, b_{2l}\, b_{2k} \\
&+ a_{12}\, a_{21}\, a_{11}\, b_{1k}\, b_{2k}\, b_{1l}\, b_{1k} \\
&+ a_{12}\, a_{21}\, a_{12}\, b_{1k}\, b_{2k}\, b_{1l}\, b_{2k} \\
&+ a_{12}\, a_{22}\, a_{21}\, b_{1k}\, b_{2k}\, b_{2l}\, b_{1k} \\
&+ a_{12}\, a_{22}\, a_{22}\, b_{1k}\, b_{2k}\, b_{2l}\, b_{2k} \\
\\
&= (a_{11})^3\, (b_{1k})^3\, b_{1l} \\
&+ (a_{11})^2\, a_{12}\, (b_{1k})^2\, b_{1l}\, b_{2k} \\
&+ a_{11}\, a_{12}\, a_{21}\, (b_{1k})^3\, b_{2l} \\
&+ a_{11}\, a_{12}\, a_{22}\, (b_{1k})^2\, b_{2k}\, b_{2l} \\
&+ a_{11}\, a_{12}\, a_{21}\, (b_{1k})^2\, b_{1l}\, b_{2k} \\
&+ (a_{12})^2\, a_{21}\, b_{1k}\, b_{1l}\, (b_{2k})^2 \\
&+ a_{12}\, a_{21}\, a_{22}\, (b_{1k})^2\, b_{2k}\, b_{2l} \\
&+ a_{12}\, (a_{22})^2\, b_{1k}\, (b_{2k})^2\, b_{2l}
\end{aligned}$$

Note that, for all possible training sequences of length $T = 4$, the A-halves of the 8 terms will remain the same, as only the B-halves are sensitive to the actual sequence of symbols observed.

Note also that, considered in isolation, each term is of the form which can be trivially optimized by the result of Section 2.3.1., reproduced here as

Theorem A

The maximum of	$x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$
subject to the constraint	$\sum_{k=1}^{n} x_k = 1$
occurs at	$x_k = \frac{r_k}{T}$
and takes the value	$\prod_{k=1}^{n} \left(\frac{r_k}{T}\right)^{r_k}$
where	$T = \sum_{k=1}^{n} r_k$

For example, applying this theorem to a single term of f_1 (say term 4, which is $a_{11}\, a_{12}\, a_{22}\, b_{1k}^{2}\, b_{2k}\, b_{2l}$), the optimizing values can be found by

recording the exponents of a_{ij} and b_{jk} in the appropriate positions of auxiliary matrices A^* and B^*

$$A^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \hline 1 & 1 & 2 \\ \hline 0 & 1 & 1 \\ \hline \end{array} \qquad B^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \hline 2 & 0 & 2 \\ \hline 1 & 1 & 2 \\ \hline \end{array}$$

and then normalizing each row by the row sum, to give

$$A = \begin{array}{|c|c|} \hline 1/2 & 1/2 \\ \hline 0 & 1 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 1/2 & 1/2 \\ \hline \end{array}$$

For the isolated term, this optimum is also the <u>global</u> optimum, because the term is a unimodal 'hump' on a domain whose dimension corresponds to the degrees of freedom present within the term. To continue the example of term 4

$$a_{11}\, a_{12}\, a_{22}\, b_{1k}^{\,2}\, b_{2k}\, b_{2l}$$

since each row of A and B must sum to 1 , then

$$\begin{aligned} a_{22} &= 1 \\ b_{1k} &= 1 \\ a_{12} &= 1 - a_{11} \\ b_{2l} &= 1 - b_{2k} \end{aligned}$$

and term 4 is a hump on the two-dimensional $(\, a_{11} \times b_{2k})$ space (see page 70) .

But whereas the individual terms are easily optimized, the present task is to optimize a sum of those terms.

That task is further complicated by the fact that there are many such sums to consider, corresponding to the various boundaries of f_1 , along with f_1 itself.

3.4. Boundary Fragments

Once again, to globally maximize $f_1\,(A,B)$ requires searching the boundary 'planes' as well as the interior of its solution space. As before, when dealing with $P(O)$, a boundary of f_1 arises when any one or more of the elements of A and/or B equals zero.

To understand what searching these boundaries entails, consider again the simple case where the number of states $N = 2$, the number of symbols received $T = 4$, and the sequence $k\,k\,l\,k$ names the codebook indices of the observed symbols.

With only 2 distinct symbols received, the optimal B matrix will contain (at most) 2 non-zero columns, so the A and B matrices are both 2×2.

Observing the requirement that the rows of A must sum to 1, the set of A matrices can be partitioned into 9 classes

$$\begin{array}{|c|c|}\hline \times & \\ \hline \times & \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \\ \hline & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \\ \hline \times & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline & \times \\ \hline \times & \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline & \times \\ \hline & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline & \times \\ \hline \times & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \times \\ \hline \times & \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \times \\ \hline & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \times \\ \hline \times & \times \\ \hline\end{array}$$

where X indicates a non-zero entry
and blank indicates a zero

Similarly, observing the requirement that each symbol must issue from at least one of the states, the B matrices can also be partitioned into 9 classes

$$\begin{array}{|c|c|}\hline \times & \times \\ \hline & \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \\ \hline & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \times \\ \hline & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline & \times \\ \hline \times & \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline & \\ \hline \times & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline & \times \\ \hline \times & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \times \\ \hline \times & \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \\ \hline \times & \times \\ \hline\end{array}\quad
\begin{array}{|c|c|}\hline \times & \times \\ \hline \times & \times \\ \hline\end{array}$$

In theory, each A class can be paired with each B class, and the resulting 81 combinations represent the complete set of possible boundary 'planes' for f_1 . There are

8	zero-dimensional boundaries (vertices)
32	one-dimensional boundaries (edges)
30	two-dimensional boundaries (planes)
10	three-dimensional boundaries (cubes)
1	four-dimensional interior space

In practice, however, not all of these pairs will be relevant to a particular training sequence, since many* combinations will not actually be capable of generating the sequence. For example, the combination

$$A = \begin{array}{c|c|c|} & 1 & 2 \\ \hline 1 & & \times \\ \hline 2 & \times & \\ \hline \end{array} \qquad B = \begin{array}{c|c|c|} & k & l \\ \hline 1 & \times & \\ \hline 2 & & \times \\ \hline \end{array}$$

can only produce the alternating sequence $k\ l\ k\ l \ldots$

* Here, at least one third of the total number can be ignored. In general, the ratio is at least $2^{N-1} - 1$ in $2^{N} - 1$

	0 $\begin{matrix}\boxtimes & \square \\ \boxtimes & \square\end{matrix}$	0 $\begin{matrix}\boxtimes & \square \\ \square & \boxtimes\end{matrix}$	1 $\begin{matrix}\boxtimes & \square \\ \boxtimes & \boxtimes\end{matrix}$	0 $\begin{matrix}\square & \boxtimes \\ \boxtimes & \square\end{matrix}$	0 $\begin{matrix}\square & \boxtimes \\ \square & \boxtimes\end{matrix}$	1 $\begin{matrix}\square & \boxtimes \\ \boxtimes & \boxtimes\end{matrix}$	1 $\begin{matrix}\boxtimes & \boxtimes \\ \boxtimes & \square\end{matrix}$	1 $\begin{matrix}\boxtimes & \boxtimes \\ \square & \boxtimes\end{matrix}$	2 $\begin{matrix}\boxtimes & \boxtimes \\ \boxtimes & \boxtimes\end{matrix}$
1 $\begin{matrix}\boxtimes & \boxtimes \\ \square & \square\end{matrix}$	1	1	2	1	1	2	2	2	3
0 $\begin{matrix}\boxtimes & \square \\ \square & \boxtimes\end{matrix}$	0	0	1	0	0	1	1	1	2
1 $\begin{matrix}\boxtimes & \boxtimes \\ \square & \boxtimes\end{matrix}$	1	1	2	1	1	2	2	2	3
0 $\begin{matrix}\square & \boxtimes \\ \boxtimes & \square\end{matrix}$	0	0	1	0	0	1	1	1	2
1 $\begin{matrix}\square & \square \\ \boxtimes & \boxtimes\end{matrix}$	1	1	2	1	1	2	2	2	3
1 $\begin{matrix}\square & \boxtimes \\ \boxtimes & \boxtimes\end{matrix}$	1	1	2	1	1	2	2	2	3
1 $\begin{matrix}\boxtimes & \boxtimes \\ \boxtimes & \square\end{matrix}$	1	1	2	1	1	2	2	2	3
1 $\begin{matrix}\boxtimes & \square \\ \boxtimes & \boxtimes\end{matrix}$	1	1	2	1	1	2	2	2	3
2 $\begin{matrix}\boxtimes & \boxtimes \\ \boxtimes & \boxtimes\end{matrix}$	2	2	3	2	2	3	3	3	4

In order to determine which of the 81 combinations are relevant and which can be ignored, notice that an A-class with a blank (zero) at position (i, j) nulls out any term of $f_1(A, B)$ which includes a_{ij} as a factor. Thus

$\begin{array}{\|c\|c\|}\hline \times & \\ \hline \times & \\ \hline\end{array}$	eliminates terms 2 through 8 and preserves term 1
$\begin{array}{\|c\|c\|}\hline \times & \\ \hline & \times \\ \hline\end{array}$	eliminates terms 2 through 8 and preserves term 1
$\begin{array}{\|c\|c\|}\hline \times & \\ \hline \times & \times \\ \hline\end{array}$	eliminates terms 2 through 8 and preserves term 1
$\begin{array}{\|c\|c\|}\hline & \times \\ \hline \times & \\ \hline\end{array}$	eliminates terms 1, 2, 3, 4, 5, 7, 8 and preserves term 6
$\begin{array}{\|c\|c\|}\hline & \times \\ \hline & \times \\ \hline\end{array}$	eliminates terms 1 through 7 and preserves term 8
$\begin{array}{\|c\|c\|}\hline & \times \\ \hline \times & \times \\ \hline\end{array}$	eliminates terms 1 through 5 and preserves terms 6, 7, 8
$\begin{array}{\|c\|c\|}\hline \times & \times \\ \hline \times & \\ \hline\end{array}$	eliminates terms 4, 7, 8 and preserves term 1, 2, 3, 5, 6
$\begin{array}{\|c\|c\|}\hline \times & \times \\ \hline & \times \\ \hline\end{array}$	eliminates terms 3, 5, 6, 7 and preserves term 1, 2, 4, 8
$\begin{array}{\|c\|c\|}\hline \times & \times \\ \hline \times & \times \\ \hline\end{array}$	preserves all terms

Likewise, a B-class with a blank at position (j, k) nulls out any term of $f_1(A, B)$ which includes b_{jk} as a factor. Thus

$\begin{array}{\|c\|c\|}\hline \times & \times \\ \hline & \\ \hline\end{array}$	eliminates terms 2 through 8 and preserves term 1
$\begin{array}{\|c\|c\|}\hline \times & \\ \hline & \times \\ \hline\end{array}$	eliminates terms 1, 2, 4, 5, 6, 7, 8 and preserves term 3
$\begin{array}{\|c\|c\|}\hline \times & \times \\ \hline & \times \\ \hline\end{array}$	eliminates terms 2, 4, 5, 6, 7, 8 and preserves terms 1, 3
$\begin{array}{\|c\|c\|}\hline & \times \\ \hline \times & \\ \hline\end{array}$	eliminates all terms
$\begin{array}{\|c\|c\|}\hline & \\ \hline \times & \times \\ \hline\end{array}$	eliminates all terms
$\begin{array}{\|c\|c\|}\hline & \times \\ \hline \times & \times \\ \hline\end{array}$	eliminates all terms
$\begin{array}{\|c\|c\|}\hline \times & \times \\ \hline \times & \\ \hline\end{array}$	eliminates terms 3, 4, 7, 8 and preserves terms 1, 2, 5, 6
$\begin{array}{\|c\|c\|}\hline \times & \\ \hline \times & \times \\ \hline\end{array}$	eliminates terms 1, 2, 5, 6 and preserves terms 3, 4, 7, 8
$\begin{array}{\|c\|c\|}\hline \times & \times \\ \hline \times & \times \\ \hline\end{array}$	preserves all terms

* Note that the details on the previous page are independent of the training sequence, and are common to all 15 possible sequences of length 4.
Only the B preservation sets on this page are sensitive to the particular sequence of symbols. A similar situation holds among symbol sequences of other lengths.

Finally, for each of the 81 pairs consider the list of terms which are preserved by <u>both</u> A- and B-classes simultaneously (i.e., the intersection of their preservation sets).

The pairs whose intersection is non-empty represent the combinations which are capable of generating the sequence $\boldsymbol{k\ k\ l\ k}$ ... while the pairs whose intersection is empty can be ignored.

The fore-going explanation serves two useful purposes:

(1) It provides a method for identifying which boundary planes are relevant to a given observation sequence.

(2) It also demonstrates that the boundary planes are equivalent to fragments of the total objective function $f_1\ (A, B)$ ...and that searching the boundary planes is accomplished by optimizing each of those fragments.

kklk	1 $\begin{array}{cc} \times & \\ \times & \end{array}$	1 $\begin{array}{cc} \times & \\ & \times \end{array}$	1 $\begin{array}{cc} \times & \\ \times & \times \end{array}$	6 $\begin{array}{cc} & \times \\ \times & \end{array}$	8 $\begin{array}{cc} & \times \\ & \times \end{array}$	678 $\begin{array}{cc} & \times \\ \times & \times \end{array}$	12356 $\begin{array}{cc} \times & \times \\ \times & \end{array}$	1248 $\begin{array}{cc} \times & \times \\ & \times \end{array}$	1—>8 $\begin{array}{cc} \times & \times \\ \times & \times \end{array}$
1 $\begin{array}{cc} \times & \times \\ & \end{array}$	1	1	1				1	1	1
3 $\begin{array}{cc} \times & \\ & \times \end{array}$							3		3
1 3 $\begin{array}{cc} \times & \times \\ & \times \end{array}$	1	1	1				1 3		1 3
none $\begin{array}{cc} & \times \\ \times & \end{array}$									
none $\begin{array}{cc} & \\ \times & \times \end{array}$									
none $\begin{array}{cc} & \times \\ \times & \times \end{array}$									
1 2 5 6 $\begin{array}{cc} \times & \times \\ \times & \end{array}$	1	1	1	6		6	1 2 5 6	1 2	1 2 5 6
3 4 7 8 $\begin{array}{cc} \times & \\ \times & \times \end{array}$					8	7 8	3	4 8	3 4 7 8
1—>8 $\begin{array}{cc} \times & \times \\ \times & \times \end{array}$	1	1	1	6	8	6 7 8	1 2 3 5 6	1 2 4 8	1—>8

3.5. Geometric Interpretation

Appendices B, C & D explore three different methods for optimizing the boundary fragments of f_1. However, before considering those methods, a few more observations will be helpful.

The collection of fragments from the previous page are listed (without duplication) below :

(1)
(3)
(6)
(8)
(1 2)
(1 3)
(4 8)
(7 8)
(6 7 8)
(1 2 4 8)
(1 2 5 6)
(3 4 7 8)
(1 2 3 5 6)
(1 --> 8)

The simplest of these fragments are the 'singletons', consisting of only a single term from $f_1(A, B)$:

$$(1) = (a_{11})^3 (b_{1k})^3 b_{1l}$$
$$(3) = a_{11} a_{12} a_{21} (b_{1k})^3 b_{2l}$$
$$(6) = (a_{12})^2 a_{21} b_{1k} b_{1l} (b_{2k})^2$$
$$(8) = a_{12} (a_{22})^2 b_{1k} (b_{2k})^2 b_{2l}$$

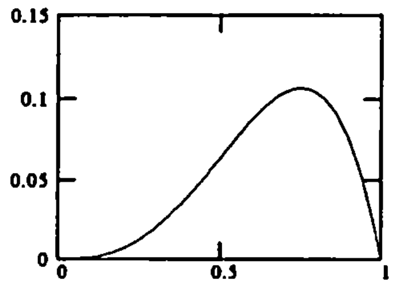
As explained on page 55 , these can be viewed as one-dimensional in the sense that each contains only one free variable. The requirement that the rows of A and B sum to 1 implies that within (3) , for example, $a_{21} = b_{1k} = b_{2l} = 1$, with only a_{11} free to vary (since $a_{12} = 1 - a_{11}$) .

In this way

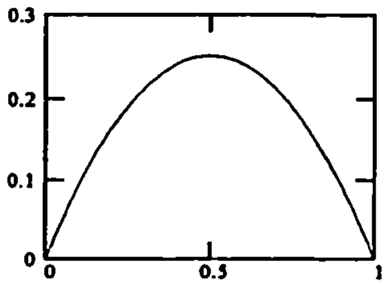
Term	...represents the one-dimensional boundary (edge)			
(1)	$0 \le b_{1k} \le 1$	with	$a_{11} = 1$	$a_{12} = 0$
(3)	$0 \le a_{11} \le 1$	with	$a_{21} = 1$	$a_{22} = 0$
			$b_{1k} = 1$	$b_{1l} = 0$
			$b_{2k} = 0$	$b_{2l} = 1$
(6)	$0 \le b_{1k} \le 1$	with	$a_{11} = 0$	$a_{12} = 1$
			$a_{21} = 1$	$a_{22} = 0$
			$b_{2k} = 1$	$b_{2l} = 0$
(8)	$0 \le b_{2k} \le 1$	with	$a_{11} = 0$	$a_{12} = 1$
			$a_{21} = 0$	$a_{22} = 1$
			$b_{1k} = 1$	$b_{1l} = 0$

(Terms (1) and (6) represent different (parallel) edges
--the first at $a_{11} = 1$, and the second at $a_{11} = 0$)

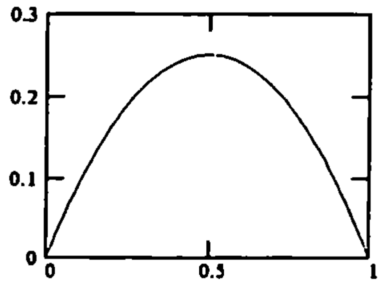
More accurately, it is the free variables which comprise the boundary planes, whereas the fragments are functions defined on those boundary planes--and invariably produce uni-modal curves (surfaces) :



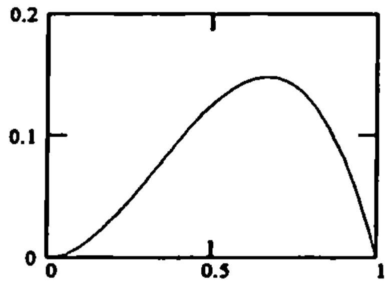
(1)



(3)



(6)



(8)

As mentioned above, these functions are easily maximized using Theorem A on page 54, with the position of the mode determined by

recording the exponents of a_{ij} and b_{jk} in the appropriate positions of auxiliary matrices A^* and B^* (given here for (3))

$$A^* = \begin{array}{cc|c} & & \text{Row Sum} \\ 1 & 1 & 2 \\ 1 & 0 & 1 \end{array} \qquad B^* = \begin{array}{cc|c} & & \text{Row Sum} \\ 3 & 0 & 3 \\ 0 & 1 & 1 \end{array}$$

and then normalizing each row by the row sum, to give

$$A = \begin{array}{|c|c|} \hline 1/2 & 1/2 \\ \hline 1 & 0 \\ \hline \end{array} \qquad B = \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$$

Finally, Appendix section D.4. uses the area (volume) under the uni-modal surface. This can be expressed in terms of the 'beta' function of advanced calculus and statistics, which is defined as

$$B(p, q) = \int_{x=0}^{1} x^{p-1}(1-x)^{q-1}dx$$

$$= \frac{(p-1)!\,(q-1)!}{(p+q-1)!} \qquad \text{for } p \text{ and } q \text{ integers}$$

It seems the single terms of f_1 can be interpreted as generalized (i.e. multi-dimensional) Beta random variables, which otherwise have applications [35]

(a) as order statistics within probability theory

(b) and also within Superstring Theory.

After the singleton fragments, the next in complexity are the compound fragments

$$(1\ \ 2) = (a_{11})^3 (b_{1k})^3 b_{1l} + (a_{11})^2 a_{12} (b_{1k})^2 b_{1l} b_{2k}$$

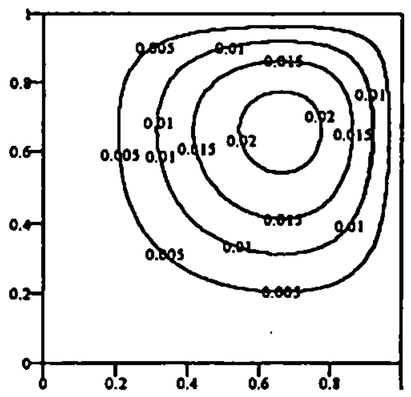
$$(4\ \ 8) = a_{11} a_{12} a_{22} (b_{1k})^2 b_{2k} b_{2l} + a_{12} (a_{22})^2 b_{1k} (b_{2k})^2 b_{2l}$$

$$(7\ \ 8) = a_{12} a_{21} a_{22} (b_{1k})^2 b_{2k} b_{2l} + a_{12} (a_{22})^2 b_{1k} (b_{2k})^2 b_{2l}$$

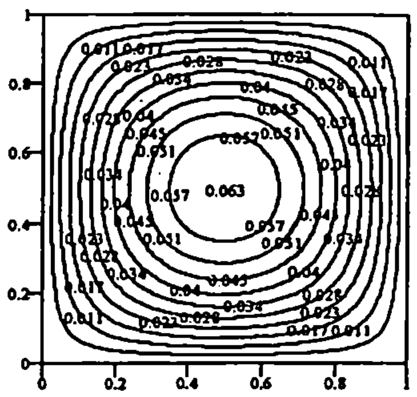
Viewed in isolation, terms 2 4 and 7 are two-dimensional, each possessing two free variables.

Term	...represents the two-dimensional boundary plane			
2	$0 \le a_{11} \le 1$ $0 \le b_{1k} \le 1$	with	$b_{2k} = 1$	$b_{2l} = 0$
4	$0 \le a_{11} \le 1$ $0 \le b_{2k} \le 1$	with	$a_{21} = 0$ $b_{1k} = 1$	$a_{22} = 1$ $b_{1l} = 0$
7	$0 \le a_{21} \le 1$ $0 \le b_{2k} \le 1$	with	$a_{11} = 0$ $b_{1k} = 1$	$a_{12} = 1$ $b_{1l} = 0$

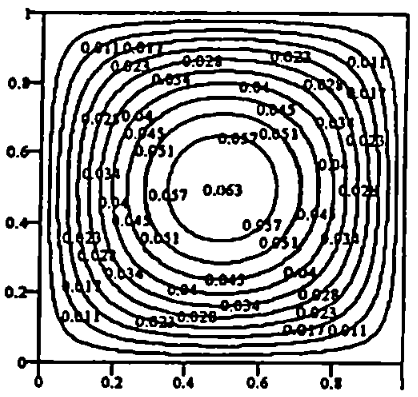
Regarded as functions defined on those free variables, the terms 2 4 and 7 are uni-modal surfaces :



2



4



7

Once again, the position of the mode can be computed by dividing the auxiliary exponents matrices by the relevant row sums :

$$2: \quad A^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \cline{1-2} 2 & 1 & 3 \\ \cline{1-2} - & - & - \\ \cline{1-2} \end{array} \qquad B^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \cline{1-2} 2 & 1 & 3 \\ \cline{1-2} 1 & 0 & 1 \\ \cline{1-2} \end{array}$$

$$4: \quad A^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \cline{1-2} 1 & 1 & 2 \\ \cline{1-2} 0 & 1 & 1 \\ \cline{1-2} \end{array} \qquad B^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \cline{1-2} 2 & 0 & 2 \\ \cline{1-2} 1 & 1 & 2 \\ \cline{1-2} \end{array}$$

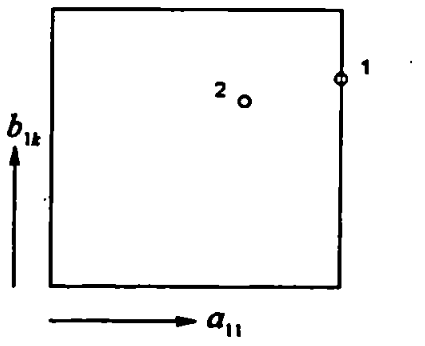
$$7: \quad A^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \cline{1-2} 0 & 1 & 1 \\ \cline{1-2} 1 & 1 & 2 \\ \cline{1-2} \end{array} \qquad B^* = \begin{array}{|c|c|c} \multicolumn{2}{c}{} & \text{Row Sum} \\ \cline{1-2} 2 & 0 & 2 \\ \cline{1-2} 1 & 1 & 2 \\ \cline{1-2} \end{array}$$

Note, however, that terms 2 4 and 7 never actually occur in isolation, but always appear (in the list of valid fragments) joined with a 'satellite' term :

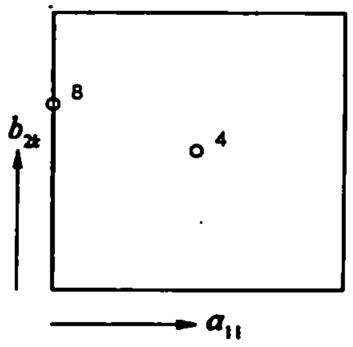
2	joined with	1
4	joined with	8
7	joined with	8

This is because the factors which comprise the satellite term are a proper subset of the factors in the main term--so that whenever those factors are non-zero (the main term is 'activated'), the satellite is activated also.

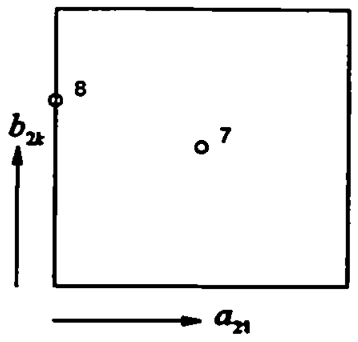
Finally, where the main term represents a two-dimensional boundary plane, the satellite term always occurs along an edge of that boundary. Therefore, *within the context of the current fragment**, the main term is classified as an 'interior term' and the satellite as a 'boundary term'.



$b_{2k} = 1 \quad b_{2l} = 0$



$a_{21} = 0 \quad a_{22} = 1$
$b_{1k} = 1 \quad b_{1l} = 0$



$a_{11} = 0 \quad a_{12} = 1$
$b_{1k} = 1 \quad b_{1l} = 0$

* However, what serves as a boundary term in a 2-D fragment will, within the context of the 1-D edge, appear as an interior term.

A second type of two-dimensional fragment is the simple union

$$(1\ \ 3) \ = \qquad (a_{11})^3\,(b_{1k})^3\,b_{1l} \ + \ a_{11}\,a_{12}\,a_{21}\,(b_{1k})^3\,b_{2l}$$

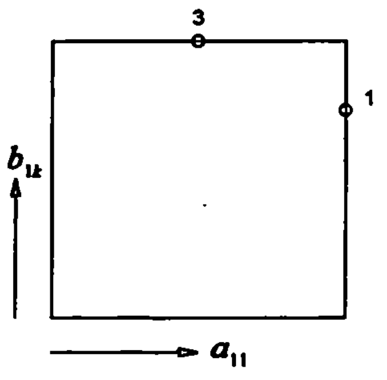
Each of the component terms is a one-dimensional singleton :

Term	...represents the one-dimensional boundary (edge)			
(1)	$0 \le b_{1k} \le 1$	with	$a_{11} = 1$	$a_{12} = 0$
(3)	$0 \le a_{11} \le 1$	with	$a_{21} = 1$	$a_{22} = 0$
			$b_{1k} = 1$	$b_{1l} = 0$
			$b_{2k} = 0$	$b_{2l} = 1$

In term (1) a_{11} is fixed and b_{1k} is free, while in term (3) b_{1k} is fixed and a_{11} is free (i.e., the edges are orthogonal). Maximizing the <u>sum</u> of the two terms entails relaxing the value of a_{11} in term (1) and the value of b_{1k} in term (3) . In this way, the fragment comes to represent the two-dimensional space spanned by the pair of orthogonal edges :

Fragment	...represents the two-dimensional boundary plane			
(1 3)	$0 \le a_{11} \le 1$	with	$a_{21} = 1$	$a_{22} = 0$
	$0 \le b_{1k} \le 1$		$b_{2k} = 0$	$b_{2l} = 1$

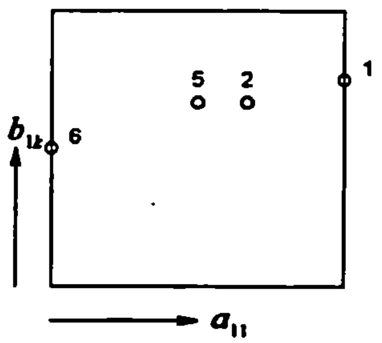
The result is a plane with two boundary terms and no interior terms :



$a_{21} = 1 \quad a_{22} = 0$
$b_{2k} = 0 \quad b_{2l} = 1$

The fragment (1 2 5 6) is a more general type of compound fragment.

The two-dimensional interior term 5 activates a second interior term 2 , as well as the <u>two</u> boundary terms 1 and 6 .



$a_{21} = 1 \quad a_{22} = 0$
$b_{2k} = 1 \quad b_{2j} = 0$

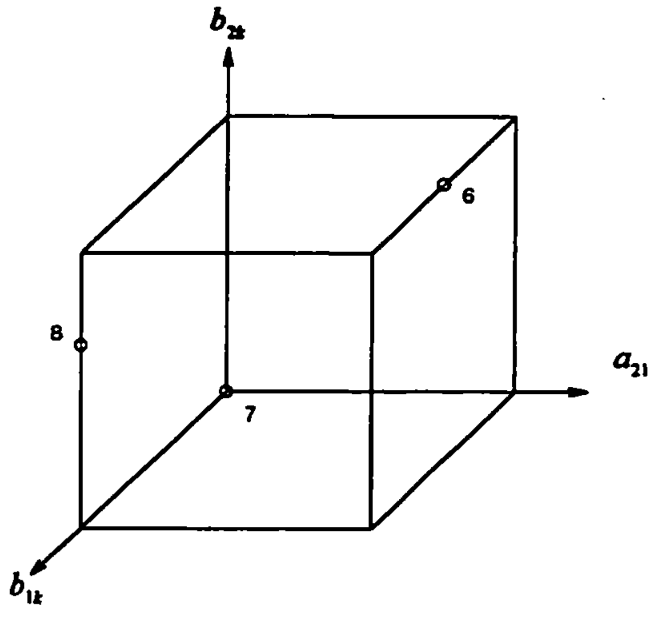
One might say that the interior term(s) activates multiple satellite terms around the edges.

Alternatively, one might say that the space spanned by the boundary terms is not empty (as with a simple union), but contains interior terms.

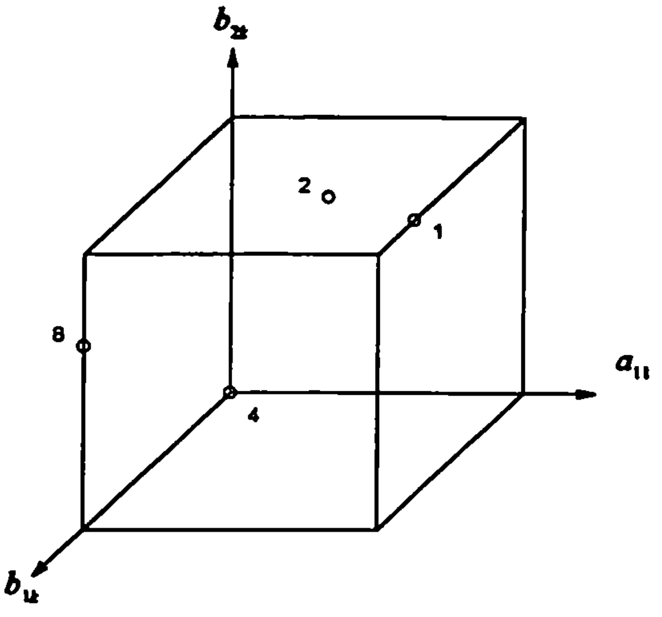
Fragments (6 7 8) and (1 2 4 8) are three-dimensional examples of the simple union.

Fragment (6 7 8) represents the three-dimensional space spanned by the planar fragment (7 8) and the orthogonal edge (6).

Fragment (1 2 4 8) represents the three-dimensional space spanned by the two orthogonal planar fragments (1 2) and (4 8).



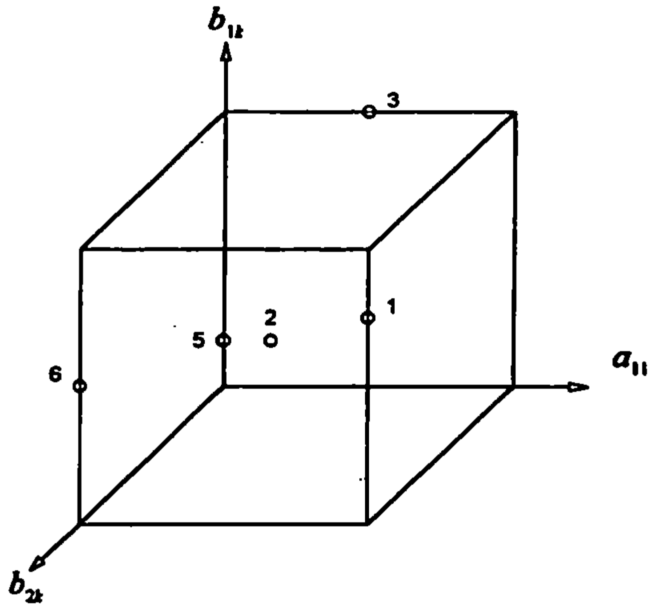
$a_{11} = 0 \quad a_{12} = 1$



$a_{21} = 0 \quad a_{22} = 1$

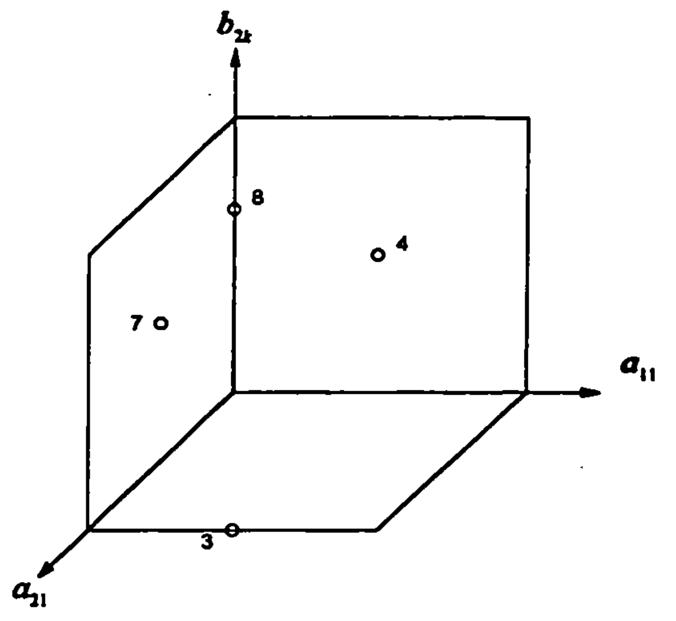
In each case, all of the participating terms are boundary terms--there are no interior terms.

The fragment (1 2 3 5 6) is also a three-dimensional simple union. However, unlike the previous examples, the lower-dimensional components (1 2 5 6) and (3) are not orthogonal, but lie parallel and opposite.



$a_{21} = 1 \quad a_{22} = 0$

Fragment (3 4 7 8) is a three-dimensional union spanned by a <u>trio</u> of fragments : (4 8), (7 8) and (3)



$b_{1k} = 1 \quad b_{1l} = 0$

(3) is orthogonal to (7 8) and parallel and opposite to (4 8).

(4 8) and (7 8) share the edge (8).

Finally, the four-dimensional 'fragment' (1 --> 8) is a simple union of the two three-dimensional fragments (1 2 3 5 6) and (3 4 7 8).

An alternative combination is (1 2 5 6), (6 7 8) and (3 4 7 8).

In neither of these unions is an interior term present.

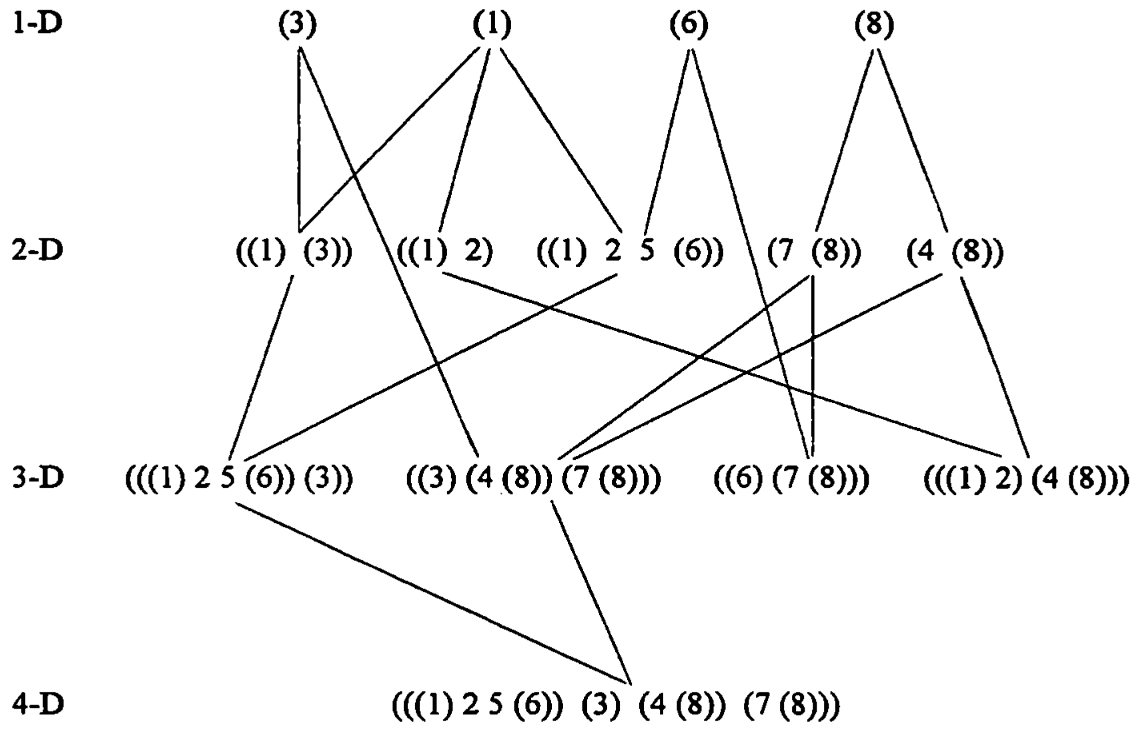
3.6. Summary

Having examined the complete list of fragments, these observations follow :

(1) The fragments fall into distinct categories:

- (A) Compound fragments, where the space is spanned by interior term(s), which may also activate lower-dimensional boundary plane(s)
- (B) Unions, where the space is spanned by the lower-dimensional boundary planes, and contains no interior terms
- (C) Singletons, defined as atomic compound fragments composed of a lone interior term

(2) A natural hierarchy exists among the fragments, with the smaller fragments serving as lower-dimensional boundary planes for the larger fragments*



* A number without enclosing brackets denotes an interior term--e.g., the 2 in ((1) 2)

(3) Each of the terms of $f_1(A, B)$ appears as an interior term in exactly one fragment* --which is either a compound fragment or a singleton.

Note that in subsequent appearances, the fragment in question forms part of the boundary of some (higher-dimensional) fragment--so that what was once an interior term now serves as a boundary term.

* There are minor exceptions, where a term may appear as an interior term more than once. Such terms are invariably 'incomplete', in the sense that at least one row of A or B fails to contribute any factors to the term--
so that there is an unspecified 'floating' dimension that is neither fixed nor free.
In the present example, term 2 is incomplete since it includes neither a_{21} nor a_{22}.
However, such exceptions to the 'rule' cause no difficulty in what follows.

These observations, and particularly the fact that the non-zero boundaries of a fragment are merely the lower-dimensional fragments in its hierarchy, inspire the following recursive formulation for locating the global optimum of a fragment (including, of course, the complete f_1) :

```
result <-- Optimum_of(fragment)

if fragment is a singleton
     apply Theorem A
     return result

else
     let result equal largest interior solution              (*)
     while unexamined boundary planes exist
          result = Max {result, Optimum_of(next boundary plane)}
     return result
```

The chief practical difficulty is that, as yet, there is no automatic procedure to identify the global optimum within the interior of a fragment, as required on line (*). Appendices B, C & D describe three different efforts to achieve this.

3.7. Isomorphic Functions

The purpose of this brief appendix section is to support the claim made in Section 3.2. that the f_i are isomorphic to each other, being identical to within a renaming of variables, and therefore share the same landscape and the same global optimum values (though not at the same time) .

Since the terms of f_1 correspond to the state sequences which begin at state 1 , it is unsurprising that the terms of f_2 correspond to the state sequences which begin at state 2 , and so forth. Then

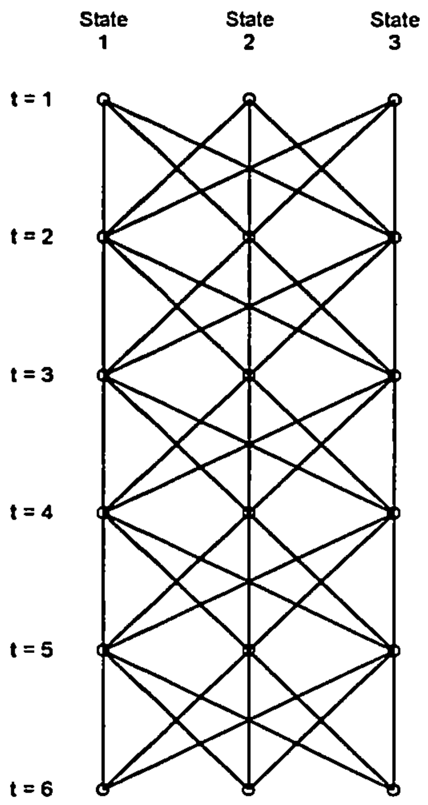
$$\begin{aligned}
f_1(A, B) &= a_{11}\,a_{11}\,a_{11}\,b_{1k}\,b_{1k}\,b_{1l}\,b_{1k} \\
&+ a_{11}\,a_{11}\,a_{12}\,b_{1k}\,b_{1k}\,b_{1l}\,b_{2k} \\
&+ a_{11}\,a_{12}\,a_{21}\,b_{1k}\,b_{1k}\,b_{2l}\,b_{1k} \\
&+ a_{11}\,a_{12}\,a_{22}\,b_{1k}\,b_{1k}\,b_{2l}\,b_{2k} \\
&+ a_{12}\,a_{21}\,a_{11}\,b_{1k}\,b_{2k}\,b_{1l}\,b_{1k} \\
&+ a_{12}\,a_{21}\,a_{12}\,b_{1k}\,b_{2k}\,b_{1l}\,b_{2k} \\
&+ a_{12}\,a_{22}\,a_{21}\,b_{1k}\,b_{2k}\,b_{2l}\,b_{1k} \\
&+ a_{12}\,a_{22}\,a_{22}\,b_{1k}\,b_{2k}\,b_{2l}\,b_{2k}
\end{aligned}$$

$$\begin{aligned}
f_2(A, B) &= a_{21}\,a_{11}\,a_{11}\,b_{2k}\,b_{1k}\,b_{1l}\,b_{1k} \\
&+ a_{21}\,a_{11}\,a_{12}\,b_{2k}\,b_{1k}\,b_{1l}\,b_{2k} \\
&+ a_{21}\,a_{12}\,a_{21}\,b_{2k}\,b_{1k}\,b_{2l}\,b_{1k} \\
&+ a_{21}\,a_{12}\,a_{22}\,b_{2k}\,b_{1k}\,b_{2l}\,b_{2k} \\
&+ a_{22}\,a_{21}\,a_{11}\,b_{2k}\,b_{2k}\,b_{1l}\,b_{1k} \\
&+ a_{22}\,a_{21}\,a_{12}\,b_{2k}\,b_{2k}\,b_{1l}\,b_{2k} \\
&+ a_{22}\,a_{22}\,a_{21}\,b_{2k}\,b_{2k}\,b_{2l}\,b_{1k} \\
&+ a_{22}\,a_{22}\,a_{22}\,b_{2k}\,b_{2k}\,b_{2l}\,b_{2k}
\end{aligned}$$

etc.

A visual inspection will confirm that, by re-labelling state 2 as state 1 and vice versa, the terms of f_2 are transformed into terms of f_1 (in reverse order) .

The transformation can be illustrated with the aid of the network diagram below, which represents all possible state sequences which the model could undergo :



Imagine that this network was once drawn on the surface of a cylinder, which was subsequently flattened. The transformation basically consists of rotating the cylinder by one (or more) nodes before flattening it again. Clearly, the new network would be identical to the first. In this way, f_2 (or f_i) is essentially identical to f_1 .

4. Summary of Boundary Optimization Methods

4.1. Introduction

The previous chapter has demonstrated that for a single training sequence, at least, the global maximum of $P(O \mid M)$ must reside on a boundary of the solution space--namely, that sub-space where the first element of π equals 1 and all of the remaining elements of π equal 0 . On this boundary, $P(O \mid M)$ can be replaced by a simpler objective function given by $f_1\,(A,B)$, which corresponds to the collection of possible state sequences that the HMM might have undergone (in the course of generating the outputs) which begin from State 1. However, as with attempting to optimize $P(O \mid M)$, the interior maxima alone are not sufficient to guarantee the global maximum of f_1 , and it is also necessary for the boundaries of this new objective function to be examined for endpoint extrema. This is equivalent to optimizing all the many fragments of f_1 which result when one or more elements of A and/or B are set equal to 0 . Chapter 3 also showed how to narrow down the complete list of fragments to include only those relevant few which are capable of generating the given training sequence. Finally, when globally optimizing an individual fragment, it seems that the interior maxima alone are sufficient in this case. This is because the boundaries of a fragment are merely those lower-dimensional ones which fall below it in the hierarchy of fragments, so that the endpoint extrema belonging to a fragment will be examined in due course when the lower-dimensional fragments are themselves optimized.

Having established the need for optimizing the boundary fragments of f_1 , and having learned how to identify the relevant ones, the obvious next step is to develop a method for performing the necessary global optimization of a single fragment. From among the numerous different attempts to find such a procedure, three have been selected for inclusion in the thesis as Appendices B, C & D .

Appendix B revisits the calculus approach first introduced in Chapter 2 . Calculus is a viable strategy here because only interior maxima are sought, so that the shortcoming noted in Section 3.1. concerning endpoint extrema does not apply. Appendix B contains a more thorough analysis using Matrix Derivatives, in which the constraints have been adapted in a way which focuses on the fragments of $f_1(A,B)$.

The appendix chapter begins with the same basic formulation as Chapter 2 , but with 3 extra Lagrange terms added to the objective function that constrain chosen elements of π B & A to equal zero, and thereby focus the analysis on a specific boundary 'plane' of the solution space. Then the tools of Matrix Derivatives are exercised in order to partially differentiate the objective function with respect to the 3 principle matrices π B & A , and the 6 Lagrange multiplier matrices. The 9 derivative equations are set to zero, and matrix algebra is brought to bear in an effort to solve first for the Lagrange multipliers and then substitute their values to arrive at a reduced system of equations in π B & A . The chapter includes a section for each of the 6 Lagrange multipliers which details the progress achieved in solving for that unknown.

Appendix C describes a second attempt at globally optimizing the fragments, which employs the optimization technique known as Geometric Programming. This is a relatively new technique which is well suited to objective functions of the present form, and which is essentially the 'dual' of the derivatives approach.

Geometric Programming was devised in the 1960s to carry out unconstrained minimizations on a certain class of objective function. Appendix C presents the theory and nomenclature of the original formulation, and then describes how the techniques have been extended to cope with a wider range of problems, including more general objective functions, maximization as well as minimization, and the application of constraints. The boundary fragments of f_1 appear to require the more complicated treatment of the extended formulation but, fortunately, thanks to the special features of these functions, their handling can be greatly streamlined, and the appendix presents a novel algorithm for automatically transforming a boundary fragment optimization problem into the appropriate G.P. equations. Unfortunately, there still remains the task of solving the G.P. equations. The strategy adopted is to systematically reduce the equations by dividing out some of the easily-found roots. The later sections of the appendix describe the progress achieved with this strategy, including some representative examples.

Appendix D makes a third attempt, using a new hill-climbing algorithm. The chapter first shows the mathematical development of the recurrence formulas. Next it gives evidence that the formulas will actually converge to local maxima. Finally it explains why it is necessary to select a starting point which is within the interior of the fragment's domain, and introduces an assortment of schemes for providing such a starting point.

What is innovative here is not so much the algorithm. Rather, the significant work consists of an exploration of the various schemes to nominate a starting point for the hill-climb--to determine whether the new algorithm together with the 'right' starting point is consistently able to produce the global optimum.

Unfortunately, none of these three attempts were successful in achieving a solution to the optimization task. In addition, their developments are heavily mathematical, and they contribute only indirectly to the remainder of the thesis. For these reasons, and in order to avoid interrupting the flow of the thesis, they have been relegated to the Appendix. On the other hand, these different attempts represent a substantial body of work. Furthermore, all three attempts have produced some interesting patterns and observations. The purpose of the present chapter is to give a brief summary of the progress made, together with a few of the more notable insights achieved.

4.2. Equivalence of the Methods

Although, superficially, the three solution methods appear to be very different, in fact, they are quite closely related. The connection between the methods is that they all depend, ultimately, upon the fact that at any interior optimum of the objective function the derivatives all equal zero.

This connection can be observed with the aid of the recurrence relations from Appendix D , which are reproduced here

$$x_{ij} = \frac{\sum_{k=1}^{L} p_{ijk} \times t_k}{\sum_{k=1}^{L} \left\{ \sum_{j=1}^{M_i} p_{ijk} \right\} \times t_k} \qquad (i = 1..N)(j = 1..M_i)$$

(4.1)

where x_{ij} is an element of either A or B

t_k is the k^{th} term of the objective, equal to

$$t_k = \prod_{i=1}^{N} \prod_{j=1}^{M_i} x_{ij}^{p_{ijk}}$$

and p_{ijk} is the exponent of x_{ij} in term k

In Section D.2., these recurrence relations are derived directly from that observation (i.e. derivatives equal to zero).

The Geometric Programming methods in Appendix C exploit the observation indirectly. Nevertheless, the above recurrence relations could also be reached through a re-working of the general ('full blown') orthogonality conditions in Section C.2.

Finally, to see the connection with the Matrix Derivatives approach, let

$$f = \sum_{k=1}^{L} t_k$$

Then the denominators of recurrence relation (4.1) can be expressed in terms of Matrix Derivatives as

$$(X \odot \frac{df}{dX}) \mathbf{1}$$

which has the precise form of the Lagrange multipliers for the Markov constraints on A and B (See Sections B.4.1. and B.5.1.)

In short, it seems that (primal) analysis, Geometric Programming and hill-climbing are all closely related, with the difference being that

analysis seeks the optimal A & B, which are then used to calculate the t_k

G.P. seeks the optimal t_k, which are then used to calculate A & B

hill-climbing seeks the optimal A & B and the optimal t_k simultaneously

4.3. Progress with the Matrix Analysis

The matrix analysis strategy was to

(a) express the objective plus the additional Lagrange (constraint) terms using matrices

(b) differentiate with respect to the 3 primary independent variables (A, B, π) plus the Lagrange multipliers

(c) solve the derivative equations for the Lagrange multipliers

(d) substitute those results to get a reduced system of equations in (A, B, π)

(e) solve this reduced system

The analysis got halfway through step (c) .

As observed in Section 4.2., the Lagrange multipliers for the Markov constraints are equal to

$$(X \odot \frac{df}{dX})\,\mathbf{1} \qquad\qquad \text{for } X = A \text{ or } B$$

which also equals the collection of denominator values appearing in the recurrence relations for the hill-climbing algorithm. These Lagrange multipliers also have a physical interpretation as sensitivity factors (or 'shadow prices'). This says that if the Markov constraint $\sum_{j} x_{ij} = 1$ could be allowed to slip by one additional unit (i.e. sum = 2), then the value of the objective would be increased by

$$\sum_{k=1}^{L} \left\{ \sum_{j=1}^{M_i} p_{ijk} \right\} \times t_k$$

The other set of Lagrange multipliers, which focus the analysis on a specific boundary, could only be solved to within a sum of their elements. However, what is intriguing is that the three sums all take the same form regardless of the manner in which the associated primary variables A, B and π enter into the objective expression. In all three cases, $\frac{1}{N}$ times the sum of the active Lagrange multipliers equals the negative of the difference between

k copies of $P(O)$ at the optimum

and another k copies in which one occurrence of X per copy is replaced by $\frac{1}{N}\mathbf{1}\,\mathbf{1}'$

(where k is the number of occurrences of X in $P(O)$).

4.4. Progress with Geometric Programming

Progress in performing the optimizations using Geometric Programming can be summarized as follows:

(1) A procedure was developed which automatically maps the (primal) fragment and constraints into a (dual) G.P. system of equations ready for solution.

(2) Evidence was found suggesting that the dual equations could be reduced into a set of simpler sub-systems, each of which contains at most one interior solution of the original system--permitting the complete set of solutions to be found by hill-climbing/descent.

(3) No way has yet been found for reliably identifying the set of simpler sub-systems.

4.5. Progress with Hill-Climbing

The investigation into hill-climbing was to determine whether the recurrence formulas together with the 'right' starting values could consistently produce the global optimum for a given fragment.

Six schemes for generating the starting values were examined. The results can be summarized as follows:

(1) When the global maximum occurs within the interior of the fragment, virtually any starting point which is also within the interior will find it--including the modal position of interior term(s) as well as the outputs of any of the six schemes.

(2) When the global maximum occurs at a boundary of the fragment, none of the six schemes was 100 percent effective in finding it.

Observation (1) tends to suggest that the 'landscape' of the fragments is not as complicated as might be assumed. Taken with other evidence (see Sections 5.3.2. and 5.3.3.), it appears that the conditions under which a fragment has an interior peak are sufficient to guarantee that it has only one interior peak.

5. Rule Induction

5.1. Overview

The methods described in Appendices B , C & D share certain features in common :

(1) They are basically deductive in nature, attempting to derive a rule which is founded upon theory and logic.

(2) They seek local maxima, and rely upon finding the complete set in order to identify the global maximum from among them.

(3) They draw no distinction between f_1 and its fragments, applying equally well to all.

The approach described in this chapter is the opposite of these. First, by analyzing a variety of different training sequences and model sizes using all available mathematical tools, one searches specifically for the global maximum of f_1 in each case . Then, by studying the resulting body of examples for common themes and patterns, one tries to induce a rule (or rules) which account for the examples, and by which all global maxima can be found*.

The chapter begins by describing some of the more striking observations derived from the body of examples. Afterwards, those observations are amalgamated into an algorithm for identifying the global maximum HMM, and finally the results of applying the new algorithm are presented.

* Finally, one hopes to be able to confirm those rules by deduction.

5.2. Collecting the Examples

Before proceeding with the observations, however, first a word about how the examples themselves were collected. For the purpose of gathering a body of examples for carrying out rule induction, the method for finding the global maximum did not have to be efficient or even systematic. Any technique or combination of techniques was allowed for any f_1, provided only that the true global maximum was located. The question is, how can one achieve certainty that an optimum is actually the global optimum for the given f_1.

In general, this is a very difficult question, and even for the smaller examples (small number of states and short training sequences) it is not always easy to be absolutely certain. Nevertheless, there are certain techniques by which the task can be made easier.

One important technique depends upon the following observation, which is introduced by example :

For the training sequence $k\,k\,l$ and the 2-state model $(N=2)$, f_1 equals

$$a_{11}^2 b_{1k}^2 b_{1l} + a_{11}a_{12}\, b_{1k}^2 b_{2l} + a_{12}\, a_{21}\, b_{1k} b_{1l} b_{2k} + a_{12}\, a_{22}\, b_{1k} b_{2k} b_{2l}$$

Notice that this can be expressed as the inner product of two vectors,

$$[\; a_{11}^2 \quad a_{11}a_{12} \quad a_{12}\, a_{21} \quad a_{12}\, a_{22} \;] \begin{bmatrix} b_{1k}^2 b_{1l} \\ b_{1k}^2 b_{2l} \\ b_{1k} b_{1l} b_{2k} \\ b_{1k} b_{2k} b_{2l} \end{bmatrix}$$

which might be described as the 'A-contributions' and the 'B-contributions' of the four terms.

Finally, evaluate the A-contributions at various points around the solution space. For example, at the modal position for term (1), which is

$$A = \begin{bmatrix} 1 & 0 \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ b_{2k} & b_{2l} \end{bmatrix}$$

the vector of A-contributions equals

$$[(1)^2 \ \ (1)(0) \ \ (0)(a_{21}) \ \ (0)(a_{22})] = [\,1 \ \ 0 \ \ 0 \ \ 0\,]$$

At the modal position for term (2), which is

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

the vector is

$$[(\tfrac{1}{2})^2 \ \ (\tfrac{1}{2})(\tfrac{1}{2}) \ \ (\tfrac{1}{2})(a_{21}) \ \ (\tfrac{1}{2})(a_{22})] = [\tfrac{1}{4} \ \ \tfrac{1}{4} \ \ \tfrac{1}{2}a_{21} \ \ \tfrac{1}{2}(1-a_{21})]$$

At the modal position for term (3), which is

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 1 & 0 \end{bmatrix}$$

the vector is

$$[(0)^2 \ \ (0)(1) \ \ (1)(1) \ \ (1)(0)] = [\,0 \ \ 0 \ \ 1 \ \ 0\,]$$

At the modal position for term (4), which is

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

the vector is

$$[(0)^2 \quad (0)(1) \quad (1)(0) \quad (1)(1)] = [0 \quad 0 \quad 0 \quad 1]$$

In all of these cases, the A-contributions sum to 1. And, in general, the sum of the A-contributions for f_1 equals 1 *at every point of the solution space*. Furthermore, with a minor qualification, this holds true for the fragments of f_1 also.

A proof of this result can be sketched out as follows :

Recall from Section 3.2. that the function $f_1 = f_1(A, B)$ which is a sum of terms made up of various a-factors and b-factors is also the first element of the vector L, whose definition is given by

$$L \;=\; \text{Diag}(B_1)\, A\, \text{Diag}(B_2)\, A \,\ldots\, A\, \text{Diag}(B_T)\, \mathbf{1}$$

$$\text{where} \quad L'\pi \;=\; P \;=\; P(o_1\, o_2 \ldots o_T)$$

To sum just the A-contributions of f_1 , it is sufficient merely to evaluate f_1 with all of its b-factors $b_{jk} = 1$.

Accordingly, imagine that the matrix B consists of all 1's . This makes

$$\text{Diag}(B_1) \;=\; \text{Diag}(B_2) \;=\; \ldots \;=\; \text{Diag}(B_T) \;=\; \mathrm{I}$$

the identity matrix.

And

$$\begin{aligned} L &= \mathrm{I}\, A\, \mathrm{I}\, A \ldots A\, \mathrm{I}\, \mathbf{1} \\ &= A^{T\text{-}1}\, \mathbf{1} \\ &= \mathbf{1} \qquad\qquad \text{(using the Markov property of } A\text{)} \end{aligned}$$

which indicates that all of the elements of L (including f_1) equal 1 , regardless of the value of A .

Finally, by employing a simple device it is possible to extend this result to include the fragments of f_1 also. The fragment has to be augmented with additional terms to bring it up to the full complement of terms required by the proof. But because the B-contributions of these extra terms are zero, the value of the fragment is not altered.

Consider the fragment (2 4)

$$a_{11}a_{12}\, b_{1k}^{2}\, b_{2l} + a_{12}\, a_{22}\, b_{1k}\, b_{2k}\, b_{2l}$$

which is the boundary fragment defined by $a_{21} = b_{1l} = 0$.

At the interior point given by

$$A = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

the A-contributions sum to only $\frac{2}{9} + \frac{2}{3} = \frac{8}{9}$. And, in general, the A-contributions of a typical fragment do <u>not</u> sum to 1 at each point of its domain.

On the other hand, by including term (1) :

$$a_{11}^{2} b_{1k}^{2}\, b_{1l}$$

the A-contributions will sum to 1 and, since for $b_{1l} = 0$ the B-contribution of term (1) equals 0 , the value of the fragment is unchanged.

The trick is to include all the terms of f_1 which result from setting $a_{21} = 0$ (in this example). In the language of Chapter 3 , this is the smallest A-preservation set which contains the fragment. When (2 4) is expressed (quite legitimately) as the inner product

$$[\; a_{11}^2 \quad a_{11}a_{12} \quad a_{12}\,a_{22} \;] \begin{bmatrix} 0 \\ b_{1k}^2\, b_{2l} \\ b_{1k}\, b_{2k}\, b_{2l} \end{bmatrix}$$

it is in a form whose A-contributions always sum to 1 .

To describe how this observation can be applied to the task of determining the global maximum of f_1, note that since f_1 is the inner product of its A- and B-contributions, and since the A-contributions always sum to 1, it follows that f_1 can be viewed as the weighted average of its B-contributions.

Recall that this idea was used in Chapter 3 to de-couple the π vector from the original optimization problem. With

$$\begin{aligned} P(O) &= L' \pi \\ &= [f_1(A,B)\ f_2(A,B)\ \dots\ f_N(A,B)]\ [\pi_1\ \pi_2 \dots \pi_N]' \end{aligned}$$

and $\sum \pi_i = 1$, it follows that P is a weighted average of the $f_i(A, B)$. The greatest value which a weighted average can attain is that of its largest component. Therefore, to maximize P it is sufficient to

(1) find the largest possible maximum among the $f_i(A, B)$, and
(2) assign a 1 to that position in π and 0's elsewhere.

This naively suggests that f_1 might be easily optimized via similar reasoning :

Since f_1 is the weighted average of its B-contributions, the way to maximize f_1 is to

(1) find the largest possible maximum among the B-contributions
(2) assign values to the a_{ij} to make the corresponding A-contribution equal to 1

Furthermore, the B-contributions from step (1) are trivially maximized using Theorem A (reproduced below), which states

Theorem A

the maximum of the function $x_1^{r_1} x_2^{r_2} \dots x_n^{r_n}$

subject to the constraint $\sum_{k=1}^{n} x_k = 1$

is achieved at $x_k = \dfrac{r_k}{\sum_{k=1}^{n} r_k}$

and takes the value $\dfrac{1}{T^T} \times \prod_{k=1}^{n} r_k^{r_k}$ where $T = \sum r_k$

However this time, unlike the π-vector, the A-contributions are not free to assume just any value. There are upper bounds upon them which, again, follow from Theorem A . (The table below gives the maximum value of each A-contribution for $N = 2$ and symbol sequences of length four and five.) As a consequence, it is generally not so easy to optimize f_1 .

Nevertheless, in the fortuitous case when the largest possible B-contribution does fall to a term whose A-contribution can be made to equal 1 , this simple approach will indeed produce a guaranteed global maximum for f_1 .

8 Terms

1	1 1 1 1	$(a_{11})^3$	1
2	1 1 1 2	$(a_{11})^2 a_{12}$	$\frac{4}{27}$
3	1 1 2 1	$a_{11} a_{12} a_{21}$	$\frac{1}{4}$
4	1 1 2 2	$a_{11} a_{12} a_{22}$	$\frac{1}{4}$
5	1 2 1 1	$a_{11} a_{12} a_{21}$	$\frac{1}{4}$
6	1 2 1 2	$(a_{12})^2 a_{21}$	1
7	1 2 2 1	$a_{12} a_{21} a_{22}$	$\frac{1}{4}$
8	1 2 2 2	$a_{12} (a_{22})^2$	1

16 Terms

1	1 1 1 1 1	$(a_{11})^4$	1
2	1 1 1 1 2	$(a_{11})^3 a_{12}$	$\frac{27}{256}$
3	1 1 1 2 1	$(a_{11})^2 a_{12} a_{21}$	$\frac{4}{27}$
4	1 1 1 2 2	$(a_{11})^2 a_{12} a_{22}$	$\frac{4}{27}$
5	1 1 2 1 1	$(a_{11})^2 a_{12} a_{21}$	$\frac{4}{27}$
6	1 1 2 1 2	$a_{11} (a_{12})^2 a_{21}$	$\frac{4}{27}$
7	1 1 2 2 1	$a_{11} a_{12} a_{21} a_{22}$	$\frac{1}{16}$
8	1 1 2 2 2	$a_{11} a_{12} (a_{22})^2$	$\frac{1}{4}$
9	1 2 1 1 1	$(a_{11})^2 a_{12} a_{21}$	$\frac{4}{27}$
10	1 2 1 1 2	$a_{11} (a_{12})^2 a_{21}$	$\frac{4}{27}$
11	1 2 1 2 1	$(a_{12})^2 (a_{21})^2$	1
12	1 2 1 2 2	$(a_{12})^2 a_{21} a_{22}$	$\frac{1}{4}$
13	1 2 2 1 1	$a_{11} a_{12} a_{21} a_{22}$	$\frac{1}{16}$
14	1 2 2 1 2	$(a_{12})^2 a_{21} a_{22}$	$\frac{1}{4}$
15	1 2 2 2 1	$a_{12} a_{21} (a_{22})^2$	$\frac{4}{27}$
16	1 2 2 2 2	$a_{12} (a_{22})^3$	1

More commonly, where this observation does prove useful is in helping to <u>eliminate</u> various fragments of f_1 as the possible source of its global maximum.

Suppose that a threshold has been given, above which the global maximum of f_1 is known to lie. In this case, let the threshold equal the largest modal value exhibited among the singleton fragments of f_1 .

If the global maximum is to exceed this threshold, it must result from a sum of terms (since the single-term fragments have been exhausted) and, consequently, is the weighted average of two or more B-contributions.

Assuming that a global maximum of this sort (i.e. multi-term) does exist, consider the values of those B-contributions as calculated at the location of the global maximum. Since a weighted average can never exceed its largest component, there must be present in the sum at least one B-contribution, $\mathrm{BCon}_{k,\mathrm{global}}$, which is at least as large as the global maximum

$$\text{global max} \le \mathrm{BCon}_{k,\mathrm{global}}$$

Finally, unless that global maximum coincides with the modal position of one of the (interior) terms, all of the global B-contributions will be less than their modal, or theoretical maximum, values as given by Theorem A . Therefore

$$\text{global max} \le \mathrm{BCon}_{k,\mathrm{global}} \le \mathrm{BCon}_{k,\mathrm{modal}}$$

The conclusion is that, for a global maximum of f_1 to exceed a given value, there must be present within the sum a term whose modal B-contribution also exceeds that threshold. And, by the contrapositive, if a fragment contains no such term, it has no hope of providing an improvement over the threshold, and so can be eliminated as a possible source of the global max.

Thus, a useful first step in locating the global maximum of f_1 can be sketched out as follows :

(1) Identify the valid fragments of f_1 for the given model size and training sequence using the procedure described in Section 3.4. and worksheet similar to page 63 .

(2) From this list of fragments, pick out the single-term fragments (singletons) and use Theorem A to calculate their modal values. The global maximum of f_1 must be at least as large as the largest of these (call it FMax) .

(3) No fragment of f_1 can exceed FMax which does not include a term k for which

$$\text{FMax} < \text{BCon}_{k,\text{modal}}$$

Therefore, find the terms of f_1 with such a B-contribution, and discard all fragments which lack such a qualifying term.

From here, the task calls for globally optimizing the surviving fragments and then selecting the greatest maximum from among them. Any of the techniques from Appendices B, C & D or elsewhere can be applied. If the fragment is one or two-dimensional, graphical methods (e.g. MathCAD) can be used to locate its maxima visually. Where the dimension exceeds two, Geometric Programming can sometimes lower the dimension--provided the number of distinct terms in the fragment does not exceed three. Calculus and/or Geometric Programming can be used to provide the interior maxima (bearing in mind that any boundary extrema will come to light among the lower-dimensional fragments in its hierarchy--see page 80) . Hill-climbing which is initiated from closely-spaced points in the domain of the fragment is the next recourse.

A variety of different regimes for carrying this out are described in Section 5.7. Finally, one tends to develop a 'feel' for the likely maximum, based upon the presence or absence of interior terms, the pattern of the exponents (see page 279), and resemblance to previous examples.

As a very simple illustration, consider the problem which opened this section, consisting of the training sequence $k\,k\,l$ and the 2-state model $(N = 2)$, for which f_1 equals

$$a_{11}^{\,2} b_{1k}^{\,2}\, b_{1l} + a_{11} a_{12}\, b_{1k}^{\,2}\, b_{2l} + a_{12}\, a_{21}\, b_{1k}\, b_{1l}\, b_{2k} + a_{12}\, a_{22}\, b_{1k}\, b_{2k}\, b_{2l}$$

Completing the procedure for identifying relevant fragments, the worksheet is shown on the following page, and the fragments are found to include :

(1)
(2)
(3)
(4)
(1 2)
(1 3)
(2 4)
(3 4)
(1 2 3)
(1 2 4)
(1 2 3 4)

Each of the four terms comprise a singleton, and their modal values are given below :

k k l	1 $\begin{array}{cc}\times & \\ \times & \end{array}$	1 $\begin{array}{cc}\times & \\ & \times\end{array}$	1 $\begin{array}{cc}\times & \\ \times & \times\end{array}$	3 $\begin{array}{cc} & \times\\ \times & \end{array}$	4 $\begin{array}{cc} & \times\\ & \times\end{array}$	3 4 $\begin{array}{cc} & \times\\ \times & \times\end{array}$	1 2 3 $\begin{array}{cc}\times & \times\\ \times & \end{array}$	1 2 4 $\begin{array}{cc}\times & \times\\ & \times\end{array}$	1—>4 $\begin{array}{cc}\times & \times\\ \times & \times\end{array}$
1 $\begin{array}{cc}\times & \times\\ & \end{array}$	1	1	1				1	1	1
2 $\begin{array}{cc}\times & \\ & \times\end{array}$							2	2	2
1 2 $\begin{array}{cc}\times & \times\\ & \times\end{array}$	1	1	1				1 2	1 2	1 2
none $\begin{array}{cc} & \times\\ \times & \end{array}$									
none $\begin{array}{cc} & \\ \times & \times\end{array}$									
none $\begin{array}{cc} & \times\\ \times & \times\end{array}$									
1 3 $\begin{array}{cc}\times & \times\\ \times & \end{array}$	1	1	1	3		3	1 3	1	1 3
2 4 $\begin{array}{cc}\times & \\ \times & \times\end{array}$					4	4	2	2 4	2 4
1—>4 $\begin{array}{cc}\times & \times\\ \times & \times\end{array}$	1	1	1	3	4	3 4	1 2 3	1 2 4	1—>4

Term k	A-contribution $\text{ACon}_{k,\text{modal}}$	B-contribution $\text{BCon}_{k,\text{modal}}$	Modal Value FMod
1	1	$\frac{4}{27}$	$\frac{4}{27}$
2	$\frac{1}{4}$	1	$\frac{1}{4}$
3	1	$\frac{1}{4}$	$\frac{1}{4}$
4	1	$\frac{1}{4}$	$\frac{1}{4}$

The largest modal value, FMax , is $\frac{1}{4}$ and only one term (2) has a B-contribution which exceeds this, so only fragments which include (2) have any chance of providing an improvement :

(1 2)
(2 4)
(1 2 3)
(1 2 4)
(1 2 3 4)

Fragments (1 2) and (2 4) are each 2-dimensional, defined on the ($a_{11} \times b_{1k}$) and the ($a_{11} \times b_{2k}$) spaces respectively, and are easily optimized by inspection using a graphics package. The first contains no interior maximum, but the second is found to have a maximum at the point

$$A = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ 0 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

The value at the peak is $\frac{8}{27}$, which exceeds FMax , and is a contender for the global maximum.

Fragments (1 2 3) and (1 2 4) are each 3-dimensional and so beyond visual inspection. However, using Geometric Programming (see Appendix C) each of their optimizations can be transformed into two equations in two unknowns :

for (1 2 3) --

$$u_1\ (1 - u_1)(1 - u_1 - u_2)(2 + u_2) = (1 + u_1 - u_2)(u_1 + u_2)(1 - u_1)(1 - u_1 - u_2)$$

$$u_2\ (1 + u_1 - u_2)(2 - u_2)(1 - u_1 - u_2) = u_2\ (u_1 + u_2)(2 - u_2)(1 - u_1 - u_2)$$

for (1 2 4) --

$$u_1\ (1 - u_1)^2\ (2 + u_1 - u_2) = u_1\ (1 + u_1 - u_2)(1 - u_1 + u_2)(1 - u_1 - u_2)$$

$$u_2\ (1 + u_1 - u_2)(2 - u_2)(1 - u_1 + u_2) = u_2\ (2 - u_2)(2 + u_1 - u_2)(1 - u_1 - u_2)$$

MathCAD finds 9 solutions to the first system, of which only 2 are interior critical points, and neither of which exceeds FMax . MathCAD finds 8 solutions to the second system, none of which is an interior point.

The remaining fragment (1 2 3 4) is 4-dimensional. To search for maxima, one option is hill-climbing which is initiated from closely-spaced points in the domain (see Section 5.7. and Appendix D). However, in this case Geometric Programming can be employed to transform the problem into a system of 3 equations in 3 unknowns, which MathCAD can then solve.

In this way, it is found that the maximum of fragment (2 4) provides the global optimum for this example. With hindsight, this solution might have been anticipated, since the two terms display a special (symmetrical mirror-image) relationship which is absent from the other fragments. More about the importance of symmetry in a later section.

5.3. The Observations

5.3.1. Observation One

Having amassed a body of examples of the global maximum of f_1 derived from a variety of different training sequences and model sizes, the next task was to examine them for common themes and patterns. Among the several points raised in this way, three observations seemed most significant.

The first of these states that for the global maximum of f_1 to result from a sum of terms (a simple union or compound fragment) is comparatively rare... in the great majority of known cases, the global maximum is provided by a single term (or, more accurately, by the modal position of a single term) .

The general explanation for this phenomenon seems to be that (a) single-term fragments tend to occupy the smallest possible number of dimensions, and (b) low dimensions are conducive to a high objective value.

Stated another way, note that each of the terms of f_1 is a product of matrix elements (T-1 factors from A , T factors from B) where the factors range from 0 to 1 . A high value for the term demands that all of its factors are as close as possible to 1 . However, when $x_{ij} = 1$ this forces the other elements from row i of the matrix to equal 0 . Consequently, the terms of f_1 which can achieve the highest value are those which include the smallest number of elements (preferably no more than a single element) from any row of A and B . Finally, with so few elements present from A and B , these terms have little opportunity to 'activate' other terms (see page 72), so tend to constitute the single-term fragments (singletons).

Of course, there are exceptions to this observation, in which the global maximum does result from a sum of terms. One becomes aware of a trade-off at work, between a single term offering a large value thanks to its low dimensions, and a sum of many terms each offering a more modest contribution as a consequence of their higher dimensions.

The equivalence between terms and state sequences has already been described (see page 53) . The A-contribution of a state sequence seems to measure the consistency with which exit paths are selected from the states. For example, if a sequence is such that from State i the model always (or nearly always) proceeds to a particular State j , this will be reflected in a larger A-contribution for the term.

In a similar way, the B-contribution seems to measure how strongly output symbols can be identified with states. For example, if a sequence is such that a particular Symbol k always (or nearly always) seems to occur whenever the model is in State j , this will be reflected in a larger B-contribution for the term.

The B-contributions are strongly dependent upon the observation sequence, while the A-contributions are independent of the sequence. Consequently, a symbol sequence which promotes a large B-contribution may not necessarily lead to a large A-contribution, and vice versa.

For example, note that when there are N states and N or fewer distinct symbols, it is always possible to find a term whose B-contribution is a perfect 1. Taking the sequence $k\,k\,l\,k$,

if we assign k generation to state 1
and assign l generation to state 2

the result is the state sequence 1-->1-->2-->1 , corresponding to the term $a_{11}\, a_{12}\, a_{21}\, (b_{1k})^3\, b_{2l}$. In this case, the B-contribution is a maximum possible 1 , but the modal value of the term is eroded by the relatively poor A-contribution of $\frac{1}{4}$.

It is cases like these (where the A- and B-contributions are out of step) that the balance of the trade-off described above becomes tipped away from the single term in favor of a sum of terms. Nevertheless, even when the global maximum is given by a sum, it is notable that the sum seems to tend towards a minimum number of dimensions.

5.3.2. Observation Two

Although the landscape of f_1 may in general be quite complicated, the fragment which provides its global maximum seems always to be a simple unimodal hump.

Clearly this is true when the fragment in question is a singleton, since these are unimodal, with the location and modal value given by Theorem A (page 54, see also the discussion on page 55) . Remarkably, it seems to hold in the case of multi-term fragments also.

Consider the problem defined by an $N = 4$ state model together with the training sequence $k\,k\,k\,k\,l$. The global optimum for f_1 is found to occur at the point

$$A = \begin{bmatrix} \frac{1}{5} & \frac{4}{5} & & \\ & \frac{1}{5} & \frac{4}{5} & \\ & & \frac{1}{5} & \frac{4}{5} \\ & & & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & \\ 1 & \\ 1 & \\ \frac{1}{5} & \frac{4}{5} \end{bmatrix}$$

which is the result of optimizing the 4-dimensional fragment

$$a_{11}\,a_{12}\,a_{23}\,a_{34}\,b_{1k}^{\,2}\,b_{2k}\,b_{3k}\,b_{4l} + a_{12}\,a_{22}\,a_{23}\,a_{34}\,b_{1k}\,b_{2k}^{\,2}\,b_{3k}\,b_{4l} + a_{12}\,a_{23}\,a_{33}\,a_{34}\,b_{1k}\,b_{2k}\,b_{3k}^{\,2}\,b_{4l}$$
$$+ \; a_{12}\,a_{23}\,a_{34}\,a_{44}\,b_{1k}\,b_{2k}\,b_{3k}\,b_{4k}\,b_{4l}$$

The unimodal shape of this function can be argued on the basis of the symmetry which exists among its terms. When the exponents matrices for the four terms are written

$$A^* = \begin{bmatrix} 1 & 1 & & \\ & & 1 & \\ & & & 1 \\ & & & \end{bmatrix} \qquad B^* = \begin{bmatrix} 2 & \\ 1 & \\ 1 & \\ & 1 \end{bmatrix}$$

$$A^* = \begin{bmatrix} & 1 & & \\ & 1 & 1 & \\ & & & 1 \\ & & & \end{bmatrix} \qquad B^* = \begin{bmatrix} 1 & \\ 2 & \\ 1 & \\ & 1 \end{bmatrix}$$

$$A^* = \begin{bmatrix} & 1 & & \\ & & 1 & \\ & & 1 & 1 \\ & & & \end{bmatrix} \qquad B^* = \begin{bmatrix} 1 & \\ 1 & \\ 2 & \\ & 1 \end{bmatrix}$$

$$A^* = \begin{bmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ & & & 1 \end{bmatrix} \qquad B^* = \begin{bmatrix} 1 & \\ 1 & \\ 1 & \\ 1 & 1 \end{bmatrix}$$

it is clear that each term is 1-dimensional, and when the fragment is re-written using x, y, z and w to distinguish those dimensions,

$$x\,(1\text{-}x)\,(1\text{-}y)\,(1\text{-}z)\,(1\text{-}w) \;+\; (1\text{-}x)\,y\,(1\text{-}y)\,(1\text{-}z)\,(1\text{-}w) \;+\; (1\text{-}x)\,(1\text{-}y)\,z\,(1\text{-}z)\,(1\text{-}w)$$
$$+\; (1\text{-}x)\,(1\text{-}y)\,(1\text{-}z)\,w\,(1\text{-}w)$$

one can observe the equal and symmetrical interactions between each term and the remaining three.

This symmetry, and the consequent unimodal shape, are further suggested by the facts that

(a) at the global maximum, each term contributes the same value, equal to $\frac{4^4}{5^5}$.

(b) the global maximum is a simple average of the four sets of exponents matrices (as in appendix Section D.4., described there as the 'pseudo-mode')

Finally, it is instructional to note what happens when fewer than four terms are included in the sum:

When any three of the terms are combined, the resulting 3-dimensional function is also a bona fide fragment of f_1, and represents a cross-section of the original 4-dimensional fragment. For example, the first three terms represent f_1 after it has been sectioned along the $b_{4k}\ (= w) = 0$ hyper-plane.

The new fragment is also unimodal, exhibiting the same sort of symmetry between its terms, but is slightly smaller, with the peak given by three times $\frac{3^3}{4^4}$ and the modal position found by averaging the three sets of exponents matrices.

This process can be continued to give a number of 2-dimensional and, finally, 1-dimensional functions. Each is a fragment of f_1 in its own right, and each is unimodal, but having a reduced peak compared with the previous (higher-dimensional) surface.

5.3.3. Observation Three

The third observation appears to be related to the second and may help to account for it :

In the case when a sum of terms (simple union or compound fragment) provides the global maximum, that sum invariably

(a) includes at least one term, k, which is a singleton of f_1 , and

(b) consists of one or more 'nearest neighbors' of the term k

--where a nearest neighbor is defined as a fragment which shares all the same dimensions as term k, but has exactly <u>one</u> additional dimension.

As a simple example, consider an $N = 2$ state model together with the training sequence $k\,l\,l\,k\,k$. The global optimum is given by maximizing the fragment (14 15 16) :

$$a_{12}^2 a_{21} a_{22} b_{1k}^2 b_{2k} b_{2l}^2 + a_{12} a_{21} a_{22}^2 b_{1k}^2 b_{2k} b_{2l}^2 + a_{12} a_{22}^3 b_{1k} b_{2k}^2 b_{2l}^2$$

Term k in this case is the third of these, for which

$$\mathrm{ACon}_{16,\,\mathrm{modal}} = 1$$

$$\mathrm{BCon}_{16,\,\mathrm{modal}} = \frac{1}{16}$$

and

$$\mathrm{FMod} = \mathrm{FMax} = \frac{1}{16}$$

and whose single dimension is evident from the profile

$$A_{16} = \begin{bmatrix} & \times \\ & \times \end{bmatrix} \qquad B_{16} = \begin{bmatrix} \times & \\ \times & \times \end{bmatrix}$$

Consistent with Observation Three, the fragment (14 15 16) is a nearest neighbor of term 16 , the extra dimension caused by the presence of factor a_{21} :

$$A = \begin{bmatrix} & \times \\ \times & \times \end{bmatrix} \qquad B = \begin{bmatrix} \times & \\ \times & \times \end{bmatrix}$$

As a second example, consider the four-term fragment described in Observation Two.

Here, <u>each</u> of the terms constitutes a valid singleton of f_1 , and each term is also a nearest neighbor of all the others, as is evident when the fragment is written in the form

$$x\,(1\text{-}x)\,(1\text{-}y)\,(1\text{-}z)\,(1\text{-}w) \;+\; (1\text{-}x)\,y\,(1\text{-}y)\,(1\text{-}z)\,(1\text{-}w) \;+\; (1\text{-}x)\,(1\text{-}y)\,z\,(1\text{-}z)\,(1\text{-}w)$$
$$+\; (1\text{-}x)\,(1\text{-}y)\,(1\text{-}z)\,w\,(1\text{-}w)$$

where x, y, z, w are again used to distinguish the 4 dimensions.

Term 1 is coupled with term 2 by the presence of the factor x in term 1 and the factor y in term 2 , and the sum of those two terms is exactly one dimension larger than either term individually.

When the fragment consists of <u>multiple</u> nearest neighbors of k such as this, it appears to be essential also for each of those to be a nearest neighbor of each other :

$$\begin{tikzcd}
\text{Term 1} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] \arrow[dr, leftrightarrow] & \text{Term 2} \arrow[d, leftrightarrow] \arrow[dl, leftrightarrow] \\
\text{Term 4} \arrow[r, leftrightarrow] & \text{Term 3}
\end{tikzcd}$$

For example, it is possible to exhibit similar fragments whose terms constitute a <u>chain</u> of nearest neighbors

$$\begin{tikzcd}
\text{Term 1} \arrow[r, leftrightarrow] \arrow[d, leftrightarrow] & \text{Term 2} \arrow[d, leftrightarrow] \\
\text{Term 4} \arrow[r, leftrightarrow] & \text{Term 3}
\end{tikzcd}$$

but where the terms on the diagonal are not nearest neighbors. In those cases, the global maximum is no larger than that provided by maximizing the sum of term 1 plus term 4, for example.

Observation Three reflects the apparent necessity for two terms to be quite close to each other geometrically before they can combine into a single hump. (See also the discussion in Appendix D, pages 278-279) This is also reflected in the unimodal shape of the global maximum fragment, as noted in Observation Two.

Consider the hypothetical fragment

$$f(x,y,z) = x^p (1-x)^q (1-y)^s (1-z)^r + (1-x)^r y^s z^p (1-z)^q$$

Notice there are two 1-dimensional terms which are coupled to each other across <u>two</u> dimensions, so that the sum forms a 3-dimensional fragment. Because there are two additional dimensions rather than one, the terms are <u>not</u> nearest neighbors.

Notice also that the exponents have been selected to make the terms symmetrical mirror images of one another. This symmetry confers two advantages :

(1) First, the position of the interior critical point is known--

$$x = z = \frac{p}{p+q+r} \quad (1-x) = (1-z) = \frac{q+r}{p+q+r} \quad y = (1-y) = \frac{s}{s+s} = \frac{1}{2}$$

(2) Second, the value at the interior critical point is as large as possible, in comparison with fragments whose terms are not symmetric.

To expand upon this second point, consider the alternative (non-symmetrical) fragment

$$g(x, y, z) = x^p (1-x)^q (1-y)^t (1-z)^r + (1-x)^r y^u (1-y)^v z^p (1-z)^q$$

Let $$s = \text{Minimum}\{ t , u \}$$

Then $(1-y)^s \geq (1-y)^t$ and $(1-y)^s \geq (1-y)^u$

from which $$x^p (1-x)^q (1-y)^s (1-z)^r \geq x^p (1-x)^q (1-y)^t (1-z)^r$$

and $$(1-x)^r y^s (1-y)^v z^p (1-z)^q \geq (1-x)^r y^u (1-y)^v z^p (1-z)^q$$

or $$h(x, y, z) = x^p (1-x)^q (1-y)^s (1-z)^r + (1-x)^r y^s (1-y)^v z^p (1-z)^q \geq g(x, y, z)$$

Finally since $$1 \geq (1-y)$$

then $$1 \geq (1-y)^v$$

$$y^s \geq y^s (1-y)^v$$

$$(1-x)^r y^s z^p (1-z)^q \geq (1-x)^r y^s (1-y)^v z^p (1-z)^q$$

$$f(x, y, z) \geq h(x, y, z) \geq g(x, y, z)$$

In other words, for any non-symmetrical fragment $g(x, y, z)$ a symmetrical fragment having the form of $f(x, y, z)$ can be found which serves as its upper bound on the domain. Note also that along the $z = 0$ boundary, the functions are equal.

Finally, back to the landscape of $f(x, y, z)$, with its symmetrical terms. The value of f at the modal position of its first term is

$$\left(\frac{1}{p+q}\right)^{p+q} p^p q^q$$

and the value of f at the position of the interior critical point is

$$\left(\frac{1}{p+q+r}\right)^{p+q+r} p^p (q+r)^{q+r} \left(\frac{1}{2}\right)^{s-1}$$

We ask, for what values of p, q, r and s does the function value at the interior critical point of f exceed the boundary peak? (When do the symmetric mirror-image terms combine to produce an interior maximum?)

After algebra, that condition can be re-expressed as

$$\left(1+\frac{p}{q+r}\right)^{q+r} \left(1+\frac{r}{p+q}\right)^{p} \left(\frac{q}{p+q}\right)^{q} < \left(\frac{1}{2}\right)^{s-1}$$

The following printout gives selected values of the left-hand expression for values of p, q and r ranging from 1 through 10 . We see there are numerous combinations for which the expression on the left of the inequality is less than 2 , but none where it is less than 1 . In other words, the condition <u>can</u> be satisfied provided $s = 0$, but is not satisfied for $s \geq 1$. However, when $s = 0$, we observe that the two terms making up $f(x, y, z)$ are nearest neighbors, while for $s \geq 1$ they are not. So the symmetrical mirror-image terms can combine to form an interior hump only if they are nearest neighbors.

```
p = 1 q = 1 r = 1 x = 1.687500
p = 1 q = 2 r = 1 x = 1.404664
p = 1 q = 2 r = 2 x = 1.808449
p = 1 q = 3 r = 1 x = 1.287460
p = 1 q = 3 r = 2 x = 1.574640
p = 1 q = 3 r = 3 x = 1.861669
p = 1 q = 4 r = 1 x = 1.223059
p = 1 q = 4 r = 2 x = 1.446001
p = 1 q = 4 r = 3 x = 1.668874
p = 1 q = 4 r = 4 x = 1.891702
p = 1 q = 5 r = 1 x = 1.182283
p = 1 q = 5 r = 2 x = 1.364508
p = 1 q = 5 r = 3 x = 1.546697
p = 1 q = 5 r = 4 x = 1.728860
p = 1 q = 5 r = 5 x = 1.911006
p = 1 q = 6 r = 1 x = 1.154130
p = 1 q = 6 r = 2 x = 1.308229
p = 1 q = 6 r = 3 x = 1.462307
p = 1 q = 6 r = 4 x = 1.616370
p = 1 q = 6 r = 5 x = 1.770421
p = 1 q = 6 r = 6 x = 1.924464
p = 1 q = 7 r = 1 x = 1.133520
p = 1 q = 7 r = 2 x = 1.267021
p = 1 q = 7 r = 3 x = 1.400509
p = 1 q = 7 r = 4 x = 1.533987
p = 1 q = 7 r = 5 x = 1.667458
p = 1 q = 7 r = 6 x = 1.800924
p = 1 q = 7 r = 7 x = 1.934384
p = 1 q = 8 r = 1 x = 1.117776
p = 1 q = 8 r = 2 x = 1.235540
p = 1 q = 8 r = 3 x = 1.353296
p = 1 q = 8 r = 4 x = 1.471045
p = 1 q = 8 r = 5 x = 1.588789
p = 1 q = 8 r = 6 x = 1.706529
p = 1 q = 8 r = 7 x = 1.824266
p = 1 q = 8 r = 8 x = 1.942000
p = 1 q = 9 r = 1 x = 1.105356
p = 1 q = 9 r = 2 x = 1.210704
p = 1 q = 9 r = 3 x = 1.316046
p = 1 q = 9 r = 4 x = 1.421384
p = 1 q = 9 r = 5 x = 1.526719
p = 1 q = 9 r = 6 x = 1.632050
p = 1 q = 9 r = 7 x = 1.737379
p = 1 q = 9 r = 8 x = 1.842706
p = 1 q = 9 r = 9 x = 1.948031
p = 1 q = 10 r = 1 x = 1.095307
p = 1 q = 10 r = 2 x = 1.190609
p = 1 q = 10 r = 3 x = 1.285906
p = 1 q = 10 r = 4 x = 1.381201
p = 1 q = 10 r = 5 x = 1.476493
p = 1 q = 10 r = 6 x = 1.571782
p = 1 q = 10 r = 7 x = 1.667070
p = 1 q = 10 r = 8 x = 1.762357
p = 1 q = 10 r = 9 x = 1.857642
p = 1 q = 10 r = 10 x = 1.952926
p = 2 q = 2 r = 1 x = 1.808449
p = 2 q = 3 r = 1 x = 1.574640
p = 2 q = 4 r = 1 x = 1.446001
p = 2 q = 4 r = 2 x = 1.973081
p = 2 q = 5 r = 1 x = 1.364508
```

```
p = 2 q = 5 r = 2 x = 1.785090
p = 2 q = 6 r = 1 x = 1.308229
p = 2 q = 6 r = 2 x = 1.657554
p = 2 q = 7 r = 1 x = 1.267021
p = 2 q = 7 r = 2 x = 1.565455
p = 2 q = 7 r = 3 x = 1.895303
p = 2 q = 8 r = 1 x = 1.235540
p = 2 q = 8 r = 2 x = 1.495873
p = 2 q = 8 r = 3 x = 1.781000
p = 2 q = 9 r = 1 x = 1.210704
p = 2 q = 9 r = 2 x = 1.441475
p = 2 q = 9 r = 3 x = 1.692313
p = 2 q = 9 r = 4 x = 1.963217
p = 2 q = 10 r = 1 x = 1.190609
p = 2 q = 10 r = 2 x = 1.397792
p = 2 q = 10 r = 3 x = 1.621550
p = 2 q = 10 r = 4 x = 1.861883
p = 3 q = 3 r = 1 x = 1.861669
p = 3 q = 4 r = 1 x = 1.668874
p = 3 q = 5 r = 1 x = 1.546697
p = 3 q = 6 r = 1 x = 1.462307
p = 3 q = 7 r = 1 x = 1.400509
p = 3 q = 7 r = 2 x = 1.895303
p = 3 q = 8 r = 1 x = 1.353296
p = 3 q = 8 r = 2 x = 1.781000
p = 3 q = 9 r = 1 x = 1.316046
p = 3 q = 9 r = 2 x = 1.692313
p = 3 q = 10 r = 1 x = 1.285906
p = 3 q = 10 r = 2 x = 1.621550
p = 4 q = 4 r = 1 x = 1.891702
p = 4 q = 5 r = 1 x = 1.728860
p = 4 q = 6 r = 1 x = 1.616370
p = 4 q = 7 r = 1 x = 1.533987
p = 4 q = 8 r = 1 x = 1.471045
p = 4 q = 9 r = 1 x = 1.421384
p = 4 q = 9 r = 2 x = 1.963217
p = 4 q = 10 r = 1 x = 1.381201
p = 4 q = 10 r = 2 x = 1.861883
p = 5 q = 5 r = 1 x = 1.911006
p = 5 q = 6 r = 1 x = 1.770421
p = 5 q = 7 r = 1 x = 1.667458
p = 5 q = 8 r = 1 x = 1.588789
p = 5 q = 9 r = 1 x = 1.526719
p = 5 q = 10 r = 1 x = 1.476493
p = 6 q = 6 r = 1 x = 1.924464
p = 6 q = 7 r = 1 x = 1.800924
p = 6 q = 8 r = 1 x = 1.706529
p = 6 q = 9 r = 1 x = 1.632050
p = 6 q = 10 r = 1 x = 1.571782
p = 7 q = 7 r = 1 x = 1.934384
p = 7 q = 8 r = 1 x = 1.824266
p = 7 q = 9 r = 1 x = 1.737379
p = 7 q = 10 r = 1 x = 1.667070
p = 8 q = 8 r = 1 x = 1.942000
p = 8 q = 9 r = 1 x = 1.842706
p = 8 q = 10 r = 1 x = 1.762357
p = 9 q = 9 r = 1 x = 1.948031
p = 9 q = 10 r = 1 x = 1.857642
p = 10 q = 10 r = 1 x = 1.952926
```

Furthermore, this situation must be roughly true also for the non-symmetrical cases. For every non-symmetrical sum of terms there exists a symmetrical upper bound such that its (equal) boundary peaks are comparable to at least one of the peaks of the non-symmetrical sum. When the interior critical point of the upper bound falls below its boundary peaks (i.e. when the symmetrical terms are not nearest neighbors), then the interior critical point of the non-symmetrical sum must do so also, since the non-symmetrical surface cannot exceed its upper bound.

This little demonstration has aimed to show that two terms cannot form an interior maximum unless they are nearest neighbors. While still a far cry from a rigorous proof of this, it does seem to add some credance to Observation Three and the suggestion that terms need to be quite close geometrically before they will form a hump.

5.4. New Algorithm

Assuming the above three observations to be typical of the general situation, it is possible to adapt them into a process for finding the global maximum of f_1 .

This process is basically the reverse of that described in Section 5.3.2. , where the maximizing hump was repeatedly sectioned into lower and lower dimensions until an (atomic) single-term fragment was reached :

> Find the set of singletons f_1
>
> For each singleton, identify the fragments which are its nearest neighbors
>
> Find the nearest neighbors of <u>those</u> fragments, and continue expanding the fragments by one additional dimension in this way for as long as each expansion produces a taller hump than the last

More formally :

For a given model size N and training sequence $O = o_1 o_2 .. o_T$

(1) Find the set of single-term fragments of f_1 and calculate their modal values

(2) For each singleton from step (1)

  (A) Identify the fragments of f_1 which are its nearest neighbors (i.e. differ from it by having one extra dimension)

  (B) Perform a hill-climb of each nearest neighbor from some convenient interior point, and retain those which result in an improved interior maximum (i.e. higher than the end-point maximum provided by the singleton)

  (C) Pairing the survivors from (B) with the fragments found in (A), identify new combinations which result in one additional dimension. Then hill-climb these larger fragments, again retaining only those which show an improved maximum over their components.

  (D) Continue the process of building fragments which are one dimension larger than their components, hill-climbing, and retaining those which show an improvement.

  (E) The process terminates when
      no new combinations are possible, or
      none of the new combinations show improvement

The resulting set of peaks essentially constitute the landscape of f_1 . The highest of these peaks represents the global maximum.

Given the list of nearest neighbors from step (2A) , the intention is to examine every possible combination of 1 , 2 , 3 , etc neighbors to see at what point the objective improvement halts. The process resembles crystal-growing, where the singleton fragments of step (1) provide the 'seeds', and where a fragment is allowed to grow another dimension only when doing so produces an improved maximum.

5.5. Finding the Singletons

Although step (2) , the process of identifying and combining the nearest neighbors, may appear very laborious, the fact that most global maxima are given by singletons (Observation One) generally means that this part of the algorithm runs quickly. Instead, it is step (1) , the identification of the singletons, which poses the more serious bottleneck.

On the face of it, the number of terms to be investigated as possible singletons must increase at the rate N^{t-1} , where each investigation then requires testing the candidate term against <u>all</u> of the remaining terms (in the worst case) for possible activation. Clearly, such an effort could render the algorithm a non-starter for all but the smallest problems.

Fortunately, it is possible to 'grow' the set of singletons for f_1 on a symbol-by-symbol basis as the successive elements of the observation sequence become known. This also opens the door for constructing the globally optimum HMM on an incremental basis.

The key observation which makes this possible goes as follows :

> If a state sequence (i.e. term) 'eclipses' some other sequence, and if both sequences share the same final state, then not only is the eclipsing sequence <u>not</u> a singleton, but also none of the sequences which can be formed by appending additional states to that sequence can ever by singletons, either.

To elaborate, consider an $N = 2$ state model, the observed symbol sequence $k\ l\ l\ k$, and the proposed state sequence 1->1->2->1 . This produces the term $a_{11}\, a_{12}\, a_{21}\, b_{1k}^{2}$ $b_{1l}\, b_{2l}$ whose modal position is computed by dividing the exponents matrices

$$A^* = \begin{array}{c|c} & \text{Row Sum} \\ \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} & \begin{array}{c} 2 \\ 1 \end{array} \end{array} \qquad B^* = \begin{array}{c|c} & \text{Row Sum} \\ \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} & \begin{array}{c} 3 \\ 1 \end{array} \end{array}$$

by their row sums as described already on pages 54-55 .

Also, consider the second state sequence 1->1->1->1 .

Sequence 1 1 2 1 is said to 'eclipse' sequence 1 1 1 1 in the sense that, when their silhouettes are compared

$$A = \begin{bmatrix} \times & \times \\ \times & \end{bmatrix} \qquad B = \begin{bmatrix} \times & \times \\ & \times \end{bmatrix}$$

versus

$$A = \begin{bmatrix} \times & \\ & \end{bmatrix} \qquad B = \begin{bmatrix} \times & \times \\ & \end{bmatrix}$$

all of the marked positions for 1 1 1 1 are covered by the marked positions for 1 1 2 1 .

Clearly, when sequence X eclipses sequence Y, X 'activates' Y (in the sense described on page 72) since the elements of A and B which are required to produce a non-zero Y are provided by X. Thus, whenever X is active (non-zero), Y is also * .

* However, the converse is false. Sequence X can activate Y without eclipsing it. The exceptions always involve terms of the sort (i.e. unspecified rows) described in the footnote on page 81 .

Now consider two state sequences $X = x_1 x_2 .. x_{t-1}\ i$ and $Y = y_1 y_2 .. y_{t-1}\ i$ which both terminate in the same final state, i, and assume that X eclipses Y.

Then X is not a singleton (single-term fragment), since it activates another term so can never appear in isolation. Furthermore, assume that the next observed symbol (at time t+1) is k.

Whatever state j this next symbol is assigned to,

$$x_1 x_2 .. x_{t-1}\ i j \quad \text{activates} \quad y_1 y_2 .. y_{t-1}\ i j$$

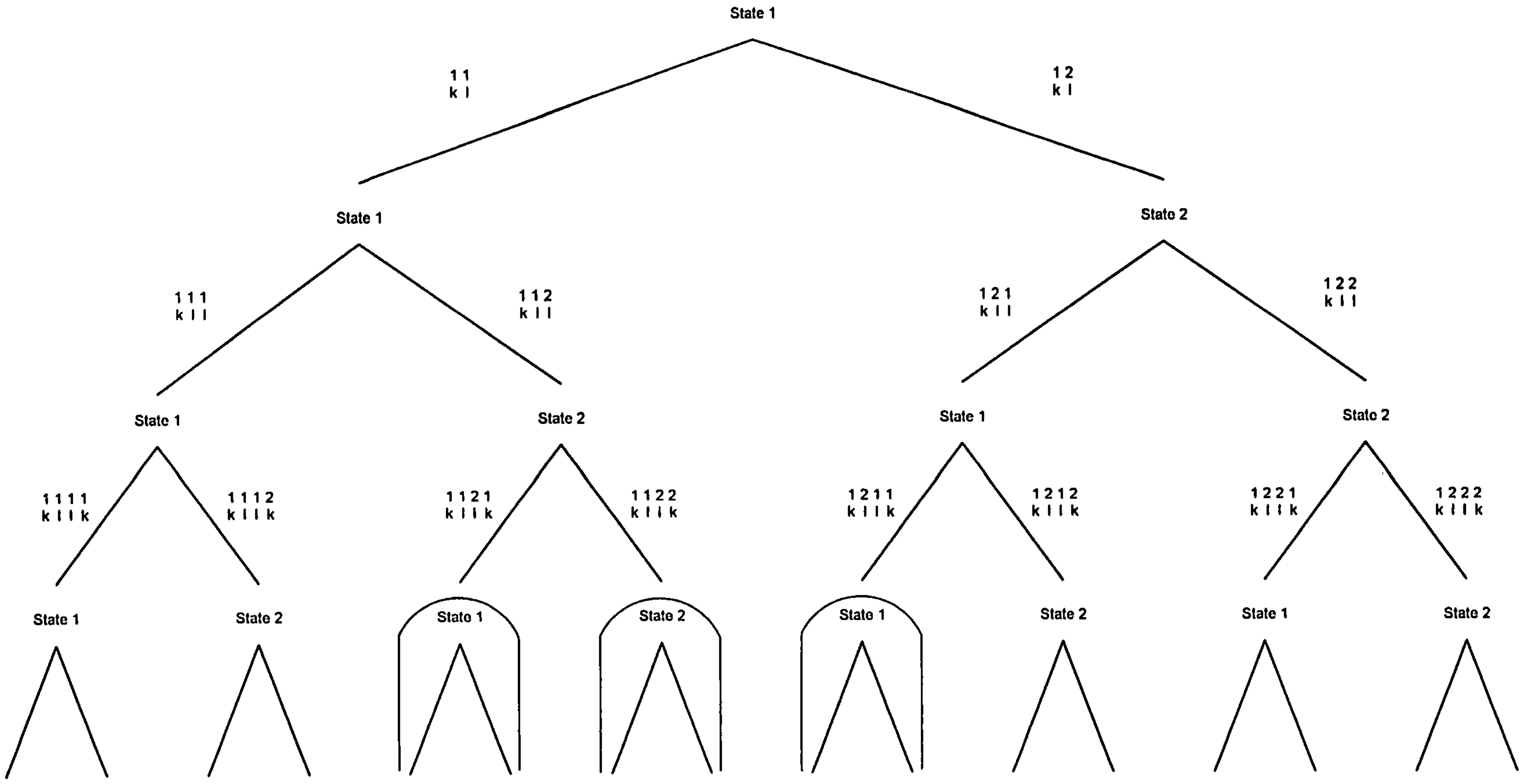
since the only possible changes to their silhouettes, i.e.

a_{ij} becomes marked in A
b_{jk} becomes marked in B

will be the same for both terms--and consequently the updated X silhouette will continue to eclipse the updated Y silhouette.

Finally, this argument can be extended to any number of additional states appended to X and Y. Thus, none of the descendents of X can be singletons, since there always exists a term (namely, the corresponding descendent of Y) which it activates (eclipses).

Based upon this result, the procedure for locating singletons as required in step (1) of the new algorithm can be illustrated using the tree diagram below. Starting from state 1 and symbol k, for each successive observed symbol in the training sequence there are N possible states it could have been issued from, resulting in N children for each of the already existing state sequences, and the given N-ary tree.


State 1
1 1 k l
1 2 k l
State 1
State 2
1 1 1 k l l
1 1 2 k l l
1 2 1 k l l
1 2 2 k l l
State 1
State 2
State 1
State 2
1 1 1 1 k l l k
1 1 1 2 k l l k
1 1 2 1 k l l k
1 1 2 2 k l l k
1 2 1 1 k l l k
1 2 1 2 k l l k
1 2 2 1 k l l k
1 2 2 2 k l l k
State 1
State 2
State 1
State 2
State 1
State 2
State 1
State 2

At each level t there are N^{t-1} state sequences, any one of which is a potential singleton. However, noting that sequence 1 1 2 1 eclipses 1 1 1 1 , then 1 1 2 1 fails the singleton test. Furthermore, since 1 1 2 1 and 1 1 1 1 both terminate in 1 , then by the above result none of the descendents of 1 1 2 1 can be singletons either, so this branch may be pruned from the tree. The same holds for sequences 1 1 2 2 and 1 2 1 1 (which eclipse 1 2 2 2 and 1 1 1 1 , respectively) .

The sequence 1 1 1 2 presents a third possibility. This eclipses the sequence 1 1 1 1 , however the two sequences don't share the same final state. While 1 1 1 2 is not itself a singleton, some of its descendents <u>may</u> be singletons, so the branch beginning from 1 1 1 2 cannot be pruned from the tree.

Thus at each level the state sequences fall into three categories :

Singletons	1 1 1 1	
	1 2 1 2	
	1 2 2 1	
	1 2 2 2	
Disposables	1 1 2 1	
	1 1 2 2	
	1 2 1 1	which can be pruned, and
Reserves	1 1 1 2	

which must be carried forward until the nature of their descendents can be established.

The process of generating, classifying and pruning the N offspring of the survivors at level t is repeated for each successive observed symbol in the training sequence. Finally, after the last symbol, the list of singletons (but not the reserves) is passed on to step (2) of the algorithm.

5.6. Algorithm Efficiency

At each level of the tree on page 127, as another observed symbol is received, the singletons and reserves carried forward from the previous level each generate N offspring whose type (singleton/disposable/reserve) must be ascertained. Fortunately, thanks to the pruning action which is possible, the rate of growth of the tree is significantly less than the value N^{t-1} it would be otherwise.

It is difficult to state the new rate of growth precisely, since it depends very closely upon the particular sequence of symbols observed. However, in tests the number of state sequences carried forward to the next level remains roughly constant from level to level, and is some small multiple of

$$K = \begin{cases} \dfrac{N(N+1)}{2} + B_M & \text{for } M \leq N \\[2ex] \dfrac{N(N+1)}{2} + B_N N^{M-N} & \text{for } M > N \end{cases}$$

where N is the number of states
M is the number of distinct symbols received at that point
and B_n is the nth Bell number, equal to

$$B_n = S_n^1 + S_n^2 + .. + S_n^n$$

where S_n^m are Stirling numbers of the second kind, defined recursively as

$$S_n^n = S_n^1 = 1$$

$$S_{n+1}^m = S_n^{m-1} + m S_n^m \qquad 1 < m \leq n$$

$$S_n^m = 0 \qquad \text{otherwise}$$

A table showing Stirling and Bell numbers for small values is given on the next page. See also [36,37].

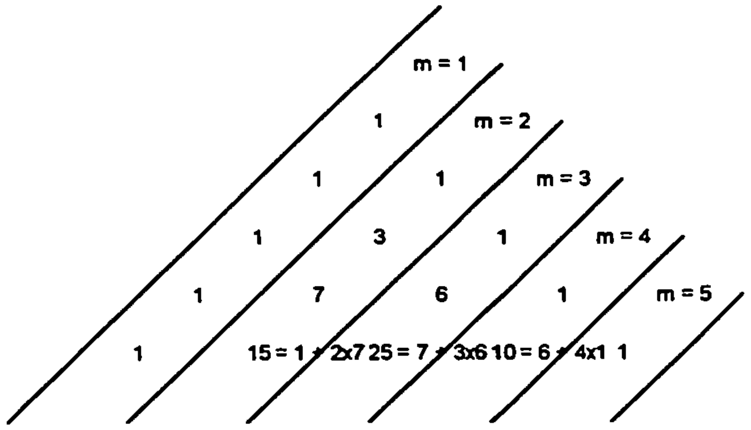
1

1 1

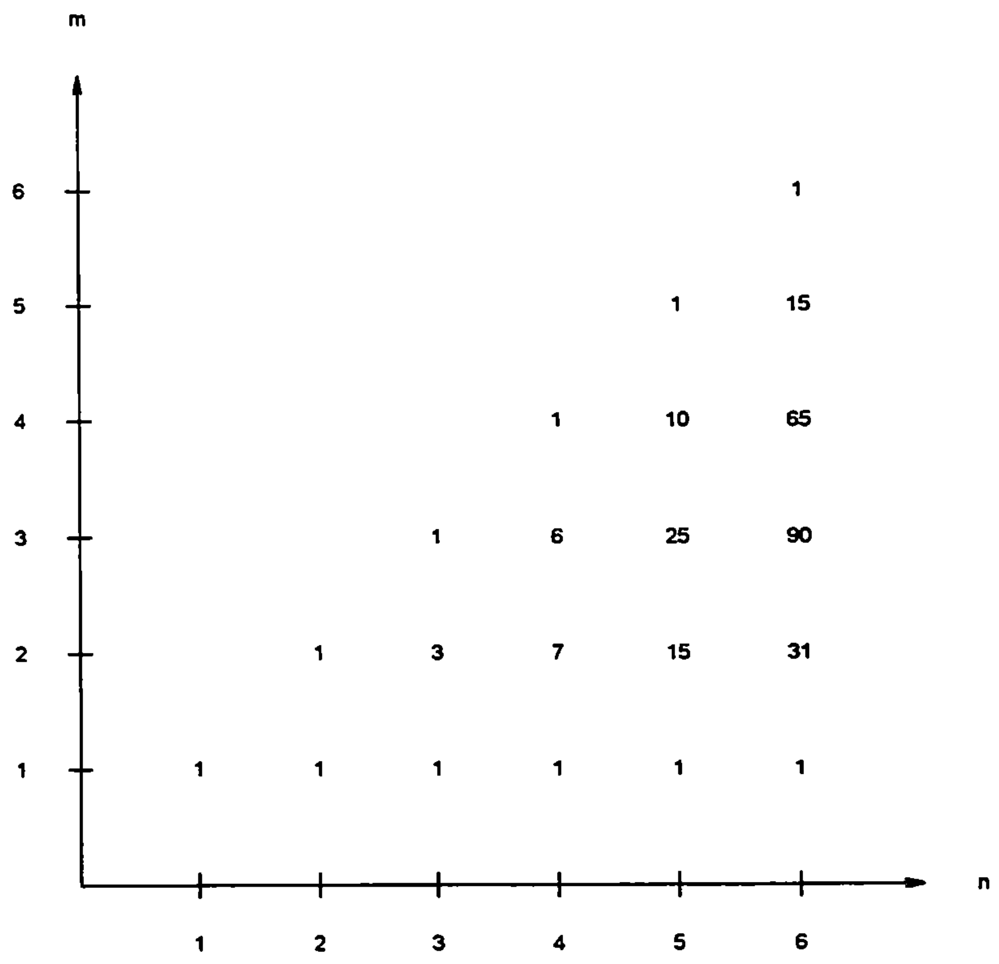
1 2 1

1 3 3 1

1 4 = 1 + 3 6 = 3 + 3 4 = 3 + 1 1

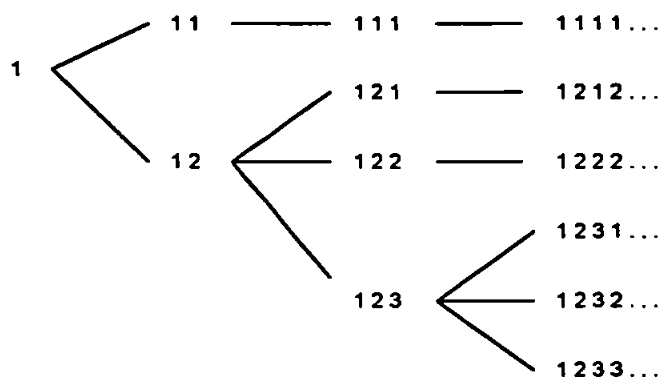


Stirling numbers of the second kind represent a generalization of Pascal's Triangle in which, rather than compute an entry by merely summing the two numbers above it, the right-hand number is first multiplied by the appropriate value of m .



n	1	2	3	4	5	6
B_n	1	2	5	15	52	203

The first term of K represents the number of ways in which a unique exit path can be assigned to each of the N states (page 111, paragraph 1). For $N = 3$ states, this number stabilizes at six, as follows :



In a similar way, the second term of K represents the number of ways in which the M distinct symbols can be identified with one of the N available states (see page 111, paragraph 2).

The sum of the two terms provides a lower bound on the number of singletons, since it represents those state sequences for which there is (at most) one non-zero value along each row of the A-matrix or one non-zero value down each column of the B-matrix, which state sequences are trivially confirmed as singletons. However, singletons other than these usually exist, and the actual number carried forward is roughly some small integer multiple of K. This number grows exponentially (in the second term), but only with M, not T. Thus, the number of survivors (width of the 'beam', in the parlance of tree searching [38]) remains roughly constant from level to level, until a previously unobserved symbol is received, whereupon that number makes a quantum jump.

Finally, at the next level of the tree the number of offspring is $N \times K$. However only $K \times (K\text{-}1)$ tests for activation (eclipsing) need to be carried out (in the worst case), since a state sequence only needs to be tested against other sequences which end in the same state. Thus, the total number of tests conducted during Step (1) of the algorithm is roughly proportional to

$$K \times (K\text{-}1) \times (T\text{-}1)$$

5.7. Test Procedure

The algorithm has been programmed in 'C' , to run on a Sun SPARCstation 5 workstation. A listing of the program (known as NHB, for New HMM Builder) has been included as Appendix E .

The program requires two inputs :

> a single training sequence in the form of a character string (commencing with the letter '*a*' *) , where the characters symbolize members of a finite alphabet of possible outputs (see Section 1.4.)
>
> the required number of HMM model states, N

NHB then generates two results based upon those inputs :

> the parameters (A and B matrices) of the proposed 'globally' optimum HMM
>
> the probability $P = P(o_1 o_2 .. o_T)$ that the proposed HMM would produce the given training sequence

As a preliminary test of the algorithm's effectiveness in returning models which are globally optimal, the body of examples for which a 'known' globally optimum solution was available were presented to the computer program. NHB succeeded in reproducing all of those solutions. However, this was not entirely surprising, since the algorithm was constructed around those examples in the first place.

* The first received output is allocated the letter '*a*' and, as previously unobserved outputs are received, they are allocated the letters '*b*', '*c*', '*d*' , etc. in order

So, in order to conduct a more independent test, a set of test data was then collected. These consisted of randomly selected character strings, ranging from $T = 5$ to 15 characters in length, generated according to the following (pseudo-code) instructions :

```
Symbol [1]  <-- 1
H <-- 1
for i = 2 to T
    choose  Symbol [i]  randomly among the
            integers from  1  to  H
    if Symbol [i]  =  H , then  H  <--  H + 1
```

The collection of sequences generated in this way are shown on the following page.

Next, these test sequences were input to NHB five times, once for each value of N ranging from 3 to 7. However, due to the long running times of the program for larger values of N and T, many of the trials had to be aborted prior to completion. The number of input combinations for which results were successfully recorded are shown in the table on page 136 .

Finally, efforts were made to confirm that the results generated by NHB were globally optimal. As noted earlier, to conclusively identify the global optimum solution for a particular objective function is generally very difficult. To make a 'reasonable' attempt to achieve this, a further series of computer programs were written which performed various systematic searches of the parameter space for each of the trials successfully conducted above. If a point within the parameter space could be found for which the value of P exceeded the value exhibited by NHB at its proposed solution, this would certainly disprove the ability of NHB to provide the global maximum. Alternatively, if no such point were ever found, this would add credence to (but not prove) its effectiveness.

T = 5	T = 6	T = 7	T = 8	T = 9	T = 10	T = 11
ababb	aabbcb	abacded	abaaaabb	abccbdaeb	aabacccabb	abaccbcdadb
abccc	aaabaa	aaaabbc	abbbbbbb	abcbabcdb	abbbbcaabd	aaaabbcadda
abcad	aaabca	abbaaaa	aaabbaba	abcbbabcb	abaabbbcca	abcadbaabac
abbaa	ababcb	abaacaa	abbaacdb	aabbcddec	aababaaabc	aaaababacde
aabac	abbbba	aaabaca	aaaaabcb	aabcbadda	aaabacccdc	abcbdbbcacd
aabaa	abbacb	aabbcbc	aaaabbbc	ababbbbcc	abcdecebca	aaabcbaaacd
aaaab	aabaaa	aaabcac	abcdbdeb	aabbacaad	aabcbdefdg	ababacdbccd
aabca	aaaabc	abbcdda	abcdedee	abbcbcddc	abccaaddae	abacbaccaaa
ababc	ababbb	aabbbac	abcacccd	abbaacbba	aabaabacab	abcbcabddcd
abbac	aabccd	abcbdea	aaabacdc	aabcbbdba	abccbdccdb	abbbccdeabe
					abbaaabbaa	

T = 12	T = 13	T = 14	T = 15
aaaaaaabbaba	abcccaccdeaec	aabbbababcacdb	abaccdaaeffeecd
aaaabcacabab	aaabccacadace	aaaaaaaaaababa	aabcbbcdbeaeffb
aabaabaccabb	aaabcdeabcbdc	abcacabdaeafce	abacaddbbbaccad
abacdbdabbce	aabbbaccacaba	abbbaacacacacd	aaaabbbbbaaacbd
aaaabacccbdc	aabaaaccbdcee	aabcbbdabbbbdc	abbcccccdddbacc
abacccbddddb	aaabbbbcbacac	aabaabbabaacda	aabcbdabcdbbcbb
abccddbcebce	abcaaaadebfeb	abacdaecacdfeb	abbacaabcdccaab
abaccdefefcb	aabbcdbeacbed	aabcadaeefafeg	aabacbaddccccee
abcccaaadadd	aaaababcabcda	abccdabeeebcbe	abbbbcdbefdeccg
abcccaccccad	abcbdddeceefb	ababcbddefadgf	aabbabaabbcbbac

The table below indicates the amount of data gathered per model size N and training sequence length T :

	N $\longrightarrow$				
T $\downarrow$	3	4	5	6	7
5	10	10	10	0	0
6	10	10	10	2	0
7	10	10	10	4	0
8	10	10	4	2	2
9	10	10	2	0	0
10	11	6	0	0	0
11	10	2	0	0	0
12	10	0	0	0	0
13	10	0	0	0	0
14	6	0	0	0	0
15	0	0	0	0	0

The first program in this series performed a Baum-Welch style hill-climb from the geometric center of the solution space, along with hill-climbs from the geometric center of each boundary 'plane' of the solution space. This strategy was, in part, guided by the possibility (see Sections D.5. and 5.3.2.) that the landscape of the objective function may be fairly simple (even uni-modal) in the vicinity of the global max. If this were true, a hill-climb conducted within the 'right' boundary plane should certainly find it. Unfortunately, this strategy quickly became overwhelmed by the size of the task. With N states and M distinct symbols in a test sequence, that could require a possible

$$(2^N - 1)^N \times (2^M - 1)^N$$

number of hill-climbs to complete the trial for that sequence. It was soon found that only the $N = 3$ set of trials were amenable to this kind of search.

The second program in this series also performed multiple hill-climbs. This time they were initiated from equally spaced starting locations within the interior of the solution space only (i.e. the boundaries were not included). The intention was that, if the grid of starting locations was sufficiently fine, some of the climbs would occur near enough to the boundary planes to detect any end-point extrema. Once again, however, the sheer enormity of the search task for reasonable grid densities rendered this strategy unusable for the larger values of N.

The third program in this series of tests abandoned the idea of hill-climbs, and simply carried out evaluations of the objective function, P, at the same grid of equally spaced locations that were used in Program 2. This met with better success, and Appendix F presents the results of that test, showing the highest located peaks for each trial, plus the elapsed computer time taken to complete each trial.

Eventually, however, even the third program became too slow to use. At that point, Program 3 was replaced by a battery of four HMM model-building programs, all performing classic Baum-Welch re-estimation (i.e. a single hill-climb), but differing in the model architecture and choice of starting point :

	Fully-connected	Left-right (2 skip states)
Initialized uniformly	BW1	BW2
Initialized randomly	BW3	BW4

Using these faster, but far less thorough, Baum-Welch programs, a full set of comparisons was eventually achieved.

Finally, the results of the various tests can be stated qualitatively as follows :

(1) No counter-example was ever found which dis-proved the ability of NHB to find the global maximum solution.

(2) The peak value for P supplied by NHB is often several orders of magnitude larger than from the other programs.

(3) However, for all but the smallest values of model size and sequence length, NHB is relatively slow (orders of magnitude slower than the Baum-Welch programs).

6. Conclusions

6.1. Review

In this final chapter we review the contents of the thesis before giving a brief summary of the contributions of the research.

In the Introduction we presented the fundamentals of Hidden Markov Model technology within the setting of Automatic Speech Recognition. We then attempted to make a case for employing the globally optimum HMM, as opposed to the locally optimum models which are produced by the Baum-Welch algorithm. We also tried to make a case for analyzing Discrete HMMs trained using a single training sequence as a first step towards understanding how to generate these globally optimum models.

In Chapter 2 we launched into a straight-forward attack upon that analysis using the standard optimization strategies of calculus, where the objective function to be optimized was the probability, $P(O)$, that a Discrete HMM would produce the given training sequence, and the independent variables were the parameters of the model, given by the elements of A, B and π. The intention was to develop formulas for finding the complete set of optima and then choosing the global optimum from among them. Efforts were complicated by the large number of (scalar) domain variables involved, so tools were developed for treating the optimization problem in matrix form.

Part-way through this analysis, we were able to prove that an upper bound exists on those optima of the objective function which occur within the interior of its domain (parameter space). However, a shortcoming of calculus-based optimization is that it tends to overlook possible endpoint extrema, which occur on the boundaries of the domain. It was fairly easy to show that, in the case of a single training sequence at least, the global optimum is not to be found in the interior but must reside on a boundary of the domain--namely, that sub-space where the first element of π equals 1 and all the other elements of π equal 0.

As a consequence of this finding, we were able to replace the original objective function with a slightly simpler function of A and B only, namely $f_1(A, B)$, which corresponds to the set of possible state sequences which the model might pass through beginning from State 1. However, as with the earlier objective function, it was still necessary to optimize f_1 over each of its boundary planes as well as the interior of its domain.

It became evident that optimizing f_1 over each of its boundary planes is equivalent to optimizing the many fragments of f_1 which result when one or more of the elements of A and/or B equal zero. Not all of the possible combinations are actually capable of reproducing the given training sequence, however, so a procedure was given which can identify the relevant fragments for a particular training sequence.

Next came a brief examination into the nature of these fragments. The boundaries to which they correspond differ in their dimensions (e.g. vertex, edge, plane) depending upon the number of free variables the fragment contains (which is similar to the number of non-zero elements in A and B). This situation leads to a natural hierarchy among the fragments--larger fragments are found to contain smaller fragments, in the way that planes contain edges, which in turn contain vertices, etc.

In addition, the fragments are observed to fall into distinct categories. The simplest are the atomic 'singletons', consisting of only a single term of f_1, while those containing multiple terms can be classified as either 'simple unions' or 'compound fragments', depending upon whether the fragment consists entirely of smaller fragments, or whether it also includes additional 'interior' terms.

On their domain, the singleton fragments are uni-modal humps. These form the most basic features of the 'landscape' of f_1 : f_1 must contain at least as many peaks as there are singletons. The question is, as more dimensions are added (as fragments are combined to create larger fragments), how many additional peaks are introduced? The situation can be pictured (very crudely) in terms of a square sheet of stiffened tent canvas supported by a set of poles, where the terms of the fragment constitute the tent poles and their modal positions provide the locations of those poles. Under what conditions do terms combine with each other to produce a new hump in the canvas which is taller than the participating poles... or when does the canvas merely sag down between the poles? This is one of the central questions of the research.

In pursuit of an answer, the research became directed towards finding a mechanical procedure for optimizing the individual fragments of f_1 . Except in simple cases, this would not be sufficient, by itself, to globally optimize f_1 since there are at least as many fragments as there are terms of f_1 , a number which grows exponentially as N^{T-1} . On the other hand, such a procedure would clearly be useful in its own right, and could shed light on the mechanics of peak creation. In this way, it might also help to predict which among the many fragments are the likely ones to provide the global maximum.

Among the many different attempts which were made to develop such a procedure, three are described in the thesis. The first of these used the Matrix Derivatives approach, revisited from Chapter 2 and now adapted to optimize the objective over one of its boundaries. This attempt became 'stuck' while trying to solve for the Lagrange multipliers which are used to enforce the constraints upon the domain variables.

Next came a promising new tool known as Geometric Programming, which had been developed specifically to optimize functions of the present form. This is also calculus-based and is shown to be the 'dual' of the standard approach. Within the thesis it was discovered how to automate the transformation from the primal to the dual G.P. equations, but there was only partial success with automating the solution of the dual equations.

The third attempt investigated a new hill-climbing method, which was applied to six different schemes for selecting the starting point. Unless the landscape of the objective function is sufficiently well known in advance, hill-climbing cannot normally be relied upon to deliver its global maximum. The intention was to learn whether the new recurrence relations used in connection with one of the six starting schemes would be consistently effective in doing so. Unfortunately, none of the six schemes passed the test. On the other hand, the investigation tended to shed still more light on the nature of peak creation.

In short, none of these attempts to globally optimize the individual fragments of f_1 could be sufficiently developed, although all of the attempts were indirectly useful.

Finally, rather than contend with the landscape of f_1 in its full generality, the research now took a different tack. The new approach was to collect examples of the global optimum of f_1 and then study the landscape in the immediate vicinity of that peak. With enough examples, it might be possible to learn by induction the rules which account for the global peak (versus the many lesser peaks), as well as identify those fragments of f_1 where it is likely to occur.

This rule induction, of course, requires that examples of the global optimum be available for study and, in general, such examples are not easy to come by. The thesis describes some useful tools for helping to locate the global optimum, but the application of those tools is largely ad hoc. Unless the number of model states is low and the length of the training sequence is small, it is difficult to achieve absolute certainty about a candidate optimum--although continued experience does guide one in forming a <u>reasonable</u> certainty.

As hoped, once a body of examples had been collected, it was possible to discern certain patterns among them. First, for the global maximum of f_1 to result from a fragment containing multiple terms is fairly unusual... in the great majority of known cases, the global maximum is given by optimizing a singleton. The explanation appears to be that singletons tend to be associated with the minimum number of dimensions, which is also conducive to a large objective value.

Second, the fragments which contribute the global maximum all seem to have unimodal landscapes. This is trivial to demonstrate when the fragment in question is a singleton, but seems to hold true also for multiple terms. Bearing in mind that each <u>term</u> of f_1 is a continuous-valued unimodal function ('bump'), and that each <u>fragment</u> of f_1 is a 'mixture' of such bumps (in the way that Continuous HMMs employ a mixture of Gaussian bumps), we find that the mixtures associated with the g.o. are particularly simple. It is as if only a single bump is required to optimally model a single training sequence.

Third, even when the globally optimum fragment is not a singleton, it contains a term which is a singleton... while the remaining terms are related to that one in a distinctive way. That is, the fragment invariably comprises one or more 'nearest neighbors' of the singleton term, where a nearest neighbor shares all the same dimensions but has exactly one additional dimension. Furthermore, when the fragment consists of multiple nearest neighbors of the singleton term, those must be nearest neighbors of each other also.

Assuming those three observations to hold for all cases, the thesis went on to construct an algorithm for producing the globally optimum HMM for a given training sequence. The algorithm works by identifying the single-term fragments of f_1 and then, for each of those singletons, finds their nearest neighbors and pieces them together to create larger and larger fragments, halting when a taller unimodal hump eventually fails to result. The process resembles crystal growing, where the singletons provide the 'seeds', and where extra dimensions are accumulated until the objective no longer shows an improvement.

The chief practical difficulty with the new algorithm is in identifying the single-term fragments of f_1. With N^{T-1} candidate terms in f_1, an exhaustive search is usually out of the question. Fortunately, it is possible to 'grow' the list of singletons on a symbol-by-symbol basis, by considering the successive elements of the training sequence in their turn. This implies a further advantage to the new algorithm in terms of the potential for building the HMM incrementally as each of the training symbols is received.

The key result which makes this possible is the realization that if term (i.e. state sequence) A 'eclipses' a second term (sequence) B, and if both end in the same state, then none of the descendants of term A which derive from subsequent training symbols can be singletons--so these can be pruned from the next generation of candidates.

Finally, the new algorithm was implemented as a 'C' program, and this was then subjected to various tests to try to confirm or refute the optimality of its outputs.

The tests consisted of independent computer searches of the objective landscape to try to locate the highest peak, which was then compared with the peak produced by the new HMM building algorithm. Different tests were necessary due to the inherent difficulties (e.g. excessive run-times) of conducting searches over a large space. As the test examples increased in difficulty, new strategies had to be devised. For example, according to one strategy, each relevant boundary plane of the objective function was identified, and a hill-climb was then initiated from the geometric center of that boundary, with the highest hill over all of the boundaries providing the results of that trial. In a later search strategy, the hill-climbs were initiated from a grid of equally-spaced starting locations limited to the interior of the parameter space. The intention was that, if the grid of starting locations was sufficiently fine, some of the climbs would start near enough to the boundaries to detect any end-point extrema. Later, as the test examples became even more difficult, that same grid of starting locations was retained, but the hill-climbs were replaced by a one-off function evaluation at each grid point.

Each trial explored the landscape unique to a given randomly-generated training sequence (ranging from 5 to 15 symbols in length) and model size (ranging from 3 to 7 states). Due to the finite number of those trials, and the inability of the computer searches to guarantee the global peak, it is still impossible to be conclusive about the effectiveness of the new algorithm. But, to date, no counter-example has yet been found which shows it failing to provide the global optimum model. On the other hand, a serious drawback of the algorithm is that it runs very slowly, typically several orders of magnitude slower than the Baum-Welch algorithm.

6.2. Contributions

The contributions of this research fall roughly into three categories, listed in increasing order of importance.

In the first category are some low-level innovations and insights, including :

(a) various tools for Matrix Differentiation and constraint expression (Appendix A)

(b) a generalized geometric progression formula (Section A.6.)

(c) a procedure for identifying the relevant fragments of f_1 (Section 3.4.)

(d) the theoretical connections between the three deductive solution methods (Section 4.2.)

(f) mathematical insights into the values of the constraint sensitivity factors (Section 4.3.)

(g) the automatic primal-to-dual transformation procedure (Section C.4.)

(h) insights into the reducibility of the dual G.P. equations (Section C.5.)

(i) the global effectiveness of the six search initiation schemes (Section D.5.)

In the second category are various insights into

the nature of the problem

the objective landscape

and the process of peak formation

as follows--

Nature of the Problem :

(a) the objective function $P(O)$ viewed as a generalization of $x_1^{r_1} x_2^{r_2} \cdots x_n^{r_n}$ (Section 2.3.3.)

(b) the individual terms of f_1 as generalized Beta random variables (Section 3.5.)

(c) at least as many relevant fragments as there are terms of f_1 (Section 3.6.)

(d) the different classes of fragments, and the natural hierarchy among the fragments (Sections 3.5. - 3.6.)

(e) sum of the A-contributions for f_1 equals 1 (Section 5.2.)

The Objective Landscape :

(a) the upper bound on the peaks within the interior of $P(O)$ (Sections 2.2. - 2.3.)

(b) in the case of a single training sequence the global optimum must fall on a boundary of $P(O)$ (Section 3.2.)

(c) the $f_i(A, B)$ are isomorphic and have essentially the same landscapes (Section 3.7.)

(d) at least as many peaks in f_1 as it has singletons (Section5.4.)

(e) a lower bound for the global maximum is easily established as that of the tallest singleton (Section 5.2.)

(f) the global maximum is nearly always equal to the tallest singleton (Section 5.3.1.)

(g) a simple mixture (unimodal hump) is adequate to model the global optimum for a single training sequence (Sections 5.3.2. - 5.3.3.)

Peak Formation :

(a) the link between low dimensionality and high objective value, which is reflected in

the large number of singleton solutions (Section 5.3.1.)

the nearest neighbor phonemonon (Section 5.3.3.)

(b) the necessity of geometrical closeness (Sections D.3., D.5., 5.3.3.)

(c) the conditions under which a fragment has an interior peak seem to be sufficient to ensure that it has only one interior peak (Sections D.5. and 5.3.3.)

Finally, in the third category of contributions are the NHB model-building algorithm itself, including the critical procedure for identifying the singletons of f_1 , and the body of test results.

6.3. Future Work

The goal of the project was to develop a procedure by which globally optimal Hidden Markov Models can be identified. On a very modest level, it seems as though the project has been at least partially successful in achieving that goal. It has produced some intriguing insights and an algorithm which appears to do the job, but which requires far more evaluation and refinement before it can be of practical use within speech recognition.

A principle weakness of the work is that, despite the attempted use of mathematics where possible, the finished product is still largely descriptive and far more qualitative than quantitative. To some extent, this is a reflection of the fact that the genesis and testing of the new HMM building algorithm represent only the final six months or so of the research period, while the bulk of the research effort is tied up in the other chapters (particularly those of the Appendix). Also, with few other recorded efforts in the literature to build upon, the work was naturally very exploratory.

In order to bring the research onto a more solid footing, far more analysis and testing is required, including as a minimum: the collection of still more global optimum examples, the use of the new algorithm on more complicated cases, as well as faster and more thorough searches of the parameter space by way of verification. Unfortunately, much of this demands more computing 'muscle' than is currently available to the author.

In the long run, however, there is an even greater need for the development of some theory capable of supporting a deductive proof of the algorithm's adequacy... since otherwise the cycle of

$$\begin{tikzcd}
\text{rule induction} \arrow[r] & \text{counter-examples} \arrow[r] & \text{refinement} \arrow[ll, bend left]
\end{tikzcd}$$

may never terminate.

Appendices

A. Matrix Derivatives

A.1. Introduction

Our ultimate goal is to find Markov matrices (A, B, π) which produce the global maximum for

$$P\,(\,o_1\,o_2 \ldots o_T)$$

the probability that the training sequence will occur. As Levinson, et al point out [28], that objective function can be expressed compactly in matrix notation as

$$P \;=\; \mathbf{1}'\; B_T\; A' \ldots A'\; B_2\; A'\; B_1\;\; \pi$$

where $\mathbf{1}$ is a column matrix of 1's

and B_t is a diagonal matrix derived from the appropriate column of B

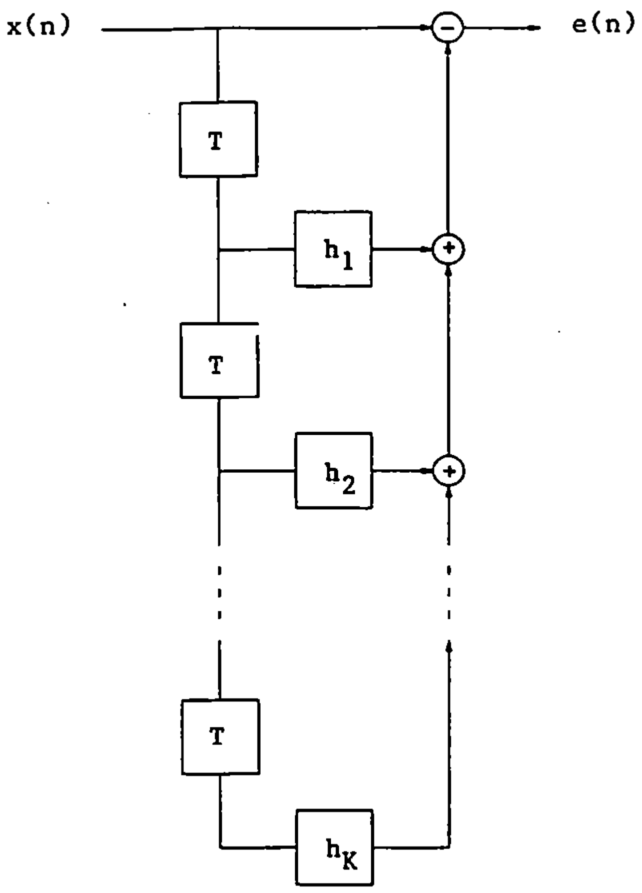
To maximize this according to the methods of calculus requires that we

perform a partial differentiation of the objective with respect to each individual matrix element

set each of those derivatives equal to zero

solve the massed system of equations simultaneously for the matrix elements.

In place of this daunting task, consider for now the FIR filter below



and the somewhat simpler job of computing the vector of filter coefficients which minimizes the error signal (as per Linear Predictive Coding, for example).

Here the objective (for minimization) is

$$f = \sum_{n=1}^{N} [e(n)]^2 = \sum_{n=1}^{N} \left[x(n) - \sum_{k=1}^{K} h_k x(n-k) \right]^2 \qquad \text{(A.1)}$$

This can also be expressed in matrix form, as follows :

Let $\quad H = [\, h_1 \; h_2 \; \dots \; h_K \,]'$

$E = [e(1) \; e(2) \; \dots \; e(N)]'$

$Y = [x(1) \; x(2) \; \dots \; x(N)]'$

and let X equal the $N \times K$ matrix where element (n, k) is $x(n - k)$ *

Then $\quad E = Y - XH$

and $\quad f = f(H) = E'E$

$= [\, Y' - H'X' \,] \, [\, Y - XH \,]$

$$= Y'Y - Y'XH - H'X'Y + H'X'XH \qquad \text{(A.2)}$$

* If $x(n - k) = 0$ for negative arguments, this corresponds to the Autocorrelation Method.
If, for $n < k$, $x(n - k)$ is available from the previous window, this leads to the Covariance Method [39].

Differentiating the scalar form (A.1) with respect to each of the unknown coefficients h_i

$$\frac{\partial f}{\partial h_i} = \sum_{n=1}^{N} \left\{ 2 \left[x(n) - \sum_{k=1}^{K} h_k\, x(n-k) \right] \times [-x(n-i)] \right\}$$

$$= -2 \sum_{n=1}^{N} \left\{ x(n-i) \left[x(n) - \sum_{k=1}^{K} h_k\, x(n-k) \right] \right\}$$

$$= 0 \qquad \text{at a stationary point}$$

from which

$$\sum_{k=1}^{K} h_k \left\{ \sum_{n=1}^{N} x(n-i)\, x(n-k) \right\} = \sum_{n=1}^{N} x(n-i)\, x(n)$$

$$i = 1..K \qquad \text{(A.3)}$$

Finally, it is possible to combine the complete set of results expressed by (A.3) into a single matrix equation which can be solved for H :

In matrix terms, the right hand side is

$$\sum_{n=1}^{N} x(n-i)\,x(n) \quad = \quad X_i'\,Y \qquad (i = 1..K)$$

where X_i is the i^{th} column of X

Similarly, the left hand side is

$$\sum_{k=1}^{K} h_k \left\{ \sum_{n=1}^{N} x(n-i)\,x(n-k) \right\} \quad = \quad \sum_{k=1}^{K} h_k \left\{ X_i' X_k \right\}$$

$$= \quad \sum_{k=1}^{K} \left\{ X_i' X_k \right\} h_k$$

$$= \quad X_i' \sum_{k=1}^{K} X_k\, h_k$$

$$= \quad X_i'\,X\,H \qquad (i = 1..K)$$

And therefore

$$X_i'\,Y \quad = \quad X_i'\,X\,H \qquad (i = 1..K)$$

Combined into a single equation, this becomes

$$X'\,Y \quad = \quad X'\,X\,H \qquad \text{(A.4)}$$

from which finally

$$H \quad = \quad (X'\,X)^{-1}\,X'\,Y$$

To summarize, we have

(a) taken an objective function (A.1) which is expressible in matrix form (A.2)

(b) differentiated it K times to produce a system of simultaneous equations (A.3), and

(c) bundled those equations into a matrix equation (A.4) for solution.

In the case of the FIR filter the amount of work required to do this was quite modest and manageable. However, in the case of maximizing $P(o_1\, o_2 \ldots o_T)$ this requires

N^2	differentiations	for	the	elements	of	A
$N \times M$	"	"	"	"	"	B
N	"	"	"	"	"	π

which then need to be assembled into 3 matrix equations for solution.

Clearly, it would be useful to be able to miss out the piecemeal differentiations and the bundling, and go directly from (A.2) to the derivative equation(s) (A.4), ready for solution. The following sections develop the rules by which this can be accomplished. The treatment of the subject is necessarily incomplete, but is sufficient for the work carried out in the remainder of the thesis.

A.2. Product Rule

We confine our attention to scalar-valued functions of a matrix (or matrices).

The result of differentiating a scalar-valued function with respect to each element of its argument, and then assembling those derivatives into a new matrix conforming to the argument, is one definition of a matrix derivative (for others see [40,41]) :

$$\frac{df}{dX} = \begin{bmatrix} \frac{\partial f}{\partial x_{11}} & \cdots & \frac{\partial f}{\partial x_{1N}} \\ \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial x_{N1}} & \cdots & \frac{\partial f}{\partial x_{NN}} \end{bmatrix}$$

Let A represent an $N \times N$ matrix. As a first example of a scalar function of A, consider the sum of its elements :

$$f(A) = \mathbf{1}' A \, \mathbf{1}$$

where $\mathbf{1}$ is the $N \times 1$ column vector of 1's

For the simple case of $N = 2$

$$\mathbf{1}' A \, \mathbf{1} = a_{11} + a_{12} + a_{21} + a_{22}$$

Then $$\frac{\partial f}{\partial a_{ij}} = 1 \qquad (i = 1..N), (j = 1..N)$$

Finally, bundling these results into a matrix produces

$$\frac{df}{dA} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \mathbf{1}\,\mathbf{1}'$$

As a second example, consider

$$f(A) = \mathbf{1}' A^2 \mathbf{1}$$

Then $$f(A) = \mathbf{1}' A \; A \, \mathbf{1}$$

$$= \mathbf{1}' [\, A_{C1} \;\; A_{C2} \,] \begin{bmatrix} A_{R1} \\ A_{R2} \end{bmatrix} \mathbf{1}$$

$$= [\, \mathbf{1}' A_{C1} \;\; \mathbf{1}' A_{C2} \,] \begin{bmatrix} A_{R1}\,\mathbf{1} \\ A_{R2}\,\mathbf{1} \end{bmatrix}$$

$$= \mathbf{1}' A_{C1} A_{R1} \mathbf{1} + \mathbf{1}' A_{C2} A_{R2} \mathbf{1}$$

where A_{Ci} = the i^{th} column of matrix A , etc.

And for general N

$$f(A) = \mathbf{1}' A_{C1} A_{R1} \mathbf{1} + \mathbf{1}' A_{C2} A_{R2} \mathbf{1} + .. + \mathbf{1}' A_{CN} A_{RN} \mathbf{1}$$

Now

$$\frac{\partial f}{\partial a_{ij}} = \mathbf{1}' A_{Ci} + A_{Rj} \mathbf{1}$$

since a_{ij} appears exactly once in exactly one row of A
and " " " " " column of A

After bundling,

$$\frac{df}{dA} = \begin{bmatrix} \mathbf{1}'A_{C1} & \mathbf{1}'A_{C1} \\ \mathbf{1}'A_{C2} & \mathbf{1}'A_{C2} \end{bmatrix} + \begin{bmatrix} A_{R1}\mathbf{1} & A_{R2}\mathbf{1} \\ A_{R1}\mathbf{1} & A_{R2}\mathbf{1} \end{bmatrix}$$

The first matrix can be written as the outer product

$$\begin{bmatrix} \mathbf{1}'A_{C1} & \mathbf{1}'A_{C1} \\ \mathbf{1}'A_{C2} & \mathbf{1}'A_{C2} \end{bmatrix} = \begin{bmatrix} \mathbf{1}'A_{C1} \\ \mathbf{1}'A_{C2} \end{bmatrix} \mathbf{1}'$$

$$= [\, \mathbf{1}'A_{C1} \;\; \mathbf{1}'A_{C2} \,]' \; \mathbf{1}'$$

$$= (\mathbf{1}'A)' \; \mathbf{1}'$$

$$= A' \mathbf{1} \, \mathbf{1}'$$

The second matrix can be written as the outer product

$$\begin{bmatrix} A_{R1}\mathbf{1} & A_{R2}\mathbf{1} \\ A_{R1}\mathbf{1} & A_{R2}\mathbf{1} \end{bmatrix} = \mathbf{1} \, [\, A_{R1}\mathbf{1} \;\; A_{R2}\mathbf{1} \,]$$

$$= \mathbf{1} \begin{bmatrix} A_{R1}\mathbf{1} \\ A_{R2}\mathbf{1} \end{bmatrix}'$$

$$= \mathbf{1} \, (A \, \mathbf{1})'$$

$$= \mathbf{1} \, \mathbf{1}' A'$$

Finally

$$\frac{df}{dA} = A' \mathbf{1} \, \mathbf{1}' + \mathbf{1} \, \mathbf{1}' A'$$

By similar methods, it can be shown that

for $$f(A) = \mathbf{1}' A^3 \mathbf{1}$$

then $$\frac{df}{dA} = \mathbf{1}\,\mathbf{1}' A' A' + A' \mathbf{1}\,\mathbf{1}' A' + A' A' \mathbf{1}\,\mathbf{1}'$$

and likewise

for $$f(A) = \mathbf{1}' A^4 \mathbf{1}$$

then $$\frac{df}{dA} = \mathbf{1}\,\mathbf{1}' (A')^3 + A' \mathbf{1}\,\mathbf{1}' (A')^2 + (A')^2 \mathbf{1}\,\mathbf{1}' A' + (A')^3 \mathbf{1}\,\mathbf{1}'$$

For a slightly different problem, consider $f(A) = \mathbf{1}' A\ A' \mathbf{1}$

Then $$f(A) = \mathbf{1}' A\ A' \mathbf{1}$$

$$= \mathbf{1}' [A_{C1}\ A_{C2}] \begin{bmatrix} A'_{C1} \\ A'_{C2} \end{bmatrix} \mathbf{1}$$

$$= [\,\mathbf{1}' A_{C1}\ \ \mathbf{1}' A_{C2}\,] \begin{bmatrix} A'_{C1}\,\mathbf{1} \\ A'_{C2}\,\mathbf{1} \end{bmatrix}$$

$$= \mathbf{1}' A_{C1} A_{C1}' \mathbf{1} + \mathbf{1}' A_{C2} A_{C2}' \mathbf{1}$$

from which

$$\frac{\partial f}{\partial a_{ij}} = 1' A_{C_j} + A_{C_j}' 1 = 2 A_{C_j}' 1$$

and

$$\frac{df}{dA} = 2 \begin{bmatrix} A_{C1}' 1 & A_{C2}' 1 \\ A_{C1}' 1 & A_{C2}' 1 \end{bmatrix}$$

$$= 2\ 1 [A_{C1}' 1 \quad A_{C2}' 1]$$

$$= 2\ 1 \begin{bmatrix} A_{C1}' 1 \\ A_{C2}' 1 \end{bmatrix}'$$

$$= 2\ 1\ (A' 1)'$$

$$= 2\ 1\ 1' A$$

Finally, from Section A.1. the function

$$f(H) = Y'Y - Y'XH - H'X'Y + H'X'XH$$

has the matrix derivative

$$\frac{df}{dA} = -2X'Y + 2X'XH$$

What rule, if any, can account for these various examples?

Where the function includes more than one term (as immediately above) there is an obvious Addition rule, analogous to the rule of scalar calculus, whereby the derivative of a sum equals the sum of the derivatives.

For differentiation of a single term, the following empirically derived rule does the job :

> Each occurrence of the independent (matrix) variable within the term will give rise to a separate term in the derivative expression, as follows--
>
> Consider the i^{th} occurrence of that variable, X, along with the factors which appear to the left of it (call them L collectively) and the factors to the right (call them R)
>
> (Note than in some cases either L or R may be the scalar 1)
>
> If this occurrence of X is being transposed, create the i^{th} derivative term by simply swapping the order of L & R, and omitting the X
>
> $$i^{th} \text{ term} \quad = \quad R\,L$$
>
> If, on the other hand, the X is not being transposed, keep the order of L & R the same but transpose them (again omitting the X)
>
> $$i^{th} \text{ term} \quad = \quad L'\;R'$$

Thus, exactly one operation is performed upon the L, R pair :

either their order is reversed, or they are transposed

And exactly one category of factor undergoes transposition :

if the X appears transposed, then the (L, R) isn't
if X isn't transposed , then the (L, R) is

X	L, R	
transposed	order reversed	transposed
yes	yes	no
no	no	yes

Let's see how this rule works in connection with an earlier example. Take

$$f(A) = \mathbf{1}'A\,A'\mathbf{1}$$

The first (left-most) occurrence of A has

$$L = \mathbf{1}' \qquad \text{and} \qquad R = A'\mathbf{1}$$

Since this A is not transposed, the L & R <u>will</u> be--while their order remains unchanged. The appropriate contribution to the derivative expression is then

$$L'\,R' = (\mathbf{1}')'\,(A'\mathbf{1})' \qquad = \mathbf{1}\,\mathbf{1}'A$$

The second occurrence of A has

$$L = \mathbf{1}'A \qquad \text{and} \qquad R = \mathbf{1}$$

Since the A is transposed in this case, the L & R won't be--however, their order will be reversed. The required contribution is

$$R\,L = \mathbf{1}\,\mathbf{1}'A$$

In this way,

$$\frac{df}{dA} = 2\;\mathbf{1}\,\mathbf{1}'A$$

as deduced earlier.

Note that the rule applies, regardless of the dimensions of the independent variable (i.e., not necessarily square), or its position within the expression.

For example, consider

$$f = a_1 x_1 + a_2 x_2 + a_3 x_3 = [\, a_1 \;\; a_2 \;\; a_3 \,] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = A' X$$

Differentiating with respect to X, we note that

$$L = A' \qquad \text{and} \qquad R \;\text{ is the scalar }\; 1$$

Since X is not transposed, the rule gives

$$\frac{df}{dX} = L' R' = (A')' \, 1 = A$$

which is clearly correct.

A.3. Determinant

A second scalar-valued function of a (square) matrix is its determinant, $f = \mathrm{Det}(A)$.

This value can be computed by forming the inner product of any row (or column) of A with the cofactors of those elements :

$$\begin{aligned}\mathrm{Det}(A) &= a_{i1}\,\mathrm{Cof}(a_{i1}) + a_{i2}\,\mathrm{Cof}(a_{i2}) + .. + a_{iN}\,\mathrm{Cof}(a_{iN}) \\ &= a_{1j}\,\mathrm{Cof}(a_{1j}) + a_{2j}\,\mathrm{Cof}(a_{2j}) + .. + a_{Nj}\,\mathrm{Cof}(a_{Nj})\end{aligned}$$

The cofactor of element a_{ij} is $(-1)^{i+j}$ times the minor of a_{ij} , computed as the determinant of A after row i and column j have been removed. Therefore, the cofactor of a_{ij} does not include any element from row i or column j in its calculation.

Consequently

$$\begin{aligned}\frac{\partial}{\partial a_{ij}}[\,\mathrm{Det}(A)\,] &= \frac{\partial}{\partial a_{ij}}[a_{i1}\,\mathrm{Cof}(a_{i1}) + .. + a_{ij}\,\mathrm{Cof}(a_{ij}) + .. + a_{iN}\,\mathrm{Cof}(a_{iN})] \\ &= \mathrm{Cof}(a_{ij})\end{aligned}$$

since none of the cofactors present in that sum use a_{ij} .

Assembling these derivatives into a matrix,

$$\frac{df}{dA} = \begin{bmatrix} \mathrm{Cof}(a_{11}) & \cdots & \mathrm{Cof}(a_{1N}) \\ \vdots & \ddots & \vdots \\ \mathrm{Cof}(a_{N1}) & \cdots & \mathrm{Cof}(a_{NN}) \end{bmatrix}$$

$$= [\,\mathrm{Adj}\,(A)\,]' = \mathrm{Adj}\,(A')$$

where the transpose of the cofactor matrix of A is known as the adjoint of A .

Finally, from Matrix Algebra

$$A^{-1} = \frac{1}{\mathrm{Det}(A)}\,\mathrm{Adj}\,(A)$$

Therefore

$$\frac{d}{dA}[\,\mathrm{Det}(A)\,] = \mathrm{Det}(A)\,(A^{-1})'$$

Other variations of this rule from the literature [42,43] include

$$\frac{d}{dA}[\,\{\,\mathrm{Det}(A)\,\}^r\,] = r\,\{\,\mathrm{Det}(A)\,\}^r\,(A^{-1})'$$

and

$$\frac{d}{dA}[\,\ln\{\,\mathrm{Det}(A)\,\}\,] = (A^{-1})'$$

Apart from the sheer elegance of these results, determinants are also helpful in formulating a Markov constraint upon A for use within the method of Lagrange Multipliers :

The method requires that we include a term in the Lagrangean function whose value equals the scalar 0 when A is a Markov matrix.

Consider $\text{Det}(\lambda I - A) = 0$, which is the characteristic equation of the matrix A , and whose roots are its eigenvalues.

When A is a Markov matrix, the one definitely known eigenvalue of A is $\lambda = 1$.

Therefore, $\text{Det}(I - A)$ provides the required term, and it merely remains to be able to differentiate it with respect to A ...

By analogy with the Chain Rule of scalar calculus, one might expect that

$$\begin{aligned} \frac{d}{dA}[\,\text{Det}(I - A)\,] &= \frac{d}{d(I-A)}[\,\text{Det}(I - A)\,] \times \frac{d}{dA}(I - A) \\ &= [\,\text{Adj}\,(I - A)\,]' \times -1 \end{aligned}$$

Application of our derivative definition confirms this to be correct.

Having differentiated Det(I - A) in the Lagrangean to produce

$$- [\ \mathrm{Adj}\ (\mathrm{I} - A)\ ]'$$

it may then become useful to eliminate this term from the analysis.

To do so, post-multiply through by (I - A) ' :

Using the result

$$X^{-1} \quad = \quad \frac{1}{\mathrm{Det}(X)}\ \mathrm{Adj}\ (X)$$

or

$$\mathrm{Det}\ (X)\ \mathrm{I} \quad = \quad X\ \mathrm{Adj}\ (X)$$

then

$$\begin{aligned} [\ \mathrm{Adj}\ (\mathrm{I} - A)\ ]'\ (\mathrm{I} - A)' \quad &= \quad [\ (\mathrm{I} - A)\ \mathrm{Adj}\ (\mathrm{I} - A)\ ]' \\ &= \quad [\ \mathrm{Det}\ (\mathrm{I} - A)\ \ \mathrm{I}\ ]' \\ &= \quad \mathbf{0} \end{aligned}$$

since Det (I - A) = 0 when A is a Markov matrix.

Likewise, it may be useful to realize that the columns of $[\text{Adj}\,(I - A)]'$ are identical, so that summing downwards produces

$$\mathbf{1}'\,[\text{Adj}\,(I - A)]' = \text{constant} \times \mathbf{1}'$$

That the columns of $[\text{Adj}\,(I - A)]'$ are identical can be demonstrated as follows :

Since, as above

$$[\text{Adj}\,(I - A)]'\,(I - A)' = \mathbf{0}$$

then

$$[\text{Adj}\,(I - A)]'\,I' = [\text{Adj}\,(I - A)]'A'$$

or

$$A\,\text{Adj}\,(I - A) = \text{Adj}\,(I - A)$$

This shows that the matrix A preserves $\text{Adj}\,(I - A)$ unaltered. This can only happen when each column of $\text{Adj}\,(I - A)$ is a multiple of the eigenvector associated with the eigenvalue 1 --which is $\mathbf{1}$.

Transposing, each <u>row</u> of $[\text{Adj}\,(I - A)]'$ is now a multiple of $\mathbf{1}'$, implying that the columns are identical. *

* Alternatively, the columns of $[\text{Adj}\,(I - A)]'$ must be identical in order to remain consistent with the other possible way of handling the Markov constraints--which is to use a separate Lagrange multiplier for each row of A , differentiate, and bundle the results into a matrix.

Finally, for purposes of the Langrangean, one needs to know how to handle non-square Markov matrices (such as the $N \times M$ B-matrix), for which a determinant is not normally defined.

The answer is to augment B by $M - N$ additional rows (converting it into a square matrix), where the extra rows are duplicates of the column averages of B, computed as

$$\frac{1}{N} \underset{1 \times N}{\mathbf{1}'} \underset{N \times M}{B}$$

Alternatively, this can be achieved by pre-multiplying B with the $M \times N$ matrix J whose first N rows are the $N \times N$ identity matrix, and whose final $M - N$ rows are filled with the scalar value $\frac{1}{N}$

$$J = \frac{1}{N} \begin{bmatrix} N\mathrm{I} \\ \mathbf{1}' \\ \vdots \\ \mathbf{1}' \end{bmatrix}$$

Then the rows of this new square matrix JB sum to 1 as required :

$$JB\mathbf{1} = \frac{1}{N} \begin{bmatrix} N\mathrm{I} \\ \mathbf{1}' \\ \vdots \\ \mathbf{1}' \end{bmatrix} B\mathbf{1}$$

$$= \frac{1}{N} \begin{bmatrix} N\mathrm{I}B\mathbf{1} \\ \mathbf{1}'B\mathbf{1} \\ \vdots \\ \mathbf{1}'B\mathbf{1} \end{bmatrix}$$

$$= \frac{1}{N}\begin{bmatrix} N B \mathbf{1} \\ \mathbf{1}' B \mathbf{1} \\ \vdots \\ \mathbf{1}' B \mathbf{1} \end{bmatrix}$$

$$= \frac{1}{N}\begin{bmatrix} N \mathbf{1} \\ N \\ \vdots \\ N \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

$$= \underset{\mathrm{M \times 1}}{\mathbf{1}}$$

Invoking our definition for matrix derivative, it can be shown that

$$\frac{d}{dB}[\,\mathrm{Det}\,(\mathrm{I} - JB)\,] \quad = \quad [\,\mathrm{Adj}\,(\mathrm{I} - JB) \times -J\,]'$$

$$= \quad -J'\,[\,\mathrm{Adj}\,(\mathrm{I} - JB)\,]'$$

This is further evidence of a Chain Rule within matrix differentiation. Note also how the order of J' in this expression adheres to the rule from the "Product Rule" discussion :

Since B doesn't appear transposed in Det (I - JB) then L (equal to $-J$) and R (equal to the scalar 1) appear in their same order (i.e., L before the main result, R after) but transposed. On the other hand,

$$\frac{d}{d\pi}[\,\mathrm{Det}\,(\mathrm{I} - \mathbf{1}\,\pi')\,] \quad = \quad [\,\mathrm{Adj}\,(\mathrm{I} - \mathbf{1}\,\pi')\,] \times -\mathbf{1}$$

A.4. Trace

Another useful scalar-valued function of a (square) matrix is the trace, $f = \mathrm{Tr}(A)$, defined as the sum of the diagonal elements.

Clearly,

$$\frac{\partial f}{\partial a_{ij}} = \begin{cases} 1 & \text{if } a_{ij} \text{ is on the diagonal (i.e. } i=j\text{)} \\ 0 & \text{otherwise} \end{cases}$$

Bundling the partial derivatives into a matrix as per the definition gives

$$\frac{df}{dA} = \text{the identity matrix I}$$

Next, let B equal a second $N \times N$ matrix.

Then

$$A\,B = \begin{bmatrix} A_{\mathrm{R}1}B_{\mathrm{C}1} & \cdots & A_{\mathrm{R}1}B_{\mathrm{C}N} \\ \vdots & \ddots & \vdots \\ A_{\mathrm{R}N}B_{\mathrm{C}1} & \cdots & A_{\mathrm{C}N}B_{\mathrm{C}N} \end{bmatrix}$$

Summing down the diagonal

$$\mathrm{Tr}(A\,B) = \sum_{i=1}^{N} A_{\mathrm{R}i}B_{\mathrm{C}i}$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{N} a_{ij}\, b_{ji}$$

and

$$\frac{\partial}{\partial a_{ij}}[\,\mathrm{Tr}(A\,B)\,] \quad = \quad b_{ji}$$

so that

$$\frac{d}{dA}[\,\mathrm{Tr}(A\,B)\,] \quad = \quad B\,'$$

Likewise

$$\frac{d}{dB}[\,\mathrm{Tr}(A\,B)\,] \quad = \quad A\,'$$

Furthermore, building on these last two results

$$\frac{d}{dA}[\,\mathrm{Tr}(A\,B\,C)\,] \quad = \quad (B\,C)\,' \quad = \quad C\,'B\,'$$

$$\frac{d}{dC}[\,\mathrm{Tr}(A\,B\,C)\,] \quad = \quad (A\,B)\,' \quad = \quad B\,'A\,'$$

Also, it can be shown that

$$\frac{d}{dB}[\,\mathrm{Tr}(A\,B\,C)\,] \qquad\qquad = \quad A\,'C\,'$$

Finally,

$$\begin{aligned}\frac{d}{dB}[\,\mathrm{Tr}(A\,B\,'C)\,] \quad &= \quad \frac{d}{dB}[\,\mathrm{Tr}(\,\{A\,B\,'\,C\}'\,)\,] \\ &= \quad \frac{d}{dB}[\,\mathrm{Tr}(C\,'B\,A\,')\,] \\ &= \quad (C\,')\,'(A\,')\,' \\ &= \quad C\,A\end{aligned}$$

The point to note is how all of these examples uphold the rules from the "Product Rule" discussion ("same order, but transposed" or "reverse order, not transposed"--depending upon whether or not the independent variable appears transposed within the expression).

As a final useful result, recall that

$$\frac{d}{dA}[\,\mathrm{Tr}(A)\,] = \mathrm{I} = \mathrm{I}'$$

In a similar way,

$$\frac{d}{dA}[\,\mathrm{Tr}(A^2)\,] = \frac{d}{dA}[\,\mathrm{Tr}(A\,A)\,] = A' + A' = 2\,A'$$

$$\frac{d}{dA}[\,\mathrm{Tr}(A^3)\,] = \frac{d}{dA}[\,\mathrm{Tr}(A\,A\,A)\,] = A'A' + A'A' + A'A' = 3(A^2)'$$

and, in general [44,45]

$$\frac{d}{dA}[\,\mathrm{Tr}(A^r)\,] = r\,(A^{r-1})'$$

providing the equivalent of the Power Rule from scalar calculus.

Where Trace can be especially helpful is in converting a statement given in matrix form into a scalar, ready for differentiation. In particular, Trace is good for expressing element-wise constraints upon a matrix, for inclusion in a Lagrangean function.

For example, to impose non-negativity constraints upon the elements of A , define

$$\text{Abs}(A) = \begin{bmatrix} a_{11}\,\text{sign}(a_{11}) & \cdots & a_{1N}\,\text{sign}(a_{1N}) \\ \vdots & \ddots & \vdots \\ a_{N1}\,\text{sign}(a_{N1}) & \cdots & a_{NN}\,\text{sign}(a_{NN}) \end{bmatrix}$$

which replaces each element of A by its absolute value.

Then the elements of A are non-negative provided

$$A = \text{Abs}(A)$$

and a suitable scalar term to reflect this statement within the Lagrangean is

$$\text{Tr}(M'\,[A - \text{Abs}(A)])$$

where M is a matrix of Lagrange multipliers.

Differentiation with respect to M leads to

$$A = \text{Abs}(A)$$

as required, while differentiation with respect to A produces

$$\text{Sign}(A) = \mathbf{1}\,\mathbf{1}'$$

Notice that M and A don't have to be square, provided that they share the same dimensions.

Note also that $\mathrm{Tr}(M'A)$ is equivalent to

$$\mathbf{1}'(M \odot A)\mathbf{1}$$

where $\odot$ denotes Hadamard (i.e. element-wise) multiplication.

As a second example of the use of Trace , imagine that one or more elements of A must equal zero (e.g. perhaps A is supposed to be upper-triangular).

Define the matrix H to have the dimensions of A , and to consist of 1's to mark the positions where a_{ij} is required to equal 0 , and 0's elsewhere :

$$h_{ij} = \begin{cases} 1, & a_{ij} = 0 \\ 0, & a_{ij} = 1 \end{cases}$$

Then a suitable term in the Lagrangean consists of

$$\mathrm{Tr}[M'(H \odot A)]$$

Note that

$$\frac{d}{dM}\{\mathrm{Tr}[M'(H \odot A)]\} = H \odot A$$

and

$$\frac{d}{dA}\{\mathrm{Tr}[M'(H \odot A)]\} = H \odot M$$

A.5. Diagonal

At the heart of all this effort is the objective function

$$P \quad = \quad \mathbf{1}' \; B_T \; A' \ldots B_2 \, A' \; B_1 \; \pi$$

where B_t is a diagonal matrix made from the k^{th} column of B , assuming codebook symbol k was the one observed at time t ($o_t = v_k$) .

Let D_t represent the $N \times 1$ column vector consisting of 0's , except for a single 1 at position k .

Then B_t can be written in matrix form as

$$\text{Diag}(BD_t)$$

where 'Diag' embeds its argument along the major diagonal of a matrix of 0's , and

$$P = \mathbf{1}' \; \text{Diag}(BD_T) \; A' \ldots \; \text{Diag}(BD_2) \; A' \; \text{Diag}(BD_1) \; \pi$$

The differentiation of P with respect to both π and A can be handled as per the discussion under "Product Rule" .

In addition, we need a rule to cope with B , buried as it is within the Diag operator. The required rule, derived empirically, goes as follows :

As before, each occurrence of B within the objective will give rise to a term in the derivative expression.

Focusing on a particular occurrence of B, designate everything to the left of the Diag as X', and everything to the right as Y.

Then

$$\frac{d}{dB}[X' \operatorname{Diag}(BD)\, Y] = \operatorname{Diag}(X)\; YD'$$

Finally, note for convenience that

$$1' \operatorname{Diag}(X) = X' \quad \text{and} \quad \operatorname{Diag}(X)\, 1 = X$$

$$1' \operatorname{Diag}(X') = X' \quad \text{and} \quad \operatorname{Diag}(X')\, 1 = X$$

A.6. Geometric Progression

This section generalizes the scalar formula for the sum of a geometric progression. Strictly speaking, this effort has little to do with Matrix Differentiation, but is included here because of a connection with other results in this chapter.

Assume that A is an $N \times N$ Markov matrix. Also, assume that A is diagonalizable *

$$A = Q D Q^{-1}$$

As a Markov matrix,

$$A\mathbf{1} = \mathbf{1}$$

which indicates that the equation

$$AX = \lambda X$$

is satisfied for the eigenvector $X = \mathbf{1}$ and eigenvalue $\lambda = 1$.

Therefore, A has at least one eigenvalue equal to 1 . Furthermore, from the literature the magnitudes of the remaining eigenvalues are all < 1 . (A has a 'spectral radius' of 1 [31,32])

* If the eigenvalues of A are distinct, then D is the diagonal matrix consisting of the eigenvalues of A , and the columns of Q are the respective eigenvectors [46,47]

Split D into the two diagonal matrices

$$D = D_1 + D_2$$

where D_1 contains the 1 eigenvalue(s) (and 0's elsewhere)
and D_2 contains the eigenvalues $\neq$ 1 (and 0's elsewhere)

$$D_1 = \begin{bmatrix} 1 & & \\ & 0 & \\ & & 0 \end{bmatrix} \qquad D_2 = \begin{bmatrix} 0 & & \\ & b & \\ & & c \end{bmatrix}$$

Clearly, $D_1 \times D_2$ is the matrix of 0's .

Then

$$\begin{aligned} A^i &= (QDQ^{-1})^i \\ &= QDQ^{-1} \times QDQ^{-1} \times QDQ^{-1} \times \ldots \times QDQ^{-1} \\ &= QD \quad \underbrace{\phantom{Q^{-1}\times Q}}_{\mathrm{I}} \quad D \quad \underbrace{\phantom{Q^{-1}\times Q}}_{\mathrm{I}} \quad D \quad \ldots \quad DQ^{-1} \\ &= QD^iQ^{-1} \\ &= Q(D_1+D_2)^iQ^{-1} \\ &= Q(D_1^i+D_2^i)Q^{-1} \end{aligned}$$

since the mixed terms must equal **0** as above.

Forming the sum,

$$\sum_{i=0}^{k-1} A^i = \sum_{i=0}^{k-1} Q\left(D_1^i + D_2^i\right)Q^{-1}$$

$$= Q\left\{\sum_{i=0}^{k-1} D_1^i + \sum_{i=0}^{k-1} D_2^i\right\} Q^{-1}$$

$$= Q\left\{\begin{bmatrix} k & & \\ & 0 & \\ & & 0 \end{bmatrix} + \begin{bmatrix} 0 & & \\ & \frac{1-b^k}{1-b} & \\ & & \frac{1-c^k}{1-c} \end{bmatrix}\right\} Q^{-1}$$

using the geometric progression rule for scalars.

Finally, since

$$D^k = \begin{bmatrix} 1 & & \\ & b^k & \\ & & c^k \end{bmatrix}$$

$$\mathrm{I} - D^k = \begin{bmatrix} 0 & & \\ & 1-b^k & \\ & & 1-c^k \end{bmatrix}$$

$$\mathrm{I} - D_2 = \begin{bmatrix} 1 & & \\ & 1-b & \\ & & 1-c \end{bmatrix}$$

$$(\mathrm{I} - D_2)^{-1} = \begin{bmatrix} 1 & & \\ & \frac{1}{1-b} & \\ & & \frac{1}{1-c} \end{bmatrix}$$

the sum can be written

$$\sum_{i=0}^{k-1} A^i = Q\{kD_1 + (\mathrm{I}-D_2)^{-1}(\mathrm{I}-D^k)\}Q^{-1}$$

$$= kQD_1Q^{-1} + (\mathrm{I}-QD_2Q^{-1})^{-1}(\mathrm{I}-A^k)$$

Notice how this generalizes the scalar case :

If all of the eigenvalues equal 1 , then $D = D_1 = \mathrm{I}$ and the second term disappears, leaving

$$kQD_1Q^{-1} = kQDQ^{-1} = kA$$

corresponding to the scalar result $\displaystyle\sum_{i=0}^{k-1} a^i = ka$ for $a = 1$.

If none of the eigenvalues equal 1 , then $D = D_2$ and the first term disappears, leaving

$$(\mathrm{I}-QD_2Q^{-1})^{-1}(\mathrm{I}-A^k) = (\mathrm{I}-QDQ^{-1})^{-1}(\mathrm{I}-A^k)$$

$$= (\mathrm{I}-A)^{-1}(\mathrm{I}-A^k)$$

corresponding to the scalar result $\displaystyle\sum_{i=0}^{k-1} a^i = \frac{1-a^k}{1-a}$ for $|a| < 1$.

Finally, a connection apears to exist with the earlier work on determinants :

Let X and Y be right and left eigenvectors of A for the eigenvalue $\lambda = 1$

i.e. $\quad AX = X \quad$ and $\quad Y'A = Y' \quad (\text{or } A'Y = Y)$

(We already know that **1** is a solution for X)

It is known from the literature [48] that

$$\text{Adj}(\text{I} - A) = \text{constant} \times \text{Det}(\text{I} - D_2)\,\frac{XY'}{Y'X}$$

where the constant equals 1 when 1 is a simple (unrepeated) eigenvalue of A .

In addition, there is evidence that *

$$\text{Adj}(\text{I} - A) = \text{Det}(\text{I} - D_2)\; Q D_1 Q^{-1}$$

* What the evidence suggests :

$$\text{Det}(\text{I} - D_2) = \mathbf{1}'Y = Y'\mathbf{1}$$

$$Q D_1 Q^{-1} = \frac{\mathbf{1}\,Y'}{\mathbf{1}'\,Y}$$

$$\text{Adj}(\text{I} - A) = \mathbf{1}\,Y'$$

$$[\,\text{Adj}(\text{I} - A)\,]' = Y\,\mathbf{1}'$$

A.7. Miscellaneous Results

For an intriguing result from the literature [49] :

let A be a real symmetric $N \times N$ matrix

λ be a simple eigenvalue of A

X be the normalized eigenvector for λ

then
$$\frac{d\lambda}{dA} = X X'$$

Also from the literature [50],

let A be an $N \times N$ matrix

X, Y be $N \times 1$ column vectors

then
$$\frac{d}{dA}[X' A^{-1} Y] = -[A^{-1} Y X' A^{-1}]'$$

A.8. An Example

One important application of Matrix Differentiation appears within Section 2.3. of the thesis, where it is used to help demonstrate that

the maximum of the function $f(x_1, x_2, .., x_n) = x_1^{r_1} x_2^{r_2} \dots x_n^{r_n}$

subject to the constraint $\sum_{k=1}^{n} x_k = 1$

occurs at $x_k = \frac{r_k}{\sum_{k=1}^{n} r_k}$

As a second instructive example, we apply it to the following optimization problem taken from *Operations Research* (3rd Ed), Hamdy A. Taha, p. 748 [51] :

minimize $z = x_1^2 + x_2^2 + x_3^2$

subject to $4x_1 + x_2^2 + 2x_3 - 14 = 0$

Let

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$Y = \begin{bmatrix} 4 \\ x_2 \\ 2 \end{bmatrix}$$

Then the objective function can be written

$$f(X, \lambda) = X'X - \lambda(X'Y - 14)$$

with the scalar Lagrange multiplier λ .

Note that, unlike before, Y is neither a simple variation of X nor a scalar (i.e. independent of X) . The differentiation of $X'Y$ requires

(a) the derivative of a vector-valued function of X, as well as

(b) a more general form of Product Rule

First, to differentiate one vector with respect to a second vector, the dependent vector must be arranged at right angles (transposed) with respect to the independent vector. For example, one can differentiate Y' with respect to X , or Y with respect to X' .

To differentiate Y' with respect to X,

> arrange Y' across the top and X down one side
>
> then differentiate each element at the top (as a scalar function) with respect to the elements down the side
>
> and fill in the columns accordingly :

$$\begin{array}{cc} & [\,4 \;\; x_2 \;\; 2\,] \\ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Alternatively, to differentiate Y with respect to X',

> arrange X' across the top and Y down one side
>
> differentiate the elements of Y as scalar functions
>
> and fill in the rows accordingly :

$$\begin{array}{cc} & [\,x_1 \;\; x_2 \;\; x_3\,] \\ \begin{bmatrix} 4 \\ x_2 \\ 2 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Clearly,

$$\frac{dY'}{dX} = \left(\frac{dY}{dX'}\right)'$$

Second, to achieve this 'right angle' orientation, it may become necessary to rearrange the term being differentiated. This means that if, while working through the list of factors, the factor currently being processed fails to display the correct orientation, simply transpose the term temporarily. In practice, this causes no problem since the term as a whole is scalar-valued and, hence, unaffected by transpose, i.e.

$$X'Y \qquad \text{is equal to} \qquad Y'X$$

This requirement appears related to the earlier version of the Product Rule (Section A.2.) whereby if the current variable is already transposed, the left and right factors aren't, and vice versa.

In what now appears very much like the Product Rule for scalar calculus, we find (differentiating with respect to X)

$$\begin{aligned}
\frac{d}{dX}[X'Y] &= \frac{dX'}{dX}Y + \frac{dY'}{dX}X \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ x_2 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \\
&= \begin{bmatrix} 4 \\ x_2 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ x_2 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 4 \\ 2x_2 \\ 2 \end{bmatrix}
\end{aligned}$$

which is clearly correct.

Armed with this new procedure, we address the problem

$$\text{minimize} \qquad f = X'X - \lambda(X'Y - 14)$$

Differentiating

<u>w.r.t. X</u>

$$\frac{df}{dX} = 2X - \lambda \begin{bmatrix} 4 \\ 2x_2 \\ 2 \end{bmatrix} = 2X - \lambda Z = 0 \qquad \text{(A.5)}$$

<u>w.r.t. λ</u>

$$\frac{df}{d\lambda} = -(Y'X - 14) = 0 \qquad \text{(A.6)}$$

The usual strategy is to solve for λ and then substitute to find X:

Pre-multiply (A.5) by Y'

$$2Y'X = \lambda Y'Z$$

Applying (A.6)

$$2 \times 14 = \lambda \; [4 \;\; x_2 \;\; 2] \begin{bmatrix} 4 \\ 2x_2 \\ 2 \end{bmatrix} = \lambda \; (20 + 2x_2^2)$$

Then

$$\lambda = \frac{14}{10 + x_2^2}$$

Substituting this back into (A.5)

$$2X = \lambda Z = \frac{14}{10+x_2^2}\begin{bmatrix} 4 \\ 2x_2 \\ 2 \end{bmatrix}$$

and

$$X = \frac{14}{10+x_2^2}\begin{bmatrix} 2 \\ x_2 \\ 1 \end{bmatrix}$$

Note that when

$$x_2 = 2, \qquad X = \frac{14}{14}\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

$$x_2 = -2, \qquad X = \frac{14}{14}\begin{bmatrix} 2 \\ -2 \\ 1 \end{bmatrix}$$

$$x_2 = 0, \qquad X = \frac{14}{10}\begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix}$$

In this way, all of the solutions from Taha are accounted for. However, the Matrix Derivative method is more simple and elegant by far, and also appears to provide a more complete solution set.

B. Matrix Analysis

B.1. Formulation

This appendix returns to the theme of analysis using Matrix Derivatives begun in Chapter 2 , now adapted to optimize over specific boundary planes of the solution space. The objective is to

Maximize $\quad \mathbf{1}' \ \mathrm{Diag}(BD_T) \ A' \ldots A' \ \mathrm{Diag}(BD_2) \ A' \ \mathrm{Diag}(BD_1) \ \pi$

$$- \lambda_1 \, (\underset{1\mathrm{xN}}{\pi'} \underset{\mathrm{Nx1}}{\mathbf{1}} - 1) - \underset{1\mathrm{xN}}{\lambda_2'} (\underset{\mathrm{NxM}}{B} \ \underset{\mathrm{Mx1}}{\mathbf{1}} - \underset{\mathrm{Nx1}}{\mathbf{1}}) - \underset{1\mathrm{xN}}{\lambda_3'} (\underset{\mathrm{NxN}}{A} \ \underset{\mathrm{Nx1}}{\mathbf{1}} - \underset{\mathrm{Nx1}}{\mathbf{1}})$$

$$- \underset{1\mathrm{xN}}{\lambda_4'} (\underset{\mathrm{Nx1}}{H_4} \odot \underset{\mathrm{Nx1}}{\pi}) - \mathrm{Tr}[\ \underset{\mathrm{MxN}}{\lambda_5'} (\underset{\mathrm{NxM}}{H_5} \odot \underset{\mathrm{NxM}}{B}) \ ] - \mathrm{Tr}[\ \underset{\mathrm{NxN}}{\lambda_6'} (\underset{\mathrm{NxN}}{H_6} \odot \underset{\mathrm{NxN}}{A}) \ ]$$

The formulation is the same as Chapter 2 , with the addition of an extra row of Lagrange terms. Once again, the expression appearing on the first line is the matrix representation of $P(o_1 \, o_2 .. o_T)$. Note that

> D_t is an $M \times 1$ column vector of zeros with a single 1 to mark the codebook symbol which occurred at time t
>
> Therefore BD_t extracts the k^{th} column of B , where symbol k occurred at time t
>
> and $\mathrm{Diag}(BD_t)$ represents the diagonal matrix which contains BD_t along the major diagonal

The terms appearing on the second line express the constraints that π B & A respectively must equal Markov matrices. Note that

$\mathbf{1}$ is a column vector (of appropriate size) consisting of all 1's

the expression $\quad X\mathbf{1} - \mathbf{1} = 0$

or $\quad X\mathbf{1} = \mathbf{1}$

demonstrates that the rows of X each sum to 1

and the λ_i are appropriate Lagrange multipliers

Finally the terms appearing on the third line, of the form

$$\mathrm{Tr}[\,\lambda'(H \odot X)\,]$$

are used to constrain chosen elements of π B and A to equal zero, and thereby focus the analysis on a specific boundary 'plane' of the solution space.

Once again, λ is an appropriate Lagrange multiplier. Matrix H has the same dimensions as X, and consists of 0's with scattered 1's to indicate where x_{ij} is intended to equal zero

$$h_{ij} = \begin{cases} 1, & x_{ij} = 0 \\ 0, & x_{ij} = 1 \end{cases}$$

Thus, for example, setting

$$H_4 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \qquad \text{forces} \qquad \pi = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

with the optimization carried out over this restricted domain.

To maximize $P(o_1 o_2 .. o_T)$ subject to the constraints, we

perform a partial differentiation of the full expression with respect to each unknown array,

set each result equal to zero,

and attempt to solve the simultaneous system of matrix equations for the unknowns

B.2. Derivative Equations

Differentiating--

w.r.t. π

$$\mathrm{Diag}(BD_1)\ A\ \mathrm{Diag}(BD_2)\ A \ldots A\ \mathrm{Diag}(BD_T)\ \mathbf{1}$$

$$- \underset{\text{scalar}}{\lambda_1}\ \underset{\text{Nx1}}{\mathbf{1}} - \underset{\text{Nx1}}{H_4} \odot \underset{\text{Nx1}}{\lambda_4} = \mathbf{0} \qquad \text{(B.1)}$$

w.r.t. B

$$\begin{aligned} & \mathrm{Diag}[\ \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \ldots A'\ \mathrm{Diag}(BD_2)\ A'\ ]\ \mathrm{x}\ \pi D_1' \\ + \quad & \mathrm{Diag}[\ \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \ldots A'\ ]\ \mathrm{x}\ A'\ \mathrm{Diag}(BD_1)\ \pi D_2' \\ \cdot \\ \cdot \\ + \quad & \mathrm{Diag}[\ \mathbf{1}'\ ]\ \mathrm{x}\ A' \ldots A'\ \mathrm{Diag}(BD_2)\ A'\ \mathrm{Diag}(BD_1)\ \pi D_T' \end{aligned}$$

$$- \underset{\text{Nx1}}{\lambda_2}\ \underset{\text{1xM}}{\mathbf{1}'} - \underset{\text{NxM}}{H_5} \odot \underset{\text{NxM}}{\lambda_5} = \mathbf{0} \qquad \text{(B.2)}$$

w.r.t. A

$$\begin{aligned} & \mathrm{Diag}(BD_1)\ \pi\ \mathrm{x}\ \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \ldots A'\ \mathrm{Diag}(BD_2) \\ + \quad & \mathrm{Diag}(BD_2)\ A'\ \mathrm{Diag}(BD_1)\ \pi\ \mathrm{x}\ \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \ldots A'\ \mathrm{Diag}(BD_3) \\ \cdot \\ \cdot \\ + \quad & \mathrm{Diag}(BD_{T\text{-}1})\ A' \ldots A'\ \mathrm{Diag}(BD_1)\ \pi\ \mathrm{x}\ \mathbf{1}'\ \mathrm{Diag}(BD_T) \end{aligned}$$

$$- \underset{\text{Nx1}}{\lambda_3}\ \underset{\text{1xN}}{\mathbf{1}'} - \underset{\text{NxN}}{H_6} \odot \underset{\text{NxN}}{\lambda_6} = \mathbf{0} \qquad \text{(B.3)}$$

w.r.t. λ_1 $$\pi' \mathbf{1} = 1 \tag{B.4}$$

w.r.t. λ_2 $$B \mathbf{1} = \mathbf{1} \tag{B.5}$$

w.r.t. λ_3 $$A \mathbf{1} = \mathbf{1} \tag{B.6}$$

w.r.t. λ_4 $$H_4 \odot \pi = \mathbf{0} \tag{B.7}$$

w.r.t. λ_5 $$H_5 \odot B = \mathbf{0} \tag{B.8}$$

w.r.t. λ_6 $$H_6 \odot A = \mathbf{0} \tag{B.9}$$

From here, the standard strategy [29] is to :

solve first for the Lagrange multipliers

then substitute their values to get a reduced system of equations in A B & π

B.3. The π Equation

B.3.1. Solving for λ_1

Transpose (B.1) ...then post-multiply by π

$$\mathbf{1}' \text{ Diag}(BD_T)\; A' \ldots A' \text{ Diag}(BD_2)\; A' \text{ Diag}(BD_1)\; \pi = \lambda_1\, \mathbf{1}'\pi + (H_4 \odot \lambda_4)'\, \pi$$

The left-hand side is $P(o_1\, o_2 .. o_T)$, while on the right-hand side

$$\mathbf{1}'\pi = 1 \qquad \text{using (B.4)}$$

Also

$$\begin{aligned}
(H_4 \odot \lambda_4)'\,\pi &= \text{Tr}[\,(H_4 \odot \lambda_4)'\,\pi\,] && \text{since it is a scalar} \\
&= \text{Tr}[\,\pi'\,(H_4 \odot \lambda_4)\,] \\
&= \text{Tr}[\,(\pi' \odot H_4')\,\lambda_4\,] && \text{see Magnus \& Neudecker p46, theorem 7(a)} \\
&= \text{Tr}[\,(H_4 \odot \pi)'\,\lambda_4\,] \\
&= \text{Tr}[\,0'\;\lambda_4\,] && \text{using (B.7)} \\
&= 0
\end{aligned}$$

Therefore λ_1 equals the value of the objective function at the stationary point.

B.3.2. Solving for λ_4

Define $\quad L = \text{Diag}(BD_1)\ A\ \text{Diag}(BD_2)\ A \ldots A\ \text{Diag}(BD_T)\ \mathbf{1} = \{\text{LHS}\}$

Then from (B.1)

$$L = \lambda_1 \mathbf{1} + H_4 \odot \lambda_4$$

In the interior of the π space,

$$H_4 = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

and $\quad L = \lambda_1 \mathbf{1}$

which implies that, at interior stationary points, L is a constant vector.

Otherwise, $\quad H_4 \odot \lambda_4 = L - \lambda_1 \mathbf{1}$

Alternatively, premultiply (B.1) by $\mathbf{1}'$

$$\mathbf{1}'\underbrace{\text{Diag}(BD_1)\ A\ \text{Diag}(BD_2)\ A \ldots A\ \text{Diag}(BD_T)}_{\text{NxN}}\ \mathbf{1} = \lambda_1 \underbrace{\mathbf{1}'\mathbf{1}}_{\text{scalar N}} - \mathbf{1}'(H_4 \odot \lambda_4)$$

Transposing and rearranging, this becomes

$$\frac{1}{N}(H_4 \odot \lambda_4)' \mathbf{1} = 1' \ \mathrm{Diag}(BD_T) \ A' \ldots A' \ \mathrm{Diag}(BD_2) \ A' \ \mathrm{Diag}(BD_1) \times \frac{1}{N}\mathbf{1} \ - \ \lambda_1$$

$$= 1' \ \mathrm{Diag}(BD_T) \ A' \ldots A' \ \mathrm{Diag}(BD_2) \ A' \ \mathrm{Diag}(BD_1) \times \frac{1}{N}\mathbf{1} \ - \ 1' \ \mathrm{Diag}(BD_T) \ A' \ldots A' \ \mathrm{Diag}(BD_2) \ A' \ \mathrm{Diag}(BD_1) \ \pi$$

$$= L' \ \times \frac{1}{N}\mathbf{1} \ - L' \pi$$

This states that $\frac{1}{N}$ times the sum of the active Lagrange multipliers equals the negative of the difference between

$P(o_1 \ o_2 .. o_T)$

and the same calculation with π replaced by $\frac{1}{N}\mathbf{1}$.

This is also the negative of the difference between

the maximum value of L

and the average value of L

when the stationary point corresponds to the global maximum * .

* Applying the definition for L, the objective $P = P(O)$ can be written

$$P(O) = L' \pi$$

Since $\sum \pi_i = 1$, P can be viewed as a weighted average of the elements of L .

The greatest value which a weighted average can attain is that of its largest element.

Therefore, to maximize P it is sufficient to

(1) find the largest possible maximum among the l_i , and
(2) assign a 1 to that position in π and 0's elsewhere.

This implies that at a global maximum, π consists of 0's and a single 1 ,
and that $L' \pi$ gives the maximum value (i.e. maximum possible element) of L .

B.4. The B Equation

B.4.1. Solving for λ_2

Post-multiply (B.2) by B'

$$\begin{aligned}
& \text{Diag}[\ \mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ ]\ \pi D_1' B' \\
+\ & \text{Diag}[\ \mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ ]\ A'\ \text{Diag}(BD_1)\ \pi D_2' B' \\
\cdot\ & \\
\cdot\ & \\
+\ & \text{Diag}[\ \mathbf{1}'\ ]\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi D_T' B'
\end{aligned}$$

$$= \lambda_2 \mathbf{1}' B' + (H \odot \lambda_5) B'$$

$$= \lambda_2 \mathbf{1}' + (H \odot \lambda_5) B'$$
using (B.5)

Then take the trace of both sides. On the right-hand side

$$\text{Tr}[\ \lambda_2 \mathbf{1}'\ ] = \mathbf{1}' \lambda_2$$ trace of outer product equals inner product

Also

$$\begin{aligned}
\text{Tr}[\ (H_5 \odot \lambda_5) B'\ ] &= \text{Tr}[\ (\lambda_5 \odot H_5) B'\ ] \\
&= \text{Tr}[\ \lambda_5 (H_5' \odot B')\ ] && \text{M \& N, p46, 7(a)} \\
&= \text{Tr}[\ \lambda_5 (H_5 \odot B)'\ ] \\
&= \text{Tr}[\ \lambda_5 \mathbf{0}\ ] && \text{using (B.8)} \\
&= 0
\end{aligned}$$

On the left-hand side :

The trace of a sum equals the sum of the traces. The result is a sum of terms of the form

$$\mathrm{Tr}[\ \mathrm{Diag}(X)\ Y\ ]$$

Invoking Magnus & Neudecker, p46, theorem 7(b)

$$\mathrm{Tr}[\ \mathrm{Diag}(X)\ Y\ ] \;=\; X'\,(\mathrm{I} \odot Y)\,\mathbf{1}$$

where I is the $N \times N$ identity matrix

In addition, Y is itself an outer product

$$Y = \underset{\mathrm{Nx1}}{Z}\ \underset{\mathrm{1xN}}{W'}$$

Therefore,

$$(\mathrm{I} \odot Y)\,\mathbf{1} = \begin{bmatrix} y_{11} & & \\ & y_{22} & \\ & & y_{33} \end{bmatrix} \mathbf{1} = \begin{bmatrix} y_{11} \\ y_{22} \\ y_{33} \end{bmatrix} = \begin{bmatrix} z_1 w_1 \\ z_2 w_2 \\ z_3 w_3 \end{bmatrix}$$

$$= \mathrm{Diag}(W)\ Z$$

Pulling together these various results, the equation becomes :

$$
\begin{array}{ll}
 & \mathbf{1}' \, \mathrm{Diag}(BD_T) \, A' \ldots A' \, \mathrm{Diag}(BD_2) \, A' \times \mathrm{Diag}(BD_1) \, \pi \\
+ & \mathbf{1}' \, \mathrm{Diag}(BD_T) \, A' \ldots A' \, \mathrm{Diag}(BD_3) \, A' \times \mathrm{Diag}(BD_2) \, A' \, \mathrm{Diag}(BD_1) \, \pi \\
\cdot & \\
\cdot & \\
+ & \mathbf{1}' \times \mathrm{Diag}(BD_T) \, A' \ldots A' \, \mathrm{Diag}(BD_2) \, A' \, \mathrm{Diag}(BD_1) \, \pi \\
 & \qquad\qquad\qquad\qquad\qquad\qquad = \mathbf{1}' \lambda_2
\end{array}
$$

The left-hand side is T copies of the value of the objective function at the stationary point.

Therefore

$$T \, P(o_1 o_2 .. o_T) = \mathbf{1}' \lambda_2$$

or

$$T \, \lambda_1 = \mathbf{1}' \lambda_2$$

Alternatively, the B-equation can be written

$$\frac{dP}{dB} = \underset{N\times 1}{\lambda_2}\,\underset{1\times M}{\mathbf{1}'} + \underset{N\times M}{H_5}\odot\underset{N\times M}{\lambda_5}$$

where $P = P(o_1 o_2 .. o_T)$

Hadamard-multiply through by B

$$\begin{aligned} B\odot\frac{dP}{dB} &= B\odot(\lambda_2\mathbf{1}') + B\odot(H_5\odot\lambda_5) \\ &= B\odot[\text{Diag}(\lambda_2)\,\mathbf{1}\,\mathbf{1}'] + (B\odot H_5)\odot\lambda_6 \\ &= \text{Diag}(\lambda_2)\,B + \mathbf{0}\odot\lambda_5 \\ &= \text{Diag}(\lambda_2)\,B \end{aligned}$$

Now post-multiply by $\mathbf{1}$

$$\begin{aligned} (B\odot\frac{dP}{dB})\,\mathbf{1} &= \text{Diag}(\lambda_2)\,B\,\mathbf{1} \\ &= \text{Diag}(\lambda_2)\,\mathbf{1} \\ &= \lambda_2 \end{aligned}$$

B.4.2. Solving for λ_5

Post-multiply (B.2) by $\underset{M\text{x}1}{\mathbf{1}}$

$$
\begin{aligned}
& \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\,]\ \pi D_1'\,\mathbf{1} \\
+\ & \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,]\ A'\ \text{Diag}(BD_1)\ \pi D_2'\,\mathbf{1} \\
\cdot\ & \\
\cdot\ & \\
+\ & \text{Diag}[\,\mathbf{1}'\,]\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi \underbrace{D_T'\,\mathbf{1}}_{\text{scalar } 1} \\
& = \lambda_2 \underbrace{\mathbf{1}'\,\mathbf{1}}_{\text{scalar } M} + (H_5 \odot \lambda_5)\,\mathbf{1}
\end{aligned}
$$

or

$$
\begin{aligned}
& \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\,]\ \pi \\
+\ & \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,]\ A'\ \text{Diag}(BD_1)\ \pi \\
\cdot\ & \\
\cdot\ & \\
+\ & \text{Diag}[\,\mathbf{1}'\,]\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi \\
& = \underset{N\text{x}1}{M \lambda_2} + (H_5 \odot \lambda_5)\,\mathbf{1}
\end{aligned}
$$

Now pre-multiply by $\underset{1\text{x}N}{\mathbf{1}'}$

$$
\begin{aligned}
& \mathbf{1}'\ \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\,]\ \pi \\
+\ & \mathbf{1}'\ \text{Diag}[\,\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,]\ A'\ \text{Diag}(BD_1)\ \pi \\
\cdot\ & \\
\cdot\ & \\
+\ & \mathbf{1}'\ \text{Diag}[\,\mathbf{1}'\,]\ A' \ldots A'\ \text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi \\
& = M\,\mathbf{1}'\,\lambda_2 + \mathbf{1}'\,(H_5 \odot \lambda_5)\,\mathbf{1}
\end{aligned}
$$

which becomes

$$
\begin{array}{llllll}
 & \mathbf{1}' \,\mathrm{Diag}(BD_T)\; A' \ldots A' \,\mathrm{Diag}(BD_2)\; A' & & & & \pi \\
+ & \mathbf{1}' \,\mathrm{Diag}(BD_T)\; A' \ldots A' & & & A' \,\mathrm{Diag}(BD_1) & \pi \\
\cdot & & & & & \\
\cdot & & & & & \\
+ & \mathbf{1}' \; A' \ldots A' \,\mathrm{Diag}(BD_2)\; A' \,\mathrm{Diag}(BD_1) & & & & \pi
\end{array}
$$

$$= M\,\mathbf{1}'\,\lambda_2 + \mathbf{1}'\,(\,H_5 \odot \lambda_5\,)\,\mathbf{1}$$

Omitting the proof, the left-hand side represents T copies of the objective function with successive occurrences of $\mathrm{Diag}(BD_t)$ replaced by the identity matrix I, and is equal to

$$P(o_2..o_T) \;+\; P(o_1\,o_3..o_T) \;+ \ldots +\; P(o_1..o_{T-1})$$

Also, from Section B.4.1.,

$$\mathbf{1}'\,\lambda_2 \;=\; T\,\lambda_1$$

And

$$\mathbf{1}'\,(\,H_5 \odot \lambda_5\,)\,\mathbf{1} \;=\; \mathrm{Tr}[\,H_5'\;\lambda_5\,]$$

$$= \text{the sum of the active Lagrange multipliers}$$

Alternatively, this can be written

$$
\begin{array}{ll}
 & \mathbf{1}' \,\mathrm{Diag}(BD_T)\, A' \ldots A' \,\mathrm{Diag}(BD_2)\, A' \quad \frac{1}{M}\mathrm{I} \quad \pi \\
+ & \mathbf{1}' \,\mathrm{Diag}(BD_T)\, A' \ldots A' \quad \frac{1}{M}\mathrm{I} \quad A' \,\mathrm{Diag}(BD_1)\, \pi \\
\cdot & \\
\cdot & \\
+ & \mathbf{1}' \quad \frac{1}{M}\mathrm{I} \quad A' \ldots A' \,\mathrm{Diag}(BD_2)\, A' \,\mathrm{Diag}(BD_1)\, \pi
\end{array}
$$

$$= \mathbf{1}'\lambda_2 + \frac{1}{M}\mathbf{1}'(H_5 \odot \lambda_5)\,\mathbf{1}$$

$$= T\lambda_1 + \frac{1}{M}\mathbf{1}'(H_5 \odot \lambda_5)\,\mathbf{1}$$

This states that $\frac{1}{M}$ times the sum of the active Lagrange multipliers equals the negative of the difference between

T copies of $P(o_1 o_2 .. o_T)$

and another T copies in which one occurrence of B per copy is replaced by $\frac{1}{M}\mathbf{1}\,\mathbf{1}'$.

B.5. The A Equation

B.5.1. Solving for λ_3

Post-multiply (B.3) by A'

$$
\begin{aligned}
& \underbrace{\text{Diag}(BD_1)\ \pi}_{\text{Nx1}} \ \text{x}\ \underbrace{\mathbf{1}'\,\text{Diag}(BD_T)\ A' \ldots A'\ \text{Diag}(BD_2)\ A'}_{\text{1xN}} \\
+\ & \underbrace{\text{Diag}(BD_2)\ A'\ \text{Diag}(BD_1)\ \pi}_{\text{Nx1}} \ \text{x}\ \underbrace{\mathbf{1}'\ \text{Diag}(BD_T)\ A' \ldots A'\,\text{Diag}(BD_3)\ A'}_{\text{1xN}} \\
\cdot\ & \\
\cdot\ & \\
\cdot\ & \\
+\ & \underbrace{\text{Diag}(BD_{T-1})\ A' \ldots A'\,\text{Diag}(BD_1)\ \pi}_{\text{Nx1}} \ \text{x}\ \underbrace{\mathbf{1}'\ \text{Diag}(BD_T)\ A'}_{\text{1xN}} \\
& \qquad = \lambda_3\,\mathbf{1}'A' + (H_6 \odot \lambda_6)\,A' \\
& \qquad = \lambda_2\,\mathbf{1}' + (H_6 \odot \lambda_6)\,A'
\end{aligned}
$$

using (B.6)

Then take the trace of both sides. Since

Ⓐ If two matrices are equal, then their traces are equal

Ⓑ The trace of a sum equals the sum of the traces

Ⓒ The trace of the outer product of two vectors equals their inner product :

$$\text{Tr}(\underset{\text{Nx1}}{X}\,\underset{\text{1xN}}{Y'}) = \underset{\text{1xN}}{Y'}\,\underset{\text{Nx1}}{X}$$

this becomes

$$
\begin{aligned}
& \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \ldots A'\ \mathrm{Diag}(BD_2)\ A' \times \mathrm{Diag}(BD_1)\ \pi \\
+ \quad & \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \ldots A'\ \mathrm{Diag}(BD_3)\ A' \times \mathrm{Diag}(BD_2)\ A'\ \mathrm{Diag}(BD_1)\ \pi \\
\cdot \quad & \\
\cdot \quad & \\
+ \quad & \mathbf{1}'\ \mathrm{Diag}(BD_T)\ A' \times \mathrm{Diag}(BD_{T-1})\ A' \ldots A'\ \mathrm{Diag}(BD_1)\ \pi
\end{aligned}
$$

$$= \mathbf{1}'\lambda_3 + \mathrm{Tr}[\,(H_6 \odot \lambda_6)\,A'\,]$$

The left-hand side is T-1 copies of the value of the objective function at the stationary point.

On the right-hand side

$$
\begin{aligned}
\mathrm{Tr}[\,(H_6 \odot \lambda_6)\,A'\,] &= \mathrm{Tr}[\,(\lambda_6 \odot H_6)\,A'\,] \\
&= \mathrm{Tr}[\,\lambda_6\,(H_6' \odot A')\,] && \text{M \& N, p46, 7(a)} \\
&= \mathrm{Tr}[\,\lambda_6\,(H_6 \odot A\,)'\,] \\
&= \mathrm{Tr}[\,\lambda_6\,\mathbf{0}\,] && \text{using (B.9)} \\
&= 0
\end{aligned}
$$

Therefore

$$(\,T - 1)\ P(o_1 o_2 .. o_T) \quad = \quad \mathbf{1}'\lambda_3$$

or

$$(\,T - 1)\ \lambda_1 \quad = \quad \mathbf{1}'\lambda_3$$

Alternatively, the A-equation can be written

$$\frac{dP}{dA} = \underset{N\mathrm{x}1}{\lambda_3}\,\underset{1\mathrm{x}N}{\mathbf{1}'} + \underset{N\mathrm{x}N}{H_6}\odot\underset{N\mathrm{x}N}{\lambda_6}$$

where $P = P(o_1 o_2 .. o_T)$

Hadamard-multiply through by A

$$\begin{aligned} A\odot\frac{dP}{dA} &= A\odot(\lambda_3\,\mathbf{1}') + A\odot(H_6\odot\lambda_6) \\ &= A\odot[\mathrm{Diag}(\lambda_3)\,\mathbf{1}\,\mathbf{1}'] + (A\odot H_6)\odot\lambda_6 \\ &= \mathrm{Diag}(\lambda_3)\,A + \mathbf{0}\odot\lambda_6 \\ &= \mathrm{Diag}(\lambda_3)\,A \end{aligned}$$

Now post-multiply by $\mathbf{1}$

$$\begin{aligned} (A\odot\frac{dP}{dA})\,\mathbf{1} &= \mathrm{Diag}(\lambda_3)\,A\,\mathbf{1} \\ &= \mathrm{Diag}(\lambda_3)\,\mathbf{1} \\ &= \lambda_3 \end{aligned}$$

B.5.2. Solving for λ_6

Post-multiply (B.3) by $\underset{N\times 1}{\mathbf{1}}$

$$
\begin{aligned}
& \operatorname{Diag}(BD_1)\ \pi \times \mathbf{1}' \operatorname{Diag}(BD_T)\ A' \ldots A' \operatorname{Diag}(BD_2)\ \mathbf{1} \\
+ \quad & \operatorname{Diag}(BD_2)\ A' \operatorname{Diag}(BD_1)\ \pi \times \mathbf{1}' \operatorname{Diag}(BD_T)\ A' \ldots A' \operatorname{Diag}(BD_3)\ \mathbf{1} \\
\cdot \quad & \\
\cdot \quad & \\
+ \quad & \operatorname{Diag}(BD_{T-1})\ A' \ldots A' \operatorname{Diag}(BD_1)\ \pi \times \mathbf{1}' \operatorname{Diag}(BD_T)\ \mathbf{1} \\
& \qquad = \lambda_3 \underbrace{\mathbf{1}'\mathbf{1}}_{\text{scalar N}} + (H_6 \odot \lambda_6)\ \mathbf{1}
\end{aligned}
$$

Now pre-multiply by $\underset{1\mathrm{x}N}{\mathbf{1}'}$

$$
\begin{aligned}
& \underbrace{\mathbf{1}' \operatorname{Diag}(BD_1)\ \pi}_{\text{scalar}} \times \underbrace{\mathbf{1}' \operatorname{Diag}(BD_T)\ A' \ldots A' \operatorname{Diag}(BD_2)\ \mathbf{1}}_{\text{scalar}} \\
+ \quad & \underbrace{\mathbf{1}' \operatorname{Diag}(BD_2)\ A' \operatorname{Diag}(BD_1)\ \pi}_{\text{scalar}} \times \underbrace{\mathbf{1}' \operatorname{Diag}(BD_T)\ A' \ldots A' \operatorname{Diag}(BD_3)\ \mathbf{1}}_{\text{scalar}} \\
\cdot \quad & \\
\cdot \quad & \\
\cdot \quad & \\
+ \quad & \underbrace{\mathbf{1}' \operatorname{Diag}(BD_{T-1})\ A' \ldots A' \operatorname{Diag}(BD_1)\ \pi}_{\text{scalar}} \times \underbrace{\mathbf{1}' \operatorname{Diag}(BD_T)\ \mathbf{1}}_{\text{scalar}} \\
& \qquad = N\,\mathbf{1}'\lambda_3 + \mathbf{1}'(H_6 \odot \lambda_6)\ \mathbf{1}
\end{aligned}
$$

Since the multiplication of scalars is commutative, this can be re-arranged as follows :

$$
\begin{aligned}
& \mathbf{1}' \,\mathrm{Diag}(BD_T)\, A' \ldots A' \,\mathrm{Diag}(BD_2)\, \frac{1}{N}\, \mathbf{1}\,\mathbf{1}' \,\mathrm{Diag}(BD_1)\, \pi \\
+ \quad & \mathbf{1}' \,\mathrm{Diag}(BD_T)\, A' \ldots A' \,\mathrm{Diag}(BD_3)\, \frac{1}{N}\, \mathbf{1}\,\mathbf{1}' \,\mathrm{Diag}(BD_2)\, A' \,\mathrm{Diag}(BD_1)\, \pi \\
\cdot \quad & \\
\cdot \quad & \\
+ \quad & \mathbf{1}' \,\mathrm{Diag}(BD_T)\, \frac{1}{N}\, \mathbf{1}\,\mathbf{1}' \,\mathrm{Diag}(BD_{T-1})\, A' \ldots A' \,\mathrm{Diag}(BD_2)\, A' \,\mathrm{Diag}(BD_1)\, \pi
\end{aligned}
$$

$$
= \mathbf{1}'\lambda_3 + \frac{1}{N}\,\mathbf{1}'\,(H_6 \odot \lambda_6)\,\mathbf{1}
$$

$$
= (T\text{-}1)\lambda_1 + \frac{1}{N}\,\mathbf{1}'\,(H_6 \odot \lambda_6)\,\mathbf{1}
$$

Just as with π and B earlier, this states that $\frac{1}{N}$ times the sum of the active Lagrange multipliers equals the negative of the difference between

T-1 copies of $P(o_1 o_2 .. o_T)$

and another T-1 copies in which one occurrence of A per copy is replaced by $\frac{1}{N}\,\mathbf{1}\,\mathbf{1}'$.

B.6. Summary

The previous sections have revealed some interesting patterns. For example, despite the different ways in which the principal unknowns A , B and π enter into the objective expression, there is remarkable similarity between the forms assumed by their respective Lagrange multipliers.

Unfortunately, the analysis is still far from complete. Of the six Lagrange multipliers, several are known only to within a sum of their elements. New insights are necessary before these can be determined and a reduced system of equations in A B & π can be solved.

C. Geometric Programming

C.1. Origins

This appendix presents a second approach to optimizing the boundary fragments of f_1 described in Chapter 3. Geometric Programming is a calculus-based optimization technique which is particularly well-suited to functions of that form [52,53,54].

The name derives from a connection with the Arithmetic-Geometric Mean Inequality (also known as Cauchy's Inequality [55]) , which states that the arithmetic mean of a set of non-negative numbers is never smaller than the geometric mean :

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \geq x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} \qquad \text{(with equality iff } x_1 = x_2 = \ldots = x_n\text{)}$$

$$\begin{array}{lcl} \text{provided} & x_k \geq 0 & (k = 1..n) \\ & \alpha_k > 0 & (k = 1..n) \\ \text{and} & \sum_{k=1}^{n} \alpha_k = 1 & \end{array}$$

Historically, the original motivation for the technique was to carry out unconstrained minimizations on a certain general class of objective function* :

$$\begin{aligned} y &= f(x_1, x_2, \ldots x_N) \\ &= c_1 x_1^{p_{11}} x_2^{p_{21}} \ldots x_N^{p_{N1}} + c_2 x_1^{p_{12}} x_2^{p_{22}} \ldots x_N^{p_{N2}} + c_M x_1^{p_{1M}} x_2^{p_{2M}} \ldots x_N^{p_{NM}} \\ &= \sum_{j=1}^{M} c_j \left\{ \prod_{i=1}^{N} x_i^{p_{ij}} \right\} \qquad \text{where } x_i > 0 \end{aligned}$$

* Since a minimum is sought, one can regard these as 'cost' functions

These functions resemble normal polynomials, with two important differences :

(1) The exponents p_{ij} are allowed to assume any real value (not restricted to the non-negative integers)

(2) The coefficients c_j must be positive

As a consequence of (2), the functions were christened 'posynomials'.

Let $$t_j = c_j x_1^{p_{1j}} x_2^{p_{2j}} \ldots x_N^{p_{Nj}}$$

$$= \text{the } j^{\text{th}} \text{ term of } y$$

and consider the situation when y is at a minimum point :

$$\frac{\partial y}{\partial x_1} = c_1 p_{11} x_1^{p_{11}-1} x_2^{p_{21}} \ldots x_N^{p_{N1}} + \ldots + c_M p_{1M} x_1^{p_{1M}-1} x_2^{p_{2M}} \ldots x_N^{p_{NM}}$$

$$= \frac{1}{x_1} \sum_{j=1}^{M} p_{1j}\, t_j \qquad \text{(since } x_i > 0\text{)}$$

$$= 0 \qquad \text{(derivative vanishes)}$$

and, in general

$$\frac{1}{x_i} \sum_{j=1}^{M} p_{ij}\, t_j = 0 \qquad (i = 1..N)$$

The classical method for optimizing y is to solve this system of N simultaneous equations in the N unknowns x_i. As an alternative to this, define a new set of variables

$$w_j = \frac{t_j}{y}$$

which represent the relative contribution of the j^{th} term to the total cost.

Then $$w_j > 0$$

and $$\sum_{j=1}^{M} w_j = \sum_{j=1}^{M} \frac{t_j}{y} = \frac{1}{y}\sum_{j=1}^{M} t_j = \frac{1}{y} y = 1 \qquad \text{(C.1)}$$

Furthermore, at the minimum point for y

$$\sum_{j=1}^{M} p_{ij} w_j = \sum_{j=1}^{M} p_{ij}\left(\frac{t_j}{y}\right) = \frac{x_i}{y} \times \frac{1}{x_i}\sum_{j=1}^{M} p_{ij} t_j = \frac{x_i}{y} \times 0 = 0$$

$$(i = 1..N) \qquad \text{(C.2)}$$

Then, using (C.1) and (C.2)

$$y = y^1 = y^{\sum w_j} = \prod_{j=1}^{M} y^{w_j} = \prod_{j=1}^{M}\left(\frac{t_j}{w_j}\right)^{w_j}$$

$$= \prod_{j=1}^{M}\left(\frac{1}{w_j} \times c_j \prod_{i=1}^{N} x_i^{p_{ij}}\right)^{w_j}$$

$$= \prod_{j=1}^{M}\left\{\left(\frac{c_j}{w_j}\right)^{w_j}\left(\prod_{i=1}^{N} x_i^{p_{ij}}\right)^{w_j}\right\}$$

$$= \left\{\prod_{j=1}^{M}\left(\frac{c_j}{w_j}\right)^{w_j}\right\}\left\{\prod_{j=1}^{M}\left(\prod_{i=1}^{N} x_i^{p_{ij}}\right)^{w_j}\right\}$$

$$= \left\{\prod_{j=1}^{M}\left(\frac{c_j}{w_j}\right)^{w_j}\right\}\left\{\prod_{j=1}^{M}\left(\prod_{i=1}^{N} x_i^{p_{ij}w_j}\right)\right\}$$

$$= \left\{\prod_{j=1}^{M}\left(\frac{c_j}{w_j}\right)^{w_j}\right\}\left\{\prod_{i=1}^{N}\left(\prod_{j=1}^{M} x_i^{p_{ij}w_j}\right)\right\}$$

$$= \left\{\prod_{j=1}^{M}\left(\frac{c_j}{w_j}\right)^{w_j}\right\}\left\{\prod_{i=1}^{N} x_i^{\sum p_{ij}w_j}\right\}$$

$$= \left\{\prod_{j=1}^{M}\left(\frac{c_j}{w_j}\right)^{w_j}\right\}\left\{\prod_{i=1}^{N} x_i^{0}\right\} \qquad \text{at the minimum point}$$

$$= \prod_{j=1}^{M}\left(\frac{c_j}{w_j}\right)^{w_j}$$

Finally, consider the new function defined on the variables w_j

$$z = d(w_1, w_2, \ldots w_M) = \prod_{j=1}^{M}\left(\frac{c_j}{w_j}\right)^{w_j}$$

It can be shown (using the Arithmetic-Geometric Mean Inequality) that the values of w_j which maximize this function subject to the constraints

$$\sum_{j=1}^{M} w_j \quad = \quad 1 \qquad\qquad (C.1)$$

and

$$\sum_{j=1}^{M} p_{ij}\, w_j \quad = \quad 0 \qquad (i = 1..N) \qquad\qquad (C.2)$$

are exactly those which minimize the original function

$$y \;=\; f(x_1, x_2, \ldots x_N) \qquad \text{(and vice versa)}$$

An intuitive 'proof' of this can be sketched out as follows :

Using the Arithmetic-Geometric Mean Inequality, one can show that z is a lower bound for y, with equality at the stationary points of y (equivalent to the constraint set of z)

$$y \;=\; \sum_{j=1}^{M} t_j \;=\; \sum_{j=1}^{M} w_j \left(\frac{t_j}{w_j}\right) \;=\; \prod_{j=1}^{M} \left(\frac{t_j}{w_j}\right)^{w_j} \qquad \text{invoking equality condition}$$

since $\frac{t_j}{w_j} = y$ for all j

$$\left.\begin{aligned} &= \prod_{j=1}^{M} \left(\frac{c_j}{w_j}\right)^{w_j} \\ &= z \text{ (constrained)} \end{aligned}\right\} \quad \text{at the stationary points of y}$$

Furthermore, as a concave function subject to a set of convex constraints, the constrained z has a unique maximum... while y has a unique minimum.

Therefore, the constrained maximum for z must equal the minimum for y.

In summary, by introducing the new variables w_j, the primal problem of minimizing y is transformed into the dual 'Geometric Programming' problem of maximizing z, which is often easier to solve.

The constraints (C.1) and (C.2) are known as the normality and orthogonality conditions, respectively.

There is one dual variable w_j for each term in the primal function y. There is also one orthogonality equation (C.2) for each primal variable x_i.

In the case when the primal problem contains one more term than variable ($M = N+1$), the result is a system of M linear equations in the M unknowns w_j, which can be solved uniquely. From this solution, the optimum y is calculated using

$$y = \prod_{j=1}^{M} \left(\frac{c_j}{w_j}\right)^{w_j}$$

The primal variables x_i are then recovered using the relations

$$w_j = \frac{t_j}{y} = \frac{c_j x_1^{p_{1j}} x_2^{p_{2j}} \cdots x_N^{p_{Nj}}}{y}$$

This task is generally easier than the form of the relations suggests and often can be accomplished by simple inspection. However, if necessary, one can take logarithms and solve the resulting simultaneous equations, which are now linear in $\log x_i$ (see [56]).

More typically $M > N+1$, in which case additional steps are employed to find the optimum set of weights w_j from among all those sets which satisfy (C.1) and (C.2) :

The difference, D, between M and $N+1$, which is conventionally called the number of degrees of freedom, is here referred to as the 'degrees of difficulty'.

The linear system of equations consisting of (C.1) and (C.2) is solved for M of the variables in terms of the remaining D variables. Then the results are substituted back into the expression for z ...which is either differentiated with respect to the D unknowns to produce a second system of (non-linear) equations for solution, or is maximized directly via hill-climbing. Either way, thanks to the nature of z as a concave function under convex constraints, a unique constrained maximum for z (minimum for y) is guaranteed.

C.2. The General Problem

It must be emphasized that the formulation presented above only applies when

(1) the objective function contains positive coefficients,
(2) a minimum is sought, and
(3) there are no side conditions.

In time, the theory of Geometric Programming was refined and extended to cope with inequality constraints of either sense, as well as negative and/or positive coefficients in either the objective function or the constraints.

This permits a maximum to be found, for example, by minimizing the negative of the objective function.

The price to be paid for this generality, however, is that [57]

(1) The dual function z no longer has a unique stationary point which is a maximum, but may also exhibit stationary points which are minima or points of inflection.

(2) The optimal set of weights may now occur at any type of dual stationary point (not simply the maxima)--which rules out hill-climbing as a solution method.

(3) Stationary points in the dual function no longer guarantee a minimum in the primal function y (could yield a saddle point or a maximum).

The full-blown Geometric Programming problem can be summarized as follows :

Take the word 'signomial' to indicate a posynomial which is permitted negative coefficients, and use the new subscript k to distinguish among a collection of such functions.

The general signomial can be written

$$y_k = f_k(x_1, x_2, \dots x_N)$$

$$= \sigma_{1k} c_{1k} x_1^{p_{11k}} x_2^{p_{21k}} \dots x_N^{p_{N1k}} + \sigma_{2k} c_{2k} x_1^{p_{12k}} x_2^{p_{22k}} \dots x_N^{p_{N2k}} + \dots$$

(M_k terms)

$$= \sum_{j=1}^{M_k} \sigma_{jk} c_{jk} \left\{ \prod_{i=1}^{N} x_i^{p_{ijk}} \right\}$$

$$= \sum_{j=1}^{M_k} \sigma_{jk} t_{jk}$$

where $c_{jk} > 0$

and $\sigma_{jk} = \pm 1$

Next assume that, along with the objective, there are L constraints. All of these constraints are inequalities, and are expressed as either

$$y_k \le 1 \qquad \text{or} \qquad y_k \ge 1 \qquad (k = 1..L)$$

where the y_k are signomials. For each constraint, define an auxiliary variable

$$\sigma_k = \begin{cases} +1, & y_k \le 1 \\ -1, & y_k \ge 1 \end{cases}$$

Assume that the problem is stated as a signomial to be <u>minimized</u>, and use y_0 to denote the objective function. Define

$$\sigma_0 = \begin{cases} +1\,, & \text{if a positive minimum (positive 'cost') is expected} \\ -1\,, & \text{if a negative minimum (positive 'profit') is expected} \end{cases}$$

Finally, define $L+1$ sets of dual variables, $w_{jk} \ge 0$,

with a set of variables for each signomial, including the objective function
$(k = 0..L)$

and a variable per set for each term of the respective signomials
$(j = 1..M_k)$

where (as before) $w_{j0} = \dfrac{t_{j0}}{y_0}$ $\qquad (j = 1..M_0)$

and in addition $w_{jk} = t_{jk} \times \displaystyle\sum_{j=1}^{M_k} \sigma_{jk}\, w_{jk}$ $\qquad (j = 1..M_k)(k = 1..L)$

(C.3)

Then the dual Geometric Programming problem (see [58]) is to maximize

$$d = \sigma_0 \left[\prod_{k=0}^{L} \prod_{j=1}^{M_k} \left(\frac{c_{jk}\, w_{0k}}{w_{jk}} \right)^{\sigma_{jk} w_{jk}} \right]^{\sigma_0}$$

subject to the normality condition

$$\sum_{j=1}^{M_0} \sigma_{j0}\, w_{j0} = \sigma_0$$

and the N orthogonality conditions

$$\sum_{k=0}^{L} \sum_{j=1}^{M_k} \sigma_{jk}\, p_{ijk}\, w_{jk} = 0 \; (i = 1..N)$$

with $$w_{0k} = \sigma_k \sum_{j=1}^{M_k} \sigma_{jk}\, w_{jk} \geq 0 \qquad (k = 1..L)$$

and $$w_{00} = 1$$

The main difference with Section C.1. is that all terms from all of the primal signomials contribute to the dual objective function and the orthogonality equations--no distinction is made between terms from the primal objective and terms from the constraints [59].

C.3. Application to Boundary Fragments

Fortunately, by exploiting the special features of f_1 and its fragments, and also by employing a useful 'trick', much of the complexity of the general formulation evaporates off and the optimizations assume a simpler form.

To illustrate, consider the observation sequence $k\,k\,l\,k$ and the task of maximizing the fragment $(6\ 7\ 8)$, or

$$(a_{12})^2\, a_{21}\, b_{1k}\, b_{1l}\, (b_{2k})^2 \;+\; a_{12}\, a_{21}\, a_{22}\, (b_{1k})^2\, b_{2k}\, b_{2l} \;+\; a_{12}\, (a_{22})^2\, b_{1k}\, (b_{2k})^2\, b_{2l}$$

First, let

$$\begin{aligned} a_{21} &= x_1 \\ a_{22} &= x_2 \\ b_{1k} &= x_3 \\ b_{1l} &= x_4 \\ b_{2k} &= x_5 \\ b_{2l} &= x_6 \end{aligned}$$

and, of course, $a_{12} = 1$

Then the fragment looks like

$$x_1\, x_3\, x_4\, (x_5)^2 \;+\; x_1\, x_2\, (x_3)^2\, x_5\, x_6 \;+\; (x_2)^2\, x_3\, (x_5)^2\, x_6$$

To maximize this function is equivalent to minimizing its negative. The primal problem is therefore

Minimize $y_0 = -[\, x_1 x_3 x_4 (x_5)^2 + x_1 x_2 (x_3)^2 x_5 x_6 + (x_2)^2 x_3 (x_5)^2 x_6 \,]$

subject to
$$\begin{aligned} x_1 + x_2 &\leq 1 \\ x_3 + x_4 &\leq 1 \\ x_5 + x_6 &\leq 1 \end{aligned}$$

Notice that

(A) There are 6 variables and a total of 9 terms. Therefore, the dual problem will have $D = 9 - (6+1) = 2$ 'degrees of difficulty' (free variables).

(B) The use of inequality constraints instead of the (more accurate) equalities causes no difficulty. The optimization will certainly drive them to equality anyway, and meanwhile they keep the form of the problem simpler.
If, alternatively, x_2 is replaced by $1 - x_1$, etc. and the results substituted, the objective would split into 10 (mixed positive and negative) terms in the 3 variables, leading to 6 degrees of difficulty in the dual.

(C) All of the p_{ijk} and the c_{jk} are whole numbers. (In <u>this</u> case, all $c_{jk} = 1$)

The constraints are already in the proper form : $\sigma_k y_k \leq \sigma_k$

with $\sigma_k = 1$ ($k = 1..3$)

and also $\sigma_{jk} = 1$ ($j = 1, 2$), ($k = 1..3$)

As -1 times the sum of probabilities, the objective will certainly take a negative minimum : $\sigma_0 = -1$

And, of course, $\sigma_{j0} = -1$ ($j = 1..3$)

Now here's the 'trick' (inspired by [60]) :

Factor out $x_1\, x_2\, (x_3)^2\, x_5\, x_6$ from the objective function* to give

$$-\, x_1\, x_2\, (x_3)^2\, x_5\, x_6 [\, (x_2)^{-1}(x_3)^{-1}\, x_4\, x_5\, (x_6)^{-1} + 1 + (x_1)^{-1}\, x_2\, (x_3)^{-1}\, x_5\,]$$

and introduce an additional variable, x_7

If $\quad x_7 \leq (x_2)^{-1}(x_3)^{-1}\, x_4\, x_5\, (x_6)^{-1} + 1 + (x_1)^{-1}\, x_2\, (x_3)^{-1}\, x_5$

then $\quad -\, x_1\, x_2\, (x_3)^2\, x_5\, x_6\, x_7 \geq$

$$-\, x_1\, x_2\, (x_3)^2\, x_5\, x_6 [\, (x_2)^{-1}(x_3)^{-1}\, x_4\, x_5\, (x_6)^{-1} + 1 + (x_1)^{-1}\, x_2\, (x_3)^{-1}\, x_5\,] = y_0$$

and minimizing the upper bound $-\, x_1\, x_2\, (x_3)^2\, x_5\, x_6\, x_7$ will minimize y_0

Therefore, replace y_0 with the simplified objective function

$$-\, x_1\, x_2\, (x_3)^2\, x_5\, x_6\, x_7$$

and append the additional constraint to the list :

$$x_7 - (x_2)^{-1}(x_3)^{-1}\, x_4\, x_5\, (x_6)^{-1} - (x_1)^{-1}\, x_2\, (x_3)^{-1}\, x_5 \leq 1$$

* Any of the 3 terms will do, but for simplicity choose one with the maximum number of different factors--e.g., an 'interior' term, if one is available

The revised problem becomes :

$$\text{Minimize} \qquad y_0 = -x_1 x_2 (x_3)^2 x_5 x_6 x_7$$

$$\begin{aligned} \text{subject to} \qquad x_1 + x_2 &\le 1 \\ x_3 + x_4 &\le 1 \\ x_5 + x_6 &\le 1 \\ x_7 - (x_2)^{-1}(x_3)^{-1} x_4 x_5 (x_6)^{-1} - (x_1)^{-1} x_2 (x_3)^{-1} x_5 &\le 1 \end{aligned}$$

Note that

(D) The number of terms and variables each increase by 1 , so that the degrees of difficulty is unchanged.

(E) With only one term in the objective function, the normality condition becomes trivial :

$$-w_{10} = -1 \quad (\text{or } w_{10} = 1) \qquad \text{since } \sigma_{10} = \sigma_0 = -1$$

(F) The 'interesting' terms are the objective expression plus the final $(D = 2)$ terms of the final constraint. These are also the terms for which $\sigma_{jk} = -1$.

Now, writing down the orthogonality conditions

for $i = 1$:

$$
\begin{aligned}
& \quad (-1)(1)\, w_{10} && (k=0) \\
& + \ (+1)(1)\, w_{11} + (+1)(0)\, w_{21} && (k=1) \\
& + \ (+1)(0)\, w_{12} + (+1)(0)\, w_{22} && (k=2) \\
& + \ (+1)(0)\, w_{13} + (+1)(0)\, w_{23} && (k=3) \\
& + \ (+1)(0)\, w_{14} + (-1)(0)\, w_{24} + (-1)(-1)\, w_{34} && (k=4) \\
& = \ 0
\end{aligned}
$$

for $i = 2$:

$$
\begin{aligned}
& \quad (-1)(1)\, w_{10} && (k=0) \\
& + \ (+1)(0)\, w_{11} + (+1)(1)\, w_{21} && (k=1) \\
& + \ (+1)(0)\, w_{12} + (+1)(0)\, w_{22} && (k=2) \\
& + \ (+1)(0)\, w_{13} + (+1)(0)\, w_{23} && (k=3) \\
& + \ (+1)(0)\, w_{14} + (-1)(-1)\, w_{24} + (-1)(1)\, w_{34} && (k=4) \\
& = \ 0
\end{aligned}
$$

etc., or

$$
\begin{array}{llllllllllll}
 & w_{10} & w_{11} & w_{21} & w_{12} & w_{22} & w_{13} & w_{23} & w_{14} & w_{24} & w_{34} & \\
i = 1 & (-1)(1)\,w_{10} & +\,(+1)(1)\,w_{11} & & & & & & & +\,(-1)(0)\,w_{24} & +\,(-1)(-1)\,w_{34} & = 0 \\
i = 2 & (-1)(1)\,w_{10} & & +\,(+1)(1)\,w_{21} & & & & & & +\,(-1)(-1)\,w_{24} & +\,(-1)(1)\,w_{34} & = 0 \\
i = 3 & (-1)(2)\,w_{10} & & & +\,(+1)(1)\,w_{12} & & & & & +\,(-1)(-1)\,w_{24} & +\,(-1)(-1)\,w_{34} & = 0 \\
i = 4 & (-1)(0)\,w_{10} & & & & +\,(+1)(1)\,w_{22} & & & & +\,(-1)(1)\,w_{24} & +\,(-1)(0)\,w_{34} & = 0 \\
i = 5 & (-1)(1)\,w_{10} & & & & & +\,(+1)(1)\,w_{13} & & & +\,(-1)(1)\,w_{24} & +\,(-1)(1)\,w_{34} & = 0 \\
i = 6 & (-1)(1)\,w_{10} & & & & & & +\,(+1)(1)\,w_{23} & & +\,(-1)(-1)\,w_{24} & +\,(-1)(0)\,w_{34} & = 0 \\
i = 7 & (-1)(1)\,w_{10} & & & & & & & +\,(+1)(1)\,w_{14} & +\,(-1)(0)\,w_{24} & +\,(-1)(0)\,w_{34} & = 0
\end{array}
$$

Using the fact that w_{10} is always 1 (see Note E above), it becomes simple to solve for the $(N = 7)$ diagonal variables in terms of the remaining $(D = 2)$ variables :

$$\begin{aligned}
w_{11} &= 1 \phantom{{}- w_{24}} - w_{34} \\
w_{21} &= 1 - w_{24} + w_{34} \\
w_{12} &= 2 - w_{24} - w_{34} \\
w_{22} &= \phantom{1 - {}} w_{24} \\
w_{13} &= 1 + w_{24} + w_{34} \\
w_{23} &= 1 - w_{24} \\
w_{14} &= 1
\end{aligned}$$

Then

$$\begin{aligned}
w_{01} &= \sigma_1 \sum_{j=1}^{M_1} \sigma_{j1}\, w_{j1} \\
&= w_{11} + w_{21} \\
&= (1 - w_{34}) + (1 - w_{24} + w_{34}) \\
&= 2 - w_{24}
\end{aligned}$$

$$\begin{aligned}
w_{02} &= w_{12} + w_{22} \\
&= (2 - w_{24} - w_{34}) + (w_{24}) \\
&= 2 - w_{34}
\end{aligned}$$

$$\begin{aligned}
w_{03} &= w_{13} + w_{23} \\
&= (1 + w_{24} + w_{34}) + (1 - w_{24}) \\
&= 2 + w_{34}
\end{aligned}$$

$$\begin{aligned}
w_{04} &= \sigma_4 \sum_{j=1}^{M_4} \sigma_{j4}\, w_{j4} \\
&= w_{14} - w_{24} - w_{34} \\
&= 1 - w_{24} - w_{34}
\end{aligned}$$

For future ease, substitute $w_{24} = u_1$ and $w_{34} = u_2$. Then the dual function becomes*

$$\left(\frac{c_{10}\,w_{00}}{w_{10}}\right)^{\sigma_{10}w_{10}} \qquad k=0$$

$$\times\left(\frac{c_{11}\,w_{01}}{w_{11}}\right)^{\sigma_{11}w_{11}}\left(\frac{c_{21}\,w_{01}}{w_{21}}\right)^{\sigma_{21}w_{21}} \qquad k=1$$

$$\times\left(\frac{c_{12}\,w_{02}}{w_{12}}\right)^{\sigma_{12}w_{12}}\left(\frac{c_{22}\,w_{02}}{w_{22}}\right)^{\sigma_{22}w_{22}} \qquad k=2$$

$$\times\left(\frac{c_{13}\,w_{03}}{w_{13}}\right)^{\sigma_{13}w_{13}}\left(\frac{c_{23}\,w_{03}}{w_{23}}\right)^{\sigma_{23}w_{23}} \qquad k=3$$

$$\times\left(\frac{c_{14}\,w_{04}}{w_{14}}\right)^{\sigma_{14}w_{14}}\left(\frac{c_{24}\,w_{04}}{w_{24}}\right)^{\sigma_{24}w_{24}}\left(\frac{c_{34}\,w_{04}}{w_{34}}\right)^{\sigma_{34}w_{34}} \qquad k=4$$

$$=\left(\frac{1}{1}\right)^{-(1)}$$
$$\times\left(\frac{2-u_1}{1-u_2}\right)^{(1-u_2)}\left(\frac{2-u_1}{1-u_1+u_2}\right)^{(1-u_1+u_2)}$$
$$\times\left(\frac{2-u_2}{2-u_1-u_2}\right)^{(2-u_1-u_2)}\left(\frac{2-u_2}{u_1}\right)^{(u_1)}$$
$$\times\left(\frac{2+u_2}{1+u_1+u_2}\right)^{(1+u_1+u_2)}\left(\frac{2+u_2}{1-u_1}\right)^{(1-u_1)}$$
$$\times\left(\frac{1-u_1-u_2}{1}\right)^{(1)}\left(\frac{1-u_1-u_2}{u_1}\right)^{-(u_1)}\left(\frac{1-u_1-u_2}{u_2}\right)^{-(u_2)}$$

$$=\left(1-u_2\right)^{-(1-u_2)}\quad\left(1-u_1+u_2\right)^{-(1-u_1+u_2)}\quad\left(2-u_1\right)^{+(2-u_1)}$$
$$\times\left(2-u_1-u_2\right)^{-(2-u_1-u_2)}\quad u_1^{-(u_1)}\quad\left(2-u_2\right)^{+(2-u_2)}$$
$$\times\left(1+u_1+u_2\right)^{-(1+u_1+u_2)}\quad\left(1-u_1\right)^{-(1-u_1)}\quad\left(2+u_2\right)^{+(2+u_2)}$$
$$\times(1)^{-(1)}\quad u_1^{+(u_1)}\quad u_2^{+(u_2)}\quad\left(1-u_1-u_2\right)^{+(1-u_1-u_2)}$$

* Maximizing $\sigma_0\left[d\left(u_1,u_2\right)\right]^{\sigma_0}=-1\left[d\left(u_1,u_2\right)\right]^{-1}$ is equivalent to maximizing $d(u_1,u_2)$

Taking logs

$$\begin{aligned}
v = \log[d(u_1, u_2)] = & - (1 - u_2) \log (1 - u_2) \\
& - (1 - u_1 + u_2) \log (1 - u_1 + u_2) \\
& + (2 - u_1) \log (2 - u_1) \\
& - (2 - u_1 - u_2) \log (2 - u_1 - u_2) \\
& - u_1 \log u_1 \\
& + (2 - u_2) \log (2 - u_2) \\
& - (1 + u_1 + u_2) \log (1 + u_1 + u_2) \\
& - (1 - u_1) \log (1 - u_1) \\
& + (2 + u_2) \log (2 + u_2) \\
& - \log 1 \\
& + u_1 \log u_1 \\
& + u_2 \log u_2 \\
& + (1 - u_1 - u_2) \log (1 - u_1 - u_2)
\end{aligned}$$

Taking partial derivatives, and equating to zero

$$
\begin{aligned}
\frac{\partial v}{\partial u_1} = \quad & - \quad [\,0\,] \\
& - (-1)\,[1 + \log(1 - u_1 + u_2)] \\
& + (-1)\,[1 + \log(2 - u_1)] \\
& - (-1)\,[1 + \log(2 - u_1 - u_2)] \\
& - (+1)\,[1 + \log\ u_1] \\
& + \quad [\,0\,] \\
& - (+1)\,[1 + \log(1 + u_1 + u_2)] \\
& - (-1)\,[1 + \log(1 - u_1)] \\
& + \quad [\,0\,] \\
& - \quad [\,0\,] \\
& + (+1)\,[1 + \log\ u_1] \\
& + \quad [\,0\,] \\
& + (-1)\,[1 + \log(1 - u_1 - u_2)] \\
& = 0
\end{aligned}
$$

Or

$$
\begin{aligned}
&\log(1 - u_1 + u_2) - \log(2 - u_1) + \log(2 - u_1 - u_2) - \log\ u_1 \\
&- \log(1 + u_1 + u_2) + \log(1 - u_1) + \ u_1 \log\ u_1 - \ \log(1 - u_1 - u_2) \ = \ 0
\end{aligned}
$$

since the constant terms always cancel [61].

Similarly,

$$\frac{\partial v}{\partial u_2} = - (-1)\,[1+\log(1-u_2)]$$

$$- (+1)\,[1+\log(1-u_1+u_2)]$$

$$+ \quad [\,0\,]$$

$$- (-1)\,[1+\log(2-u_1-u_2)]$$

$$- \quad [\,0\,]$$

$$+ (-1)\,[1+\log(2-u_2)]$$

$$- (+1)\,[1+\log(1+u_1+u_2)]$$

$$- \quad [\,0\,]$$

$$+ (+1)[1+\log(2+u_2)]$$

$$- \quad [\,0\,]$$

$$+ \quad [\,0\,]$$

$$+ (+1)\,[1+\log\ u_2]$$

$$+ (-1)\,[1+\log(1-u_1-u_2)]$$

$$= 0$$

Or

$$\log(1-u_2) - \log(1-u_1+u_2) + \log(2-u_1-u_2) - \log(2-u_2)$$

$$- \log(1+u_1+u_2) + \log(2+u_2) + \log\ u_2 - \log(1-u_1-u_2) = 0$$

Rearranging and taking exponentials, the final system of equations is

$$(1 - u_1 + u_2)\,(2 - u_1 - u_2)\,(1 - u_1)\,u_1 \;=\; (2 - u_1)\,u_1\,(1 + u_1 + u_2)\,(1 - u_1 - u_2)$$

$$(1 - u_2)\,(2 - u_1 - u_2)\,(2 + u_2)\,u_2 \;=\; (1 - u_1 + u_2)\,(2 - u_2)\,(1 + u_1 + u_2)\,(1 - u_1 - u_2)$$

Note further that

(G) Geometric Programming offers a reduction in complexity in this case, since applying calculus to the primal problem would produce 3 non-linear equations in 3 unknowns instead of the 2 x 2 above .

(H) Once this system is solved, the form of the problem makes it simple to recover the original variables. Using definition (C.3) from page 222 and the terms from the first 3 constraints

$$a_{21} = x_1 = \frac{w_{11}}{w_{01}} \qquad a_{22} = x_2 = \frac{w_{21}}{w_{01}}$$

$$b_{1k} = x_3 = \frac{w_{12}}{w_{02}} \qquad b_{1l} = x_4 = \frac{w_{22}}{w_{02}}$$

$$b_{2k} = x_5 = \frac{w_{13}}{w_{03}} \qquad b_{2l} = x_6 = \frac{w_{23}}{w_{03}}$$

(I) Although non-linear, each equation contains only linear factors.

There are equal numbers of factors on either side of the equation.

Each of the factors is non-negative since they equal the w_{jk}, which must be ≥ 0 .

Also, the final factor in each equation is always of the form

$$(1 - u_1 - \ldots - u_D)$$

The non-negativity of the factors can be used to deduce bounds on the unknowns $u_1 \ldots u_D$. In particular,

$$u_1 + \ldots + u_D \leq 1$$

Therefore, Geometric Programming has transformed the problem from a (primal) domain of x 's where several small groups of unknowns satisfy

$$\sum_j x_{ij} \leq 1$$

to a (dual) domain of u 's where all of the unknowns simultaneously satisfy

$$u_1 + \ldots + u_D \leq 1$$

C.4. Automating the Transformation

Finally, thanks to the special features of these boundary fragment problems and to the 'trick' presented earlier, it is possible to automate the process of transforming from the primal to the dual problem, and so proceed directly to the final system of dual equations.

Consider the annotated 'tableau' shown below :

k k l k (6 7 8)

$$(a_{12})^2\, a_{21}\, b_{1k}\, b_{1l}\, (b_{2k})^2 \;+\; a_{12}\, a_{21}\, a_{22}\, (b_{1k})^2\, b_{2k}\, b_{2l} \;+\; a_{12}\, (a_{22})^2\, b_{1k}\, (b_{2k})^2\, b_{2l}$$

① Let

$$\begin{aligned} a_{21} &= x_1 \\ a_{22} &= x_2 \\[6pt] b_{1k} &= x_3 \\ b_{1l} &= x_4 \\[6pt] b_{2k} &= x_5 \\ b_{2l} &= x_6 \\[6pt] a_{12} &= 1 \end{aligned}$$

$$x_1\, x_3\, x_4\, (x_5)^2 \;+\; x_1\, x_2\, (x_3)^2\, x_5\, x_6 \;+\; (x_2)^2\, x_3\, (x_5)^2\, x_6$$

② $x_1\, x_2\, (x_3)^2\, x_5\, x_6\; [\, (x_2)^{-1} (x_3)^{-1}\, x_4\, x_5\, (x_6)^{-1} + \;1 + \;(x_1)^{-1}\, x_2\, (x_3)^{-1}\, x_5 \,]$

③ ⑤ ⑥ ⑦

$$\left.\begin{aligned} w_{11} &= 1 \phantom{{}-u_1} - u_2 \\ w_{21} &= 1 - u_1 + u_2 \end{aligned}\right\}\quad w_{01} = 2 - u_1$$

$$\left.\begin{aligned} w_{12} &= 2 - u_1 - u_2 \\ w_{22} &= \phantom{2 - {}} u_1 \end{aligned}\right\}\quad w_{02} = 2 - u_2$$

$$\left.\begin{aligned} w_{13} &= 1 + u_1 + u_2 \\ w_{23} &= 1 - u_1 \end{aligned}\right\}\quad w_{03} = 2 + u_2$$

④

$$\left.\begin{aligned} w_{14} &= 1 \\ -w_{24} &= - u_1 \\ -w_{34} &= - u_2 \end{aligned}\right\}\quad w_{04} = 1 - u_1 - u_2$$

⑧ $(1 - u_1 + u_2)\,(2 - u_1 - u_2)\,(1 - u_1)\,u_1 \;=\; (2 - u_1)\,u_1\,(1 + u_1 + u_2)\,(1 - u_1 - u_2)$

$(1 - u_2)\,(2 - u_1 - u_2)\,(2 + u_2)\,u_2 \;=\; (1 - u_1 + u_2)\,(2 - u_2)\,(1 + u_1 + u_2)\,(1 - u_1 - u_2)$

① List the variables and group them by Markov constraint. Rename them using x's, and substitute into the original fragment.

② Factor out one term from the expression. For later ease, this term should include a maximum number of different factors (e.g., an 'interior' term if one is present).

③ List double-subscripted variables ('w_{jk} = ') and group them exacly as per step ①, with the k subscript indicating the number of the group and the j subscript indicating position within the group.

④ Below this list, include an additional group consisting of the equations

$$\begin{array}{rcl} w_{1\,(L+1)} & = & 1 \\ -w_{2\,(L+1)} & = & \quad -u_1 \\ & & \qquad \ddots \\ -w_{(D+1)\,(L+1)} & = & \qquad\qquad -u_D \end{array}$$

where (as above) L = number of Markov constraints

D = degrees of difficulty = $M - (N+1)$

⑤ In the constant column, list the exponents of the x_i $(i = 1..N)$ from the factored term (outside the brackets) of step ②.

⑥ In the u_1 column, multiply u_1 by the exponents of the x_i from the first 'interesting' term (i.e., not equaling 1) inside the brackets. In this way, generate D columns based upon the interesting terms within the brackets.

⑦ To the right of the braces, define w_{0k} to equal the sum of the expressions in group k.

⑧ There will be D equations, each having the form

$$-[\mathrm{L}] \quad = \quad +[\mathrm{R}]$$

where L and R are products of the linear expressions generated in steps ③ through ⑦.

The equations are derived from those expressions, as follows :

In the area to the left of the braces--

If an expression contains u_d preceded by a positive coefficient then include the expression, raised to the absolute value of that coefficient*, within the R side of equation d ...alternatively,

If the expression contains u_d preceded by a negative coefficient then include the expression, raised to the absolute value of that coefficient, within the L side of equation d.

To the right of the braces, these rules are reversed--

The expression goes into the L (R) side if the sign is positive (negative).

⑨ A final adjustment is necessary if any of the terms inside the brackets in step ② include a coefficient other than 1. Such a coefficient will derive from duplicate terms in the original fragment, and therefore must be a rational number.

Referring again to 'interesting' term d, include the numerator of the coefficient in the R side of equation d and the denominator in the L side of equation d.

* In the present example, all coefficients happen to equal 1.

C.5. Solving the Dual Equations

C.5.1. Description

Being able to generate the dual equations automatically is one thing... being able to solve them automatically is another matter. The purpose of the present section is to review the (modest) progress which has been made in this direction.

The basic thrust has been to mimic the strategy for polynomials whereby, once roots are known, they can be factored out to leave a simpler polynomial which is more easily solved for the remaining roots.

To see how this might be accomplished, refer once more to the observation sequence *k k l k* and fragment (6 7 8) whose 'tableau' and dual equations were presented in the previous section.

Mathcad PLUS version 6.0 identifies ten solutions for this system--

(1)	$u_1 = 0$	$u_2 = 1$
(2)	$u_1 = 0$	$u_2 = 2$
(3)	$u_1 = 2$	$u_2 = 1$
(4)	$u_1 = 1$	$u_2 = 0$
(5)	$u_1 = 1$	$u_2 = -2$
(6)	$u_1 = -1 + \sqrt{2}$	$u_2 = 6 - 4\sqrt{2}$
(7)	$u_1 = -1 - \sqrt{2}$	$u_2 = 6 + 4\sqrt{2}$
(8)	$u_1 = 1.2999$	$u_2 = -0.9387$
(9)	$u_1 = -0.4833 + \mathrm{j}\,0.1512$	$u_2 = 0.8027 + \mathrm{j}\,1.4820$
(10)	$u_1 = -0.4833 - \mathrm{j}\,0.1512$	$u_2 = 0.8027 - \mathrm{j}\,1.4820$

The first point to note about these ten solutions is that most of them (in fact, all but three: 1, 4 & 6) fail to provide a feasible solution, when mapped back to the primal domain.

The linear factors which comprise the dual equations are merely the dual variables

$$w_{jk} \qquad (k = 0..L)\ (j = 0..M_k)$$

expressed in terms of u_1 and u_2 (which are themselves the dual variables $w_{2(L+1)}$ .. $w_{(D+1)\ (L+1)}$). The dual variables must be ≥ 0 since, otherwise, the original primal variables won't fall between 0 and 1 , as required.

To satisfy simultaneously all the non-negativity requirements forces u_1 and u_2 to inhabit the (hyper) triangle bounded by

$$u_d = 0 \qquad (d = 1..D)$$

and by $$u_1 + \ldots + u_D = 1 \qquad \text{(see Note (I) above)}$$

Furthermore, it appears that under the Geometric Programming transformation :

infeasible primal points	map to the	exterior	of the triangle
interior primal points	" " "	interior	" " "
boundary points	" " "	boundary	" " "

Finally, only those dual solutions which are interior to the triangle (solution 6) are of immediate interest, since the solutions on the boundary (1 & 4) will be identified in the course of processing the boundary fragments lower down the hierarchy--while the infeasible solutions can be ignored* .

* Complex solutions appear to have no physical interpretation within this problem and are treated as infeasible.

The second point to note about the Mathcad solutions is that several of them are readily identifiable.

One simple strategy for solving an equation (or system of equations) given in the form $g_1(u_1, u_2,...) = g_2(u_1, u_2,...)$ is to look for values which cause both sides of the equation to simultaneously equal zero. The fact that the dual equations are already expressed in linear-factored form makes the process even simpler. The following plot shows the set of lines in the (u_1 , u_2) plane which result when all of the linear factors contained in the dual equations are equated to zero. Appearing among the points of intersection are Mathcad solutions (1) through (5) because corresponding to each of those points there exists a set of factors which makes both sides of all dual equations equal to zero :

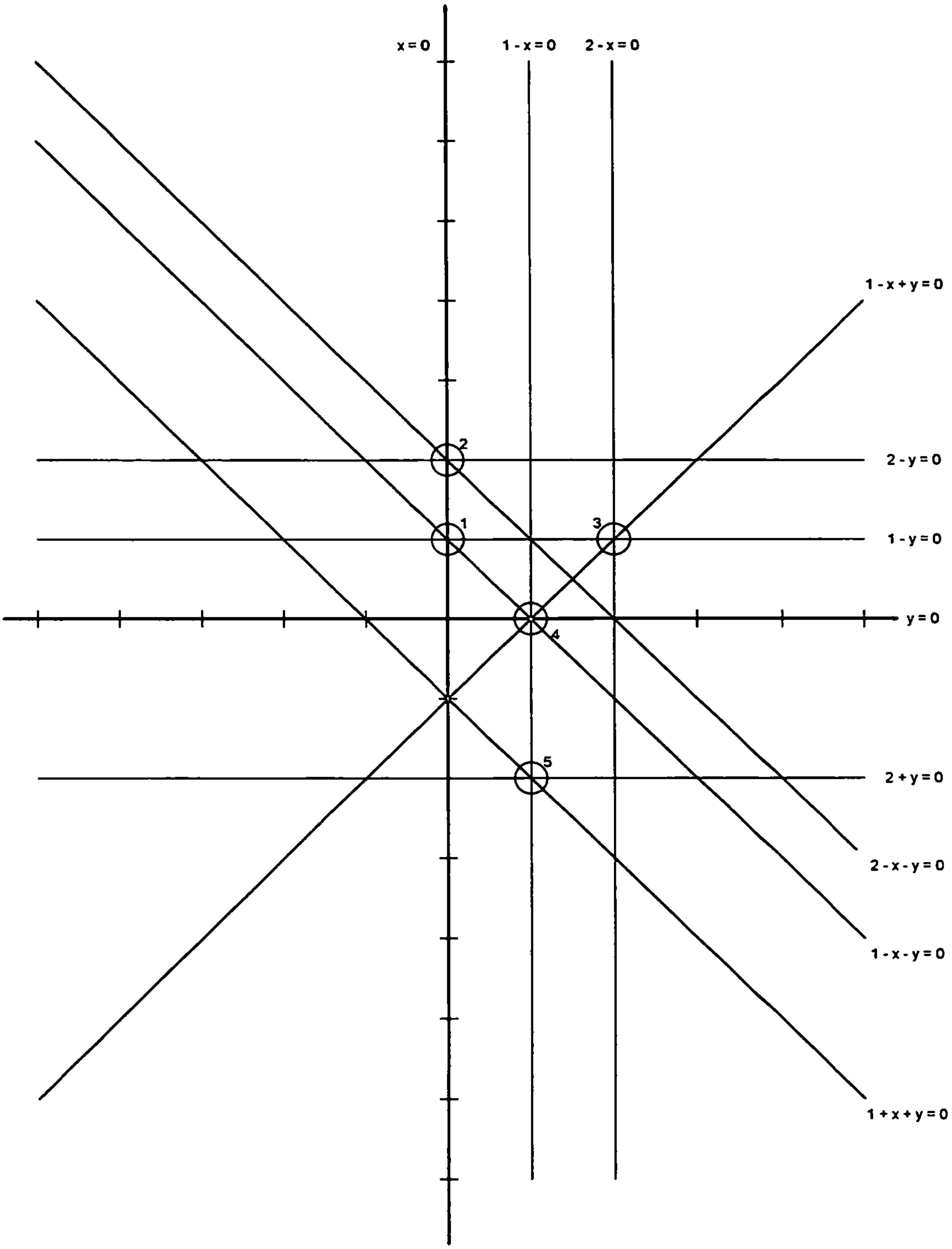
$$\left.\begin{array}{l} x = 0 \\ 1 - y = 0 \\ 1 - x - y = 0 \end{array}\right\} \qquad (1)$$

$$\left.\begin{array}{l} x = 0 \\ 2 - y = 0 \\ 2 - x - y = 0 \end{array}\right\} \qquad (2)$$

$$\left.\begin{array}{l} 2 - x = 0 \\ 1 - y = 0 \\ 1 - x + y = 0 \end{array}\right\} \qquad (3)$$

$$\left.\begin{array}{l} 1 - x = 0 \\ y = 0 \\ 1 - x + y = 0 \\ 1 - x - y = 0 \end{array}\right\} \qquad (4)$$

$$\left.\begin{array}{l} 1 - x = 0 \\ 2 + y = 0 \\ 1 + x + y = 0 \end{array}\right\} \qquad (5)$$


x = 0
1 - x = 0
2 - x = 0
1 - x + y = 0
2 - y = 0
1 - y = 0
y = 0
2 + y = 0
2 - x - y = 0
1 - x - y = 0
1 + x + y = 0
1
2
3
4
5

The rule which finds these solutions is :

If a pair of factors

(a) are not parallel, and

(b) either they *or their sum or difference* appear on opposite sides of all equations

then their intersection will provide a solution.

Another simple approach to finding solutions is to examine the lower-dimensional fragments which fall within its hierarchy. The lower-dimensional fragments form the boundaries, and therefore constitute special cases, of the fragment. For this reason, solutions to the dual equations of the sub-fragments might be expected to 'work' for the larger fragment also.

Again, the expression for the fragment (6 7 8) is

$$(a_{12})^2 a_{21} b_{1k} b_{1l} (b_{2k})^2 + a_{12} a_{21} a_{22} (b_{1k})^2 b_{2k} b_{2l} + a_{12} (a_{22})^2 b_{1k} (b_{2k})^2 b_{2l}$$

and the sub-tree attached to (6 7 8) looks like

$$\begin{tikzcd}
 & & (8) \arrow[d, dash] \\
(6) \arrow[dr, dash] & & (7\ (8)) \arrow[dl, dash] \\
 & ((6)\ (7\ (8))) &
\end{tikzcd}$$

Analysis confirms that information about those lower-dimensional fragments is 'encoded' in the equations for (6 7 8). For example, what distinguishes the terms 7 & 8 within the above expression is

$$b_{1k} = 1 \qquad \text{and} \qquad b_{1l} = 0$$

which implies that $\dfrac{w_{12}}{w_{02}} = 1$ and $\dfrac{w_{22}}{w_{02}} = 0$

or $w_{12} = w_{02}$ and $w_{22} = 0$

$$\underbrace{2 - u_1 - u_2 = 2 - u_2 \qquad \text{and} \qquad u_1 = 0}_{u_1 \;=\; 0}$$

Substituting this into the pair of dual equations produces

$$(1 + u_2)\,(2 - u_2)\,(1)\,(0) \;=\; (2)\,(0)\,(1 + u_2)\,(1 - u_2)$$

$$(1 - u_2)\,(2 - u_2)\,(2 + u_2)\,u_2 \;=\; (1 + u_2)\,(2 - u_2)\,(1 + u_2)\,(1 - u_2)$$

The first of these is trivially satisfied for any value of u_2 . The second (minus the factors $(2 - u_2)$) turns out to be the dual equation for the fragment (7 8) :

$$(1 - u_2)\,(2 + u_2)\,u_2 \;=\; (1 + u_2)\,(1 + u_2)\,(1 - u_2)$$

And, conversely, any solutions of this equation for u_2 (i.e. any solution of the fragment (7 8)), coupled with $u_1 = 0$, will clearly satisfy the equations for (6 7 8) .

This example demonstrates that

(1) The equations for a fragment contain the lower-dimensional component fragments as special cases, and that

(2) These special cases contribute solutions to the higher-dimensional fragment.

This offers a second bootstrap approach for finding solutions to the dual equations.

Now it is well known that, given a single polynomial in one unknown, plus a knowledge of some of its roots, it is possible to 'divide out' the associated factors and thereby produce a simpler polynomial for which the remaining roots are more easily found.

This prompts the question : Given a system of polynomial equations as per Geometric Programming, plus a knowledge of some of its solutions derived by either of the two methods above, is it likewise possible to somehow divide out those solutions to provide a reduced system?

The answer seems to be, 'Yes' . Even more promising, based upon a series of examples it appears that :

> For each interior solution which a system of dual equations possesses, a reduced system of equations can be found which contains only that solution --plus, at most, a minimal collection of infeasible (possibly complex) conjugates.

In this way, it seems possible to isolate the interior solutions into their own simplified systems.

For example, by a judicious pruning of factors, the system of equations for (6 7 8) is reduced to

$$(1 - u_1 + u_2)\, u_1 = (2 - u_1)(1 - u_1 - u_2)$$

$$(2 - u_1 - u_2)(1 - u_1) = u_1 (1 + u_1 + u_2)$$

which has only the interior solution (6) (plus the infeasible conjugate (7)) for solutions, and ignores the other eight original Mathcad solutions.

From Higher Algebra [62,63], a 'polynomial over a field F (or a ring R) in the indeterminate x' is an expression of the form

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

where all of the coefficients a_i belong to F . Such a polynomial is said to be 'reducible' [64] if it can be expressed as a product of polynomials of smaller degree, all of whose coefficients also belong to F . Similar to the prime factor theorem for integers, it seems that every polynomial can be expressed as a product of irreducible factors [65] and, for a field F , the factorization is unique [66] (while for a ring R it is not [67]). Also, it can be shown that irreducibility over the integers is equivalent to irreducibility over the rationals [68]. Finally, when a polynomial over the integers has been split into its irreducible factors and these factors are then individually equated to zero and solved for their complex roots, the factors which are linear will yield a single rational root, while the higher factors will yield a cluster of (irrational and/or complex) conjugate roots.

In some kind of analog which is not very well understood, it seems that the system of dual equations from G.P. can be split into various 'irreducible' sub-systems. When solved for their complex roots, these produce non-overlapping subsets of the original solutions. These subsets may contain either a single rational root, or a cluster of irrational and/or complex conjugate roots. Furthermore, it appears (based upon limited evidence) that each sub-system will contain at most one solution which is interior to the feasible region. Thus, while a system of dual equations may have more than one interior solution, each of these interior solutions can be identified by solving an appropriate sub-system.

Unfortunately, at this stage, the procedure for creating these sub-systems is still largely trial-and-error. The body of successful examples referred to above depend upon a variety of techniques, all of which require a prior knowledge of the solutions.

C.5.2. Examples

The purpose of this section is to record some representative examples of dual equations being split into sub-systems, along with the technique(s) by which it was accomplished :

The observation sequence $k\,k\,l\,k$ and fragment (4 8) has the dual equation

$$u_1\,(1 - u_1)\,(2 + u_1) \;=\; (1 + u_1)\,(2 - u_1)\,(1 - u_1)$$

with the three solutions $u_1 = 1, -\frac{1}{4}\left(1 \pm \sqrt{17}\right)$.

Quite obviously, removing the factor $(1 - u_1)$ from both sides eliminates the $u_1 = 1$ root and leaves just the interior solution $-\frac{1}{4}\left(1 - \sqrt{17}\right)$ and its conjugate.

Not so obviously, splitting the equation into <u>either</u> of the two simpler equations

$$u_1 \;=\; 2\,(1 + u_1)\,(1 - u_1)$$

$$2\,(1 - u_1)\,(2 + u_1) \;=\; (2 - u_1)$$

also has the same result.

Our old familiar example consisting of the observation sequence *k k l k* and fragment (6 7 8) has the pair of dual equations

$$(1 - u_1 + u_2)\,(2 - u_1 - u_2)\,(1 - u_1)\,u_1 \;=\; (2 - u_1)\,u_1\,(1 + u_1 + u_2)\,(1 - u_1 - u_2)$$

$$(1 - u_2)\,(2 - u_1 - u_2)\,(2 + u_2)\,u_2 \;=\; (1 - u_1 + u_2)\,(2 - u_2)\,(1 + u_1 + u_2)\,(1 - u_1 - u_2)$$

with the ten solutions listed above.

The simplified system of equations (also seen earlier)

$$(1 - u_1 + u_2)\,u_1 \;=\; (2 - u_1)\,(1 - u_1 - u_2) \qquad \text{(C.4)}$$

$$(2 - u_1 - u_2)\,(1 - u_1) \;=\; u_1\,(1 + u_1 + u_2) \qquad \text{(C.5)}$$

is generated by splitting just the <u>first</u> of the original equations--

> equating the two outer factors from both sides to produce (C.4) ,
> and then equating the two inner factors from both sides to produce (C.5) .

The observation sequence *k l k k* and fragment (1 2 4) has the pair of dual equations

$$u_1 (1 - u_1)(2 + u_1 + u_2)(2 + 2u_1 + u_2)^2 = (1 + 2u_1 + u_2)^4 (1 - u_1 - u_2)$$

$$u_2 (2 + u_1 + u_2)(2 + 2u_1 + u_2)^2 = (1 + 2u_1 + u_2)^2 (1 - u_1 - u_2)$$

which has six solutions, including the interior solution

$$u_1 = 0.7181 \qquad\qquad u_2 = 0.1107$$

Divide these equations and rearrange to get

$$u_1 (1 - u_1)(2 + 2u_1 + u_2) = u_2 (1 + 2u_1 + u_2)^2$$

Then split this into the two simpler equations

$$u_1 (2 + 2u_1 + u_2) = (1 + 2u_1 + u_2)$$

$$(1 - u_1) = u_2 (1 + 2u_1 + u_2)$$

These capture the interior solution plus two other complex conjugate solutions.

The fragment $k\,k\,l\,l$ (1 2 3) has the dual equations

$$u_1(1-u_1)(2+u_1+u_2)(3+u_1) = (1+2u_1+u_2)^2(1+u_1)(1-u_1-u_2)$$

$$u_2(2+u_1+u_2) = (1+2u_1+u_2)(1-u_1-u_2)$$

Among the four solutions from Mathcad is the single interior solution

$$u_1 = \frac{1}{3} \qquad\qquad u_2 = -\frac{1}{6}\left(5 - 3\sqrt{5}\right)$$

This system is typical of a class of problems where several of the linear factors are conjugates of one another. For example,

$$(1+2u_1+u_2) = \frac{1}{6}\left(5+3\sqrt{5}\right) = [-u_2]^* = -[u_2^*]$$

$$(2+u_1+u_2) = \frac{1}{2}\left(3+\sqrt{5}\right)$$

$$(1-u_1-u_2) = \frac{1}{2}\left(3-\sqrt{5}\right) = (2+u_1+u_2)^*$$

In addition,

$$u_2(2+u_1+u_2) = \frac{1}{3}\sqrt{5}$$

and

$$u_2(1+2u_1+u_2) = \frac{5}{9}$$

$$= [u_2(2+u_1+u_2)]^2$$

$$= u_2(2+u_1+u_2)(1+2u_1+u_2)(1-u_1-u_2)$$

(using the 2nd equation of the original pair)

Then $$1 = (2 + u_1 + u_2)(1 - u_1 - u_2)$$

and the reduced system consisting of this plus the second original equation, i.e.

$$u_2 (2 + u_1 + u_2) = (1 + 2u_1 + u_2)(1 - u_1 - u_2)$$
$$(2 + u_1 + u_2)(1 - u_1 - u_2) = 1$$

captures just the interior solution (and its conjugate) .

Clearly, what is needed are some unifying principles linking these and the other examples.

The ultimate goal is a procedure for splitting the dual equations into their simple sub-systems without prior knowledge of the solutions. Another worthy goal is a procedure to first test whether the equations possess an interior feasible solution. It is hoped that progress might follow from Group Theory (or Galois Theory), which currently offer tests to determine whether a polynomial is reducible [69].

D. Hill-climbing

D.1. Overview

Appendices B & C have explored two optimization methods which are based on calculus. Methods like these are sometimes classed as 'indirect' approaches [70], in the sense that they seek their optimum via the solution of what is actually a different (though related) problem--in this case, the solution of derivative equations. The chief advantage of such methods is that they offer the means of achieving the global optimum--provided all of the auxiliary solutions can be found. The chief disadvantage (as evident from those appendices) is that often finding even a single auxiliary solution may prove to be very difficult.

In contrast, hill-climbing is a 'direct' approach which employs a recurrence formula to make successive improvements to the objective function, until a local optimum is reached. Unless the objective 'landscape' is sufficiently well known, it is impossible to say whether that local optimum is also the global optimum--but at least a result is always forthcoming.

This chapter presents a simple but useful hill-climbing algorithm which is well-suited to f_1 and its fragments. Its description includes (1) the recurrence formula, (2) its convergence properties and (3) the choice of starting point. Afterwards, its effectiveness in capturing the global maximum is also briefly discussed.

D.2. Recurrence Formula

Consider the highly simplified objective

Maximize $\quad x_1^{p_{11}} x_2^{p_{21}} + x_1^{p_{12}} x_2^{p_{22}}$

subject to $\quad x_1 + x_2 = 1$

Define the function

$$f(x_1, x_2, \lambda) = x_1^{p_{11}} x_2^{p_{21}} + x_1^{p_{12}} x_2^{p_{22}} - \lambda (x_1 + x_2 - 1)$$

where λ is a Lagrange multiplier for the constraint.

After differentiating with respect to the three unknowns x_1 , x_2 and λ , and equating to zero, a stationary point occurs where

$$p_{11} x_1^{p_{11}-1} x_2^{p_{21}} + p_{12} x_1^{p_{12}-1} x_2^{p_{22}} = \lambda \tag{D.1}$$

$$p_{21} x_1^{p_{11}} x_2^{p_{21}-1} + p_{22} x_1^{p_{12}} x_2^{p_{22}-1} = \lambda \tag{D.2}$$

$$x_1 + x_2 = 1 \tag{D.3}$$

Multiplying (D.1) by x_1 gives

$$p_{11} x_1^{p_{11}} x_2^{p_{21}} + p_{12} x_1^{p_{12}} x_2^{p_{22}} = \lambda x_1 \tag{D.4}$$

Likewise, multiplying (D.2) by x_2 gives

$$p_{21}\, x_1^{p_{11}}\, x_2^{p_{21}} + p_{22}\, x_1^{p_{12}}\, x_2^{p_{22}} \quad = \lambda\, x_2 \tag{D.5}$$

By (D.4)

$$x_1 \quad = \frac{p_{11} x_1^{p_{11}} x_2^{p_{21}} + p_{12}\, x_1^{p_{11}} x_2^{p_{21}}}{\lambda}$$

However, by adding (D.4) and (D.5), and invoking (D.3)

$$\begin{aligned} \lambda &= \lambda\,(x_1 + x_2) \\ &= \left(p_{11} + p_{21}\right) x_1^{p_{11}}\, x_2^{p_{21}} + \left(p_{12} + p_{22}\right) x_1^{p_{12}}\, x_2^{p_{22}} \end{aligned} \tag{D.6}$$

Therefore,

$$x_1 \quad = \frac{p_{11}\, x_1^{p_{11}} x_2^{p_{21}} + p_{12}\, x_1^{p_{12}}\, x_2^{p_{22}}}{\left(p_{11} + p_{21}\right) x_1^{p_{11}} x_2^{p_{21}} + \left(p_{12} + p_{22}\right) x_1^{p_{12}}\, x_2^{p_{22}}} \tag{D.7}$$

Similarly,

$$x_2 \quad = \frac{p_{21}\, x_1^{p_{11}} x_2^{p_{21}} + p_{22}\, x_1^{p_{12}}\, x_2^{p_{22}}}{\left(p_{11} + p_{21}\right) x_1^{p_{11}} x_2^{p_{21}} + \left(p_{12} + p_{22}\right) x_1^{p_{12}}\, x_2^{p_{22}}} \tag{D.8}$$

Equations (D.7) & (D.8) define recurrence relations for x_1 and x_2. By substituting old values for these variables, i.e. $x_j^{(h)}$, into the formulas on the right, new values $x_j^{(h+1)}$ are produced on the left.

There are two schemes by which the values may be substituted [71]. According to the Gauss-Jacobi scheme, the values from generation (h) are used for calculating generation $(h+1)$ only after the complete set of $x_j^{(h)}$ are computed.

According to the Gauss-Seidel scheme, new values are substituted into the right as soon as each becomes available.

Using similar reasoning to that above, formulas (D.7) & (D.8) can be extended in various directions--

(A) To go from 2 to L terms :

$$f(x_1, x_2) = x_1^{p_{11}} x_2^{p_{21}} + x_1^{p_{12}} x_2^{p_{22}} + \cdots + x_1^{p_{1L}} x_2^{p_{2L}}$$

where $\quad x_1 + x_2 = 1$

Then

$$x_1 = \frac{p_{11} x_1^{p_{11}} x_2^{p_{21}} + p_{12} x_1^{p_{12}} x_2^{p_{22}} + \cdots + p_{1L} x_1^{p_{1L}} x_2^{p_{2L}}}{(p_{11}+p_{21}) x_1^{p_{11}} x_2^{p_{21}} + (p_{12}+p_{22}) x_1^{p_{12}} x_2^{p_{22}} + \cdots + (p_{1L}+p_{2L}) x_1^{p_{1L}} x_2^{p_{2L}}}$$

$$x_2 = \frac{p_{21} x_1^{p_{11}} x_2^{p_{21}} + p_{22} x_1^{p_{12}} x_2^{p_{22}} + \cdots + p_{2L} x_1^{p_{1L}} x_2^{p_{2L}}}{(p_{11}+p_{21}) x_1^{p_{11}} x_2^{p_{21}} + (p_{12}+p_{22}) x_1^{p_{12}} x_2^{p_{22}} + \cdots + (p_{1L}+p_{2L}) x_1^{p_{1L}} x_2^{p_{2L}}}$$

(B) To go from 2 to M variables :

$$f(x_1,\ x_2 \ldots x_M) = x_1^{p_{11}}\, x_2^{p_{21}} \ldots x_M^{p_{M1}} + x_1^{p_{12}}\, x_2^{p_{22}} \ldots x_M^{p_{M2}}$$

where $x_1 + x_2 + \ldots + x_M = 1$

Then

$$x_1 = \frac{p_{11}\, x_1^{p_{11}} x_2^{p_{21}} \ldots x_M^{p_{M1}} + p_{12}\, x_1^{p_{12}} x_2^{p_{22}} \ldots x_M^{p_{M2}}}{\left(p_{11} + p_{21} + \cdots + p_{M1}\right) x_1^{p_{11}} x_2^{p_{21}} \ldots x_M^{p_{M1}} + \left(p_{12} + p_{22} + \cdots + p_{M2}\right) x_1^{p_{12}} x_2^{p_{22}} \ldots x_M^{p_{M2}}}$$

$$x_M = \frac{p_{M1}\, x_1^{p_{11}} x_2^{p_{21}} \ldots x_M^{p_{M1}} + p_{M2}\, x_1^{p_{12}} x_2^{p_{22}} \ldots x_M^{p_{M2}}}{\left(p_{11} + p_{21} + \cdots + p_{M1}\right) x_1^{p_{11}} x_2^{p_{21}} \ldots x_M^{p_{M1}} + \left(p_{12} + p_{22} + \cdots + p_{M2}\right) x_1^{p_{12}} x_2^{p_{22}} \ldots x_M^{p_{M2}}}$$

(C) To go from 1 to N sets of variables :

$$f(x_1,\ x_2,\ y_1,\ y_2) = x_1^{p_{11}}\, x_2^{p_{21}}\, y_1^{q_{11}}\, y_2^{q_{21}} + x_1^{p_{12}}\, x_2^{p_{22}}\, y_1^{q_{12}}\, y_2^{q_{22}}$$

where

$$x_1 + x_2 = 1$$
$$y_1 + y_2 = 1$$

Then

$$x_1 = \frac{p_{11}\, x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + p_{12}\, x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}{\left(p_{11} + p_{21}\right) x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + \left(p_{12} + p_{22}\right) x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}$$

$$x_2 = \frac{p_{21} x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + p_{22} x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}{\left(p_{11} + p_{21}\right) x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + \left(p_{12} + p_{22}\right) x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}$$

$$y_1 = \frac{q_{11} x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + q_{12} x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}{\left(q_{11} + q_{21}\right) x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + \left(q_{12} + q_{22}\right) x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}$$

$$y_2 = \frac{q_{21} x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + q_{22} x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}{\left(q_{11} + q_{21}\right) x_1^{p_{11}} x_2^{p_{21}} y_1^{q_{11}} y_2^{q_{21}} + \left(q_{12} + q_{22}\right) x_1^{p_{12}} x_2^{p_{22}} y_1^{q_{12}} y_2^{q_{22}}}$$

Extending the problem in all three directions simultaneously, i.e. for N sets of variables, M_i variables per set ($i = 1..N$) and L terms, one arrives at a formulation which matches f_1 and its fragments :

Maximize $$\sum_{k=1}^{L} \prod_{i=1}^{N} \prod_{j=1}^{M_i} x_{ij}^{p_{ijk}} \qquad \text{(D.9)}$$

subject to the constraints

$$\sum_{j=1}^{M_i} x_{ij} = 1 \qquad (i = 1..N)$$

for which the recurrence formulas are

$$x_{ij}^{(h+1)} = \frac{\sum_{k=1}^{L} p_{ijk} \prod_{i=1}^{N} \prod_{j=1}^{M_i} \left[x_{ij}^{(h)} \right]^{p_{ijk}}}{\sum_{k=1}^{L} \left\{ \sum_{j=1}^{M_i} p_{ijk} \right\} \prod_{i=1}^{N} \prod_{j=1}^{M_i} \left[x_{ij}^{(h)} \right]^{p_{ijk}}}$$

$(i = 1..N)$

$(j = 1..M_i)$

$$= \frac{\sum_{k=1}^{L} p_{ijk} \times t_k^{(h)}}{\sum_{k=1}^{L} \left\{ \sum_{j=1}^{M_i} p_{ijk} \right\} \times t_k^{(h)}} \qquad \text{(D.10)}$$

where t_k represents term k of the objective, consisting of a product of powers of the j th unknown in constraint i

D.3. Convergence

The convergence properties of the recurrence formulas (D.10) have yet to be worked out in full detail. But there is evidence that the regions over which they converge are also the regions over which the objective function (D.9) is concave down... suggesting that the recurrence formulas carry out hill-climbing upon those functions.

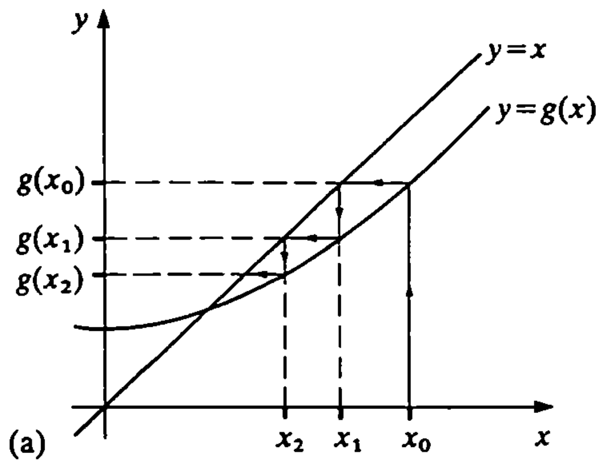
To keep the analysis simple, we restrict the discussion to functions of a single variable :

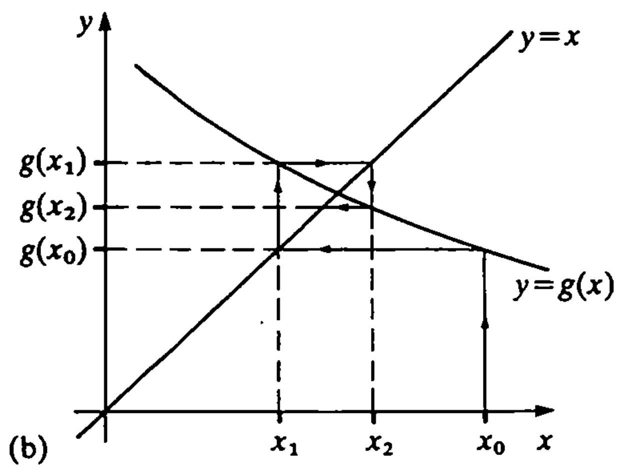
Let the general recurrence formula for a single variable be written

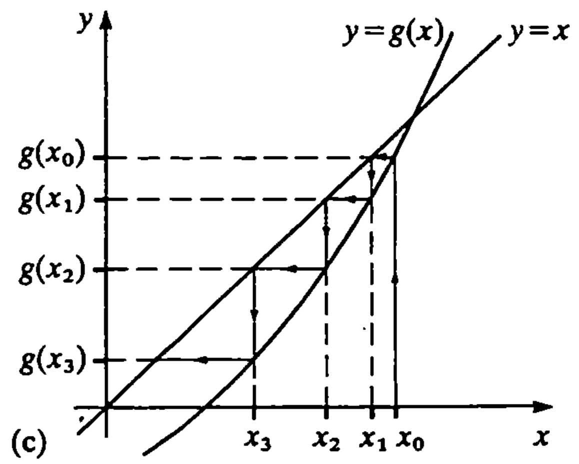
$$x^{(h+1)} = g(x^{(h)})$$

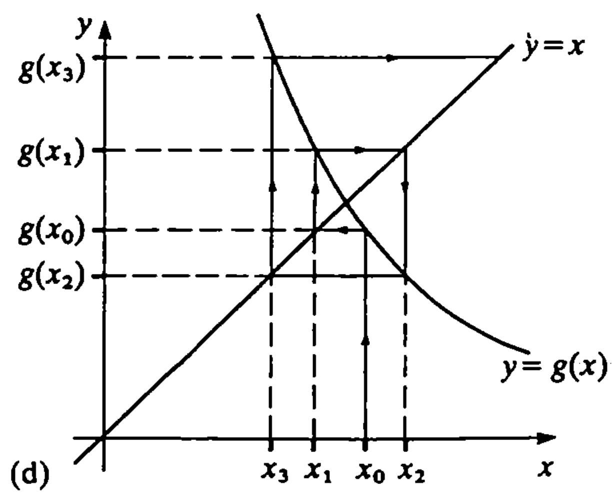
and let a represent a point of intersection between the functions $y = x$ and $y = g(x)$.

Then according to the theory [72], the iterations will converge to the point a , provided $g(x)$ performs a contraction mapping in a neighborhood of a . The 'staircase' and 'cobweb' diagrams (a) and (b) below illustrate that this condition is satisfied provided $| g'(a) | < 1$. On the other hand, diagrams (c) and (d) illustrate that divergence occurs whenever $| g'(a) | > 1$. (Illustrations reproduced from [73])


y
y=x
y=g(x)
g(x₀)
g(x₁)
g(x₂)
(a)
x₂
x₁
x₀
x


y
y=x
g(x₁)
g(x₂)
g(x₀)
y=g(x)
(b)
x₁
x₂
x₀
x


y
y=g(x)
y=x
g(x₀)
g(x₁)
g(x₂)
g(x₃)
(c)
x₃
x₂
x₁
x₀
x


y
y=x
g(x₃)
g(x₁)
g(x₀)
g(x₂)
y=g(x)
(d)
x₃
x₁
x₀
x₂
x

Now consider the original simplified objective function from Section D.2. Upon making the substitution $x_2 = 1 - x_1$ (and choosing simpler variable names), this objective can be written in terms of a single variable, as

$$f(x) = x^p (1 - x)^q + x^r (1 - x)^s$$

and the recurrence formula of Section D.2. becomes

$$g(x) = \frac{p\, x^p (1-x)^q + r\, x^r (1-x)^s}{(p+q)\, x^p (1-x)^q + (r+s)\, x^r (1-x)^s}$$

From the derivation of this formula it is clear that, if a is a stationary point of $f(x)$, then

$$a = g(a) .$$

implying that a is a point of intersection between the functions $y = x$ and $y = g(x)$.

Therefore, what we would like to show is that if, in addition,

$$f''(a) < 0 \qquad \text{(i.e. } f(x) \text{ is concave down, or max, at } a\text{)}$$

then

$$| g'(a) | < 1 .$$

Below we prove something <u>close</u> to that.

First, consider the landscape of $f(x)$.

Assuming that the exponents p, q, r, s are all $\geq$ 1, then $f(x)$ is a sum of two unimodal humps

$$f_a(x) = x^p (1 - x)^q \qquad \text{and} \qquad f_b(x) = x^r (1 - x)^s$$

with modes at $\frac{p}{p+q}$ and $\frac{r}{r+s}$, respectively.

Furthermore, assume $\frac{p}{p+q} < \frac{r}{r+s}$ (i.e. f_a to the left of f_b),

in which case the separation between the modes is equal to

$$\frac{1}{(p+q)(r+s)} [qr - ps]$$

Clearly, the function $f(x)$

either has a single stationary point (single max)
or has three stationary points (a min between two max's)

depending upon the separation between f_a and f_b .

All of these stationary points (however many)

lie to the right of $\frac{p}{p+q}$ (where f_a has a negative slope)

and to the left of $\frac{r}{r+s}$ (where f_b has a positive slope) .

If there are three stationary points, the left one lies near $\frac{p}{p+q}$ to a first approximation, and the right one lies near $\frac{r}{r+s}$.

To approximate the middle (or the single) stationary point a , assume that it occurs near the point where f_a and f_b cross each other*

Then

$$a^p(1-a)^q \overset{\text{approx}}{=} a^r(1-a)^s$$

and also

$$\begin{aligned} 0 &= f'(a) \\ &= a^p(1-a)^q\left[\frac{p}{a} - \frac{q}{(1-a)}\right] + a^r(1-a)^s\left[\frac{r}{a} - \frac{s}{(1-a)}\right] \\ &\overset{\text{approx}}{=} a^p(1-a)^q\left[\frac{p}{a} - \frac{q}{(1-a)}\right] + a^p(1-a)^q\left[\frac{r}{a} - \frac{s}{(1-a)}\right] \\ &= a^p(1-a)^q\left[\frac{p+r}{a} - \frac{q+s}{(1-a)}\right] \end{aligned}$$

from which

$$\frac{p+r}{a} \overset{\text{approx}}{=} \frac{q+s}{(1-a)}$$

and

$$a \overset{\text{approx}}{=} \frac{p+r}{p+q+r+s}$$

* The approximation is only exact when f_a and f_b are symmetrical mirror images.

In addition, consider $f''(x)$. This is

$$f''(x) = x^p(1-x)^q\left\{\left[\frac{p}{x}-\frac{q}{(1-x)}\right]^2-\left[\frac{p}{x^2}-\frac{q}{(1-x)^2}\right]\right\}$$

$$+ \; x^r(1-x)^s\left\{\left[\frac{r}{x}-\frac{s}{(1-x)}\right]^2-\left[\frac{r}{x^2}-\frac{s}{(1-x)^2}\right]\right\}$$

and, after manipulation, the requirement that $f''(a) < 0$ at some stationary point a becomes equivalent to

$$[\,r(1-a)-s\,a]\,[\,p(1-a)-q\,a]\left\{\left[\frac{p}{a}-\frac{q}{(1-a)}\right]-\left[\frac{r}{a}-\frac{s}{(1-a)}\right]\right\} < [\,qr-ps\,]$$

Assuming that the middle stationary point occurs <u>at</u> (and not merely near)

$$a = \frac{p+r}{p+q+r+s}$$

then, after substitution, $f''(a) < 0$ iff

$$\left[\frac{qr-ps}{(p+r)(q+s)}\right]^2 < \frac{1}{2}\frac{(p+q+r+s)}{(p+r)(q+s)}$$

It has always been intuitively clear that f has a single maximum provided the modes of f_a and f_b were sufficiently close, and now this result gives an idea of <u>how</u> close--namely, if their separation is less than

$$\frac{1}{2}\frac{(p+q+r+s)(p+r)(q+s)}{(qr-ps)(p+q)(r+s)}$$

Finally, consider $|g'(x)|$.

After more pages of manipulation, it can be shown that $|g'(x)|$ is a fraction with numerator

$$\{f_a(x)\, f_b'(x) - f_a'(x)\, f_b(x)\}\,[qr - ps]$$

$$= \quad x^p (1-x)^q x^r (1-x)^s \left\{\left[\frac{r}{x} - \frac{s}{(1-x)}\right] - \left[\frac{p}{x} - \frac{q}{(1-x)}\right]\right\} [qr - ps]$$

and denominator

$$[\,(p+q)\; x^p (1-x)^q \;+\; (r+s)\, x^r (1-x)^s\,]^2$$

Using the two facts that, at any stationary point a

$$f_a'(a) \;=\; -f_b'(a)$$

and also

$$a \;=\; \frac{p\, a^p\, (1-a)^q + r\, a^r\, (1-a)^s}{(p+q)\, a^p\, (1-a)^q + (r+s)\, a^r\, (1-a)^s}$$

the condition that $|g'(a)| < 1$ becomes

$$f(a)\; f_b'(a)\;[\,qr - ps\,] \;<\; [\,p\, a^{p-1} (1-a)^q \;+\; r\, a^{r-1} (1-a)^s\,]^2$$

Staying with the middle stationary point (which is the interesting one), and assuming once more that it occurs at (and not merely near)

$$a = \frac{p+r}{p+q+r+s}$$

Then, after substitution, $|g'(a)| < 1$ iff

$$\left[\frac{qr - ps}{(p+r)(q+s)}\right]^2 < \frac{1}{2}\frac{(p+q+r+s)}{(p+r)(q+s)}$$

which is identical to the criterion for $f''(a) < 0$.

In summary, the middle stationary point is either a maximum or a minimum depending upon the separation of the humps. In those cases, at least, where the humps are symmetrical mirror-images, so that the location of the stationary point is given accurately by

$$a = \frac{p+r}{p+q+r+s}$$

then

$$|g'(a)| < 1 \quad \text{iff} \quad f''(a) < 0$$

which implies that the recurrence formula converges when the stationary point occurs on a hill rather than a valley.

D.4. Starting Points

The third main issue connected with any iterative optimization scheme is the choice of starting point.

A hill-climbing algorithm which is applied to a closed, bounded solution space is guaranteed to produce a locally optimum solution by climbing to the nearest hill-top or to the nearest boundary [74,75]. But whether this solution is also the global optimum depends upon whether the starting point was fortunate enough to be located on the tallest hill. In general, unless the objective landscape is sufficiently well known (i.e. the location or, at least, the number of hills), it is impossible to be confident of a global maximum [76].

The purpose of this section and the next is to describe a number of possible schemes for generating starting points, along with their effectiveness in providing the global maximum.

Recall from Chapter 3 how each fragment of f_1 corresponds to a subset of the solution space, and how that sub-space is spanned either by interior term(s) in the case of a compound fragment or by boundary terms in the case of a simple union. One characteristic* of recurrence formulas (D.10) is that, while a starting point which is within the interior of the sub-space may converge to the boundary, a starting point which is on the boundary can never move into the interior. Once a boundary point, always a boundary point. Consequently, in optimizing these fragments it is necessary to choose a starting point from the interior.

In the case where a fragment includes interior term(s), a natural choice of starting point is the modal position of one of these. Below we consider different options when the fragment includes no interior term and a starting point must be nominated.

* The reasoning goes like this :

A starting point lies on a boundary of the sub-space if it assigns the value zero to a variable that appears in at least one term of the fragment. Moving from the boundary into the interior requires that this variable subsequently assume a non-zero value.

Now consider the recurrence formula

$$x_{ij}^{(h+1)} = \frac{\sum_{k=1}^{L} p_{ijk} \times t_k^{(h)}}{\sum_{k=1}^{L} \left\{ \sum_{j=1}^{M_i} p_{ijk} \right\} \times t_k^{(h)}}$$

where t_k is the k^{th} term of the fragment.

If a variable equals zero, then only the terms which don't include that variable can escape being zero also... but for these terms $p_{ijk} = 0$. One way or the other, the numerator of the formula is destined to remain a sum of zeros.

(a) The easiest, and perhaps most obvious, starting point is at the simple center of the full solution space, where all of the matrix elements are equal.

For example, if A is an $N \times N$ matrix and B is $N \times M$, then simply let all of the $a_{ij} = \frac{1}{N}$ and all of the $b_{jk} = \frac{1}{M}$.

(b) Another easy option is the simple center of the sub-space spanned by the fragment. This is similar to (a), except that the fractions are adjusted to reflect the number of elements from each row which participate in the fragment.

For example, take the input sequence $k\,k\,l\,k$ and fragment $(6\ \ 7\ \ 8)$:

$$(a_{12})^2\, a_{21}\, b_{1k}\, b_{1l}\, (b_{2k})^2 \;+\; a_{12}\, a_{21}\, a_{22}\, (b_{1k})^2\, b_{2k}\, b_{2l} \;+\; a_{12}\, (a_{22})^2\, b_{1k}\, (b_{2k})^2\, b_{2l}$$

Mark position (i, j) of the auxiliary matrix with a 1 if x_{ij} appears anywhere in this expression, or with a 0 otherwise...

$$A^* = \begin{array}{|c|c|c} \hline & & \text{Row Sum} \\ \hline 0 & 1 & 1 \\ \hline 1 & 1 & 2 \\ \hline \end{array} \qquad B^* = \begin{array}{|c|c|c} \hline & & \text{Row Sum} \\ \hline 1 & 1 & 2 \\ \hline 1 & 1 & 2 \\ \hline \end{array}$$

...then divide each element by its respective row sum.

(c) A second refinement is to mark each position, not with a 1 or 0 as in (b), but with a tally of the times x_{ij} appears throughout the fragment. Note that this is identical to the procedure for finding the modal position of a single term explained in Section 3.3., except that now all of the terms are included in the count instead of just the one term.

This scheme actually has several features to recommend it :

(i) There is a 'natural' connection with the given recurrence formulas in the sense that, if 1 is substituted for each of the x_{ij} in those formulas, this is the set of values that results.

This implies that the simple center given by option (a) (and usually the local center of option (b) also) are transformed into the 'pseudo-mode' of option (c) after a single iteration of the formulas, anyway.

(ii) Even more interesting, recall from the discussion within Geometric Programming how a primal solution space where

$$\sum_j x_{ij} \leq 1 \qquad (i = 1..N)$$

becomes transformed into a dual u-space where all of the unknowns satisfy a single constraint of the form

$$u_1 + .. + u_D \leq 1$$

This u solution space is a D-dimensional (hyper-)triangle whose centroid, or center of gravity, occurs at

$$u_1 = .. = u_D = \frac{1}{D+1}$$

If these values for u_d are back-substituted, one discovers that the primal values which map to the dual centroid are those given by our 'pseudo-mode' !

(iii) In actual trials, a critical point of the objective function (fragment) seems to occur at the pseudo-mode an impressive number of times.

(d) We have just observed that the pseudo-mode values can be derived from the recurrence formulas

$$x_{ij}^{(h+1)} = \frac{\sum_{k=1}^{L} p_{ijk} \times t_k^{(h)}}{\sum_{k=1}^{L} \left\{ \sum_{j=1}^{M_i} p_{ijk} \right\} \times t_k^{(h)}}$$

by substituting $t_k = 1$ for all k.

A further option is to substitute the value of each term (height of the uni-modal hump) at its modal position.

This produces a type of weighted pseudo-mode.

(e) A somewhat different approach is to regard each term as a 'point mass' which is located at the mode of the hump and which has a mass equal to the peak value of the hump (the value of the weight used in (d)) .

The starting point is then calculated as the center of gravity of this discrete system along the lines of the classic formula [77]

$$\bar{x} = \frac{\sum_i \{(x\text{-coordinate of point } i) \times (\text{mass of point } i)\}}{\sum_i (\text{mass of point } i)}$$

The result is

$$x_{ij}^{(0)} = \frac{\sum_{k=1}^{L} \left\{ \frac{p_{ijk}}{\sum_{j=1}^{M_i} p_{ijk}} \right\} \times w_k}{\sum_{k=1}^{L} w_k}$$

where w_k is the peak value of the k^{th} term

$$= \prod_{i=1}^{N} \prod_{j=1}^{M_i} \left[\frac{p_{ijk}}{\sum_{j=1}^{M_i} p_{ijk}} \right]^{p_{ijk}}$$

(f) The previous formula only approximates the center of gravity of the fragment since it replaces a continuous shape (hump) with a point (mode of hump), and a continuous density with the peak value of the hump.

Starting from the classic formula for a continuous system [78]

$$\bar{x} = \frac{\int x f(x)\, dx}{\int f(x)\, dx}$$

the true center of gravity looks similar in form to the discrete result

$$x_{ij}^{(0)} = \frac{\sum_{k=1}^{L} \left\{ \frac{\left(1+p_{ijk}\right)}{\sum_{j=1}^{M_i} \left(1+p_{ijk}\right)} \right\} \times w_k}{\sum_{k=1}^{L} w_k}$$

However, this time

$$w_k = \prod_{i=1}^{N} \mathrm{B}(\,(1+p_{i1k}),\,(1+p_{i2k}),\,\ldots\,) \qquad \text{list of } M_i \text{ elements}$$

where $\mathrm{B}(p_1, p_2 \ldots p_M)$ is the generalized 'beta' function found by integrating $x_1^{p_1-1}\, x_2^{p_2-1} \ldots x_M^{p_M-1}$ over the hyperplane $x_1 + x_2 + \ldots + x_M = 1$

$$= \frac{\prod_{j=1}^{M} \left(p_j - 1\right)!}{\left(\left\{\sum_{j=1}^{M} p_j\right\} - 1\right)!} \qquad \text{(compare with Section 3.5.)}$$

D.5. Global Effectiveness

Finally, a word about these starting points relative to the global maximum :

In tests, it is observed that

(1) When the global maximum occurs within the interior of the fragment, virtually any starting point which is also within the interior will find it-- including the modal position of interior term(s) as well as any of the schemes (a) through (f) above, whereas

(2) When the global maximum occurs at a boundary of the fragment, <u>none</u> of the schemes (a) through (f) above is 100 percent effective in finding it.

Result (2) is disappointing but perhaps not surprising. However what is somewhat surprising is result (1) . It tends to suggest that the 'landscape' of the fragments is not as complicated as might be assumed. For example, there is evidence that

(3) No fragment ever exhibits more than one interior peak, even though the fragment contains more than one interior term, and also

(4) If an interior peak is present, it provides the global maximum

To shed some light on these observations, recall the simplified objective function of Section D.3.

$$f(x) = x^p(1-x)^q + x^r(1-x)^s$$

consisting of the two unimodal humps

$$f_a(x) = x^p (1 - x)^q \qquad \text{and} \qquad f_b(x) = x^r (1 - x)^s$$

The function

either has a single stationary point (single max)
or has three stationary points (a min between two max's)

depending upon the separation between f_a and f_b
as manifested by the difference between their exponents.

Now consider the observation sequence $k\,k\,k\,l$ and fragment (2 4 8) :

$$(a_{11})^2 a_{12} (b_{1k})^3 b_{2l} + a_{11} a_{12} a_{22} (b_{1k})^2 b_{2k} b_{2l} + a_{12} (a_{22})^2 b_{1k} (b_{2k})^2 b_{2l}$$

This fragment has an interior maximum coincident with the modal position of its middle term, at $a_{11} = b_{2k} = \frac{1}{2}$.

Re-writing this fragment for clarity as

$$x^2 (1 - x) (1 - y) + x (1 - x) y (1 - y) + (1 - x) y^2 (1 - y)$$

observe how, as one moves from term to term, there is a smooth progression (increase/decrease) in the exponents of x (and likewise for y).

This situation is typical of all known fragments containing interior maxima. The terms form a sort of continuous chain in the sense that, relating each consecutive pair of terms to f_a and f_b above, there is not enough 'daylight' between their exponents to permit two separate humps to appear.

E. Source Code for <u>N</u>ew <u>H</u>MM <u>B</u>uilding Algorithm

```
/*  NHB.c :      New HMM building algorithm which examines the neighbors  */
/*               of the singletons  */
/*  B.F. McKee   09/11/98  */


/*  Include ANSI C standard libraries  */

#include        <stdio.h>
#include        <string.h>
#include        <stdlib.h>


/*  Include local header info  */

struct  snode
{
        int             *sequence ;
        struct snode    *next ;
} ;

typedef struct snode    svec ;          /* Simple list of state sequences */
typedef svec            *slist ;

struct  book
{
        int             planes ;
        int             *rows ;
        int             **cols ;
        int             ***value ;
} ;

typedef struct book     block ;


void    ProcessArguments        () ;
int     UniqueCount             () ;
int     *SymbolCode             () ;
int     *SymbolCount            () ;
void    CreateGlobals           () ;
void    MakeMasterList          () ;
int     NoEclipse               () ;
int     Singleton               () ;
int     NoFit                   () ;
int     NoAct                   () ;
void    ClimbHills              () ;
slist   NeighborsOf             () ;
void    Circumscribe            () ;
void    SAppend                 () ;
int     SLocate                 () ;
void    LAppend                 () ;
int     LengthOf                () ;
block   *BuildStateBlock        () ;
block   *BuildBinsBlock         () ;
block   *BuildBinsPlusOneBlock  () ;
int     **PermutationsOf        () ;
int     Perms                   () ;
int     Splits                  () ;
void    Item                    () ;
void    Enum                    () ;
int     **Flatten               () ;
```

```
void    Nest            () ;
void    Process         () ;
int     *Merge          () ;
void    AppendNeighborsOf () ;
int     CountRows       () ;
int     **MakeAMat      () ;
int     **MakeBMat      () ;
int     Distance        () ;
int     **Permute       () ;
void    IAbsorb         () ;
void    DAbsorb         () ;
void    RecycleBlock    () ;
void    RecyclePair     () ;
void    RecycleSList    () ;
double  Contribution    () ;
double  **Convert       () ;
long    Fact            () ;
void    Climb           () ;
void    CreateVars      () ;
void    InitVars        () ;
void    CalcCOG         () ;
void    RecordOld       () ;
double  CalcNew         () ;
double  FullValue       () ;
void    DisplayMats     () ;
void    ReleaseVars     () ;



/*  Global defines  */



/*  Global variables  */

FILE    *fp ;                   /* output file pointer
*/
char    **arg ;                 /* argv copy
*/

int     N ;                     /* number of states
*/
int     T ;                     /* length of symbol sequence
*/
int     M ;                     /* count of unique symbols in sequence
*/

int     *Symbol ;               /* symbol column assignment in B-matrix
*/
int     *Tally ;                /* tally of symbols in sequence
*/

double  **AMax ;
double  **BMax ;
double  FMax ;

slist   List ;
```

```
/*  Main program starts here  */

void    main(argc, argv)
        int     argc ;
        char    **argv ;
{

        if ((fp = fopen("dump.c","w")) == NULL)
        {
                printf("\nCan't open dump file\n") ;
                exit(1) ;
        }

        arg = argv ;
        ProcessArguments() ;
        CreateGlobals() ;
        MakeMasterList() ;
        ClimbHills() ;
        DisplayMats(AMax, BMax, FMax) ;
        fprintf(fp, "\n\n") ;
        fclose(fp) ;
        system("cat dump.c") ;
}

/*  Abstract argument information into useful global variables  */

void    ProcessArguments()
{
        int     i ;

        fprintf(fp, "\n\n%s : ", arg[1]) ;

        T = strlen(arg[1]) ;
        M = UniqueCount(arg[1]) ;

        Symbol = (int *) malloc(T * sizeof(int)) ;
        Symbol = SymbolCode(arg[1], Symbol) ;

        Tally = (int *) malloc(M * sizeof(int)) ;
        for (i=0 ; i<M ; ++i)
                Tally[i] = 0 ;
        Tally = SymbolCount(Symbol, Tally) ;
}

/*  Count the number of different characters in string  */

int     UniqueCount(s)
        char    *s ;
{
        char    *t ;
        int     k, n ;

        n = strlen(s) ;
        t = (char *) calloc(n+1, sizeof(char)) ;
        strcpy(t,s) ;
```

```
        t[1] = NULL ;

        k = strspn(s,t) ;
        while( k != n )
        {
                t = strncat(t, s+k, 1) ;
                k = strspn(s,t) ;
        }

        return(strlen(t)) ;
}

/*  Allocate symbols to columns  */

int     *SymbolCode(s, Symbol)
        char    *s ;
        int     *Symbol;
{
        int     t, j ;

        for (t=0 ; t<T ; ++t)
        {
                j = s[t] ;
                Symbol[t] = j - 'a' ;
        }
        return(Symbol) ;
}

/*  Count numbers of symbols  */

int     *SymbolCount(Symbol, Tally)
        int     *Symbol ;
        int     *Tally ;
{
        int     s, t ;

        for (t=0 ; t<T ; ++t)
        {
                s = Symbol[t] ;
                ++Tally[s] ;
        }
        return(Tally) ;
}

/*  Create additional global variables  */

void    CreateGlobals()
{
        int     i ;

        N = 3 ;                         /* N! must be integer */
/*      if ( N > M )
                N = M ;                 */

        AMax = (double **) malloc(N * sizeof(double *)) ;
        BMax = (double **) malloc(N * sizeof(double *)) ;
        for (i=0 ; i<N ; ++i)
        {
                AMax[i] = (double *) malloc(N * sizeof(double)) ;
                BMax[i] = (double *) malloc(M * sizeof(double)) ;
        }
        FMax = 0.0 ;
}

/*  Grow the list of singletons generation by generation  */

void    MakeMasterList()
{
        int     i, n, t ;
        int     *Term ;
        slist   Head, Tail, Current, Parent ;

        Head = Tail = NULL ;
        Term = (int *) malloc(T * sizeof(int)) ;
        for (t=0 ; t<T ; ++t)
                Term[t] = 0 ;
        SAppend(Term, &Head, &Tail) ;
        free(Term) ;

        for (t=1 ; t<T ; ++t)
        {
                Current = Head ;
                Head = Tail = NULL ;
                while ( Current != NULL )
                {
                        Parent = Current ;
                        Term = Parent->sequence ;
                        n = 0 ;
                        for (i=0 ; i<t ; ++i)
                                if ( Term[i] > n )
                                        n = Term[i] ;
                        if ( (n+2) > N )
                                n = N ;
                        else
                                n = n+2 ;
                        for (i=0 ; i<n ; ++i)
                        {
                                Term[t] = i ;
                                SAppend(Term, &Head, &Tail) ;
                        }
                        Current = Parent->next ;
                        free(Parent->sequence) ;
                        free(Parent) ;
                }

                Current = Parent = Head ;
                Head = Tail = NULL ;
                while ( Current != NULL )
                {
                        if ( NoEclipse(Current, Parent, t) )
                                SAppend(Current->sequence, &Head, &Tail) ;
                        Current = Current->next ;
                }
                RecycleSList(Parent) ;
        }
```

```
        Current = Parent = Head ;
        Head = Tail = NULL ;
        while ( Current != NULL )
        {
                if ( Singleton(Current, Parent, T-1) )
                        SAppend(Current->sequence, &Head, &Tail) ;
                Current = Current->next ;
        }
        RecycleSList(Parent) ;
        List = Head ;
}

/*  Test whether sequence eclipses another with same final state  */

int     NoEclipse(Node, Head, t)
        slist   Node ;
        slist   Head ;
        int     t ;
{
        int     nonefound, moreleft ;
        slist   Current ;

        Current = Head ;
        nonefound = moreleft = 1 ;
        while ( nonefound && moreleft )
        {
                if ( Current == NULL )
                        moreleft = 0 ;
                else if ( Current == Node )
                        Current = Current->next ;
                else if ( Current->sequence[t] != Node->sequence[t] )
                        Current = Current->next ;
                else if ( NoFit(Node->sequence, Current->sequence, t) )
                        Current = Current->next ;
                else
                        nonefound = 0 ;
        }
        return(nonefound) ;
}

/*  Test whether sequence activates another in list  */

int     Singleton(Node, Head, t)
        slist   Node ;
        slist   Head ;
        int     t ;
{
        int     nonefound, moreleft ;
        slist   Current ;

        Current = Head ;
        nonefound = moreleft = 1 ;
        while ( nonefound && moreleft )
        {
                if ( Current == NULL )
                        moreleft = 0 ;
                else if ( Current == Node )
                        Current = Current->next ;
                else if ( NoAct(Node->sequence, Current->sequence, t) )
                        Current = Current->next ;
                else
                        nonefound = 0 ;
        }
        return(nonefound) ;
}

/*  Test whether term1 eclipses term2 and return 1 for failure  */

int     NoFit(term1, term2, t)
        int     *term1 ;
        int     *term2 ;
        int     t ;
{
        int     i, j ;
        int     sw ;
        int     **Mat1, **Mat2 ;

        Mat1 = MakeAMat(term1, t+1) ;
        Mat2 = MakeAMat(term2, t+1) ;

        sw = i = 0 ;
        while ( ( !sw ) && ( i<N ) )
        {
                j = 0 ;
                while ( ( !sw ) && ( j<N ) )
                {
                        if ( ( Mat1[i][j] == 0 ) && ( Mat2[i][j] != 0 ) )
                                sw = 1 ;
                        else
                                ++j ;
                }
                if ( !sw )      ++i ;
        }
        IAbsorb(Mat1) ;
        IAbsorb(Mat2) ;

        if ( sw )       return(1) ;
        else
        {
                Mat1 = MakeBMat(term1, t+1) ;
                Mat2 = MakeBMat(term2, t+1) ;

                i = 0 ;
                while ( ( !sw ) && ( i<N ) )
                {
                        j = 0 ;
                        while ( ( !sw ) && ( j<M ) )
                        {
                                if ( ( Mat1[i][j] == 0 ) && ( Mat2[i][j] != 0 )
)
                                        sw = 1 ;
                                else
                                        ++j ;
                        }
                        if ( !sw )      ++i ;
                }
                IAbsorb(Mat1) ;
```

```
                IAbsorb(Mat2) ;
                return(sw) ;
        }
}

/*  Test whether term1 activates term2 and return 1 for failure  */

int     NoAct(term1, term2, t)
        int     *term1 ;
        int     *term2 ;
        int     t ;
{
        int     i, j ;
        int     sw ;
        int     *rowsum ;
        int     **Mat1, **Mat2 ;

        Mat1 = MakeAMat(term1, t+1) ;
        Mat2 = MakeAMat(term2, t+1) ;
        rowsum = (int *) malloc(N * sizeof(int)) ;
        for (i=0 ; i<N ; ++i)
        {
                rowsum[i] = 0 ;
                for (j=0 ; j<N ; ++j)
                        if ( Mat1[i][j] != 0 )
                                rowsum[i] = rowsum[i] + Mat1[i][j] ;
        }

        sw = i = 0 ;
        while ( ( !sw ) && ( i<N ) )
        {
                j = 0 ;
                while ( ( !sw ) && ( j<N ) )
                {
                        if ( ( Mat1[i][j] == 0 ) && ( Mat2[i][j] != 0 ) && ( row
sum[i] != 0 ) )
                                sw = 1 ;
                        else
                                ++j ;
                }
                if ( !sw )      ++i ;
        }
        IAbsorb(Mat1) ;
        IAbsorb(Mat2) ;
        free(rowsum) ;

        if ( sw )       return(1) ;
        else
        {
                Mat1 = MakeBMat(term1, t+1) ;
                Mat2 = MakeBMat(term2, t+1) ;
                rowsum = (int *) malloc(N * sizeof(int)) ;
                for (i=0 ; i<N ; ++i)
                {
                        rowsum[i] = 0 ;
                        for (j=0 ; j<N ; ++j)
                                if ( Mat1[i][j] != 0 )
                                        rowsum[i] = rowsum[i] + Mat1[i][j] ;
                }
                i = 0 ;
                while ( ( !sw ) && ( i<N ) )
                {
                        j = 0 ;
                        while ( ( !sw ) && ( j<M ) )
                        {
                                if ( ( Mat1[i][j] == 0 ) && ( Mat2[i][j] != 0 )
&& ( rowsum[i] != 0 ) )
                                        sw = 1 ;
                                else
                                        ++j ;
                        }
                        if ( !sw )      ++i ;
                }
                IAbsorb(Mat1) ;
                IAbsorb(Mat2) ;
                free(rowsum) ;
                return(sw) ;
        }
}

/*  For each valid term, climb among activations/neighbors  */

void    ClimbHills()
{
        int     **AMat, **BMat ;
        double  ACon, BCon, FMod ;
        slist   Neighbors, Current ;

        Current = List ;
        while ( Current != NULL )
        {

                AMat = MakeAMat(Current->sequence, T) ;
                ACon = Contribution(AMat, N, N) ;
                BMat = MakeBMat(Current->sequence, T) ;
                BCon = Contribution(BMat, N, M) ;
                FMod = ACon * BCon ;

                if ( FMod > FMax )
                {
                        AMax = Convert(AMat, AMax, N, N) ;
                        BMax = Convert(BMat, BMax, N, M) ;
                        FMax = FMod ;
                }
                IAbsorb(AMat) ;
                IAbsorb(BMat) ;
                Current = Current->next ;
        }

        while ( List != NULL )
        {

                Current = List ;
                Neighbors = NeighborsOf(Current->sequence) ;
                if ( LengthOf(Neighbors) != 1 )
                        Climb(Neighbors, 0) ;
                RecycleSList(Neighbors) ;
                List = Current->next ;
```

```
                        free(Current->sequence) ;
                        free(Current) ;
                }
}


/*  Return linked list of immediate neighbors of the given term  */

slist   NeighborsOf(Term)
        int     *Term ;
{
        int     **AMat, **BMat, **FMat ;
        int     i, j, k, l, m ;
        int     *lim ;
        slist   Head, Tail ;
        block   *NBlock ;

        Head = Tail = NULL ;
        SAppend(Term, &Head, &Tail) ;

        AMat = MakeAMat(Term, T) ; BMat = MakeBMat(Term, T) ;
        FMat = (int **) malloc(N * sizeof(int *)) ;
        for (i=0 ; i<N ; ++i)
        {
                FMat[i] = (int *) malloc(M * sizeof(int)) ;
                for (j=0 ; j<M ; ++j)
                        FMat[i][j] = BMat[i][j] ;
        }

        NBlock = BuildBinsPlusOneBlock(BMat) ;
        lim = NBlock->rows ;

        for (m=0 ; m<M ; ++m)
                for (l=0 ; l<lim[m] ; ++l)
                {
                        k = 0 ;
                        j = m ;
                        for (i=0 ; i<N ; ++i)
                                if ( BMat[i][j] != 0 )
                                {
                                        FMat[i][j] = NBlock->value[m][l][k] ;
                                        ++k ;
                                }
                        for (i=0 ; i<N ; ++i)
                                if ( BMat[i][j] == 0 )
                                {
                                        FMat[i][j] = NBlock->value[m][l][k] ;
                                        Circumscribe(AMat, BMat, FMat, &Head, &T
ail) ;
                                        FMat[i][j] = 0 ;
                                }
                        for (i=0 ; i<N ; ++i)
                                FMat[i][j] = BMat[i][j] ;
                }

        IAbsorb(AMat) ; IAbsorb(BMat) ; IAbsorb(FMat) ;
        RecycleBlock(NBlock) ;
        return(Head) ;
}
```

```
/*  Examine neighborhood of term  */

void    Circumscribe(AMat, BMat, FMat, Head, Tail)
        int     **AMat ;
        int     **BMat ;
        int     **FMat ;
        slist   *Head ;
        slist   *Tail ;
{
        int     level, *vec, *lim ;
        block   *NBlock ;

        NBlock = BuildBinsBlock(FMat) ;
        level = 0 ;
        vec = (int *) malloc(M * sizeof(int)) ;
        lim = NBlock->rows ;

        Process(AMat, BMat, FMat, NBlock, level, vec, lim, Head, Tail) ;

        RecycleBlock(NBlock) ;
}

/*  Adds State to end of linked slist  */

void    SAppend(State, Head, Tail)
        int     *State ;
        slist   *Head ;
        slist   *Tail ;
{
        int     i ;
        slist   node ;

        node = (svec *) malloc(sizeof(svec)) ;
        node->next = NULL ;

        node->sequence = (int *) malloc(T * sizeof(int)) ;
        for (i=0 ; i<T ; ++i)
                node->sequence[i] = State[i] ;

        if ( *Head == NULL )
                *Head = *Tail = node ;
        else
        {
                (*Tail)->next = node ;
                *Tail = node ;
        }
}

/*  Find state sequence in a linked list of state sequences  */

int     SLocate(State, List)
        int     *State ;
        slist   List ;
{
        int     i, sw, result ;

        result = 0 ;
```

```
        if ( List != NULL )
        {
                i = sw = 0 ;
                while ( ( !sw ) && ( i<T ) )
                        if ( State[i] != List->sequence[i] )
                                sw = 1 ;
                        else
                                ++i ;
                if ( !sw )
                        result = 1 ;
                else
                {
                        result = SLocate(State, List->next) ;
                        if ( result != 0 )
                                ++result ;
                }
        }
        return(result) ;
}

/*  Add State to end of linked slist, avoiding duplicates  */

void    LAppend(State, Head, Tail)
        int     *State ;
        slist   *Head ;
        slist   *Tail ;
{
        if ( SLocate(State, *Head) == 0 )
                SAppend(State, Head, Tail) ;
}

/*  Determine length of a linked list  */

int     LengthOf(List)
        slist   List ;
{
        if ( List == NULL )
                return(0) ;
        else
                return(1+ LengthOf(List->next)) ;
}

/*  Assembles state permutations into 3-D block, one plane per col of BMat  */

block   *BuildStateBlock(BMat)
        int     **BMat ;
{
        int     i, j ;
        int     sum, *rep, len, *vec, **mat, next, lim ;
        block   *result ;

        result = (block *) malloc(sizeof(block)) ;
        result->planes = M ;
        result->rows = (int *) malloc(M * sizeof(int)) ;
        result->cols = (int **) malloc(M * sizeof(int *)) ;
        result->value = (int ***) malloc(M * sizeof(int **)) ;
        rep = (int *) malloc(N * sizeof(int)) ;
        for (j=0 ; j<M ; ++j)
        {
                sum = 0 ;
                for (i=0 ; i<N ; ++i)
                {
                        rep[i] = BMat[i][j] ;
                        sum = sum + rep[i] ;
                }
                len = sum ;
                if ( j == 0 )
                {
                        --sum ;
                        --rep[0] ;
                }
                lim = Perms(sum, rep) ;
                next = 0 ;
                vec = (int *) malloc(len * sizeof(int)) ;
                mat = (int **) malloc(lim * sizeof(int *)) ;

                Item(sum, rep, len, vec, mat, &next) ;
                if ( j == 0 )
                        for (i=0 ; i<lim ; ++i)
                                mat[i][0] = 0 ;

                result->rows[j] = lim ;
                result->cols[j] = (int *) malloc(lim * sizeof(int)) ;
                for (i=0 ; i<lim ; ++i)
                        result->cols[j][i] = len ;
                result->value[j] = mat ;
                free(vec) ;
        }
        return(result) ;
}

/*  Assembles symbol allocations into 3-D block, one plane per col of BMat  */

block   *BuildBinsBlock(BMat)
        int     **BMat ;
{
        int     i, j ;
        int     sum, *bins, len, *vec, **mat, next, lim ;
        block   *result ;

        result = (block *) malloc(sizeof(block)) ;
        result->planes = M ;
        result->rows = (int *) malloc(M * sizeof(int)) ;
        result->cols = (int **) malloc(M * sizeof(int *)) ;
        result->value = (int ***) malloc(M * sizeof(int **)) ;

        bins = (int *) malloc(M * sizeof(int)) ;
        for (j=0 ; j<M ; ++j)
        {
                bins[j] = 0 ;
                for (i=0 ; i<N ; ++i)
                        if ( BMat[i][j] != 0 )
                                ++bins[j] ;

                sum = Tally[j] ;
```

```
                if ( j == 0 )
                        --sum ;

                lim = Splits(sum, bins[j]) ;
                next = 0 ;
                len = bins[j] ;
                vec = (int *) malloc(len * sizeof(int)) ;
                mat = (int **) malloc(lim * sizeof(int *)) ;

                Enum(sum, bins[j], len, vec, mat, &next) ;
                if ( j == 0 )
                        for (i=0 ; i<lim ; ++i)
                                ++mat[i][0] ;

                result->rows[j] = lim ;
                result->cols[j] = (int *) malloc(lim * sizeof(int)) ;
                for (i=0 ; i<lim ; ++i)
                        result->cols[j][i] = len ;
                result->value[j] = mat ;
                free(vec) ;
        }
        return(result) ;
}

/*  Assembles symbol allocations into 3-D block, one plane per col of BMat  */

block   *BuildBinsPlusOneBlock(BMat)
        int     **BMat ;
{
        int     i, j ;
        int     sum, *bins, len, *vec, **mat, next, lim ;
        block   *result ;

        result = (block *) malloc(sizeof(block)) ;
        result->planes = M ;
        result->rows = (int *) malloc(M * sizeof(int)) ;
        result->cols = (int **) malloc(M * sizeof(int *)) ;
        result->value = (int ***) malloc(M * sizeof(int **)) ;

        bins = (int *) malloc(M * sizeof(int)) ;
        for (j=0 ; j<M ; ++j)
        {
                bins[j] = 0 ;
                for (i=0 ; i<N ; ++i)
                        if ( BMat[i][j] != 0 )
                                ++bins[j] ;

                sum = Tally[j] ;
                if ( j == 0 )
                        --sum ;

                lim = Splits(sum, (1+bins[j])) ;
                next = 0 ;
                len = 1+bins[j] ;
                vec = (int *) malloc(len * sizeof(int)) ;
                mat = (int **) malloc(lim * sizeof(int *)) ;

                Enum(sum, (1+bins[j]), len, vec, mat, &next) ;
                if ( j == 0 )
```

```
                        for (i=0 ; i<lim ; ++i)
                                ++mat[i][0] ;

                result->rows[j] = lim ;
                result->cols[j] = (int *) malloc(lim * sizeof(int)) ;
                for (i=0 ; i<lim ; ++i)
                        result->cols[j][i] = len ;
                result->value[j] = mat ;
                free(vec) ;
        }
        return(result) ;
}

/*  Lists permutations of n things taken n at a time  */

int     **PermutationsOf(n)
        int     n ;
{
        int     i ;
        int     sum, *rep, len, *vec, **mat, next, lim ;

        rep = (int *) malloc(n * sizeof(int)) ;
        for (i=0 ; i<n ; ++i)
                rep[i] = 1 ;
        len = sum = n ;
        lim = (int) Fact(n) ;
        next = 0 ;
        vec = (int *) malloc(len * sizeof(int)) ;
        mat = (int **) malloc(lim * sizeof(int *)) ;

        Item(sum, rep, len, vec, mat, &next) ;
        free(vec) ;
        return(mat) ;
}

/*  Computes the permutations of sum items, allowing for repetitions  */

int     Perms(sum, rep)
        int     sum ;
        int     *rep ;
{
        long    num, dom ;
        int     result ;
        int     i ;

        num = Fact(sum) ; dom = 1 ;
        for (i=0 ; i<N ; ++i)
                dom = dom * Fact(rep[i]) ;
        result = (int) (num/dom) ;
        return(result) ;
}

/*  Computes the ways of splitting n items among r bins  */

int     Splits(n, r)
        int     n ;
        int     r ;
```

```
{
        int     i ;
        int     result ;

        result = 1 ;
        for (i=1 ; i<r ; ++i)
                result = (result * (n+i)) / i ;
        return(result) ;
}

/*  Lists the permutations of sum items, with repetitions as per rep  */

void    Item(sum, rep, len, vec, mat, next)
        int     sum ;
        int     *rep ;
        int     len ;
        int     *vec ;
        int     **mat ;
        int     *next ;
{
        int     i, n ;

        if (sum == 0)
        {
                n = *next ;
                mat[n] = (int *) malloc(len * sizeof(int)) ;
                for (i=0 ; i<len ; ++i)
                        mat[n][i] = vec[i] ;
                *next = n+1 ;
        }
        else
                for (i=0 ; i<N ; ++i)
                        if ( rep[i] > 0 )
                        {
                                vec[len-sum] = i ;
                                rep[i] = rep[i] - 1 ;
                                Item(sum-1, rep, len, vec, mat, next) ;
                                rep[i] = rep[i] + 1 ;
                        }
}

/*  Enumerates the ways of splitting sum items among a number of bins  */

void    Enum(sum, bins, len, vec, mat, next)
        int     sum ;
        int     bins ;
        int     len ;
        int     *vec ;
        int     **mat ;
        int     *next ;
{
        int     i, n ;

        if (bins == 1)
        {
                vec[len-1] = sum ;
                n = *next ;
                mat[n] = (int *) malloc(len * sizeof(int)) ;
                for (i=0 ; i<len ; ++i)
                        mat[n][i] = vec[i] ;
                *next = n+1 ;
        }
        else
                for (i=sum ; i>=0 ; --i)
                {
                        vec[len-bins] = i ;
                        Enum(sum-i, bins-1, len, vec, mat, next) ;
                }
}

/*  Combines the state permutations for each symbol into a set of sequences  */

int     **Flatten(SBlock)
        block   *SBlock ;
{
        int     level, *vec, *lim, **mat, next ;

        next = CountRows(SBlock) ;
        vec = (int *) malloc(M * sizeof(int)) ;
        mat = (int **) malloc(next * sizeof(int *)) ;
        lim = SBlock->rows ;

        next = level = 0 ;
        Nest(SBlock, level, vec, lim, mat, &next) ;
        free(vec) ;
        return(mat) ;
}

/*  Carries out M levels of nesting, each up to lim, with indices in vec  */

void    Nest(SBlock, level, vec, lim, mat, next)
        block   *SBlock ;
        int     level ;
        int     *vec ;
        int     *lim ;
        int     **mat ;
        int     *next ;
{
        int     i, n ;

        if ( level == M )
        {
                n = *next ;
                mat[n] = Merge(SBlock, vec) ;
                *next = n+1 ;
        }
        else
        {
                n = lim[level] ;
                for (i=0 ; i<n ; ++i)
                {
                        vec[level] = i ;
                        Nest(SBlock, level+1, vec, lim, mat, next) ;
                }
        }
}
```

```
/*  Performs recursive processing of all possible combinations within NBlock  */

void    Process(AMat, BMat, FMat, NBlock, level, vec, lim, Head, Tail)
        int     **AMat ;
        int     **BMat ;
        int     **FMat ;
        block   *NBlock ;
        int     level ;
        int     *vec ;
        int     *lim ;
        slist   *Head ;
        slist   *Tail ;
{
        int     i, n ;

        if ( level == M )
                AppendNeighborsOf(AMat, BMat, FMat, NBlock, vec, Head, Tail) ;
        else
        {
                n = lim[level] ;
                for (i=0 ; i<n ; ++i)
                {
                        vec[level] = i ;
                        Process(AMat, BMat, FMat, NBlock, level+1, vec, lim, Hea
d, Tail) ;
                }
        }
}

/*  Merge values for each symbol into final state sequence  */

int     *Merge(SBlock, vec)
        block   *SBlock ;
        int     *vec ;
{
        int     j, s, t ;
        int     *plane, *row, *col ;
        int     *State ;

        plane = (int *) malloc(M * sizeof(int)) ;
        row   = (int *) malloc(M * sizeof(int)) ;
        col   = (int *) malloc(M * sizeof(int)) ;

        for (j=0 ; j<M ; ++j)
        {
                plane[j] = j ;
                row[j] = vec[j] ;
                col[j] = 0 ;
        }

        State = (int *) malloc(T * sizeof(int)) ;
        for (t=0 ; t<T ; ++t)
        {
                s = Symbol[t] ;
                State[t] = SBlock->value[plane[s]][row[s]][col[s]] ;
                ++col[s] ;
        }
        free(plane) ;
        free(row) ;
        free(col) ;
        return(State) ;
}

/*  Capture the sequences which are immediate neighbors of AMat & BMat  */

void    AppendNeighborsOf(AMat, BMat, FMat, NBlock, vec, Head, Tail)
        int     **AMat ;
        int     **BMat ;
        int     **FMat ;
        block   *NBlock ;
        int     *vec ;
        slist   *Head ;
        slist   *Tail ;
{
        int     **IMat, **JMat, **SMat ;
        int     *Sequence ;
        int     i, j, k, n ;
        int     *plane, *row, *col ;
        block   *SBlock ;

        plane = (int *) malloc(M * sizeof(int)) ;
        row   = (int *) malloc(M * sizeof(int)) ;
        col   = (int *) malloc(M * sizeof(int)) ;

        for (j=0 ; j<M ; ++j)
        {
                plane[j] = j ;
                row[j] = vec[j] ;
                col[j] = 0 ;
        }

        JMat = (int **) malloc(N * sizeof(int *)) ;
        for (i=0 ; i<N ; ++i)
                JMat[i] = (int *) malloc(M * sizeof(int)) ;

        for (j=0 ; j<M ; ++j)
                for (i=0 ; i<N ; ++i)
                        if ( FMat[i][j] == 0 )
                                JMat[i][j] = 0 ;
                        else
                        {
                                JMat[i][j] = NBlock->value[plane[j]][row[j]][col
[j]] ;
                                ++col[j] ;
                        }

        free(plane) ;
        free(row) ;
        free(col) ;

        SBlock = BuildStateBlock(JMat) ;
        SMat = Flatten(SBlock) ;

        n = CountRows(SBlock) ;
        for (k=0 ; k<n ; ++k)
```

```
        {
                Sequence = SMat[k] ;
                IMat = MakeAMat(Sequence, T) ;

                if ( (Distance(AMat,IMat,N,N) + Distance(BMat,JMat,N,M)) == 1 )
                        LAppend(Sequence, Head, Tail) ;

                IAbsorb(IMat) ;
        }

        IAbsorb(JMat) ;
        RecyclePair(SBlock, SMat) ;
}

/*  Count the total number of possible state sequences within SBlock  */

int     CountRows(SBlock)
        block   *SBlock ;
{
        int     j, tot ;

        tot = 1 ;
        for (j=0 ; j<M ; ++j)
                tot = tot * SBlock->rows[j] ;
        return(tot) ;
}

/*  Returns A-matrix associated with state sequence  */

int     **MakeAMat(State, t)
        int     *State ;
        int     t ;
{
        int     **AMat ;
        int     i, j, s ;

        AMat = (int **) malloc(N * sizeof(int *)) ;
        for (i=0 ; i<N ; ++i)
        {
                AMat[i] = (int *) malloc(N * sizeof(int)) ;
                for (j=0 ; j<N ; ++j)
                        AMat[i][j] = 0 ;
        }
        for (s=1 ; s<t ; ++s)
        {
                i = State[s-1] ;
                j = State[s] ;
                ++AMat[i][j] ;
        }
        return(AMat) ;
}

/*  Returns B-matrix associated with state sequence  */

int     **MakeBMat(State, t)
        int     *State ;
        int     t ;
```

```
{
        int     **BMat ;
        int     i, j, s ;

        BMat = (int **) malloc(N * sizeof(int *)) ;
        for (i=0 ; i<N ; ++i)
        {
                BMat[i] = (int *) malloc(M * sizeof(int)) ;
                for (j=0 ; j<M ; ++j)
                        BMat[i][j] = 0 ;
        }
        for (s=0 ; s<t ; ++s)
        {
                i = State[s] ;
                j = Symbol[s] ;
                ++BMat[i][j] ;
        }
        return(BMat) ;
}

/*  By how many dimensions do two matrices differ  */

int     Distance(Mat1, Mat2, n, m)
        int     **Mat1 ;
        int     **Mat2 ;
        int     n ;
        int     m ;
{
        int     i, j ;
        int     dim1, dim2, bins ;

        dim1 = 0 ;
        for (i=0 ; i<n ; ++i)
        {
                bins = 0 ;
                for (j=0 ; j<m ; ++j)
                        if ( Mat1[i][j] > 0 )
                                ++bins ;
                if ( bins > 1 )
                        dim1 = dim1 + (bins-1) ;
        }

        dim2 = 0 ;
        for (i=0 ; i<n ; ++i)
        {
                bins = 0 ;
                for (j=0 ; j<m ; ++j)
                        if ( (Mat1[i][j] + Mat2[i][j]) > 0 )
                                ++bins ;
                if ( bins > 1 )
                        dim2 = dim2 + (bins-1) ;
        }

        return(dim2 - dim1) ;
}

/*  Permute the rows of Mat as per row  */
```

```
int     **Permute(Mat, row)
        int     **Mat ;
        int     *row ;
{
        int     i, j ;
        int     **BMat ;

        BMat = (int **) malloc(N * sizeof(int *)) ;
        for (i=0 ; i<N ; ++i)
        {
                BMat[i] = (int *) malloc(M * sizeof(int)) ;
                for (j=0 ; j<M ; ++j)
                        BMat[i][j] = Mat[row[i]][j] ;
        }
        return(BMat) ;
}

/*  Free up memory occupied by integer matrix  */

void    IAbsorb(Mat)
        int     **Mat ;
{
        int     i ;

        for (i=0 ; i<N ; ++i)
                free(Mat[i]) ;

        free(Mat) ;
}

/*  Free up memory occupied by floating point matrix  */

void    DAbsorb(Mat)
        double  **Mat ;
{
        int     i ;

        for (i=0 ; i<N ; ++i)
                free(Mat[i]) ;

        free(Mat) ;
}

/*  Free up memory occupied by block of data  */

void    RecycleBlock(Block)
        block   *Block ;
{
        int     i, j ;
        int     *lim ;

        lim = Block->rows ;
        for (j=0 ; j<M ; ++j)
        {
                for (i=0 ; i<lim[j] ; ++i)
                        free(Block->value[j][i]) ;
                free(Block->cols[j]) ;
        }
        free(Block->rows) ;
        free(Block) ;
}

/*  Free up memory occupied by large variables  */

void    RecyclePair(Block, Mat)
        block   *Block ;
        int     **Mat ;
{
        int     i, j ;
        int     tot, *lim ;

        tot = 1 ;
        lim = Block->rows ;
        for (j=0 ; j<M ; ++j)
        {
                tot = tot * lim[j] ;
                for (i=0 ; i<lim[j] ; ++i)
                        free(Block->value[j][i]) ;
                free(Block->cols[j]) ;
        }
        free(Block->rows) ;
        free(Block) ;

        for (j=0 ; j<tot ; ++j)
                free(Mat[j]) ;
        free(Mat) ;
}

/*  Free up memory occupied by slist  */

void    RecycleSList(List)
        slist   List ;
{
        slist   Current, Node ;

        Current = List ;
        while ( Current != NULL )
        {
                Node = Current ;
                Current = Current->next ;
                free(Node->sequence) ;
                free(Node) ;
        }
}

/*  Contribution given by Mat (having the stated dimensions)  */

double  Contribution(Mat, n, m)
        int     **Mat ;
        int     n ;
        int     m ;
{
        int     i, j, k ;
        double  **Max ;
```

```
        double  result ;

        Max = (double **) malloc(n * sizeof(double *)) ;
        for (i=0 ; i<n ; ++i)
                Max[i] = (double *) malloc(m * sizeof(double)) ;

        Max = Convert(Mat, Max, n, m) ;
        result = 1.0 ;
        for (i=0 ; i<n ; ++i)
                for (j=0 ; j<m ; ++j)
                        for (k=0 ; k<Mat[i][j] ; ++k)
                                result = result * Max[i][j] ;

        DAbsorb(Max) ;
        return(result) ;
}

/*  Generate floating point version of markov matrix  */

double  **Convert(Mat, Max, n, m)
        int     **Mat ;
        double  **Max ;
        int     n ;
        int     m ;
{
        int     i, j ;
        int     *Sum ;

        Sum = (int *) malloc(n * sizeof(int)) ;
        for (i=0 ; i<n ; ++i)
        {
                Sum[i] = 0 ;
                for (j=0 ; j<m ; ++j)
                        Sum[i] = Sum[i] + Mat[i][j] ;
        }
        for (i=0 ; i<n ; ++i)
                for (j=0 ; j<m ; ++j)
                        if ( Sum[i] != 0 )
                                Max[i][j] = ((double) Mat[i][j]) / ((double) Sum
[i]) ;
                        else
                                Max[i][j] = 0.0 ;
        free(Sum) ;
        return(Max) ;
}

/*  Calculate factorial  */

long    Fact(n)
        int     n ;
{
        long    result ;

        result = 1 ;
        if ( n > 1 )
                result = n * Fact(n-1) ;
        return(result) ;
}
```

```
/*  Perform B-W reestimation, either from given term or from pseudo-COG  */

void    Climb(List, pos)
        slist   List ;
        int     pos ;
{
        int     i, j, converged ;
        double  **AOld, **BOld, **ANew, **BNew ;
        double  **Alfa, **Beta, **Gama, **Neta ;
        double  Func, Prev ;

        CreateVars(&AOld, &BOld, &ANew, &BNew, &Alfa, &Beta, &Gama, &Neta) ;
        InitVars(ANew, BNew, List, pos) ;

        Func = 0.0 ;
        i = converged = 0 ;
        while ( ( !converged ) && ( i < 20 ) )
        {
                RecordOld(AOld, BOld, ANew, BNew) ;
                Prev = Func ;
                Func = CalcNew(ANew, BNew, Alfa, Beta, Gama, Neta) ;
                if ( (Func - Prev) < 0.000001 )
                        converged = 1 ;
                else
                        ++i ;
        }

/*      DisplayMats(AOld, BOld, Prev) ;         */

        if ( Prev > FMax )
        {
                for (i=0 ; i<N ; ++i)
                        for (j=0 ; j<N ; ++j)
                                AMax[i][j] = AOld[i][j] ;
                for (i=0 ; i<N ; ++i)
                        for (j=0 ; j<M ; ++j)
                                BMax[i][j] = BOld[i][j] ;
                FMax = Prev ;
        }

        ReleaseVars(AOld, BOld, ANew, BNew, Alfa, Beta, Gama, Neta) ;
}


/*  Allocate memory for main variables  */

void    CreateVars(aold, bold, anew, bnew, alfa, beta, gama, neta)
        double  ***aold ;
        double  ***bold ;
        double  ***anew ;
        double  ***bnew ;
        double  ***alfa ;
        double  ***beta ;
        double  ***gama ;
        double  ***neta ;
{
        double  **AOld, **BOld, **ANew, **BNew ;
        double  **Alfa, **Beta, **Gama, **Neta ;
```

```
        int     i, t ;

        AOld = (double **) malloc(N * sizeof(double *)) ;
        BOld = (double **) malloc(N * sizeof(double *)) ;
        ANew = (double **) malloc(N * sizeof(double *)) ;
        BNew = (double **) malloc(N * sizeof(double *)) ;

        for (i=0 ; i<N ; ++i)
        {
                AOld[i] = (double *) malloc(N * sizeof(double)) ;
                ANew[i] = (double *) malloc(N * sizeof(double)) ;
                BOld[i] = (double *) malloc(M * sizeof(double)) ;
                BNew[i] = (double *) malloc(M * sizeof(double)) ;
        }

        Alfa = (double **) malloc(T * sizeof(double *)) ;
        Beta = (double **) malloc(T * sizeof(double *)) ;
        Gama = (double **) malloc(T * sizeof(double *)) ;

        for (t=0 ; t<T ; ++t)
        {
                Alfa[t] = (double *) malloc(N * sizeof(double)) ;
                Beta[t] = (double *) malloc(N * sizeof(double)) ;
                Gama[t] = (double *) malloc(N * sizeof(double)) ;
        }

        Neta = (double **) malloc(N * sizeof(double *)) ;
        for (i=0 ; i<N ; ++i)
                Neta[i] = (double *) malloc(N * sizeof(double)) ;

        *aold = AOld ;
        *bold = BOld ;
        *anew = ANew ;
        *bnew = BNew ;
        *alfa = Alfa ;
        *beta = Beta ;
        *gama = Gama ;
        *neta = Neta ;
}


/*  Assign initial values to model  */

void    InitVars(ANew, BNew, List, pos)
        double  **ANew ;
        double  **BNew ;
        slist   List ;
        int     pos ;
{
        int     **AMat, **BMat ;
        int     *State ;
        int     h ;
        slist   Current ;

        if ( pos == 0 )
                CalcCOG(ANew, BNew, List) ;
        else
        {
                Current = List ;
                h = 1 ;
```

```
			while ( h < pos )
			{
				Current = Current->next ;
				++h ;
			}
			State = Current->sequence ;
			AMat = MakeAMat(State, T) ;
			BMat = MakeBMat(State, T) ;
			ANew = Convert(AMat, ANew, N, N) ;
			BNew = Convert(BMat, BNew, N, M) ;
			IAbsorb(AMat) ;
			IAbsorb(BMat) ;
		}
}

/*  Calculate pseudo-Center of Gravity of the participating terms  */

void	CalcCOG(ANew, BNew, List)
	double	**ANew ;
	double	**BNew ;
	slist	List ;
{
	int	***AExp, ***BExp ;
	int	**ASum, **BSum ;
	int	P, *State ;
	int	h, i, j ;
	double	num, dom ;
	slist	Current ;

	P = LengthOf(List) ;
	AExp = (int ***) malloc(P * sizeof(int **)) ;
	BExp = (int ***) malloc(P * sizeof(int **)) ;
	ASum = (int **)  malloc(P * sizeof(int * )) ;
	BSum = (int **)  malloc(P * sizeof(int * )) ;

	Current = List ;
	for (h=0 ; h<P ; ++h)
	{
		State = Current->sequence ;
		AExp[h] = MakeAMat(State, T) ;
		BExp[h] = MakeBMat(State, T) ;
		ASum[h] = (int *) malloc(N* sizeof(int)) ;
		BSum[h] = (int *) malloc(N * sizeof(int)) ;
		for (i=0 ; i<N ; ++i)
		{
			ASum[h][i] = 0 ;
			BSum[h][i] = 0 ;
			for (j=0; j<N ; ++j)
				ASum[h][i] = ASum[h][i] + AExp[h][i][j] ;
			for (j=0; j<M ; ++j)
				BSum[h][i] = BSum[h][i] + BExp[h][i][j] ;
		}
		Current = Current->next ;
	}

	for (i=0 ; i<N ; ++i)
	{
		dom = 0.0 ;
		for (h=0 ; h<P ; ++h)
```

```
				dom = dom + ((double) ASum[h][i]) ;
			for (j=0 ; j<N ; ++j)
			{
				num = 0.0 ;
				for (h=0 ; h<P ; ++h)
					num = num + ((double) AExp[h][i][j]) ;
				if (num < 0.000001)
					ANew[i][j] = 0.0 ;
				else
					ANew[i][j] = num / dom ;
			}
		}

		for (i=0 ; i<N ; ++i)
		{
			dom = 0.0 ;
			for (h=0 ; h<P ; ++h)
				dom = dom + ((double) BSum[h][i]) ;
			for (j=0 ; j<M ; ++j)
			{
				num = 0.0 ;
				for (h=0 ; h<P ; ++h)
					num = num + ((double) BExp[h][i][j]) ;
				if (num < 0.000001)
					BNew[i][j] = 0.0 ;
				else
					BNew[i][j] = num / dom ;
			}
		}

		for (h=0 ; h<P ; ++h)
			for (i=0 ; i<N ; ++i)
			{
				free(AExp[h][i]) ;
				free(BExp[h][i]) ;
			}
		for (h=0 ; h<P ; ++h)
		{
			free(AExp[h]) ;
			free(BExp[h]) ;
		}
		free(AExp) ;
		free(BExp) ;

		for (h=0 ; h<P ; ++h)
		{
			free(ASum[h]) ;
			free(BSum[h]) ;
		}
		free(ASum) ;
		free(BSum) ;
}

/*  Retain old  A & B  matrices prior to next iteration  */

void	RecordOld(AOld, BOld, ANew, BNew)
	double	**AOld ;
	double	**BOld ;
	double	**ANew ;
```

```
        double  **BNew ;
{
        int     i, j ;

        for (i=0 ; i<N ; ++i)
        {
                for (j=0 ; j<N ; ++j)
                        AOld[i][j] = ANew[i][j] ;
                for (j=0 ; j<M ; ++j)
                        BOld[i][j] = BNew[i][j] ;
        }
}

/*  Calculate revised elements of the  A & B  matrices  */

double  CalcNew(a, b, alfa, beta, gama, neta)
        double  **a ;
        double  **b ;
        double  **alfa ;
        double  **beta ;
        double  **gama ;
        double  **neta ;
{
        int     i, j, k, t ;
        int     *O ;
        double  sum, num, dom ;
        double  P ;

        O = Symbol ;

        alfa[0][0] = b[0][O[0]] ;
        for (i=1 ; i<N ; ++i)
                alfa[0][i] = 0.0 ;
        for (t=0 ; t<(T-1) ; ++t)
                for (j=0 ; j<N ; ++j)
                        if ( b[j][O[t+1]] > 0.000001 )
                        {
                                sum = 0.0 ;
                                for (i=0 ; i<N ; ++i)
                                        if ( a[i][j] > 0.000001)
                                                sum = sum + alfa[t][i] * a[i][j]
 ;
                                alfa[t+1][j] = sum * b[j][O[t+1]] ;
                        }
                        else
                                alfa[t+1][j] = 0.0 ;

        for (j=0 ; j<N ; ++j)
                beta[T-1][j] = 1.0 ;
        for (t=T-2 ; t>=0 ; --t)
                for (i=0 ; i<N ; ++i)
                {
                        sum = 0.0 ;
                        for (j=0 ; j<N ; ++j)
                                if ( ( a[i][j] > 0.000001 ) && ( b[j][O[t+1]] >
0.000001 ) )
                                        sum = sum + a[i][j] * b[j][O[t+1]] * bet
a[t+1][j] ;
                        beta[t][i] = sum ;
```

```
                }

        for (i=0 ; i<N ; ++i)
                for (j=0 ; j<N ; ++j)
                        if ( a[i][j] > 0.000001 )
                        {
                                sum = 0 ;
                                for (t=0 ; t<(T-1) ; ++t)
                                        if ( b[j][O[t+1]] > 0.000001 )
                                                sum = sum + alfa[t][i] * a[i][j]
 * b[j][O[t+1]] * beta[t+1][j] ;
                                neta[i][j] = sum ;
                        }
                        else
                                neta[i][j] = 0.0 ;

        for (t=0 ; t<T ; ++t)
                for (i=0 ; i<N ; ++i)
                        gama[t][i] = alfa[t][i] * beta[t][i] ;

        for (i=0 ; i<N ; ++i)
        {
                dom = 0.0 ;
                for (t=0 ; t<(T-1) ; ++t)
                        dom = dom + gama[t][i] ;
                for (j=0 ; j<N ; ++j)
                {
                        num = neta[i][j] ;
                        if ( num > 0.000001 )
                                a[i][j] = num / dom ;
                        else
                                a[i][j] = 0.0 ;
                }
        }

        for (j=0 ; j<N ; ++j)
        {
                dom = 0.0 ;
                for (t=0 ; t<T ; ++t)
                        dom = dom + gama[t][j] ;
                for (k=0 ; k<M ; ++k)
                {
                        num = 0.0 ;
                        for (t=0 ; t<T ; ++t)
                                if ( O[t] == k )
                                        num = num + gama[t][j] ;
                        if ( num > 0.000001 )
                                b[j][k] = num / dom ;
                        else
                                b[j][k] = 0.0 ;
                }
        }

        sum = 0.0 ;
        for (i=0 ; i<N ; ++i)
                sum = sum + alfa[T-1][i] ;
        P = sum ;
        return(P) ;
}
```

```
/*  Compute objective value at modal position using B-W forward calculation  */

double  FullValue(AMat, BMat)
        int     **AMat ;
        int     **BMat ;
{
        int     i, j, t ;
        int     *O ;
        double  **a, **b ;
        double  **alfa, sum ;
        double  P ;

        O = Symbol ;

        a = (double **) malloc(N * sizeof(double *)) ;
        b = (double **) malloc(N * sizeof(double *)) ;
        for (i=0 ; i<N ; ++i)
        {
                a[i] = (double *) malloc(N * sizeof(double)) ;
                b[i] = (double *) malloc(M * sizeof(double)) ;
        }
        a = Convert(AMat, a, N, N) ;
        b = Convert(BMat, b, N, M) ;

        alfa = (double **) malloc(T * sizeof(double *)) ;
        for (t=0 ; t<T ; ++t)
                alfa[t] = (double *) malloc(N * sizeof(double)) ;
        alfa[0][0] = b[0][O[0]] ;
        for (i=1 ; i<N ; ++i)
                alfa[0][i] = 0.0 ;
        for (t=0 ; t<(T-1) ; ++t)
                for (j=0 ; j<N ; ++j)
                        if ( b[j][O[t+1]] > 0.000001 )
                        {
                                sum = 0.0 ;
                                for (i=0 ; i<N ; ++i)
                                        if ( a[i][j] > 0.000001)
                                                sum = sum + alfa[t][i] * a[i][j]
 ;
                                alfa[t+1][j] = sum * b[j][O[t+1]] ;
                        }
                        else
                                alfa[t+1][j] = 0.0 ;

        sum = 0.0 ;
        for (i=0 ; i<N ; ++i)
                sum = sum + alfa[T-1][i] ;
        P = sum ;

        for (i=0 ; i<N ; ++i)
        {
                free(a[i]) ;
                free(b[i]) ;
        }
        free(a) ;
        free(b) ;

        for (t=0 ; t<T ; ++t)
                free(alfa[t]) ;
```

```
        free(alfa) ;

        return(P) ;
}


/*  Write  A & B  matrices to file  */

void    DisplayMats(AOld, BOld, Prev)
        double  **AOld ;
        double  **BOld ;
        double  Prev ;
{
        int     i, j ;
        double  Func, Round ;

        fprintf(fp, "\n\nA-matrix : \n") ;
        for (i=0 ; i<N ; ++i)
        {
                fprintf(fp, "\n") ;
                for (j=0 ; j<N ; ++j)
                        fprintf(fp, "%lf ", AOld[i][j]) ;
        }

        fprintf(fp, "\n\nB-matrix : \n") ;
        for (i=0 ; i<N ; ++i)
        {
                fprintf(fp, "\n") ;
                for (j=0 ; j<M ; ++j)
                        fprintf(fp, "%lf ", BOld[i][j]) ;
        }
        Func = Prev ;
        Round = ((double) ((long)(((1.0/Func) * 100.0) + 0.5))) / 100.0 ;
        fprintf(fp, "\n\nFunc = %lf = 1/%lf", Func, Round) ;
}


/*  Free up memory occupied by large variables  */

void    ReleaseVars(AOld, BOld, ANew, BNew, alpha, beta, gamma, neta)
        double  **AOld ;
        double  **BOld ;
        double  **ANew ;
        double  **BNew ;
        double  **alpha ;
        double  **beta ;
        double  **gamma ;
        double  **neta ;
{
        int     i, t ;

        for (i=0 ; i<N ; ++i)
        {
                free(AOld[i]) ;
                free(BOld[i]) ;
                free(ANew[i]) ;
                free(BNew[i]) ;
        }
        free(AOld) ;
        free(BOld) ;
        free(ANew) ;
        free(BNew) ;

        for (t=0 ; t<T ; ++t)
        {
                free(alpha[t]) ;
                free(beta[t])  ;
                free(gamma[t]) ;
        }
        free(alpha) ;
        free(beta) ;
        free(gamma) ;

        for (i=0 ; i<N ; ++i)
                free(neta[i]) ;
        free(neta) ;

}
```

F. Summary of Test Results

F.1. Function Evaluations on Grid

This section presents the results of test three, described originally in Section 5.7. in the final paragraph of page 137 .

During this test, the parameter space was searched for the global optimum solution by performing evaluations of the objective function* over a grid of equally-spaced points in the interior of the space, and the highest located function value was then recorded.

Each trial required three inputs: a single character sequence having length T (these are listed down the side), the model size N , and a parameter D which governs the grid density.

The grid of points for evaluation was constructed as follows (see the example on the next page) :

First, the number of ways in which D items can be distributed among N boxes (M for the B matrix) is enumerated. There are

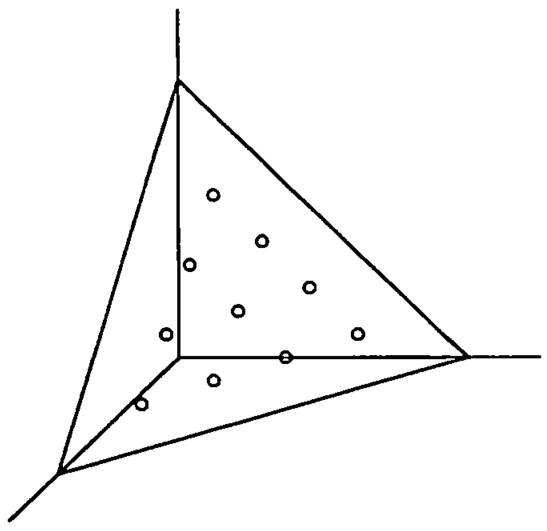
$$K = \binom{D+N-1}{N-1} = \frac{(D+N-1)!}{D!(N-1)!}$$

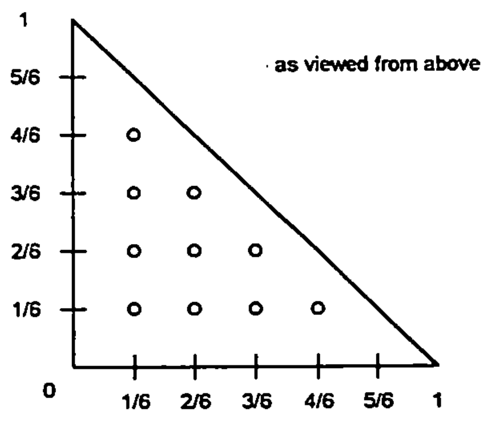
such ways. Then the number 1 is added to each box to bring the grid point into the interior of the parameter space. Finally, the contents of the boxes are divided by $D+N$ (or $D+M$ for the B matrix) . The result is a collection of K N-tuples constituting the coordinates of the grid points applicable to a single row of the A (or B) matrix.

* a different objective function for each training sequence and model size

D=3 items among N=3 boxes				To move into interior						
3	0	0		1	1	1		4	1	1
0	3	0		1	1	1		1	4	1
0	0	3		1	1	1		1	1	4
2	1	0		1	1	1		3	2	1
2	0	1		1	1	1		3	1	2
1	2	0	plus	1	1	1	equals	2	3	1
0	2	1		1	1	1		1	3	2
1	0	2		1	1	1		2	1	3
0	1	2		1	1	1		1	2	3
1	1	1		1	1	1		2	2	2

the plane $x + y + z = 1$





Obviously, the higher the grid density, the more effective the test. However, in the case for simplicity when $N = M$, this involves K^{2N} function evaluations (where each of the rows of A and B must independently receive each of the K N-tuples). For larger values of N, M and D, this number of evaluations soon becomes prohibitive.

Two sets of numbers are presented in the tables. For ease of comparison, the large (lower) numbers give the *reciprocal* of $P(O)$. The smaller this value, the higher the objective value. The little numbers appearing above them represent the CPU elapsed time in hours required to perform that trial.

N = 3
T = 5

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
ababb			0:1:45.5 7.66	0:6:24.7 7.00	0:20:44.1 6.58	aborted	2:23:20.0 5.98	5:26:41.6 5.76	3.76
abccc				7:11:58.7 8.53	32:34:25.8 6.82	aborted	aborted	aborted	1.00
abcad			26:29:54.6 61.08	aborted	aborted		aborted	aborted	4.00
abbaa								5:29:01.5 7.12	4.00
aabac				7:15:02.8 23.78	32:28:48.3 20.14	aborted	aborted	aborted	4.00
aabaa								5:26:50.1 2.62	1.00
aaaab								5:26:12.6 7.15	4.00
aabca				7:13:18.8 24.99	32:34:41.0 20.65	aborted			4.00
ababc				7:15:28.8 21.04	32:33:50.6 17.71				4.00
abbac				7:12:12.9 25.06	32:35:23.8 21.02				4.00

N = 3
T = 6

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
aabbcb				8:15:03.2 67.53				aborted	9.48
aaabaa								6:16:24.5 7.77	3.38
aaabca				8:15:44.4 57.53				aborted	6.75
ababcb				8:16:02.0 37.37				aborted	4.00
abbbba								6:16:05.4 9.36	4.00
abbacb				8:22:44.2 39.19					4.00
aabaaa								6:19:44.2 6.47	4.00
aaaabc				8:15:58.6 63.59					9.48
ababbb								6:15:47.2 7.21	4.00
aabccd		2:37:33.4 529.52	29:08:19.6 391.70						27.00

N = 3
T = 7

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
abacded			aborted						108.00
aaaabbc				9:18:37.4 180.00					27.00
abbaaaa								7:04:23.7 11.48	4.00
abaacaa				9:23:22.7 56.93					4.00
aaabaca				9:19:42.2 108.05					16.00
aabbcbc				9:23:26.4 188.49					27.00
aaabcac				9:17:37.8 193.98					27.00
abbcdda			33:20:17.7 1650.68						108.00
aabbbac				9:27:13.6 216.34					27.00
abcbdea			aborted						108.00

N = 3
T = 8

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
abaaaabb								7:57:01.1 51.86	27.00
abbbbbbb								7:57:05.2 3.33	1.00
aaabbaba								7:56:51.3 54.50	27.00
abbaacdb		3:14:17.3 8001.71	36:21:13.1 5594.86						182.25
aaaaabcb				10:49:40.9 199.70					12.21
aaaabbbc				11:06:30.1 342.76					37.93
abcdbdeb	0:59:52.6 42,168.71	26:58:27.5 21,069.74							108.00
abcdedee	1:00:35.5 45,730.64	27:15:39.3 26,597.99							432.00
abcacccd		3:14:30.7 3480.95	36:16:54.3 2683.41						182.25
aaabacdc		3:14:04.6 4240.48	36:22:42.3 3113.59						149.01

N = 3
T = 9

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
abccbdaeb	1:06:07.8 231,962.74								1230.19
abcbabcdb		3:35:18.4 10,743.98							151.70
abcbbabcb				12:27:20.1 666.74					45.56
aabbcddec	1:06:11.5 327,374.83								3125.00
aabcbadda		3:35:25.5 28,442.12							1025.01
ababbbbcc				12:10:11.9 922.24					115.74
aabbacaad		3:35:31.0 15,352.50							182.25
abbcbcddc		3:35:13.5 28,220.35							837.24
abbaacbba				12:09:55.6 1398.25					182.25
aabcbbdba		3:35:01.1 16,564.20							606.81

N = 3
T = 10

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
aabacccabb				13:27:40.3 9789.45					1728.00
abbbbcaabd		3:52:48.4 39,962.61	43:46:40.3 27,755.33						781.25
abaabbbcca				13:07:25.3 6251.27					607.18
aababaaabc				13:28:18.5 1389.66					115.74
aaabacccdc		3:54:12.9 29,162.40	43:46:54.9 21,152.21						329.59
abcdecebca	1:11:09.3 718,067.04	32:00:04.9 291,971.98							432.00
aabcbdefdg	17:24:46.9 21,474,836.47								46,656.00
abccaaddae	1:11:21.3 1,200,011.14	32:19:58.7 765,844.40							6912.00
aabaabacab				13:27:43.6 1136.49					64.00
abccbdccdb		3:51:10.5 77,310.33	43:55:14.7 52,308.67						2075.94
abbaaabbaa								9:38:27.4 142.73	28.94

N = 3
T = 11

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
abaccbcdadb		4:15:31.9 733,349.38							27,648.00
aaaabbcadda		4:15:04.3 206,384.45							5273.44
abcadbaabac		4:15:28.9 184,471.57							1728.00
aaaababacde		34:36:24.4 713,597.88							4096.00
abcbdbbcacd		4:15:24.0 368,983.76							6912.00
aaabcbaaacd		4:15:49.1 181,489.48							4651.74
ababacdbccd		4:15:25.4 275,375.47							2916.00
abacbaccaaa				14:24:32.4 9024.91					508.26
abcbcabddcd		4:15:51.3 329,406.08	47:02:54.0 202,117.52						2601.23
abbbccdeabe		34:39:56.3 2,909,207.25							16,093.25

N = 3
T = 12

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
aaaaaaabbaba								11:18:26.5 251.29	100.78
aaaabcacabab			2:45:15.2 22,915.56						1438.38
aabaabaccabb			2:41:21.3 29,204.58						606.81
abacdbdabbce	1:22:34.1 21,474,836.47	37:17:36.0 16,130,614.53							152,587.89
aaaabacccbdc		4:36:44.4 486,119.38	50:35:21.0 343,450.51						8303.77
abacccbddddb		4:40:33.7 1,313,505.77	50:32:23.5 867,159.51						14,012.60
abccddbcebce	1:22:23.9 14,891,699.57	37:18:25.3 6,300,783.09							17,558.30
abaccdefefcb	5:45:09.3 21,474,836.47								569,531.25
abcccaaadadd		4:41:05.0 1,100,482.12							10,711.12
abcccaccccad		4:40:26.2 133,632.13	50:32:31.6 83,620.48						1230.19

N = 3
T = 13

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
abcccaccdeaec	1:27:40.9 21,474,836.47	43:21:02.9 15,722,938.32							81,048.98
aaabccacadace	1:27:41.3 20,094,333.35	39:55:05.0 11,850,812.85							84,375.00
aaabcdeabcbdc	1:27:54.7 21,474,836.47								361,689.81
aabbbaccacaba			2:53:59.3 195,646.04	16:31:36.5 162,162.86					12,601.99
aabaaaccbdcee	1:27:47.1 21,474,836.47								314,928.00
aaabbbbcbacac			2:57:49.7 198,616.37	16:27:43.7 168,317.93					12,988.01
abcaaaadebfeb	6:07:07.3 21,474,836.47	aborted after 109:52:29.5							214,334.71
aabbcdbeacbed	1:27:30.9 21,474,836.47								361,689.81
aaaababcabcda		4:57:01.9 1,609,403.89	54:07:19.6 1,054,282.31						16,384.00
abcbdddeceefb	6:05:48.7 21,474,836.47								2,153,581.37

N = 4
T = 5

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
ababb	0:55:19.5 9.66	21:08:28.4 8.05			aborted after 119:54:48.2	aborted after 120 hours	aborted after 120 hours	aborted after 120 hours	1.00
abccc	37:56:11.2 24.50								1.00
abcad	aborted after 168 hours								4.00
abbaa	0:55:18.2 11.17	21:09:31.5 8.49							1.00
aabac	37:57:52.7 56.43								4.00
aabaa	0:55:23.7 8.49	21:09:09.3 6.57							1.00
aaaab	0:55:18.1 10.49	21:08:52.8 9.24							3.05
aabca	37:57:09.7 36.64								1.00
ababc	37:55:10.5 49.99								3.38
abbac	37:56:45.2 63.31								4.00

N = 5
T = 5

	D = 3	D = 4	D = 5	D = 6	D = 7	D = 8	D = 9	D = 10	NHB
ababb	aborted after 144:07:15.2								1.00
abccc									1.00
abcad									1.00
abbaa	aborted after 144:06:39.7								1.00
aabac									1.00
aabaa	aborted after 144:06:59.3								1.00
aaaab	aborted after 144:06:25.3								1.00
aabca									1.00
ababc									1.00
abbac									1.00

F.2. Baum-Welch Programs

This section presents the results of the final test, described originally in Section 5.7. on page 138 .

During this test, the <u>N</u>ew <u>HMM</u> <u>B</u>uilding algorithm was set against a battery of four HMM model-building programs, all performing classic Baum-Welch re-estimation (i.e. a single hill-climb), but differing in the model architecture and choice of starting point :

	Fully-connected	Left-right (2 skip states)
Initialized uniformly	BW1	BW2
Initialized randomly	BW3	BW4

The following few pages do not attempt to give the complete set of results, but merely present a typical cross-section. For ease of comparison, the numerical values displayed are the *reciprocal* of $P(O)$.

number of states N = 3
sequence length T = 5

Training Sequence	BW1	BW2	Algorithm BW3	BW4	NHB
a b a b b	28.94	9.22	3.70	9.22	3.76
a b c c c	115.74	1.00	1.00	1.00	1.00
a b c a d	781.25	27.00	27.00	27.00	4.00
a b b a a	28.94	4.00	4.00	4.00	4.00
a a b a c	115.74	16.00	4.00	16.00	4.00
a a b a a	12.21	4.00	1.00	4.00	1.00
a a a a b	12.21	4.82	4.82	4.82	4.00
a a b c a	115.74	16.00	4.00	16.00	4.00
a b a b c	195.31	36.00	4.00	27.00	4.00
a b b a c	195.31	16.00	4.00	16.00	4.00

number of states N = 3
sequence length T = 6

	Algorithm				
Training Sequence	BW1	BW2	BW3	BW4	NHB
a a b b c b	432.00	9.48	9.48	9.48	9.48
a a a b a a	14.93	9.22	3.38	11.42	3.38
a a a b c a	182.25	53.87	6.75	53.87	6.75
a b a b c b	432.00	74.82	6.75	74.82	4.00
a b b b b a	45.56	7.23	4.00	7.23	4.00
a b b a c b	432.00	55.90	4.00	55.90	4.00
a a b a a a	14.93	4.00	4.00	4.00	4.00
a a a a b c	182.25	21.33	9.48	21.33	9.48
a b a b b b	45.56	10.59	4.00	10.59	4.00
a a b c c d	2,916.00	27.00	27.00	27.00	27.00

number of states N = 3
sequence length T = 7

	Algorithm				
Training Sequence	BW1	BW2	BW3	BW4	NHB
a b a c d e d	51,471.44	307.55	182.25	307.55	108.00
a a a a b b c	804.24	36.00	36.00	36.00	27.00
a b b a a a a	65.88	4.00	9.48	4.00	4.00
a b a a c a a	263.53	12.21	4.00	12.21	4.00
a a a b a c a	263.53	45.56	27.00	105.93	16.00
a a b b c b c	1,906.35	28.94	28.94	55.91	27.00
a a a b c a c	804.24	45.56	27.00	45.56	27.00
a b b c d d a	12,867.86	256.73	307.63	256.80	108.00
a a b b b a c	1,129.69	108.07	27.00	108.07	27.00
a b c b d e a	51,471.44	1,233.43	182.26	1,234.28	108.00

References

[1] M. Russell (1996), "Advances in Speech Recognition", *Proc. 1996 Autumn Conf. Inst. Acoust., Windermere, UK, 21-24 November 1996*, Vol. 18, Part 9, pp. 267-274.

[2] G. Koprowski (1996), "Hello, is anyone there?", *New Scientist*, 18 May 1996, Vol. 150, No. 2030, pp. 36-39.

[3] Q. Huo, C. Chan and C.-H. Lee (1995), "Bayesian Adaptive Learning of the Parameters of Hidden Markov Model for Speech Recognition", *IEEE Trans. Speech and Audio Process.*, September 1995, Vol. 3, No. 5, p. 334.

[4] G. Zavaliagkos, Y. Zhao, R. Schwartz and J. Makhoul (1994), "A Hybrid Segmental Neural Net/Hidden Markov Model System for Continuous Speech Recognition", *IEEE Trans. Speech and Audio Process.*, January 1994, Vol. 2, No. 1, Part II, p. 151.

[5] L.T. Niles and H.F. Silverman (1990), "Combining Hidden Markov Model and Neural Network Classifiers", *Proc. IEEE Internat. Conf. Acoust. Speech Signal Process., Albuquerque, New Mexico*, p. 417.

[6] S.J. Cox (1988), "Hidden Markov models for automatic speech recognition: theory and application", *Br. Telecom Technol. J.*, Vol. 6, No. 2, April 1988, p. 105.

[7] J. Picone (1990), "Continuous Speech Recognition Using Hidden Markov Models", *IEEE ASSP Magazine*, July 1990, pp. 26-41.

[8] K.-F. Lee and H.-W. Hon (1989), "Speaker-Independent Phone Recognition Using Hidden Markov Models", *IEEE Trans. Acoust. Speech Signal Process.*, November 1989, Vol. 37, No. 11, p. 1641.

[9] S.E. Levinson, L.R. Rabiner and M.M. Sondhi (1983), "An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition", *Bell System Tech. J.*, April 1983, Vol. 62, No. 4, pp. 1040, 1069-1070.

[10] L.R. Rabiner (1989), "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proc. IEEE*, February 1989, Vol. 77, No. 2, p. 264.

[11] J.N. Holmes (1988), *Speech Synthesis and Recognition* (Chapman & Hall), p. 138.

[12] J.L. Flanagan (1972), *Speech Analysis Synthesis and Perception* (2nd Ed) (Springer-Verlag).

[13] M. Allerhand (1987), *Knowledge-Based Speech Pattern Recognition* (Kogan Page).

[14] W.A. Ainsworth (1988), *Speech Recognition by Machine* (Peter Peregrinus), pp. 79-89, 146-149.

[15] D. O'Shaughnessy (1987), *Speech Communication, Human and Machine* (Addison-Wesley).

[16] D.P. Morgan and C.L. Scofield (1991), *Neural Networks and Speech Processing* (Kluwer Academic Publishers).

[17] N. Morgan and H. Bourlard (1995), "Continuous Speech Recognition", *IEEE Signal Process. Magazine*, May 1995, pp. 25-42.

[18] L.R. Rabiner (1989), p. 259.

[19] Ibid, p. 267.

[20] J. Hertz, A. Krogh and R.G. Palmer (1991), *Introduction to the Theory of Neural Computation* (Addison-Wesley), pp. 142-144, 248-250.

[21] A.M. Peinado, J.C. Segura, A.J. Rubio, V.E. Sanchez and P. Garcia (1994), "Use of multiple vector quantisation for semicontinuous-HMM speech recognition", *IEE Proc. Vis. Image Signal Process.*, December 1994, Vol. 141, No. 6, pp. 391-396.

[22] M.J. Russell and R.K. Moore (1984), "Explicit Modelling of State Occupancy in Hidden Markov Models for Automatic Speech Recognition" (HMSO)

[23] J.J. Nijtmans (1992), "Fast Calculation of Statistical Properties for Languages Based on the New Recursive Markov Model", *Theories and Applications, Proc. 6th European Signal Process. Conf., Eusipco-92*, ed. by J. Vandewalle, R. Boite, M. Moonen and A. Oosterlinck (Elsevier), pp. 443-446.

[24] L.R. Rabiner (1989), pp. 261-266.

[25] S.E. Dreyfus and A.M. Law (1977), *The Art and Theory of Dynamic Programming* (Academic Press), pp. 19-23, 73.

[26] S.E. Levinson, L.R. Rabiner and M.M. Sondhi (1983), pp. 1059-1067.

[27] S. Gong (1992), "Visual observation as reactive learning", *Adaptive and Learning Systems, Proc. SPIE–Internat. Soc. for Optical Eng., Orlando, Florida, 20-21 April 1992*, Vol. 1706, pp. 175-186.

[28] S.E. Levinson, L.R. Rabiner and M.M. Sondhi (1983), pp. 1040, 1069-1070.

[29] J.R. Magnus and H. Neudecker (1988), *Matrix Differential Calculus with Applications in Statustics and Econometrics* (John Wiley & Sons), p. 323.

[30] Ibid, pp. 202-203.

[31] N.J. Pullman (1976), *Matrix Theory and its Applications, Selected Topics* (Marcel Dekker), p. 89.

[32] J.M. Ortega (1987), *Matrix Theory, A Second Course* (Plenum Press), p96.

[33] Open University (1987), *M203 Introduction to Pure Mathematics, Linear Algebra Block, Unit 4 Eigenvectors* (Open University Press), pp. 31-37.

[34] Ibid, pp. 5-17.

[35] T.W. Koerner (1993), *Exercises for Fourier Analysis* (Cambridge University Press), pp. 273-275.

[36] V. Krishnamurthy (1986), *Combinatorics: Theory and Applications* (Ellis Horwood), pp. 15-16.

[37] D. Zwillinger, ed. (1996), *CRC Standard Mathematical Tables and Formulae* (30th Ed) (CRC Press), p. 175.

[38] C. Thornton and B. du Boulay (1992), *Artificial Intelligence Through Search* (Intellect Books).

[39] I.H. Witten (1982), *Principles of Computer Speech* (Academic Press), pp. 128-136.

[40] G.S. Rogers (1980), *Matrix Derivatives* (Marcel Dekker).

[41] J.R. Magnus and H. Neudecker (1988), *Matrix Differential Calculus with Applications in Statustics and Econometrics* (John Wiley & Sons).

[42] Ibid, p. 180.

[43] G.S. Rogers (1980), p. 51.

[44] J.R. Magnus and H. Neudecker (1988), p. 178.

[45] G.S. Rogers (1980), p. 147.

[46] Open University (1987), pp. 31, 48.

[47] F. Ayres (1962), *Theory and Problems of Matrices* (Schaum), pp. 158-159.

[48] J.R. Magnus and H. Neudecker (1988), pp. 40-43.

[49] Ibid, p. 180.

[50] G.S. Rogers (1980), p. 52.

[51] H.A. Taha (1982), *Operations Research, An Introduction* (3rd Ed) (Macmillan), pp. 748-750.

[52] Ibid, pp. 780-784.

[53] D.J. Wilde and C.S. Beightler (1967), *Foundations of Optimization* (Prentice-Hall), pp. 27-30, 99-133.

[54] C.S. Beightler, D.T. Phillips and D.J. Wilde (1979), *Foundations of Optimization* (2nd Ed) (Prentice-Hall), pp. 331-383.

[55] D.J. Wilde and C.S. Beightler (1967), pp. 100-101.

[56] Ibid, pp. 126-127.

[57] Ibid, pp. 101-102.

[58] Ibid, pp. 125-126.

[59] Ibid, p. 111.

[60] C.S. Beightler, D.T. Phillips and D.J. Wilde (1979), pp. 366-375.

[61] D.J. Wilde and C.S. Beightler (1967), p. 107.

[62] L. Childs (1979), *A Concrete Introduction to Higher Algebra* (Springer-Verlag), p. 126.

[63] I. Stewart (1989), *Galois Theory* (2nd Ed) (Chapman and Hall), pp. 6-9.

[64] Ibid, p. 16.

[65] Ibid, p. 17.

[66] Ibid, p. 18.

[67] Ibid, p. 17.

[68] Ibid, pp. 19-20.

[69] Ibid, pp. 18-22.

[70] D.J. Wilde and C.S. Beightler (1967), p. 18.

[71] Open University (1988a), *M371 Computational Mathematics, Block 1, Unit 3, Iterative Methods for Systems of Equations* (Open University Press), pp. 22-24.

[72] Open University (1988a), *M371 Computational Mathematics, Block 1, Unit 1, Introduction to Numerical Methods* (Open University Press), pp. 27-32.

[73] Ibid, p. 27.

[74] D.J. Wilde and C.S. Beightler (1967), p. 14.

[75] J.R. Magnus and H. Neudecker (1988), pp. 118-119.

[76] D.J. Wilde and C.S. Beightler (1967), p. 15.

[77] E.W. Swokowski (1979), *Calculus, with Analytic Geometry* (2nd Ed) (Prindle, Weber & Schmidt), p. 488.

[78] Ibid, p. 492.