MULTIVARIATE ADAPTIVE REGRESSION SPLINE BASED FRAMEWORK FOR STATISTICALLY PARSIMONIOUS ADAPTIVE DYNAMIC PROGRAMMING

by

SUBRAT SAHU

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2011

ACKNOWLEDGEMENTS

First of all, I would like to thank God for all His blessings and my parents for bringing me to life and teaching me to take baby steps towards success with patience and perseverance. Now standing at the doorstep of reaching a career milestone and setting the next one, as I reflect on my past in quest for the defining moment which transformed my career and my life, it is my introduction to and my association with Dr Victoria Chen that stands out conspicuously. I would like to quote Dan Rather who had once said "The dream begins with a teacher who believes in you, who tugs and pushes and leads you to the next plateau". I had also read somewhere “A good teacher is like a candle; it consumes itself to light the way for others.” I have held these sayings dear to my heart; these sayings have been so true in my life and Dr. Chen is the physical embodiment of the “teacher” these sayings refer to. I take this opportunity to express my heart-felt gratitude to Dr. Chen who has been my mentor, my advisor, and my source of inspiration.

I would also like to thank my professors Dr. Jay Rosenberger, Dr. Brian Huff, and Dr. Li Zeng for their consent to be in my doctoral committee.

I would like to acknowledge and express my gratitude for Dr. Donald Liles, Dr. Victoria Chen, the DFW International Airport Authority and the National Science Foundation for the financial support I received from during my doctoral study.

I would also like to thank my friends and my colleagues at the COSMOS Lab for being my support system. I would like to specially mention my gratitude for Mr. Ariyajunya Bancha, Dr. Wei-che Hsu (Michael), Dr. Passakorn Phananiramai, Dr. Chingfeng Lin, Dr. Duang Chatabush, Ms. Misa Faezipour, Ms. Diana Martinez, and Ms. Nadia Martinez who made my life at COSMOS enjoyable. I would like to express my gratitude to my seniors at COSMOS for their

support and guidance. I would like to specially mention my gratitude for Dr. Huiyuan Fan, Dr. Durai Sundaramoorthi, and Dr. Prashant Tarun for their help and support.

Last but not the least, I would like to thank my wife, Shweta for her unflinching faith in me and her ever-enthusiastic encouragement to go one more step farther in pursuit of excellence in my professional as well as my personal life. I would like to thank my newborn baby girl Eesha for bringing some smiles to our faces. I would like to take this opportunity to express my gratitude to my in-laws, my younger brother, Sushant, my younger sister, Nirjharini, and all my other friends and relatives. Without their love and support, I would not have gone so far in life.

May 16, 2011

ABSTRACT

MULTIVARIATE ADAPTIVE REGRESSION SPLINE BASED FRAMEWORK
FOR STATISTICALLY PARSIMONIOUS
ADAPTIVE DYNAMIC PROGRAMMING

Subrat Sahu, PhD

The University of Texas at Arlington, 2011

Supervising Professor: Victoria C. P. Chen

Central to Dynamic Programming (DP) is the 'cost-to-go' or 'future value' function, which is obtained via solving the Bellman's equation, and central to many Approximate Dynamic Programming (ADP) methods is the approximation of the future value function. The exact DP algorithm seeks to compute and store a table consisting of one cost-to-go value for each point in the state space, and its usefulness is limited by the *curse of dimensionality*, which renders the methodology computationally intractable in the face of real life problems with high-dimensional state spaces and in the face of continuous state variables. ADP methodology seeks to address and redress the issue of the curse of dimensionality by not seeking to compute the future value function exactly and at each point of the state space; rather opting for an approximation of the future value function in the domain of the state space. Existing ADP methodologies have successfully handled 'continuous' state variables through *discretization* of the state space and *estimation* of the cost-to-go or future value function and have been challenged in scenarios involving a mix of 'continuous' and 'categorical' or qualitative state variables.

The first part of this dissertation research seeks to develop a flexible, nonparametric statistical modeling method which can capture complex nonlinearity in data comprised of a mix of continuous and categorical variables and can be used to approximate future value functions in stochastic dynamic programming (SDP) problems with a mix of continuous and categorical state variables. This dissertation proposes a statistical modeling method, called 'TreeMARS' which combines the versatility of tree-models with the flexibility of multivariate adaptive regression splines (MARS). An extension of the proposed model, called 'Boosted TreeMARS', is also presented. Comparisons are made to the tree-regression model that uses a similar concept, but only permits the use of linear regression at the terminal nodes. Comparisons are presented on a 10-dimensional simulated data set.

The second part of the dissertation research, seeking statistical parsimony, proposes a sequential statistical modeling methodology utilizing the 'sequential' concept from Design and Analysis of Computer Experiments (DACE) to make the grid 'only fine enough' for the 'efficient' discretization and then use MARS methods to approximate future value functions. This methodology can be extended in the future to use tree-based MARS models to approximate future value functions involving a mix of continuous and categorical state variables. This sequential grid discretization is nothing but sequential exploration of the state space and this concept of sequential exploration of the state space provides a statistically parsimonious ADP methodology which 'adaptively' captures the important variables from the state space and builds approximations around these quantities while seeking approximation of the future value functions, by using adaptive and flexible modeling.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

The concept of Dynamic Programming (DP) put forward by Bellman (1957) provides a paradigm for mathematical modeling and a strategy for solving a multi-stage decision-making problem. It has been popular amongst researchers and practitioners in different areas ranging from engineering to economics, and finance to social science. However, the elegance of this methodology is stymied by the *curse of dimensionality* (Bellman [5], Powell [42]). Wide ranging applications and diversity of the objectives and constraints in different applications has caused the emergence of modified paradigms based on Bellman's original dynamic programming methodology for modeling and solving multi-stage decision-making problems. One such paradigm shift is the emergence of the concept of Approximate Dynamic Programming (ADP) which seeks to address the issue of curse of dimensionality and redress the issue of computational intractability in the exact dynamic programming solution approach, in the face of high-dimensional state and decision spaces.

1.1 Brief Overview of Classical DP Methodology

In a typical DP problem, a system evolves through a series of consecutive *stages*. At each stage the system can be defined by a set of *state variables* or *state vector* (*state* for short). At each stage, one or more *decisions* must be made. These decisions (also called *decision variables* or the *decision vector*) may depend on either the stage or the state or both. At any stage, the past history of the system has no effect on the future evolution of the system, i.e., how it got to the current stage and state has no effect on where it will go in future stages . When a decision is made, a *return* (either profit or cost) is obtained and the system undergoes a *transition* to the next stage. The return is assumed to be a known single-valued function of the state and decision. The *objective* of the DP formulation is to maximize the total profit or

minimize the total cost over all the stages, and the output is the *policy* (series of decisions made in each of the consecutive stages) that achieves the objective. When the state transition is stochastic (in contrast with a deterministic transition given a state and a decision), the DP formulation is called Stochastic Dynamic Programming (SDP).

1.1.1 Typical SDP Formulation

$$\text{Objective}: \min_{u_1,\ldots,u_T} \mathrm{E}\left\{\sum_{t=1}^{T} c_t(x_t, u_t, \varepsilon_t)\right\}$$

$$\text{Transition: s.t. } x_{t+1} = f_t(x_t, u_t, \varepsilon_t), \text{for } t = 1,\ldots,T-1$$

$$\text{Constraints}: u_t \in \Gamma_t, \text{for } t = 1,\ldots,T$$

For stage t: x_t is the vector of state variables.

u_t is the vector of decision variables.

ε_t is the vector of random variables.

$c_t(.)$ is the Cost Function

$f_t(.)$ is the multivariate Transition Function

Γ_t is the set of constraints.

1.1.2 Stages of DP

Dynamic programming seeks to solve complex problems by breaking them down into simpler sub-problems. However, DP is applicable to problems exhibiting the properties of optimal substructure and overlapping sub-problems. These sub-problems correspond to the 'stages' in the DP formulation. Each one-stage problem is dealt with as an ordinary optimization problem, and its solution helps solving the next one-stage problem in the sequence.

1.1.3 State

Powell [42] succinctly defines a state variable as the 'minimally' dimensioned function of history that is necessary and sufficient to compute the decision function, the transition function,

and the contribution function. Here, the word ‘minimally’ is of significance. It can address the issue of the curse of dimensionality associated with the increase in the size of the state space and increasing the fineness of the grid discretization. In our research, we have sought to address this issue by adopting a sequential approach to approximate the future value function using tree-based multivariate adaptive regression splines.

1.1.4 Decision

It is referred to as action, or control, or decision by different users of DP in different fields. A decision is made in ‘each’ stage so as to achieve the minimum ‘total’ cost or maximum ‘total’ profit over all the stages.

1.1.5 State Transition

A state transition function defines the transition of the state variables from the current stage to the next. It models how the system evolves over time or over stages. It can be static over time, i.e., the same for all the stages or it can also be dynamic, i.e., different for different stages. It goes by different names such as ‘transfer function’, ‘system dynamics’, ‘system model’, ‘transition law’, and ‘transition function’, and is modeled differently in different fields (Powell [42]). In the Operations Research (OR) community, the evolution of the system is modeled by a system of equations, such as: $A_t x_t + B_{t-1} x_{t-1} = b_t$. In the Markov Decision Process (MDP) community, system evolution is modeled by a one-step transition matrix: $p(s'|s,a)$. In the control theory community, system evolution is expressed as a transition function that maps the state and decision (and the unknown information which is represented by the random variable) to a new state: $x_{t+1} = F(x_t, u_t, \varepsilon_t)$.

However, most of the time it is assumed specified or known; in rare cases it needs to be computed (Powell [42]). In this research, it is assumed specified and in future research, the proposed methodology can be extended to data-driven estimation of an unknown state transition function.

1.1.6 Exogenous Information

In Stochastic DP, we seek to model problems that are driven by some sort of exogenous information process (for example, observed prices, demand, or rainfall, etc). These are modeled as random variables. However, there is an important distinction between whether the underlying probability distribution is known or not.

The communities which deal with stochasticity in DP as Markov Decision Process (MDP) have standard notation for states and actions; however they do not explicitly model exogenous information; they do not have any standard notation for exogenous information; typically preferring the more compact representation of the one-step transition function.

1.1.7 Policy

A policy is an ordered set of decisions by stage and state. For example, for the inventory problem (Chen [17]), a policy can be the twelve ordering amounts of the commodity dependent on the inventory levels for each of the twelve months. An optimal policy is one that maximizes total profit or minimizes total cost over all the stages.

1.1.8 Recursive Formulation of SDP for Solving Numerically

For stages *t* through *T*:

Future value function, $V_t(x_t) = \min_{(\boldsymbol{u}_1,\dots,\boldsymbol{u}_T)} E\{\sum_{\tau=t}^{T} C_\tau(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{\varepsilon}_\tau)\}$

s.t. $x_{\tau+1} = f_\tau(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{\varepsilon}_\tau)$. for $\tau = t, \dots, T-1$; $u_\tau \in \Gamma_\tau$, for $\tau = t, \dots, T$.

The DP formulation in recursive form for numerical solution is

$V_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_t} E\{c_t(\boldsymbol{x}_t + \boldsymbol{u}_t) + V_{t+1}(\boldsymbol{x}_{t+1})\}$,

where the time horizon, $t = 1, \dots, T$; $\boldsymbol{x}_t$ is the state vector; $\boldsymbol{u}_t$ is the decision vector; $c_t(.)$ is the known cost function for stage t, and $V_t(.)$ is the future value function.

Central to the DP paradigm is the 'cost-to-go' or 'future value' function, which is obtained via solving Bellman's equation. It represents the optimal return from a given stage and state to the last stage or the end of the time horizon. The algorithm seeks to compute and store

a table consisting of one cost-to-go value for each point in the state space. Unfortunately, the size of a state space typically grows exponentially in the number of state variables. The usefulness of this algorithm is limited by the *curse of dimensionality*, which renders the dynamic programming methodology computationally intractable in the face of real life problems with high-dimensional state space. Moreover, this 'exact' (in contrast with 'approximate') DP methodology is relevant when the state of the system is defined completely by a finite set of discrete variables. It breaks down when the variables that define the state of the system include any continuous variables.

These limitations have motivated the research community to look and move beyond the classical Bellman's approach to formulate and solve DP problems.

1.2. Motivation

The essence of the DP paradigm is the 'cost-to-go' or 'future value' function, which is obtained via solving Bellman's equation. The original algorithm computes and stores a table consisting of one cost-to-go value for each point in the state space. Unfortunately, the size of a state space typically grows exponentially in the number of state variables. This limitation, known as the *curse of dimensionality*, renders the original dynamic programming methodology computationally intractable in the face of real life problems with high-dimensional state space.

The exact DP methodology is relevant when the state of the system is defined completely by the discrete variables only. It breaks down when the variables that define the state of the system include any continuous variables. Continuous-state problems require an approximate solution through *discretization* of the state space and *estimation* of the cost-to-go or future value functions. The most basic technique for approximating the solution to a continuous-state SDP problem is to form a finite grid of discretization points in the state space, then to use different interpolation methods to approximate the future value function. Foufoula-Georgiou and Kitanidis [23] used multi-linear interpolation and Johnson et al. [32] used tensor-product cubic spline interpolation for the purpose.

Grid discretization also faces the same *curse of dimensionality*; the size of the state space again grows exponentially with the finer grid and higher dimensionality of the state space. Seeking 'efficiency' in the grid discretization, Chen, Ruppert, and Shoemaker [15] introduced a *statistical modeling* approach using Orthogonal Array (OA) experimental designs to discretize the state space and Multivariate Adaptive Regression Splines (MARS, Friedman [26]) to estimate the future value functions. The existing literature on solving continuous-state DP (e.g. Chen et al. [15] and Fan [22]) handles categorical variables by coding them as discrete variables and then treating them as continuous variables. Thus the existing literature either assumes the state space involves only continuous state variables or is found lacking in handling categorical or qualitative state variables effectively when the state space is comprised of a mix of continuous and categorical state variables. This motivates research to develop a flexible statistical modeling method which can handle a mix of continuous and categorical or qualitative variables for future value function approximation. The logical next step to '*efficient*' state space discretization using a statistical approach (Chen et al. [15]) is to seek 'parsimony' in the statistical approach itself. In pursuit of statistical parsimony, this research proposes a sequential framework for state space discretization using concepts from Design and Analysis of Computer Experiments (DACE) to make the grid finer sequentially and use tree-based Multivariate Adaptive Regression Splines (MARS) to estimate the cost-to-go or future value functions and to approximate the state transition functions when the state and decision space is defined by a mix of continuous and qualitative variables, extending the work of Chen et al. [15] and Fan [22].

1.3. Research Overview

The first part of this dissertation's research seeks to develop a flexible, nonparametric statistical modeling method which can capture complex nonlinearity in data comprised of a mix of continuous and categorical variables and can be used to approximate future value functions in SDP problems with a mix of continuous and categorical state variables. We propose a statistical modeling method, called 'TreeMARS' which combines the versatility of tree-models

with the flexibility of multivariate adaptive regression splines (MARS). An extension of the proposed model, called 'Boosted TreeMARS', is also presented. Comparisons are made to the tree-regression model that uses a similar concept, but only permits the use of linear regression at the terminal nodes. Comparisons are presented on a 10-dimensional simulated data set.

The second part of the dissertation research proposes a sequential statistical modeling methodology utilizing the 'sequential' concept from Design and Analysis of Computer Experiments (DACE) to make the grid 'only fine enough' for the 'efficient' discretization and the tree-based MARS methods to approximate future value functions involving a mix of continuous and categorical state and decision variables. This sequential state space discretization is sequential exploration of state and decision spaces, and this concept of sequential exploration of state and decision space provides a statistically parsimonious ADP methodology which 'adaptively' captures the important variables from the state space and builds approximations around these quantities while seeking approximation of the future value functions, by using the adaptive and flexible modeling methodology, TreeMARS.

CHAPTER 2

REVIEW OF LITERATURE

This section presents a review of existing literature that motivated and inspired this dissertation research.

2.1 Bellman's DP and Beyond

The foundation of dynamic programming is the Bellman's equation and its standard form (Puterman [44]) is written as

$$V_t(S_t) = \max_{x_t}(C(S_t, x_t) + \gamma \sum_{s' \in S} p(s'|S_t, x_t)\, V_{t+1}(s'))$$

where S_t is the state of the system and x_t is the decision taken at stage *t* and its equivalent expectation form (Powell (2007)) is written as

$$V_t(s) = \max_{x_t}(C(S_t, x_t) + \gamma \mathrm{E}(V_{t+1}\{(S_{t+1})|S_t = s\})$$

The classical texts of DP (e.g., Puterman [44]) assume that the state space is discrete and can be represented as *S* = (1,2,…,|*S*|). Additionally, the "textbook" solution to DP assumes that V_{t+1} is known and computes $V_t(s)$ for each $s \in S_t$. (A word of explanation for the sake of notational difference: Control theorists use x for state and u for decision variables which we had followed in our earlier representation.)

If S_t is a discrete, scalar (one-dimensional) variable, enumerating the states is not too difficult. But if it is vector (multi-dimensional) valued (even if discrete), then the number of states grows exponentially with the number of dimensions. If S_t is continuous (even if scalar), the classical DP strategy cannot be applied at all (Chen, et al. [15]), Powell [42]). The essence of approximate dynamic programming (ADP) is to replace the true value function $V_t(S_t)$ with some sort of statistical approximation, $\hat{V}_t(S_t)$. This idea was suggested in Bellman and Dreyfus [5] and

implemented in Chen, et al. [15]. Further complications arise when the decision space is continuous and/or vector valued.

There are two major limitations of the classical exact DP approach. Firstly, it assumes a fully observed system which implies that decisions are based on full knowledge of state space. Secondly, it faces the curse of dimensionality as the size of the state and decision space becomes large. In the real world, the complete knowledge of complex system is rarely known *a priori*. Hence in real-world high-dimensional continuous-state (and continuous-decision) stochastic DP problems, one or both of the limitations become significant and render the classical solution method lacking and computationally intractable. This led to the emergence of a new paradigm for multi-stage decision-making based on Bellman's DP strategy. The new paradigm is popularly known as Approximate Dynamic Programming (ADP) which seeks to achieve the DP optimization by intelligent 'exploration' of the state space and 'exploitation' of the already derived information.

2.2. Approximate Dynamic Programming

Approximate dynamic programming (ADP) is both a modeling methodology and an algorithmic framework for solving multi-stage stochastic optimization problems. Most of the literature has focused on the problem of approximating $V_t(s)$ to overcome the problem of multidimensional state variables. In addition to the problem of multidimensional state variables, there are many problems with multidimensional random variables (i.e., when the expectation has to be taken over multiple random variables), and multidimensional decision variables. These three challenges make up what has been called the three curses of dimensionality (Powell [42]). The concept of ADP has evolved, been extended and enriched to address these curses of dimensionality.

Although the idea of 'approximate' dynamic programming was suggested in the early work by Bellman (see, for example, Bellman and Kalaba [4]) to overcome the curse of dimensionality, the ideas evolved independently within different fields or communities of

researchers, such as the control theory community where problems are often modeled in continuous time, with decision variables (controls) that are typically continuous (Werbos [59], Werbos [60], White and Sofge [61]) and machine learning community where the focus is on exploring the unknown state space. The work in machine learning community has evolved under the name 'Reinforcement Learning' (RL) with seminal contributions from Minsky ([37], [38]), Mendel and McLaren [36], and Sutton and Barto [53]. Work in control theory took place under a variety of names, including reinforcement learning, adaptive dynamic programming and (later) approximate dynamic programming, with important early contributions by Werbos ([59], [60]), White and Sofge [61] and Si et al. [51]. Bertsekas and Tsitsiklis (1996) introduced the name 'neuro-dynamic programming' and used the term interchangeably with 'approximate dynamic programming' (Bertsekas [9]).

The need and emergence of the ADP modeling and algorithmic framework is attributed to the computational difficulties in solving Bellman's equation. There are three computational challenges that arise in solving Bellman's equation which have been called as the *three curses of dimensionality* by Powell [42]. While solving Bellman's equation

$$V_t(x) = \max_{u_t}\left(C(x_t, u_t) + \gamma \, \mathrm{E}\{V_{t+1}(x_{t+1}) \mid x_t = x\}\right); \quad x_{t+1} = F(x_t, u_t, \varepsilon_t),$$

there are three computational challenges (Powell (2007)):

1) Finding the value function $V_t(x_t)$.
2) Computing the expectation.
3) Finding the best action (optimization over the decision variable, u_t).

The nature of these challenges arises from the characteristics of three variables: the state variable, x_t , the exogenous information variable, ε_t, and the decision variable, u_t.

State variables defining the state space can be broadly classified as:

1) Discrete and easy to enumerate (up to thousands of states, but not millions),
2) Scalar and continuous, or
3) Vector valued (which can be either discrete or continuous).

Case (1) is the simplest and Bellman's equation can be solved exactly using classical techniques (see, e.g., Puterman [44]). Case (2) has traditionally been handled through state space discretization (Foufoula-Georgiou and Kitanidis [23], Johnson et al. [32], Chen et al.[15]). Case (3) spans from discrete vectors (e.g., how many cars of each model are in inventory), to continuous vectors (e.g., how much money is invested in each investment choice), to vectors of categorical attributes, and of course to a mixture of all of these.

For the random vector ε_t, we are primarily interested in whether we can compute the expectation in Bellman's equation. Vectors of random variables may be easy for expectation computation if they are independent, and of course the problem is easiest when there are no autocorrelations linking observations over time. It is possible that the random information is simple, but we do not know the probability distribution. For example, the random variable may simply capture whether a person accepts a bid in an auction or not; the random variable is Bernoulli, but we do not know the probability distribution describing the person's behavior. It is also important to separate problems where ε_t is independent of all prior history; problems where ε_t depends on the state, x_t ; and problems where $\varepsilon_t$depends on both the state x_t and the decision u_t.

Decision variables defining the decision space or action space can be broadly classified as

1) Small number of discrete and easy to enumerate decision choices,
2) Scalar and continuous, or
3) Vector valued (which can be either discrete or continuous).

For vector-valued decision space, typically some sort of search algorithm, such as a linear program or genetic algorithm are employed to find the optimal, u_t.

The traditional way to handle continuous state or decision variables is to discretize the state and decision space, and then assume that we can compute expectations. When the number of states and actions is small enough to enumerate when discretized, and when we can

compute expectations, we typically can solve Bellman's equation exactly using classical techniques (Puterman [44]). If the state variable is a vector, discretizing it can produce an exponentially large number of states, a problem that is routinely referred to as the 'curse of dimensionality', a term coined by Bellman [5]. The same problem arises with the information variable ε_t and the decision variable u_t. When the decision variable is a vector, it may be impossible to enumerate all the decision choices. Sometimes, it is possible that the expectation can be computed easily despite the random information ε_t being continuous and sometimes, the expectation cannot be computed because the distribution for the ε_t is unknown. When all three of these problems arise (vector-valued states, incomputable expectation and vector-valued decisions), it is said to have all the *three curses of dimensionality* (Powell [42]). This research focuses on vector-valued state space comprising of a mix of continuous and categorical variables.

2.3. Statistical Approach to Approximate Dynamic Programming

Approximate dynamic programming (ADP) approach seeks to solve a continuous-state DP problem numerically by discretizing the state space, so that the optimization procedure to find the optimal decision state u_t can be carried out only for a finite number of states. Bellman (1957) introduced the idea of 'uniform grid discretization' and immediately realized the *curse of dimensionality*. The researchers, thereafter, followed the same framework of finite grid of discretization of the state space, but sought to *exploit* the underlying statistical structure intelligently by adopting different interpolation methods to approximate the future value function for approximating the solution to a continuous-state SDP problem. Thus, the statistical approach to ADP seeks to replace the true future value function with some sort of statistical model that approximates the future value function.

Foufoula-Georgiou and Kitanidis [23] used multi-linear interpolation in their proposed multilinear, Hermite gradient DP. Johnson et al., [32] used tensor-product cubic spline interpolation for the purpose on a four-reservoir problem and showed that cubic splines required

fewer grid levels in each dimension, hence, reducing computational time. Chen et al. [15] and Chen [17] observed that the above methods were based on a full grid of points and were statistically equivalent to a full factorial experimental design. They proposed an approach based on an Orthogonal Array (OA) experimental design and a Multivariate Adaptive Regression Splines (MARS) approximation. OAs are special subsets of full factorial experimental designs. They observed that while the number of points in a full grid discretization grows exponentially with the number of dimensions, OAs grow only polynomially with the number of dimensions. This greatly reduced the computational effort for high-dimensional problems. Chen et al. (1999) and Chen (1999) achieved good accuracy compared to using a full factorial design with tensor-product cubic spline interpolation by Johnson et al. [32]. Cervellera et al. [12] introduced the use of an alternate experimental design, Latin Hypercube (LH), and an alternate statistical modeling method, neural networks approximation methods into Chen's [17] approach, and obtained comparable results to using OAs and MARS on a nine-dimensional inventory forecasting problem and an eight-dimensional water reservoir problem. Cervellera et al. [12] studied experimental designs based on number-theoretic methods (NTMs) for the purpose and successfully solved a thirty-dimensional water reservoir problem. Cervellera et.al [13], [14] proposed a methodology called semilocal approximate minimization (SLAM) which introduced a semi-local approach based on kernel functions to approximate the solution of *T*-stage stochastic optimization (TSO) problems, which is a typical paradigm of Markovian decision processes. The approach was characterized by less demanding computational requirements and sought to exploit the properties of semilocal approximation through kernel models and efficient sampling of the state space. Fan [22] used the NTM method combined with Neural Network (NN) approximation for the nine-dimensional inventory forecasting problem described in Chen [17].

2.4. Sequential Approach in Approximate Dynamic Programming

If the state and action spaces in a DP model are large, it is often convenient to use an approximate model in order to apply a DP algorithm to obtain an approximate solution. Whitt [62] observed that a natural way to construct an approximate model is to let the new state and action spaces be the subsets of the original state and action spaces; then define the new transition and reward structure using the transition and reward structure of the original model. Having defined the smaller model, calculate the optimal return function and optimal policies for the smaller model and use them to define approximately optimal return functions and approximately optimal policies for the original model by a straightforward extension. In this context, the coarser grid will be a subset of the finer grid. Bertsekas [9] showed that the solution of the discretized algorithm converges to the solution of the continuous algorithm, as the discretization grids become finer and finer. Hence, there is a potential equilibrium in the tradeoff between computational cost of finer grid size and convergence of the solution of the discretized algorithm to the optimal solution.

2.4.1. Design and Analysis of Computer Experiments:

This research seeks to find the optimum discretization in the face of tradeoff between computational cost of finer grid size and convergence of the solution of the discretized algorithm to the optimal solution using the sequential approach based on concepts from Design and Analysis of Computer Experiments (DACE) due to Sacks et al [49]. DACE was developed to replace time-consuming computer models or expensive physical experiments. DACE uses design of experiments (DoE) to explore the sample space and statistical modeling as a meta-modeling tool to model the output from a computer model (Sacks et al [49]). The computer model is run at sample points determined by DoE, and the output responses are used to fit the meta-model.

In DACE meta-modeling, sample points can be generated in batch, and a statistical model is constructed based the whole batch. In a sequential DACE approach, sample points are

selected sequentially; the meta-model is updated sequentially with new sample points; the performance of the meta-model is evaluated each time the meta-model is updated and the sampling process is stopped as soon as the performance of the meta-model meets the set stopping criteria. Sacks et al. [49] presented a simple example of two-stage optimal design of a transistor circuit using a Krigging-based DACE approach to sequentially explore the design choices to optimize the performance criteria of the transistor circuit. They demonstrated that the sequential design was computationally cheaper than the corresponding factorial design. Since then different researchers have contributed to the DACE methodology seeking to make the sampling process more efficient. Lin et al. [35] proposed a sequential exploratory experimental design method in which data points were identified sequentially in regions with large expected prediction errors.

2.4.2. Sequential Sampling Methods

As mentioned before, the traditional full grid discretization is equivalent to the full factorial design and can be too large in practice for high-dimensions, and the state space discretization based on efficient design of experiments is equivalent to a fractional factorial design. In the sequential sampling approach, the sample size is increased by adding newly sampled data batches in each sampling iteration. Therefore, in a sequential sampling approach, the 'space-filling' property which essentially means 'uniformly scattered over the state space', is critical for the success of fractional factorial equivalent state space discretization. Several space-filling methods have been proposed in the literature in recent years which include orthogonal array (OA), Latin hypercube (LH), and number-theoretic method (NTM) (Chen et al [16]).

2.4.3. Quasi-random Sequences

Among the various space-filling methods, NTMs are most suitable for sequential sampling based algorithms. An NTM generates a quasi-random sequence or low-discrepancy sequence which is defined as a sequence of n-tuples that fills the n-dimensional space more

uniformly than uncorrelated random points. Such a quasi-random sequence is of immense utility in computational problems where numbers are generated on a grid, but it is not known in advance how fine the grid must be to achieve a desired accuracy in some parameter which is dependent on the fineness of the grid. Using a quasi-random sequence provides a computationally efficient way to stop when convergence is observed (Press et al. [43]). Although the ordinary uniform random numbers and quasi-random sequences both produce uniformly distributed sequences, there is a fundamental difference between the two. A uniform random number generator on [0,1] produces outputs so that each trial has the same probability of generating a point on equal subintervals, for example [0,1/2] and [1/2,1]. Therefore, it is theoretically possible for all the n-trials to coincidentally all lie in the first half of the interval, while the $(n+1)th$ point still falls within the other of the two halves with probability 0.5. This eventuality is undesirable in a sequential sampling algorithm which specifically requires the space-filling property in each subsequent sampling for its success. The quasi-random sequences preclude this eventuality because the outputs are constrained by a low-discrepancy requirement that has a net effect of points being generated in a highly correlated manner i.e., the next point "knows" where the previous points are (Trandafir et al. [55]).

There are a number of popular quasi-random or low discrepancy sequences, such as the Faure sequence (Henri Faure [30]), Halton sequence (Halton [28]), Hammersley sequence (Hammersley [29]), Niederreiter sequence (Niederreiter [40]) and Sobol' sequences (Sobol' [52]). These quasi-random or low discrepancy sequences are "less random" than a pseudorandom number sequence, but more useful for such tasks as approximation of integrals in higher dimensions, and in global optimization where the convergence of the algorithm depends on the 'space-filling' property of the sampling. Algorithms that use such sequences may have superior convergence because low discrepancy sequences tend to sample space "more uniformly" than random numbers (Niederreiter [40]).

2.4.4. Space-filling Property of Quasi-random Sequences

In the following graphical illustrations in Figure 2.1 and 2.2, the number of sampled points have been increased from 100 to 1000 to 10,000 sequentially using a Sobol' sequence and in the graphical illustration in Figure 2.3, 10,000 points have been sampled using a pseudorandom number sequence. The regions of higher and lower density are evident in illustration D and it is observable that the quasi-random sequences have better space filling property than the ordinary pseudorandom number sequences. In this research, we have used Sobol' and Halton sequences for the sequential sampling of the state space.

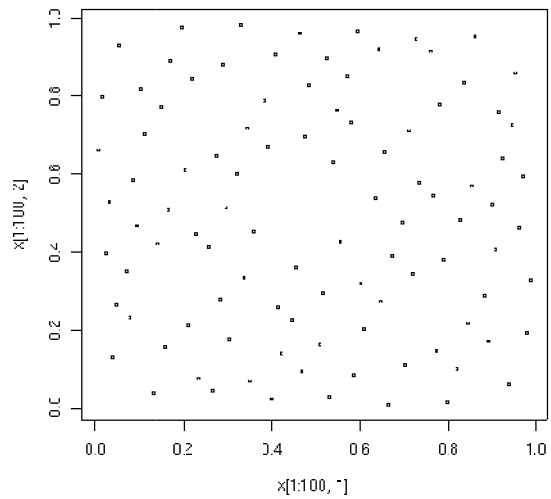
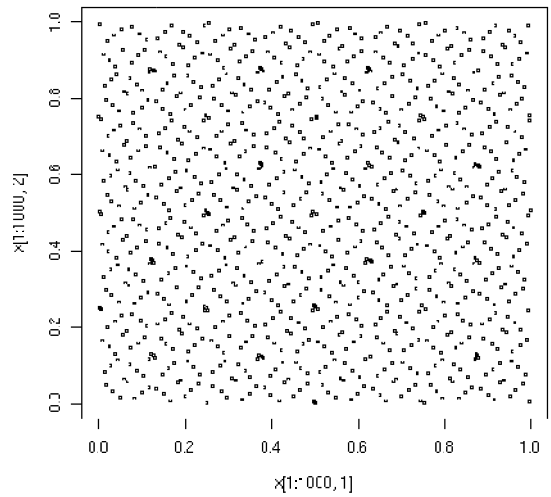
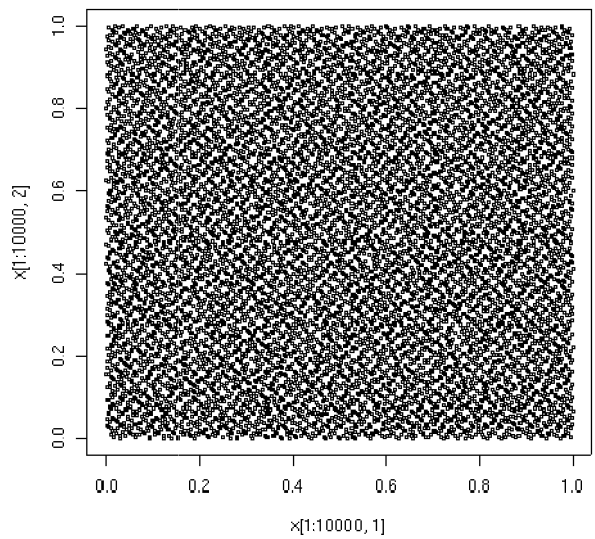


Figure 2.1: Graphical visualization of first 100 points in a low-discrepancy sequence of the Sobol' type
Source: Trandafir and Weisstein (2009)



(a)



(b)

Figure 2.2: Graphical visualization of (a) Adding another 900 points of the same low-discrepancy (b) Adding the next 9000 points of the Sobol' sequence to the existing 1000 points
Source: Trandafir and Weisstein (2009)

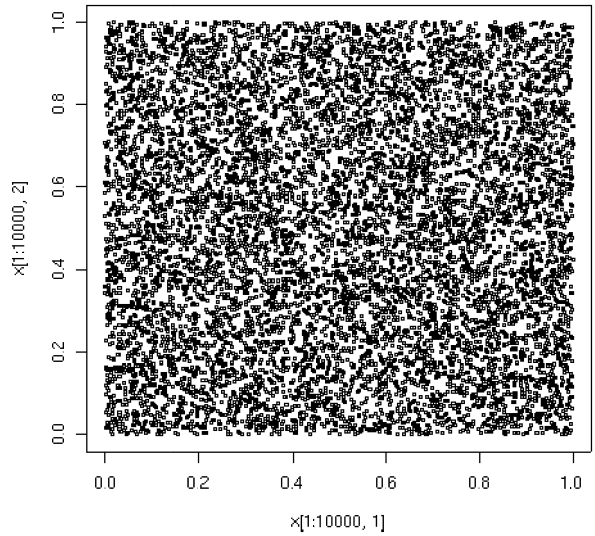


Figure 2.3: Graphical visualization of the first 10000 points in a sequence of uniformly distributed pseudorandom numbers.
Source: Trandafir and Weisstein (2009)

2.5. Flexible Data Modeling

The utility of parametric statistical modeling is limited by the structural assumptions that it imposes on the data and this very fact provides a motivation for moving beyond parametric modeling and exploring the realm of nonparametric statistical modeling when the underlying structure of the data is unknown. The nonparametric models differ from the parametric models in the sense that the model structure is not specified a priori, but it is determined from the data. However, the term 'nonparametric' is not meant to imply that the nonparametric models completely lack parameters, but that the number and nature of the parameters are flexible and not fixed in advance (Corder and Foreman [19]). The nonparametric statistical modeling methods are flexible in the sense that they do not require the structural assumption that data been drawn from a given probability distribution; thus, they are regarded as distribution free modeling methods. The nonparametric statistical modeling methods are also flexible in the sense that they do not assume that the structure of a model is fixed and typically, the model grows in size as the size of the data set grows, to accommodate the complexity of the data.

2.5.1. Moving Beyond Linear Regression

A linear regression model assumes that the regression function $f(X)$ is linear in the inputs X. In real life, the regression function can be nonlinear. The idea of 'basis expansions' in an attempt to deal with nonlinearity, augments or replaces the vector of inputs X with new variables which are transformations of X and the model is represented as

$$f(X) = \sum_{m=1}^{M} \beta_m h_m(X)$$

The model is linear in the basis functions $h_m(X)$. The choice of $h_m(X)$ is used as an instrument to achieve more flexible representation for $f(X)$. The choice of 'polynomials' as a basis expansion is limited by their being global in nature and their inability to capture the local variations in nonlinearity. The choice of 'piecewise polynomials' and 'splines' is used to capture the local variations in nonlinearity (de Boor [21]). However, these choices of 'piecewise polynomials' or 'splines' used to capture the local variations in nonlinearity are 'predefined', which limits their usefulness.

2.5.2. Generalized Additive Models

Tibshirani, and Friedman [29] proposed a more data driven flexible statistical modeling method to capture complex nonlinearity in the data and called it generalized additive models (GAM) which has the form

$$Y=f(X_1, X_2, \dots, X_p) = \alpha + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p) + \varepsilon$$

where $X_1, X_2, \dots, X_p$ are the predictors; Y is the response and f_j's are the nonparametric functions which are modeled by an expansion of basis functions. These additive models provide more flexibility than the linear models in capturing underlying nonlinearity in the data while retaining much of the interpretability of the linear models. The 'backfitting' algorithm proposed by Tibshirani, and Friedman (2002) to model the f_j's fits all the predictors and hence is not suitable for large data-mining applications.

2.5.3. Tree-based Methods

Hastie et al. [29] proposed a tree-based method called classification and regression trees (CART) which recursively partitions the input vector space into a set of hyper-rectangles and fits a simple model in each one. Classification trees can model a categorical response and categorical, as well as quantitative, predictors. Regression trees can model a quantitative response and categorical, as well as quantitative, predictors. The utility of regression trees is limited by its high variance and lack of smoothness. In addition, the regression trees may not capture the additive structure in the data.

Friedman and Fisher [25] proposed the patient rule induction method (PRIM) which slightly differs from tree-based partitioning. PRIM can handle categorical variables like CART, but was designed for regression problems, i.e., quantitative response. Two-class classification problems can be handled by coding the response as 0 and 1, but classification problems with more than 2 classes cannot be handled with PRIM.

Friedman [26] proposed the multivariate adaptive regression splines (MARS) method which is an adaptive procedure for regression. It is well suited for high-dimensional problems, unlike generalized additive models. It can capture the additive structure in the data, which CART could not. However, the MARS method is not effective in modeling categorical predictors.

Jordan and Jacobs [33] proposed the hierarchical mixture of experts (HME) procedure which is a variant of tree-based methods in that the tree-split decisions are probabilistic in nature and can be multi-way, not just binary. In the HME procedure, a linear model is fit in each terminal node, instead of a constant as in CART. However, the research in HMEs has not yet yielded a robust method for finding the tree-topology.

2.6. Insight for Research

The uniform grid discretization is in fact an effort to *explore* the state space; the finer the grid, the more extensive is the exploration. More extensive exploration, however, comes at a cost of computational effort (which sometimes can be prohibitive with the increase in the

dimension of the state space). This high computational cost made researchers to look for ways to intelligently exploit this expensive exploration. A statistical approach seems promising in generalizing the underlying structure of the information provided by the expensive explorations. This generalization of the underlying structure of the information provided by the expensive explorations can be termed as ‘exploitation’ of the accumulated information. A sequential approach seems promising to control the extent of exploration that would accumulate information just enough for effective exploitation.

This research seeks to develop an ADP algorithm using the sequential concepts of DACE for computationally efficient exploration of vector-valued state spaces comprised of a mix of continuous and categorical state variables, and to develop a flexible statistical modeling methodology that would effectively ‘exploit’ the sequentially accumulated information by capturing the underlying generalization in the face of a mix of continuous and categorical variables.

CHAPTER 3

TreeMARS MODELS

This section elaborates on proposed algorithms for flexible statistical modeling to exploit the state space exploration by effectively capturing the underlying complexity.

3.1. Motivation

Among the nonparametric statistical modeling methods, tree-based models proposed by Hastie et al. (2002) in general and the classification and regression tree (CART) models in particular provide for data-driven model fitting while retaining model interpretability. Moreover, CART models can handle categorical as well as quantitative predictors. However, CART models have limitations in that they may not capture the additive structure in the data. Multivariate adaptive regression spline (MARS) models proposed by Friedman [26] can effectively capture the additive structure in the data and also provide for data-driven model fitting while retaining model interpretability. However, MARS models cannot handle categorical predictors effectively. This research presents a flexible statistical modeling method combining the CART and MARS. The proposed Tree-MARS models brings together the best of CART and MARS and provides for a data-driven model fitting method which can handle the mix of categorical and quantitative predictors and effectively capture the additive structure in the data.

This section briefly elaborates on the regression tree models, treed regression models and the MARS models which have motivated Tree-MARS models and the next section presents the proposed TreeMARS models and its variants.

3.1.1. Regression Tree Model

The regression tree model introduced by Breiman et al. [34] in 1984 partitions the whole, and possibly highly heterogeneous, vector space of the data into a number of relatively homogeneous regions through recursive partitioning of the vector space of the predictor

variables. These homogeneous regions are represented as terminal nodes of the regression tree. The tree model fits piecewise constant basis functions by fitting a constant function at each of the terminal nodes. Its strength is its versatility in handling both quantitative and qualitative predictor variables; but the drawback is its poor predictive performance.

A regression tree partitions the predictor variable space into disjoint regions, R_j, $j = 1,\ldots,J$ represented as the terminal nodes of the tree; then fits a constant function at each of the terminal nodes. The predictive rule for assigning an observation $\boldsymbol{x}$ to a terminal node is $\boldsymbol{x} \in R_j \implies f(\boldsymbol{x}) = \gamma_j$. A Tree model can be formally expressed as

$$T(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{J} \gamma_j I(\boldsymbol{x} \in R_j), \text{ with parameter } \boldsymbol{\theta} = \{R_j, \gamma_j\}_{j=1}^{J}.$$

3.1.2. MARS Model

Multivariate Adaptive Regression Splines (MARS), introduced by Friedman (1991), yield an adaptive continuous approximation that does not impose any structural assumption on the data. It fits basis functions composed of single or products of truncated linear functions, with optional smoothing, using least squares estimation. MARS models do capture complex nonlinearity in the data and do provide a data driven and adaptive modeling method for approximation of the future value functions, but they do well with only quantitative predictor variables. Categorical variables can be modeled as 0-1 binary variables, as in linear regression, but this form of modeling does not take advantage of spline structure or tree structure modeling.

MARS is essentially a flexible regression procedure that automatically models interactions between variables. It fits a model that is linear in the basis functions, $\{h_m(\boldsymbol{x})\}_{m=1}^{M}$. MARS model can be formally expressed as

$$\eta(x) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(x)$$

The MARS fitting procedure determines both the basis functions $h_m(x)$ and the parameters, $\{\beta_m\}_{m=0}^{M}$. This is similar in style to regression trees, however while regression trees create basis functions based on piecewise constant functions, MARS creates basis functions

using combinations of truncated linear or hinge functions: $(x-t)_+$and $(t-x)_+$ with a knot or hinge at *t*, defined as:

$$(x-t)_+ = \begin{cases} x-t & for\ x > t \\ 0 & otherwise \end{cases}$$

$$(t-x)_+ = \begin{cases} t-x & for\ x < t \\ 0 & otherwise \end{cases}.$$

3.1.3. Treed Regression Model

The concept of treed regression (Alexander and Grimshaw [1]) was the first attempt to combine regression trees with the traditional linear regression model to capture continuous structure in the data which otherwise would not have been possible using only a regression tree model. Treed regression simply replaces the constant functions at the terminal nodes with linear regression models fit only to the quantitative predictors. This approach effectively fits several localized regression models to capture complexity in the data which would otherwise have remained hidden using a single global regression model.

3.2. Proposed Models

3.2.1. TreeMARS Model

Our proposed TreeMARS modeling method combines the concepts of regression trees and MARS and is an extension of the concept of treed regression. The methodology grows traditional regression trees based on all the predictors, comprised of a possible mix of quantitative and qualitative predictor variables and then fits MARS at the terminal nodes using only the quantitative predictors.

Given the disjoint regions R_j, *j* = 1, 2, …, *J*, as represented by the terminal nodes of the regression tree, let the vector of predictor variables be,

$$\boldsymbol{x} = \{\boldsymbol{x}_c, \boldsymbol{x}_q\},$$

where the subscript ‘c’ represents categorical or qualitative predictors and ‘q’ represents quantitative predictors.

The predictive rule for terminal node j is $\boldsymbol{x} \in R_j \Rightarrow f(\boldsymbol{x}) = \eta_j(\boldsymbol{x})$, where the MARS model on the quantitative predictors is

$$\eta_j(\boldsymbol{x}) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{x}_q); \boldsymbol{x} \in R_j.$$

Thus, a TreeMARS model can be formally expressed as

$$T_{MARS}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{J} \eta_j(\boldsymbol{x}) . I(\boldsymbol{x} \in R_j),$$

with parameter vector $\boldsymbol{\theta} = \{R_j, \boldsymbol{\eta}_j\}_{j=1}^{J}$, where $\boldsymbol{\eta}_j$ is the set of basis functions and their coefficients that define the MARS model fit at the terminal node *j*. The *J* can be treated as the meta-parameter which denotes the number of terminal nodes and defines the size of the tree that we may choose to grow.

Schematically,

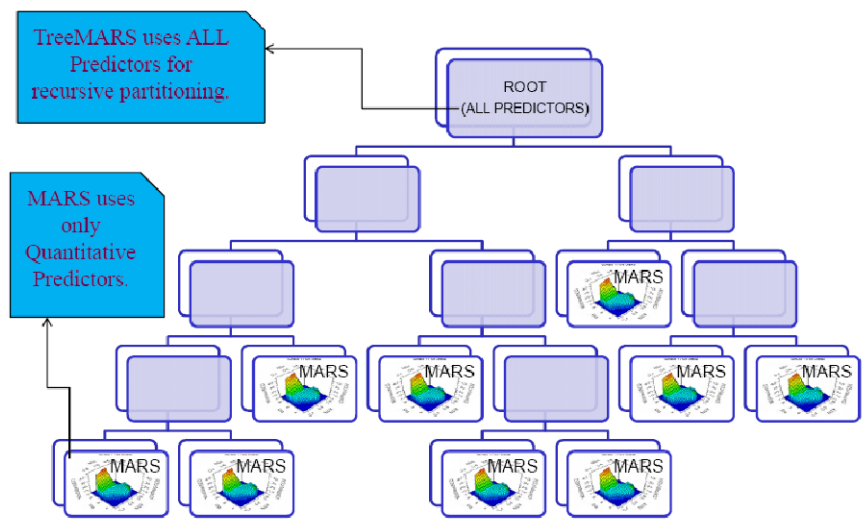


Figure 3.1: Schematic Illustration of TreeMARS Methodology

3.2.1.1. Algorithm

The parameter components $\{R_j\}_{j=1}^{J}$ pertain to growing the regression tree, and the components $\{\boldsymbol{\eta}_j\}_{j=1}^{J}$ pertain to fitting a MARS model at each terminal node *j*. The recursive partitioning algorithm by Breiman et al. [34] is used to grow the regression tree. After the tree is

grown to a desired size, a MARS models is fit at each of the terminal nodes using the MARS algorithm due to Friedman (1992).

3.2.2. CATreeMARS Model

A variant on the above TreeMARS model, called 'CATreeMARS' seeks to grow the regression tree using ONLY the categorical predictors and then fit MARS at the terminal nodes using only the quantitative predictors. This methodology also provides local continuous approximations for the continuous predictors.

Given the disjoint regions R_j, j = 1, 2, … , J, as represented by the terminal nodes of the regression tree grown using only the categorical predictors, let the vector of predictor variables be

$$\boldsymbol{x} = \{\boldsymbol{x}_c, \boldsymbol{x}_q\},$$

where the subscript 'c' represents categorical or qualitative predictors and 'q' represents quantitative predictors.

The predictive rule for terminal node j is $\boldsymbol{x}_c \in R_j \Longrightarrow \boldsymbol{x} \in R_j \Longrightarrow f(\boldsymbol{x}) = \eta_j(\boldsymbol{x})$, where the MARS model on the quantitative predictors is

$$\eta_j(\boldsymbol{x}) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{x}_q);\ \boldsymbol{x} \in R_j.$$

Thus, a CATreeMARS model can be formally expressed as

$$CT_{MARS}(\boldsymbol{x}, \boldsymbol{\theta}) = \sum_{j=1}^{J} \eta_j(\boldsymbol{x}) \cdot I(\boldsymbol{x} \in R_j),$$

with parameter vector $\boldsymbol{\theta} = \{R_j, \boldsymbol{\eta}_j\}_{j=1}^{J}$, where $\boldsymbol{\eta}_j$ is the set of basis functions and their coefficients that define the MARS model fit at the terminal node j. The J can be treated as the meta-parameter which denotes the number of terminal nodes and defines the size of the tree that we may choose to grow.

Schematically,

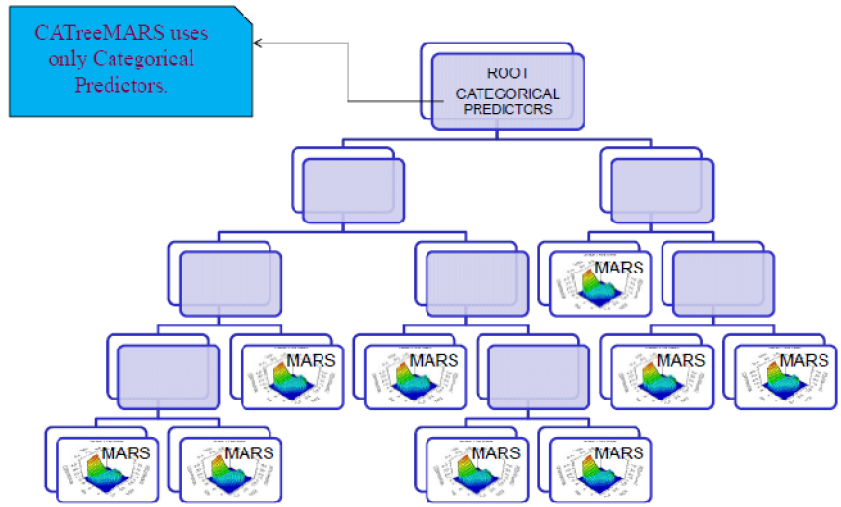


Figure 3.2: Schematic Illustration of CATreeMARS Methodology

3.2.2.1. Algorithm

The parameter components $\{R_j\}_{j=1}^{J}$ pertain to growing the usual regression tree, but with only the categorical predictors at the root. The components $\{\boldsymbol{\eta}_j\}_{j=1}^{J}$ pertain to fitting a MARS model at each terminal node *j* with only the quantitative predictors. The recursive partitioning algorithm by Breiman et al. (1984) is used to grow the regression tree. After the tree is grown to a desired size, a MARS models is fit at each of the terminal nodes using the MARS algorithm due to Friedman [26].

3.2.3. Boosted TreeMARS Model

Gradient boosted regression trees (Friedman [25]) provide a highly flexible statistical modeling method. We extend this idea and propose another modeling method that not only has the flexibility and predictive performance of gradient boosted regression trees, but also results in a smoother model. We continue to use gradient boosting, but now we boost TreeMARS that fits MARS models at the terminal nodes of the regression trees. The boosted TreeMARS model is a sum of multiple MARS regression trees (TreeMARS) and can be represented as

$$f_{BTM}(\boldsymbol{x}) = \sum_{m=1}^{M} T_{MARS}(\boldsymbol{x}, \boldsymbol{\theta}_m).$$

The model is grown in a forward stage-wise manner solving, at each stage,

$$\hat{\theta}_m = arg\,\min_{\theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T_{MARS}(x_i, \boldsymbol{\theta}_m)),$$

for the parameters $\boldsymbol{\theta}_m = \{R_{jm}, \boldsymbol{\eta}_{jm}\}_{j=1}^{J}$ of the next tree, given the current model $f_{m-1}(\boldsymbol{x})$.

Schematically,

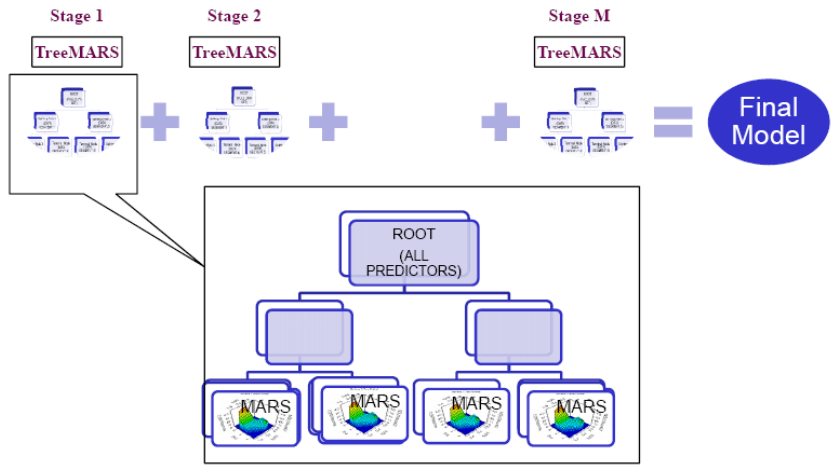


Figure 3.3: Schematic Illustration of Boosted TreeMARS Methodology

3.2.3.1. Algorithm

The MARS regression trees (TreeMARS) are induced in a forward stage-wise manner, and at each stage we solve

$$\hat{\theta}_m = arg\,\min_{\theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T_{MARS}(x_i, \theta_m))$$

for the parameters $\boldsymbol{\theta}_m = \{R_{jm}, \boldsymbol{\eta}_{jm}\}_{j=1}^{J}$ of the next tree, given the current model $f_{m-1}(\boldsymbol{x})$. For squared-error loss, the solution to the above equation simplifies, and it is simply the MARS regression tree (TreeMARS) that best predicts the current residuals $(y_i - f_{m-1}(x_i))$.

In the m^{th}-boosting iteration, the parameter components $\theta_R = \{R_j\}_{j=1}^{J}$, pertaining to growing the regression tree and the components are found by the recursive partitioning algorithm due to Breiman et al. [14]. After fitting the tree, we determine the parameter

components $\boldsymbol{\theta}_\eta = \{\boldsymbol{\eta}_j\}_{j=1}^{J}$ pertaining to fitting the MARS model at each terminal node '*j*' using the MARS algorithm due to Friedman [26].

3.2.4. Boosted CATreeMARS Model

A variant on the above TreeMARS model, called 'CATreeMARS', seeks to boost CATreeMARS models. The boosted CATreeMARS model is a sum of multiple MARS regression trees, where the regression trees are grown with only the categorical predictors at the root and MARS models fitted at the terminal nodes (CATreeMARS),

$$f_{BCTM}(\boldsymbol{x}) = \sum_{m=1}^{M} CT_{MARS}(\boldsymbol{x}, \boldsymbol{\theta}_m).$$

The model is grown in a forward stage-wise manner solving, at each stage,

$$\hat{\theta}_m = arg\ \min_{\theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + CT_{MARS}(x_i, \boldsymbol{\theta}_m)),$$

for the parameters $\boldsymbol{\theta}_m = \{R_{jm}, \boldsymbol{\eta}_{jm}\}_{j=1}^{J}$ of the next tree, given the current model $f_{m-1}(\boldsymbol{x})$.

3.2.4.1. Algorithm

The CATreeMARS models are added in a forward stage-wise manner, and at each stage we solve

$$\hat{\theta}_m = arg\ \min_{\theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + CT_{MARS}(x_i, \theta_m)),$$

for the parameters $\boldsymbol{\theta}_m = \{R_{jm}, \boldsymbol{\eta}_{jm}\}_{j=1}^{J}$ of the next tree, given the current model $f_{m-1}(\boldsymbol{x})$. For squared-error loss, the solution to the above equation simplifies, and it is simply the CATreeMARS model that best predicts the current residuals $(y_i - f_{m-1}(x_i))$.

In the m^{th}-boosting iteration, the parameter components $\theta_R = \{R_j\}_{j=1}^{J}$, pertaining to growing the regression tree with only the categorical predictors at the root and the components are found by the recursive partitioning algorithm due to Breiman et al. [14]. After fitting the tree, we determine the parameter components $\boldsymbol{\theta}_\eta = \{\boldsymbol{\eta}_j\}_{j=1}^{J}$ pertaining to fitting MARS model at each of the terminal node '*j*' using the MARS algorithm due to Friedman (1992) with only the quantitative predictors.

3.3 Evaluation of Proposed Models

The simulated dataset is based on an example from Friedman [6] and is modified to consist of 5 continuous predictors $x_1, \ldots, x_5$ and 5 categorical predictors $x_6, \ldots, x_{10}$ with continuous response *Y*, where

$$\begin{aligned}\mathbf{Y} = 10\mathbf{Sin}\pi(x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \\ 5 * I[x_6 = A] + 5 * I[x_6 = B] + 5 * I[x_6 = C] + \\ 25 * I[x_8 = A] + 27 * I[x_8 = B] + 29 * I[x_8 = C] + \\ 15 * I[x_9 = A] + 17 * I[x_9 = B] + 19 * I[x_9 = C] + \varepsilon; \quad \varepsilon \sim N(0,0.1).\end{aligned}$$

This example was chosen to exhibit complexity due to continuous as well as categorical variables. The simulation run consisted of 100 iterations and the root mean square error (RMSE) was used as the measure to evaluate and compare the performance of the proposed models against the published models.

3.3.1. TreeMARS and CATreeMARS Models

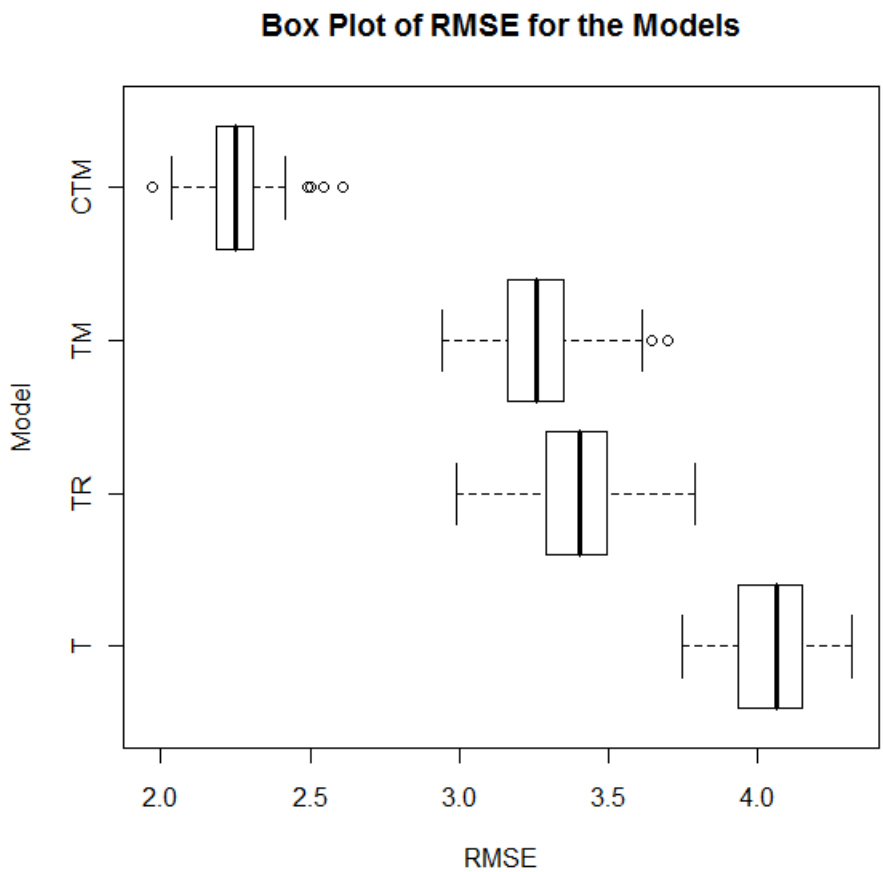


Figure 3.4: Box plots of RMSE after 100 iterations; T = Tree, TR = Tree Regression, TM = TreeMARS, and CTM = CATreeMARS

The results in Figure 3.4 clearly show that the proposed CATreeMARS model performs the best. It also shows that the treed regression model put forward by Alexander and Grimshaw [1] performs better than the traditional regression tree model developed by Breiman et al. [34]. The possible explanation can be that the tree model first captures the nonlinearity and then the linear regression or MARS serve to refine the model at the terminal nodes. These plots also reveal that fitting a regression tree only with the categorical predictors at the root and then fitting the MARS models at the terminal nodes perform much better than fitting regression tree with all the predictors. It can be explained by the fact that traditional tree models do not capture the additive structure effectively, and modeling errors occurring at the top nodes of the tree

propagate to the bottom due to the hierarchical structure of the tree models. This inefficiency results in ineffective capture of additive structure in the data in the presence of inadequate number of observations and strong additive structure in the data. CATreeMARS builds a tree model only on the categorical variables and avoids modeling the additive structure in the quantitative variables while the MARS models effectively capture the additive structure in the quantitative variables in the terminal nodes.

3.3.2. Boosted TreeMARS and Boosted CATreeMARS Models

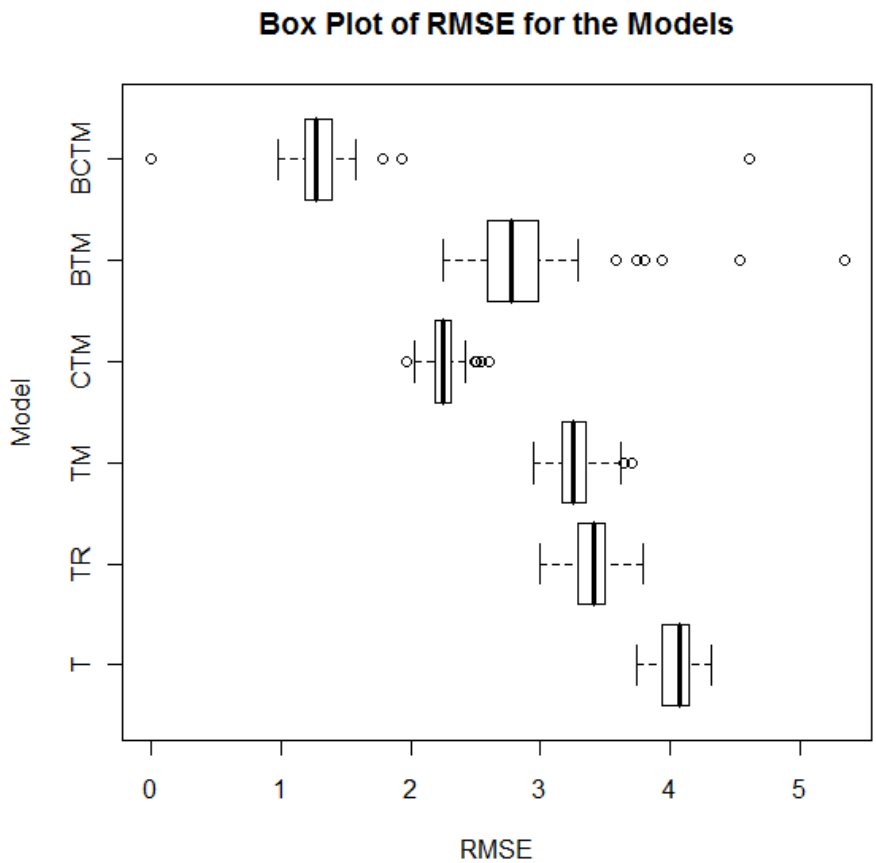


Figure 3.5: Box plots of RMSE after 100 iterations; T = Tree, TR = Tree Regression, TM = TreeMARS, CTM = CATreeMARS, BTM = Boosted TreeMARS, and BCTM = Boosted CATreeMARS.

The results in Figure 3.5 reveal that boosting improves the performance of TreeMARS and CATreeMARS models. The CATreeMARS model is the clear winner. However, CATreeMARS model performs better than the boosted TreeMARS model.

3.4 Discussion of Results

The simulation results establish the superiority of TreeMARS models over treed regression models and traditional regression trees. Boosting further improves the performance of TreeMARS models, but it comes at the cost of much increased computational effort. CATreeMARS models are computationally less expensive than the rest, yet perform better than the rest except its own boosted cousin. Overall, the CATreeMARS models can be ideal for use in a SDP framework to approximate future value functions and state transition functions when the data involve a mix of continuous and categorical variables and exhibits complex nonlinearity.

This research reports the results of boosting methods with only 5 boosting iterations. The performance of boosting methods improves with the increase in boosting iterations, but they become computationally more expensive as well. It will be interesting to study the trade-off between the performance and cost of increase in boosting iterations and find the optimal number of iterations. It will also be interesting to apply the proposed CATreeMARS modeling method to an SDP problem with a large state space with a mix of continuous and categorical variables.

CHAPTER 4

MULTIVARIATE ADAPTIVE REGRESSION SPLINE BASED FRAMEWORK FOR STATISTICALLY PARSIMONIOUS ADAPTIVE DYNAMIC PROGRAMMING

This section elaborates on a proposed algorithm for efficient state space *exploration* and effective *exploitation* of the information obtained through exploration. The proposed methodology adopts a sequential approach for state space exploration inspired by concepts from design and analysis of computer experiments (DACE) by Sacks et al. [49]) and the concept of adaptive value function approximation (AVFA) by Fan [22], which employs neural networks (NN) to '*automate*' the data driven approximation of future value functions. The proposed framework presented in this section is based on multivariate adaptive regression splines (MARS) modeling to achieve '*statistical parsimony*' in data-driven (adaptive) future value function approximation. The proposed framework can be extended to use the tree-MARS models presented in chapter 3 in the presence of categorical variables in the state space.

4. 1 Motivation

Central to dynamic programming (DP) is the 'cost-to-go' or 'future value' function, which is obtained via solving Bellman's equation, and central to approximate dynamic programming (ADP) is the use of an 'approximation' of the future value function. In this context, the 13th century maxim of 'Occam's razor', which symbolized the 'principle of parsimony', has found a renewed relevance in the ADP research community after the statisticians who had already professed their allegiance to the maxim. In the statistics community, model selection boils down to compromises between different aspects of a model. In the compromise, Occam's razor becomes the guiding principle; the principle of parsimony prevails; and the model that fits observations sufficiently well in the least complex way is preferred.

In the context of ADP, the principle of parsimony assumes added significance. Usefulness of the ADP algorithm is limited by its computational cost. Complexity of the model adds to the computational cost, as does the exploration of the state space. Thus, the usefulness of the whole ADP algorithm hinges on finding the approximation model with optimal complexity using minimum state space exploration.

4.1.1 Bias-Variance Tradeoff

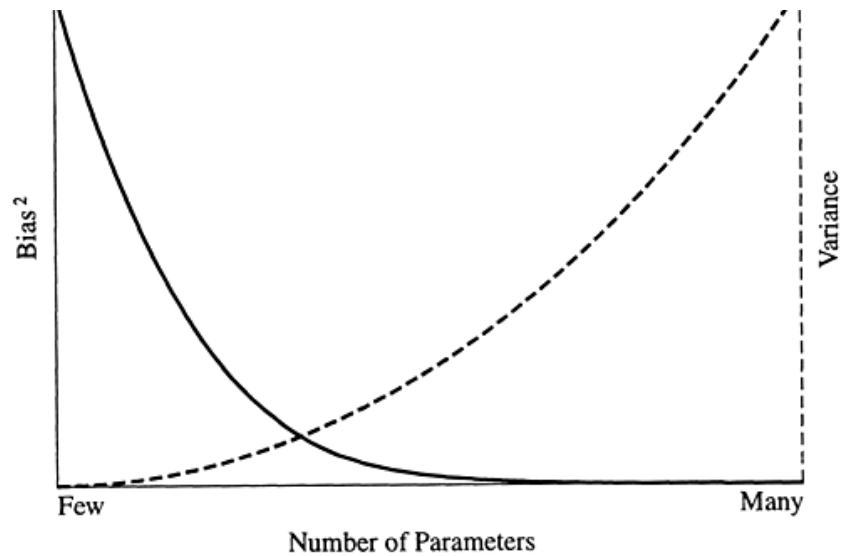


Figure 4.1: Schematic Illustration of Bias-Variance Tradeoff

The real reason behind this resurrection of Occam's razor is, however, more technical than philosophical. It has been observed that the increase in complexity of the model may reduce the bias, which measures how well the model approximates the true function, but it increases the variance in the predictive performance of the model, which makes the quality of prediction poor in terms of reliability or precision (Hastie et al. [29]). That is why the principle of parsimony is ardently adhered to by the research community involved in statistical modeling. In the real world, formalization of this principle is, however, nontrivial.

The two important aspects of the principle of parsimony, in the statistical context, are: 'fitting observations sufficiently well' and 'in the least complex way'. The aspect of 'fitting

observations sufficiently well' has traditionally been quantified by 'goodness-of-fit (GOF)' measures and the aspect of 'model complexity' has traditionally been quantified by 'degrees of freedom (df)' measure which is essentially the number of parameters in the model. In the context of Multivariate Adaptive Regression Splines (MARS), the degrees of freedom (df) is the number of basic functions (BF) in the model, and it indicates the complexity of the model. The GOF statistic can be made to look good by increasing the complexity of the model which at the same time increases the degrees of freedom (df) of the model, i.e., if we increase the complexity of the model, the GOF statistic may indicate that the model fits the data better, hence the model appears superior without any reference to the degree of freedom. Thus, the GOF statistic does not capture the conflicting objectives of increasing model fit and decreasing model complexity and does not tell much about the predictive performance of the model.

4.1.2 Generalization Error

Let f be a regression function, Y=$f(\mathrm{X}) + \varepsilon$, where ε is the random error independent of X and with mean zero and $\hat{f}$ be an estimate of the function f. Let's define the loss function for measuring the error between f and $\hat{f}$ as

$$L(f, \hat{f}) = \left(f(\mathrm{X})\text{-}\hat{f}(\mathrm{X})\right)^2$$

The 'generalization error' is defined as the expected loss or expected prediction error over an independent test sample. This is also referred to as 'test error'. This expectation averages anything that is random including the randomness in the training sample that produced $\hat{f}$.

Test Error = $E\left[L(f, \hat{f})\right] = E\left|\left(f(\mathrm{X})\text{-}\hat{f}(\mathrm{X})\right)^2\right| = E\left|\left(\mathrm{Y}\text{-}\hat{f}(\mathrm{X})\right)^2\right|$ over an independent test sample.

The training error is defined as the average loss or average training error over the training sample.

Training Error = $\frac{1}{\mathrm{N}}\sum_{i=1}^{\mathrm{N}} L(f, \hat{f}) = \frac{1}{\mathrm{N}}\sum_{i=1}^{\mathrm{N}} L(f(x_i), \hat{f}(x_i)) = \frac{1}{\mathrm{N}}\sum_{i=1}^{\mathrm{N}}\left(y_i\text{-}\hat{f}(x_i)\right)^2$ over the training sample.

The beauty of the generalization error is that it can be decomposed into bias and variance, such that Test Error = $Bias^2$ + Variance, and it is not true in the case of training error (Hastie et al. [29]). If we go on increasing the model complexity, the training error decreases monotonously, but not the test error.

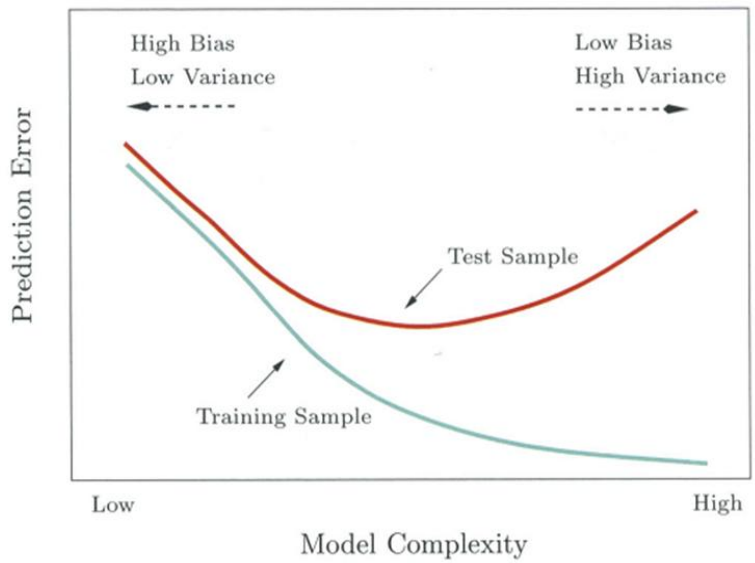


Figure 4.2: Illustration of interplay between training error and test error.
Source: T. Hastie, R. Tibshirani , J. H. Friedman (2001)

When we increase the model complexity too much, the model adapts too closely to the training data and loses the 'generalization ability', and the test error increases in the independent test data. This due to the fact that, as the model becomes complex, it can capture more complex underlying structure in the data which results in the decrease in the bias of the estimating function, but it also leads to increase in the variance of the estimating function. As long as the decrease in the bias component more than compensates for the increase in the variance component, the estimation error in the test data set decreases. Beyond certain model complexity, the rate of decrease in bias is more than offset by the rate of increase in variance, and the estimation error increases in the training dataset. In the continuum of model complexity, there is an optimal model complexity that gives the minimum test error.

This research will seek to incorporate the test error in the model selection criterion in developing a data driven algorithm to build flexible statistical models of optimal complexity.

4.1.3 Consistency Trace

'Consistency' is defined as the asymptotic convergence of an estimator to the object of estimation. Analytical studies reveal that most nonparametric regression algorithms, which include multivariate adaptive regression splines (MARS), are consistent for approximating any regression function and the rate of convergence, depending on the particular algorithm and the underlying function it seeks to approximate (Geman et al [27]). This is a reassuring property of the MARS approximation that we will seek to exploit. The observation of Fan [22] that the nonparametric regression function approximations follow a 'consistency trace' is of practical significance.

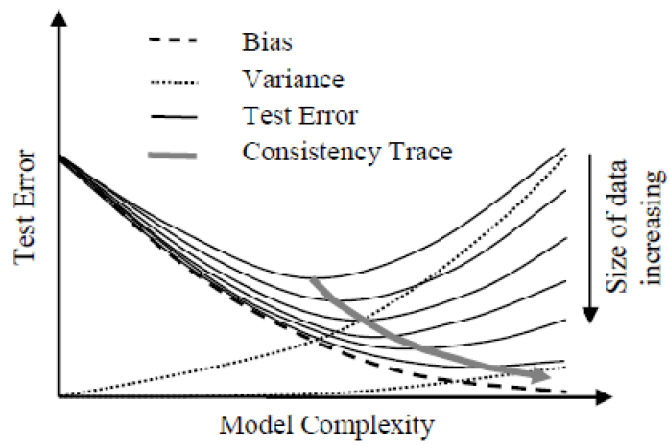


Figure 4.3: Illustration of Consistency Trace.
Source: Fan, Huiyuan (2008)

The above consistency trace reveals that the simultaneous increase in model complexity and the size of training data can force the nonparametric regression function approximation models to follow the consistency trace, if the nonparametric regression function approximation algorithm is consistent. The rate of convergence may depend on the optimality of the model complexity. This research will seek to exploit this observation regarding the

consistency trace to develop data driven stopping rules for optimal model complexity to achieve statistical parsimony.

4.2 Choice of MARS

This research seeks to develop a data driven algorithm to build flexible statistical models of optimal complexity for use in ADP to adaptively and adequately approximate future value functions.

In the nonparametric statistical modeling community, multivariate adaptive regression splines (MARS) and neural network (NN) have emerged as modeling methods of choice (Francis [24]). NNs have been touted to have the ability to approximate any arbitrary function by learning from the observed data (Francis [24]). However, MARS has emerged as a viable alternative to NN in that it can capture complex nonlinearity and the interactions in the data like NN, but it does not work in a black box fashion (Berry and Linoff [6]). The structural components of MARS are explicit whereas NN works in a black box fashion where the input enters the black box and gets transformed into the output.

The users of NN claim that the greatest advantage of NNs is their ability to approximate any arbitrary function by learning from the observed data. Francis [24] claims that a NN architecture with a sufficient number of hidden nodes in the NN's hidden layer can approximate any deterministic nonlinear continuous function accurately. However, the 'trial and error' process of identifying the appropriate NN architecture may be computationally too expensive. Other than that, there are numerous algorithms available for training neural network models and using them is not so straightforward. Selecting and tuning an algorithm for training on unseen data requires a significant amount of experimentation and is data specific. This experimentation is not trivial enough to be generalized for use in a framework that can be used to make the model follow a consistency trace and the extensive study using NN models by Fan [22] establishes this very fact. The study by Fan [22] also does not conclusively establish that NN models can be controlled for complexity solely through the number of hidden nodes without

doing the data specific experimentation to tune the algorithm used. If a nonparametric model cannot be directly controlled for complexity, then it cannot be made to follow the consistency trace efficiently; so the convergence rate in the consistency trace will be poor.

The very structure of MARS is such that it lends itself naturally to control for complexity from the model building stage and hence lends itself to control for consistency. Apart from that, MARS can capture complex nonlinearity and the underlying interactions in the data efficiently and can be used to approximate any arbitrary regression function (Friedman [26]). Therefore, this research has decided on MARS as the nonparametric statistical modeling of choice for use in the adaptive dynamic programming framework.

Shen et al. [50] propose an adaptive model selection procedure aimed at reducing selection bias and improving performance by introducing a complexity penalty determined by the data itself. However, this complexity penalty is not used at the model building stage and is used during the model selection stage, which might be too late from the perspective of computational cost savings. In MARS, the model complexity parameter is built-in into the model building process.

4.3 Features of MARS

The MARS procedure for estimating any arbitrary regression function consists of a forward stepwise algorithm to select certain spline basis functions followed by a backward stepwise algorithm to delete basis functions until the best set of basis functions is found that has the lowest generalized cross validation (GCV) error among all the possible sets of basis functions. Algorithms 2 and 3 of Friedman (1992) describe the forward stepwise selection and backward stepwise deletion procedures, respectively. The purpose of the backward stepwise algorithm is to prevent over-fitting by decreasing the complexity of the model without degrading the fit to the data. Algorithm 3 loops through the set of basis functions chosen by Algorithm 2 to find ones to delete, so the advantages of Algorithm 3 must be weighed against the computational effort required in this search.

The MARS forward stepwise algorithm is used to create the basis functions of the MARS model. The forward stepwise algorithm loops through the possible choices for basis functions, covariates and knot locations in selecting the next two basis functions to add to the model. MARS is an *adaptive* procedure because the selection of basis functions is data-based and specific to the problem at hand. The forward stepwise algorithm stops when *M*max basis functions have been selected, where *M*max is a user-specified constant. The MARS approximation approaches interpolation as the number of basis functions increases, but there is a tradeoff between *M*max and computational time.

As mentioned, the MARS forward stepwise algorithm is used to create the basis functions of the MARS model, and the process stops when a user-specified value, *M*max is reached. However, there does not exist a rule to select an appropriate value of *M*max. An improper *M*max may cause problems of over-fitting or under-fitting. Meanwhile, using an *M*max larger than necessary may be a waste of time, which is a critical issue for SDP. Tsai and Chen [56] have presented automatic stopping rules to provide an efficient way of choosing *M*max.

4.4 Structure of MARS

Let f be a regression function, Y=$f(\mathbf{X}) + \varepsilon$, where ε is the random error independent of $\mathbf{X}$ and with mean zero, and $\hat{f}$ be a MARS approximation of the function f. The MARS approximation has the form

$$\hat{f}(\boldsymbol{X}) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{X}),$$

where $\boldsymbol{X}$ is a v-variate vector of predictors and $h_m(\boldsymbol{X})$ is a basis function, M is the number of linearly independent basis functions, and β_m is the unknown coefficient for the m-th basis function $h_m(\boldsymbol{X})$. The basis function $h_m(\boldsymbol{X})$ has the form

$$h_m(\boldsymbol{X}) = \prod_{l}^{L} \left[s_{l,k} \left(x_{v(l,k)} - k_{l,m} \right) \right]_+.$$

where $[z]_+$ is the hinge function defined as max{0, z}, L is the maximum order of predictor-predictor interaction allowed, $x_{v(l,k)}$ is the input variable corresponding to the l-th hinge function in the m-th basis function, $k_{l,m}$ is the knot value corresponding to the $x_{v(l,k)}$, and $s_{l,k}$ takes the

values -1 and +1 corresponding to the pair of basis functions for the each combination of $x_{v(l,k)}$ and $k_{l,m}$. The forward stepwise algorithm in MARS adds basis functions in pairs in each iteration and loops through the possible choices for basis functions (m), covariate (v) and the knot location (k) to select the next two basis functions to add to the model. The process stops when a user-specified value, *M*max is reached.

4.5 General Framework for Solving Continuous SDP

Typical SDP formulation is

$$\text{Objective}: \min_{u_1,\ldots,u_T} \mathrm{E}\left\{\sum_{t=1}^{T} c_t(x_t, u_t, \varepsilon_t)\right\}$$

$$\text{Transition}: \text{ s.t. } x_{t+1} = f_t(x_t, u_t, \varepsilon_t), \text{ for } t = 1,\ldots,T-1$$

$$\text{Constraints}: u_t \in \Gamma_t, \text{ for } t = 1,\ldots,T$$

For stage t: x_t is the vector of state variables.

u_t is the vector of decision variables.

ε_t is the vector of random variables.

$c_t(.)$ is the Cost Function

$f_t(.)$ is the multivariate Transition Function

Γ_t is the set of constraints.

For numerical solution, the SDP is formulated in a recursive form, and for stages *t* through *T*, we define future value function as

$$V_t(x_t) = \min_{(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_T)} E\{\textstyle\sum_{\tau=t}^{T} C_\tau(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{\varepsilon}_\tau)\}$$

$$\text{s.t. } x_{\tau+1} = f_\tau(\boldsymbol{x}_\tau, \boldsymbol{u}_\tau, \boldsymbol{\varepsilon}_\tau). \text{ for } \tau = t, \ldots, T-1 \text{ ; } u_\tau \in \Gamma_\tau, \text{for } \tau = t, \ldots, T.$$

The DP formulation in recursive form for numerical solution is

$$V_t(\boldsymbol{x}_t) = \min_{\boldsymbol{u}_t} E\{c_t(\boldsymbol{x}_t + \boldsymbol{u}_t) + V_{t+1}(\boldsymbol{x}_{t+1})\},$$

where the time horizon, $t = 1, \dots, T$; $\boldsymbol{x}_t$ is the state vector; $\boldsymbol{u}_t$ is the decision vector; $c_t(.)$ is the known cost function for stage t, and $V_t(.)$ is the future value function.

The future value function $V_t(.)$ represents the minimal cost of operation from stage t through T. given the system is in state x_t and entering stage t. The goal is to compute the future value function $V_t(.)$ and find the optimal decision u_t^* . The DP is solved backwards recursively, first finding $V_T(.)$ for the last stage and subsequently using it recursively to find $V_{T-1}(.)$ through $V_1(.)$. At the end of this exercise, for each point in the state space for stages 1 through T, we have (x_t , $V_t(x_t)$) and (x_t , u_t^*), and this exercise involves as many minimization problems as the number of state points in each stage. This is essential the solution to the SDP problem; given the initial state of the system x_1, we have the $V_1(x_1)$ which gives the optimal cost of operation of the system from stage 1 through T.

This traditional recursive solution framework for SDP becomes computationally expensive as the number of state points for stages 1 through T increases. In the presence of continuous state variables, this traditional recursive framework completely breaks down. Discretization of the state space coupled with some interpolation technique is adopted to approximate the future value function for any state point x_t in stage t. The typical algorithm for solving an continuous SDP problem numerically is as follows:

Step-1: Choose N discretization points in the state space $\{\boldsymbol{x}_{jt}\}_{j=1}^{N}$ for the t-th stage; $t = 1, \dots, T$; and $\boldsymbol{x}_{jt} \in R^n$

Step-2. In the last period T,

(a) for each discretization point $\boldsymbol{x}_{jT}$, $j = 1, \dots, N$, solve for

$V_T(\boldsymbol{x}_{jT}) = \min_{\boldsymbol{u}_{jT}} E\{C_T(\boldsymbol{x}_{jT}, \boldsymbol{u}_{jT}, \boldsymbol{\varepsilon}_{jT})\}$, then

(b) approximate $V_T(\boldsymbol{x}_T)$ with $\hat{V}_T(\boldsymbol{x}_T)$ for all $\boldsymbol{x}_T \in R^n$.

Step-3. In each period $t = T-1, \dots, 1$,

(a) for each discretization point $\boldsymbol{x}_{jt}$, $j = 1, \dots, N$, solve for

$\tilde{V}_t(\boldsymbol{x}_{jt}) = \min_{\boldsymbol{u}_{jt}} E\{C_t(\boldsymbol{x}_{jt}, \boldsymbol{u}_{jt}, \boldsymbol{\varepsilon}_{jt}) + \hat{V}_{t+1}(f(\boldsymbol{x}_{jt}, \boldsymbol{u}_{jt}, \boldsymbol{\varepsilon}_{jt}))\}$, then

(b) approximate $\tilde{V}_t(\boldsymbol{x}_t)$ with $\hat{V}_t(\boldsymbol{x}_t)$ for all $\boldsymbol{x}_t \in R^n$ as in step 2(b).

This discretization of the state space makes the recursive solution framework work for continuous-state SDP. Then some kind of approximation or interpolation is required to de-discretize any inference over the state space. The quality of approximation is enhanced with the increase in fineness of the discretization grid. However, the computational cost of the recursive solution process increases with the increase in the number of discretization points. This tradeoff between the quality of approximation and the cost associated with it has motivated this research to seek ways to find the optimal discretization that would give the optimal quality of approximation.

To achieve the goal of optimal discretization and optimal quality of approximation, this research seeks to develop an algorithm that would make the statistical modeling method used for approximating the future value function to follow the consistency trace.

4.6 Proposed Framework

The objective of this research is to develop an algorithm that would make the statistical modeling method used for approximating the future value function to follow the consistency trace. This research selected the MARS method as the nonparametric statistical modeling method of choice because the MARS modeling algorithm is consistent, i.e., it can be made to follow the consistency trace, and the very structure of MARS lends itself to easy control for complexity from the model building stage and hence promises to lends itself to efficient control for consistency. However, the quality of convergence in the consistency trace depends on the algorithm used. This research explores three algorithms for building MARS models including the original MARS algorithm proposed by Friedman [26] to study their amenability to follow the consistency trace.

The structure and very way of model building gives the opportunity to control for complexity from the model building stage. There are different ways we can implement the

MARS algorithm. We can manually control the *M*max value in the forward training state to control for complexity and then allow for backward pruning as in the original MARS algorithm proposed by Friedman [26]. We can also use the modified MARS algorithm with automatic stopping rule proposed by Tsai and Chen [56] to let the training data control the *M*max value.

In the following proposed sequential framework for finding out the model with optimal complexity using minimal state space exploration, this research studies three versions of the MARS algorithm for their amenability to follow the consistency trace.

4.6.1. Flowchart of the Proposed Framework

In the proposed framework, the algorithm starts with the input of the initial size of the training data taken from a low-discrepancy quasi-random sequence and fits a model of optimal complexity for the size of the training dataset. At the end of the sequential step, the model with optimal complexity is evaluated in a fixed test dataset, and stopping criteria is checked for compliance. The percentage of variance in the test response NOT explained by the model is used as the stopping criterion. If the stopping criteron has not been met, then the algorithm proceeds to the next sequential step with a pre-specified increase in the size of the training dataset. The sequential iterations continue until the stopping criterion is met which implies that the percentage of variability in the test data not explained by the model is less than the specified limit.

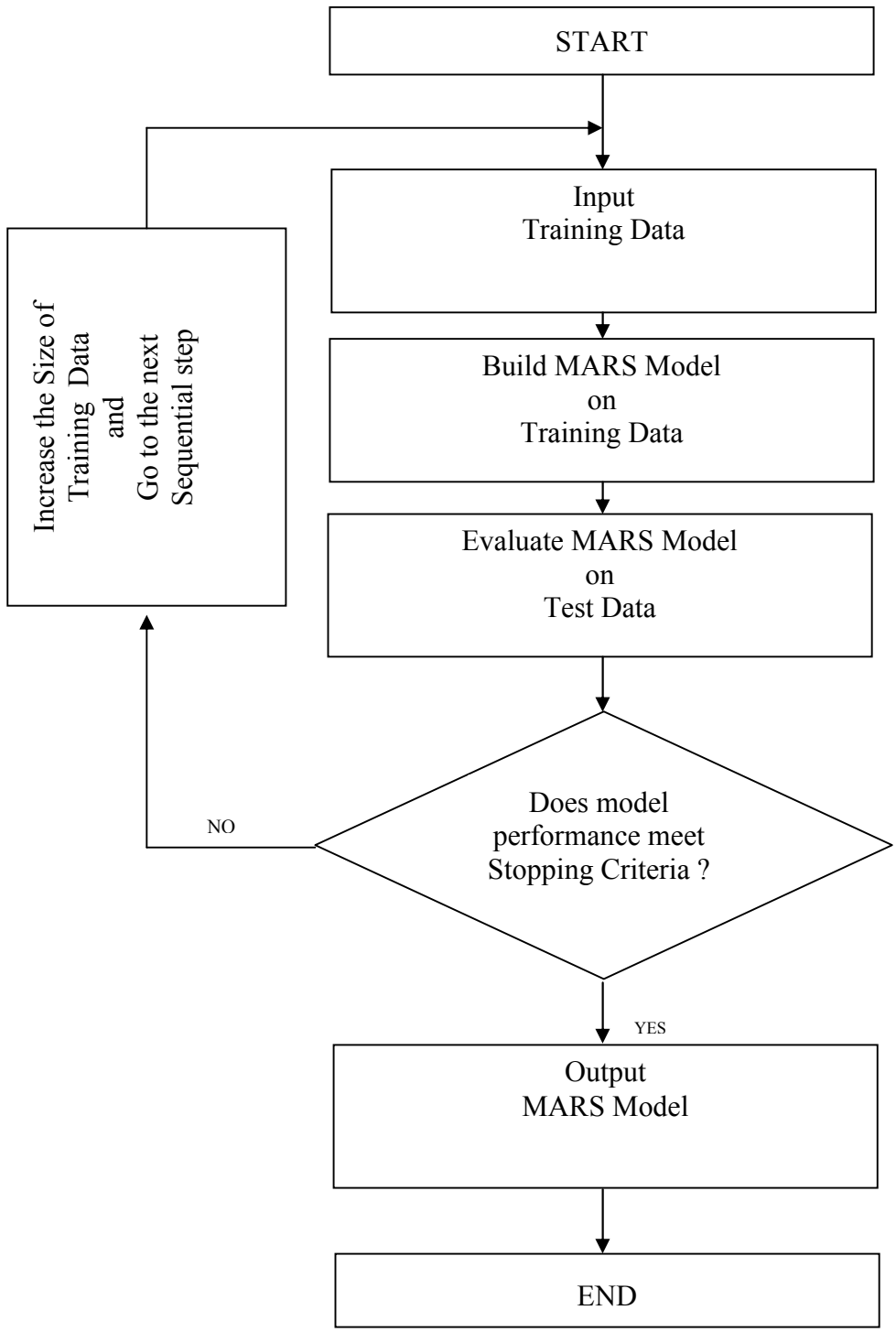


Figure 4.4: Flowchart of Proposed Sequential Algorithm

4.6.2. Three Versions of MARS Algorithm

As mentioned, there are different ways the MARS algorithm can be implemented. The first version is the original MARS algorithm proposed by Friedman [26] with a default value for Mmax as twice the number of covariates. The forward pass stops once the *M*max number of basis functions has been selected and is followed by a backward pruning pass. In this version, the *M*max parameter is fixed for each sequential run of the algorithm, and the backward pruning controls the optimal complexity of the final model in each sequential run. The rationale behind this version of algorithm is that it provides one of the possibilities to follow consistency trace without having to manually control the model complexity. The study will reveal if this forced cap is adequate to attain optimal model complexity and eventually follow consistency trace as the size of the training dataset increases. The study will also reveal how this limit on the model complexity influences the generalization ability of the model in the sequential methodology.

The second version is a modified implementation of the original MARS algorithm with the value of *M*max = $\left\lfloor \frac{2n+c}{2+c} \right\rfloor$, where n is the number of data points in the training data set, and c is the penalty parameter set to the default value of 3. The rationale behind this cap on the maximum number of basis functions is that this is the maximum number of basis functions a model can have after the backward pruning algorithm. The backward pruning algorithm chooses a model with minimum generalized cross validation (GCV) error. The GCV is defined as

$$GCV = \frac{MSE_{train}}{(1 - {enp}/{n})^2}$$

The enp is the effective number of parameters which is defined as

$$enp = m + c * (m - 1)/2$$

For model with $enp = n$, GCV is infinite and any model with $enp \geq n$ is considered infinitely bad (Jekabsons [31]). Hence, a model can have a maximum of $\left\lfloor \frac{2n+c}{2+c} \right\rfloor$ basis functions in order for the effective number of parameters to be less than n. This study will reveal how this floating

limit on the model complexity influences the generalization ability of the model in the sequential methodology.

The third version is the modified MARS algorithm with automatic stopping rule proposed by Tsai and Chen [56] that lets the training data control the *M*max value.

4.7 Application to Inventory Forecasting Problem

The proposed sequential framework to approximate future value function using three algorithms for MARS modeling can be applied to traditional backward solved ADP problems. This research seeks to study the performance of the resulting sequential ADP methodology based on MARS modeling on a nine-dimensional stochastic inventory forecasting problem (Chen [17]). The choice of this particular SDP problem is for easy comparison and benchmarking, as this nine-dimensional stochastic inventory forecasting problem has been extensively studied and frequently used for performance comparison by researchers in the past (Chen et al. [15]; Chen [17]; Cervellera et al. [12]; Fan [22]).

4.7.1. Overview of Stochastic Inventory Forecasting Problem

The nine-dimensional stochastic inventory forecasting problem (Chen [17]) is concerned with optimal order quantities for three items over two forecast periods given the inventory level and demand forecast for each item.

The state variables consist of current inventory level for each item and the demand forecast for each item over two time periods.

Define, at the beginning of the time period (stage) t:

$I_t^{(i)}$: Inventory level for item i at the beginning of current period t

$D_{t,t}^{(i)}$: Forecast of demand for item i over the current period t

$D_{t,t+1}^{(i)}$: Forecast of demand for item i over the next period $(t+1)$

$u_t^{(i)}$: Amount of item i ordered at the beginning of current period t

The state vector at the beginning of stage t is represented by

$$x_t = (I_t^{(1)}, I_t^{(2)}, I_t^{(3)}, D_{t,t}^{(1)}, D_{t,t}^{(2)}, D_{t,t}^{(3)}, D_{t,t+1}^{(1)}, D_{t,t+1}^{(2)}, D_{t,t+1}^{(3)})^T .$$

The decision vector at the beginning of stage t is represented by

$$u_t = (u_t^{(1)}, u_t^{(2)}, u_t^{(3)})^T.$$

The transition function from period t to period $(t+1)$ is given by

$$x_{t+1} = x_t + u_t - D\ .$$

The associated stochasticity in transitions is modeled using the multiplicative Martingale model of forecast evolution (See Chen [17] for details). The constraints on the decision variable (amount ordered) and the state variables (inventory levels) are placed in the form of capacity constraints.

The objective function is a cost function involving inventory holding costs and backorder costs. The cost function is typically V-shaped and is represented as

$$c_v(x_t, u_t) = \sum_{i=1}^{3} \left(h_i \left\lfloor I_{t+1}^{(i)} \right\rfloor_+ + \pi_i \left\lfloor -I_{t+1}^{(i)} \right\rfloor_+ \right),$$

where $\lfloor z \rfloor_+ = \max\{0, z\}$; h_i is the inventory holding cost for item i ; π_i is the backorder cost for item i. For optimization purposes, a smoothed version of the cost function has been used (See Chen [17] for details).

4.7.2. Computational Setup

MATLAB software (www.mathworks.com) was used to code the sequential algorithms to approximate the future value functions for the above mentioned inventory forecasting problem. "ARESLab: Adaptive Regression Splines toolbox for MATLAB" developed by Jekabsons [31] was used to implement the three versions of MARS outlined in section 4.6.2.

The nine-dimensional inventory problem involved three time periods or stages. So according to the general framework for solving continuous-state SDP in section 4.5, only two future value functions, i.e., one for the second stage and one for the third (last) stage, needed to be approximated. The future value function for the third stage was approximated first using the MARS modeling method. The future value function approximation for the second stage required the use of the already built future value function approximation model for the third stage. For both the stages, the training data used to build the future value function approximation models

came from a nine-dimensional Sobol' sequence. The test data consisted of 2000 design points from a nine-dimensional Halton sequence.

Experimentation was undertaken to study the amenability of the three versions of the MARS algorithm to follow the consistency trace and then the performances of the proposed sequential algorithms were compared with the performance of one of the algorithms (Algorithm A) presented by Fan [22] and the fixed structure algorithm presented by Cervellera et al. [12]. The algorithm by Fan [22] was sequential in nature, but used NN modeling for the future value function, whereas the algorithm by Cervellera et al. [12] used NN modeling for the future value function, but was not sequential in nature; involving one-step modeling and a fixed training dataset.

4.8 Results and Discussion

In the results presented in this section, the Algorithm-I corresponds to implementation of the first version of the MARS algorithm, as explained in section 4.6.2, which uses the original MARS algorithm presented by Friedman [26] with a default fixed value of *M*max set at twice the number of covariates. Algorithm-II corresponds to implementation of the second version of the MARS algorithm, as explained in section 4.6.2, which uses the original MARS algorithm presented by Friedman [26] with *M*max = $\left\lfloor \frac{2n+c}{2+c} \right\rfloor$, where n is the number of data points in the training data set, and c is the penalty parameter set to the default value of 3. Finally, Algorithm-III corresponds to implementation of the third version of the MARS algorithm, as explained in section 4.6.2, with *M*max adaptively set by the modified MARS algorithm with automatic stopping rule presented by Tsai and Chen [56].

In the results presented in this section, 'stage two' corresponds to the model being trained or built on the data for the future value function in stage 2 of the stochastic inventory forecasting problem outlined in section 4.7.1, and 'stage three' corresponds to the model being trained or built on the data for the future value function in stage 3 (last stage) of the stochastic inventory forecasting problem outlined in section 4.7.1. It may be noted that the future value

function data in stage two is more noisy than the future value function data in stage three because it also includes the modeling error in the stage three.

Keeping the above mentioned information in mind, the following sections present and discuss the results of this research.

4.8.1. Comparison of Consistency Trace

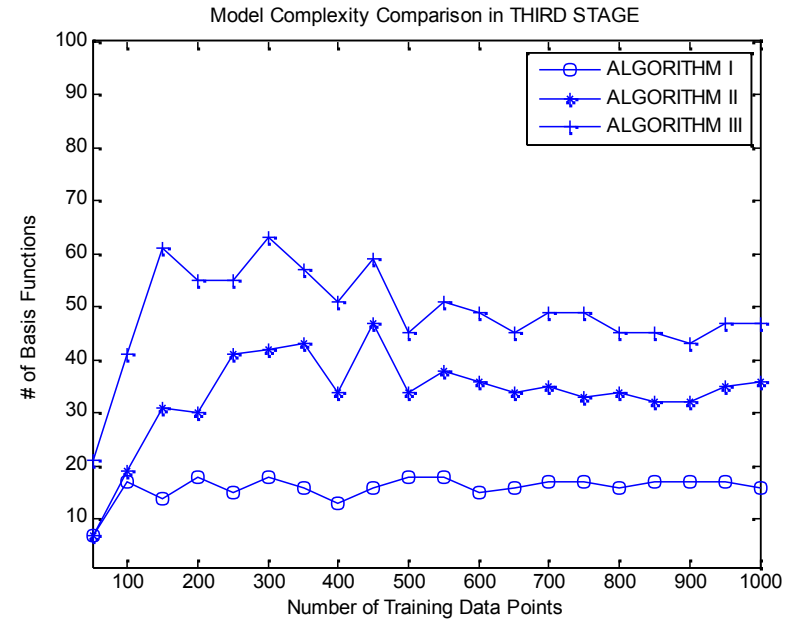
This research seeks to study the amenability of the MARS modeling method to follow the consistency trace when used in the proposed sequential framework (see section 4.6.1).

The results in Figure 4.5(b) and Figure 4.6(b) reveal that Algorithm-II and Algorithm-III follow the consistency trace nicely with their model generalization performance improving with the simultaneous increase in the size of the training data and the model complexity. Figure 4.5(a) and Figure 4.6(a) reveal that, for Algorithm-II and Algorithm-III, the optimal model complexity increases with the increase in the number of training data and then levels off, indicating that new training data points do not add to the generalization ability of the model. For Algorithm-I, the model complexity does not improve the generalization ability of the model in the test data because of the forced cap on the maximum number of basis functions in the model. The consequence of inadequacy of the forced cap in Algorithm-I is revealed in Figure 4.5(b) and Figure 4.6(b) in terms of inferior performance than Algorithm-II and Algorithm-III.

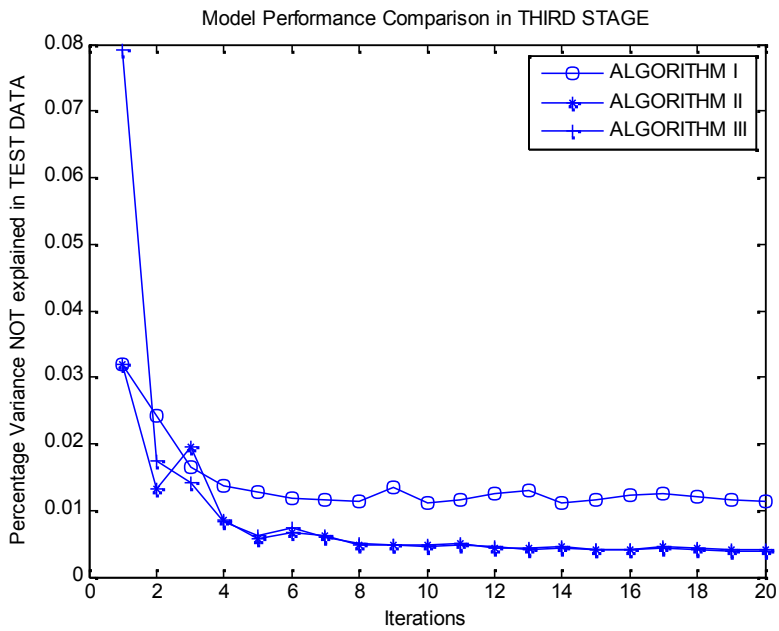
The results in Figure 4.5(b) and Figure 4.6(b) reveal that the model performance of Algorithm-II and Algorithm-III in terms of generalization ability in the test dataset is close and comparable; however, the results in Figure 4.5(a) and Figure 4.6(a) reveal that the optimal models selected, in each sequential step or for a size of the training dataset, by Algorithm-III are more complex than those selected by the Algorithm-II. This reduction in optimal model complexity comes from the back-pruning allowed in Algorithm-II. Even though this reduction in complexity does not come at the cost of the model performance, it does come at the cost of the computational expense associated with the back-pruning algorithm (backward stepwise deletion algorithm in Friedman [26]).

The results in Figures 4.7(a) and (b) reveal that the proposed sequential Algorithm-II follows the consistency trace in stage two as well stage three with respect to generalized error in terms of test MSE, test RMSE and test relative RMSE, following a decreasing trend with simultaneous increase in the model complexity and size of the training dataset and eventually leveling off when the increase in the size training dataset does not improve the generalization ability of the model. The higher generalized error curve in stage two compared to stage three can be attributed to the increased noise in the stage two training data due to the modeling error in stage three (last stage).

The results in Figures 4.8(a) and (b) reveal that the proposed sequential Algorithm-III also follows the consistency trace in stage two, as well stage three, as does Algorithm-II, with respect to generalized error in terms of test MSE, test RMSE and test relative RMSE, following a decreasing trend with simultaneous increase in the model complexity and size of the training dataset and eventually leveling off when the increase in the size training dataset does not improve the generalization ability of the model. The higher generalized error curve in stage two compared to stage three can be explained similarly to Algorithm-II.

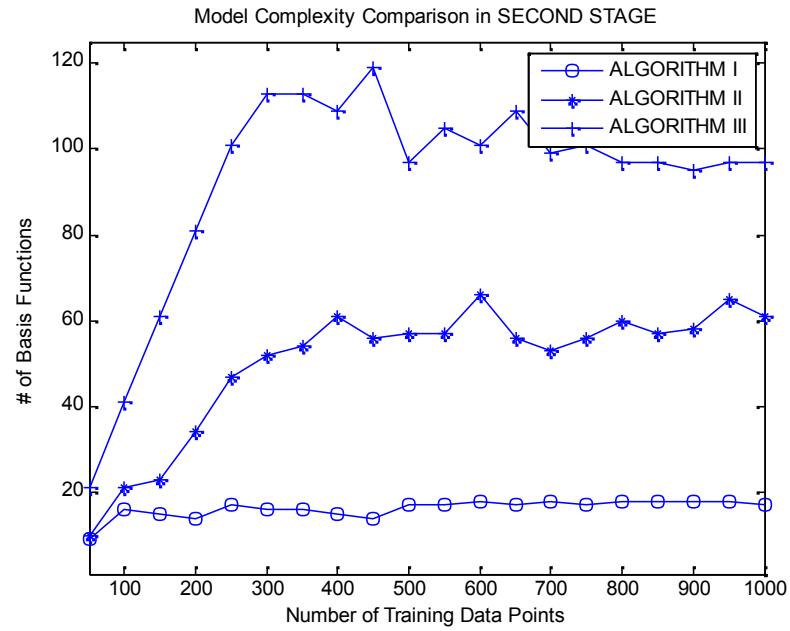


(a)

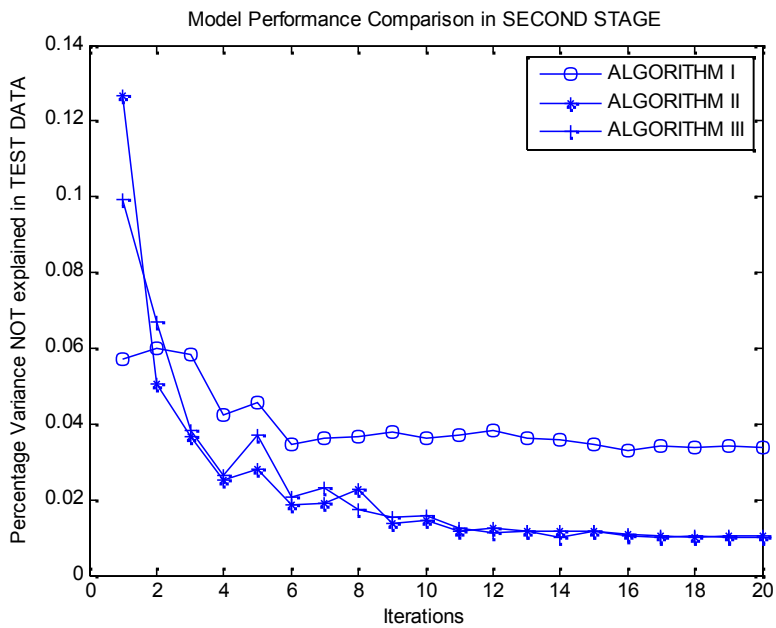


(b)

Figure 4.5: Comparison of (a) Model Complexity in Third Stage (b) Model Performance in Third Stage.

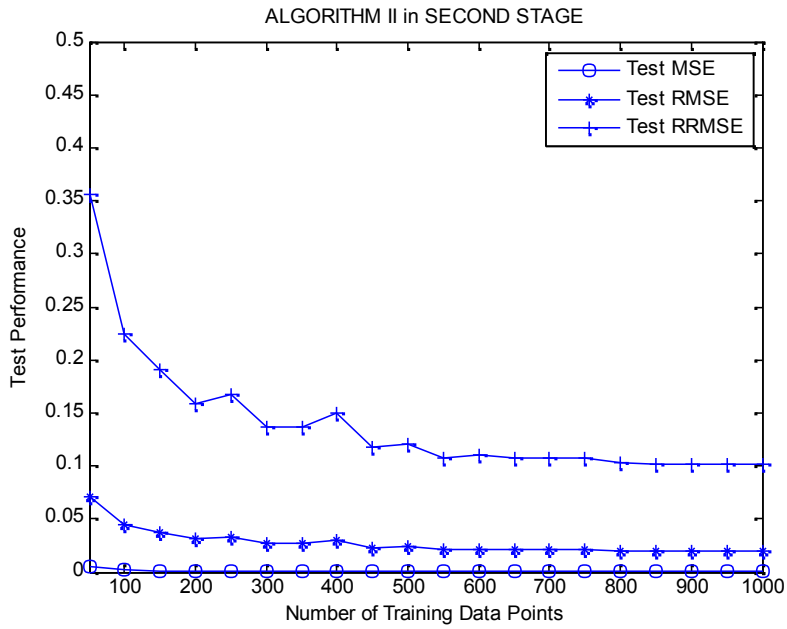


(a)

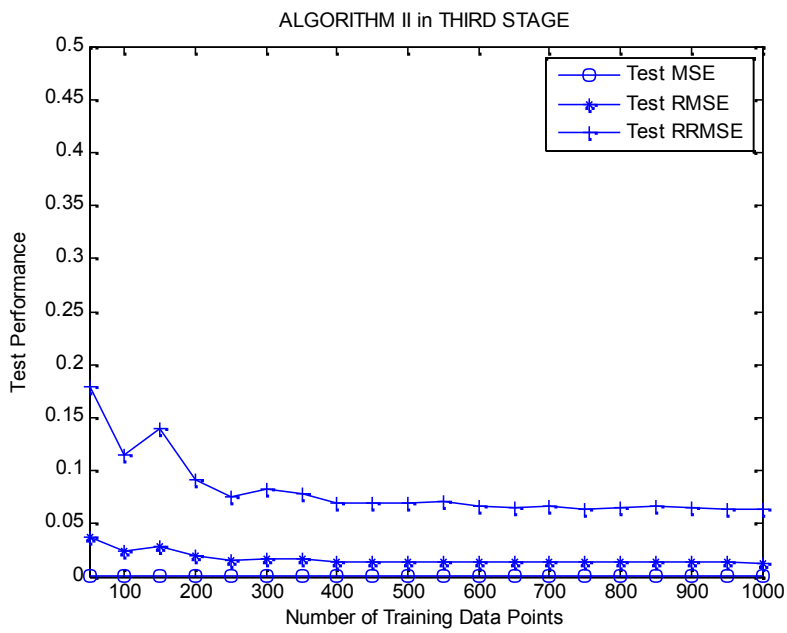


(b)

Figure 4.6: Comparison of (a) Model Complexity in Second Stage (b) Model Performance in Second Stage.

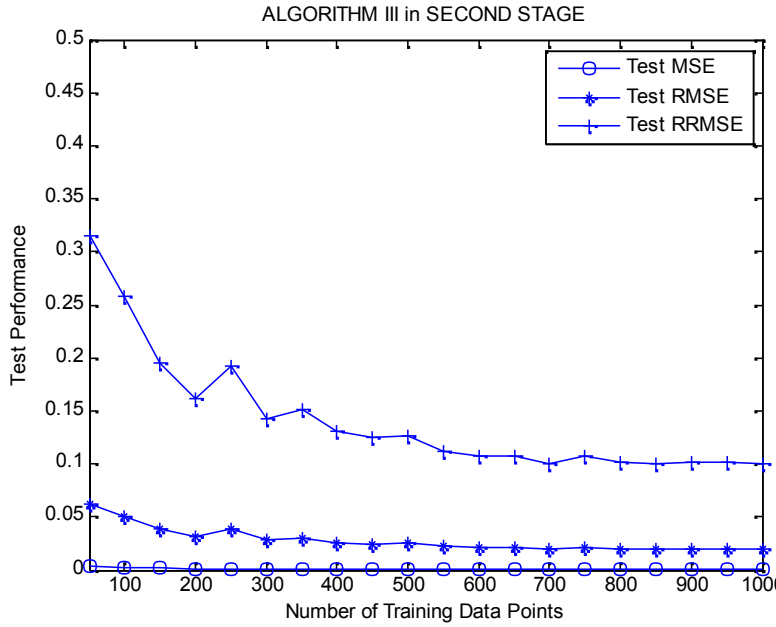


(a)

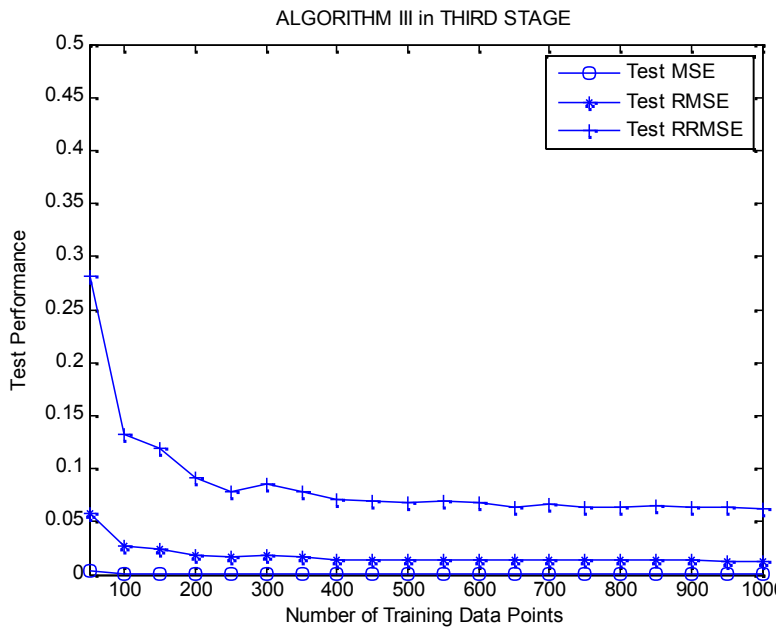


(b)

Figure 4.7: Comparison of Generalization Ability of the Algorithm-II in (a) Second Stage (b) Third Stage.



(a)



(b)

Figure 4.8: Comparison of Generalization Ability of the Algorithm-III in (a) Second Stage (b) Third Stage.

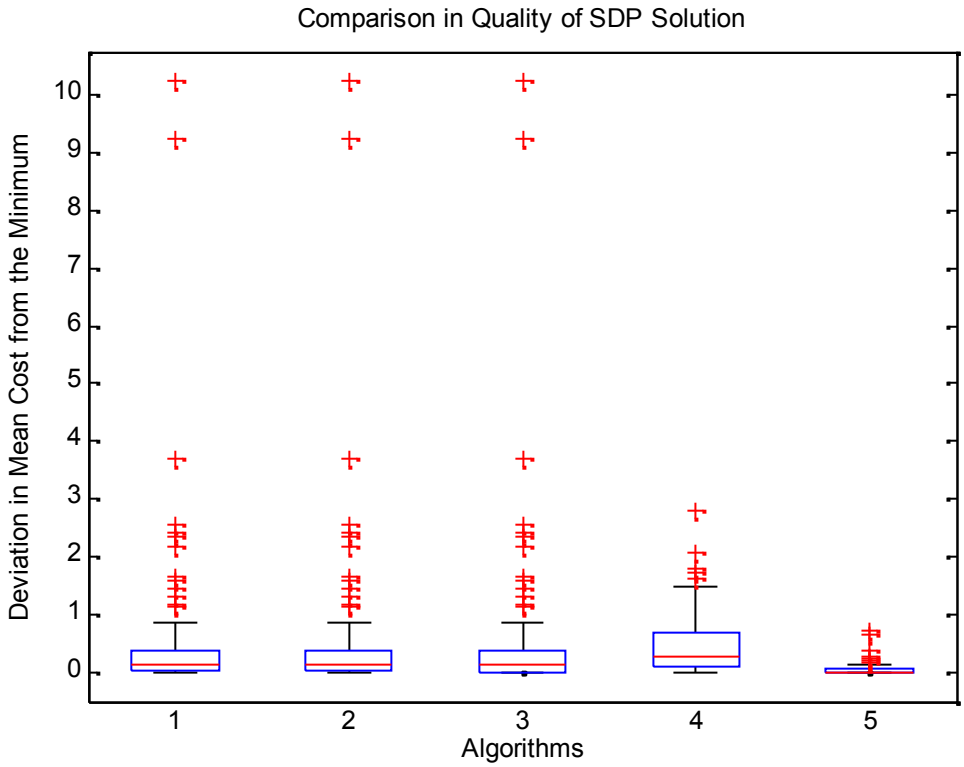
4.8.2. Comparison of SDP Solution Quality

The results in Figure 4.9 reveal that the quality of the SDP solution in terms of deviation in mean cost from the minimum is similar for Algorithm-II and Algorithm-III. This was expected because both algorithms result in models which have comparable generalization ability after saturation despite different model complexity.

The similar performance of Algorithm-I is not expected, but can be explained as a problem specific exception, where the models built by the Algorithm-I, and Algorithms-II and III, were different, but of similar shape; so that the minimization process in the SDP optimization chose the identical decision as the optimal decision that minimized cost, and the identical decision translated into identical cost for a given realization of the random error because the cost function is deterministic given an initial state point, decision, and realization of the random error.

The results in Figure 4.9 reveal that the proposed sequential algorithms using MARS modeling are superior than the sequential algorithm using NN models (comparison was made with the Algorithm A by Fan [22] with $\Delta H = 1$ and $n_{max} = 5$) in terms of time to train the algorithm.

The results in the Figure 4.9 also reveal that the quality of the SDP solution in terms of deviation in mean cost from the minimum was the best for the 'one-step' (in contrast with 'sequential') algorithm with fixed training dataset (which is the entire state space) using NN presented by Tsai and Chen [56]. However, this gain in performance comes at the expense of consequence of excessively exploring the state space.



1 - MARS Algorithm I

2 - MARS Algorithm II

3 - MARS Algorithm III

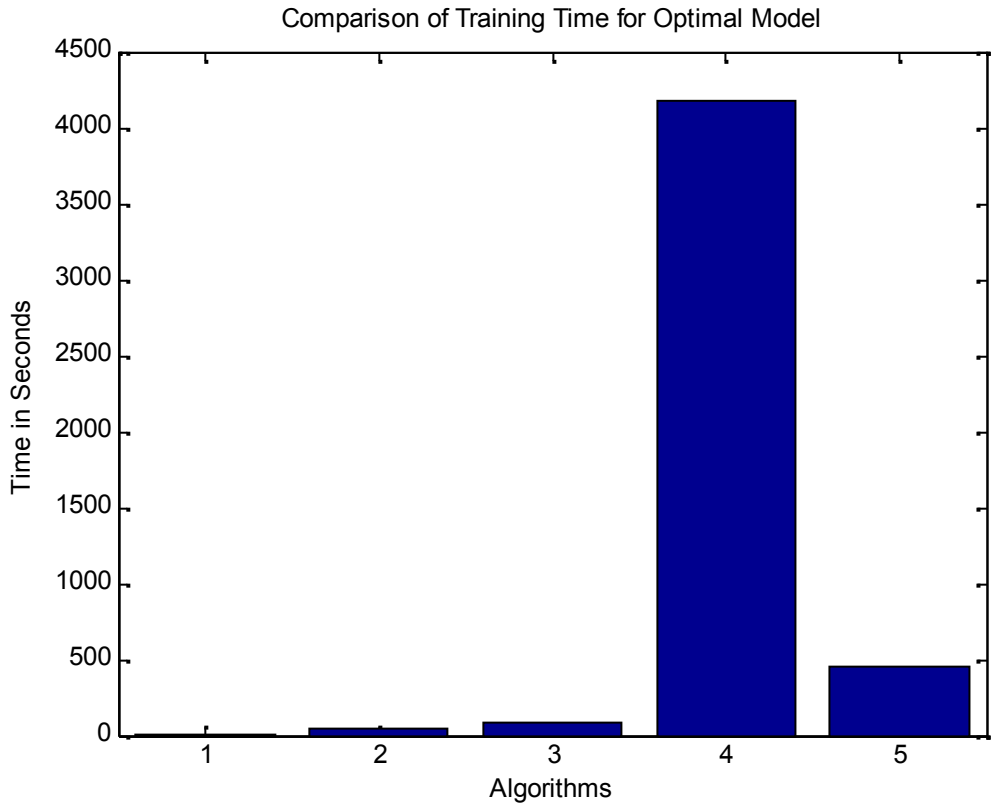
4 - NN Algorithm I by Huiyuan

5 - NN Fixed Structure Algorithm

Figure 4.9: Comparison of Quality of SDP solution:

In the Figure 4.10, the CPU times for training the optimal model have been presented for comparison. Algorithm - I takes 1.2 seconds, whereas Algorithm - II and III take 46.5 seconds and 87.5 seconds respectively. Algorithm - IV, which corresponds to the Algorithm A by Fan [22] with $\Delta H = 1$ and $n_{max} = 5$, takes 4176.9 seconds. Algorithm - V, which corresponds to the 'one-step' algorithm with a fixed training dataset (with 1331 observations) and using NN with 40 hidden nodes, presented by Tsai and Chen [56], takes 450.6 seconds for training the optimal model. The results in the Figure 4.10 reveal that this 'one-step' algorithm with fixed training dataset using NN performs better than the 'sequential' algorithm using NN, but worse than the 'sequential' algorithm using MARS in terms of the time to train the model. However, this performance assessment in terms of time to train may not be fair for this 'one-step' algorithm with fixed training dataset using NN. This is because the 'trial and error' process to identify the appropriate NN architecture and sample size is not included in the computational time. This 'trial and error' process can require days of investigation. By contrast, the sequential algorithms automatically identify structure and sample; hence, the entire computational effort is accurately recorded.

Hence, above discussion suggests that the prospect of the sequential algorithm using MARS modeling is promising for use in the framework of adaptive dynamic programming in terms of statistical parsimony it will bring in and the ensuing computational cost savings.



1 - MARS Algorithm I

2 - MARS Algorithm II

3 - MARS Algorithm III

4 - NN Algorithm I Huiyuan

5 - NN Algorithm Fixed Structure

Figure 4.10: Comparison of Training Time for the Algorithms.

CHAPTER 5

SUMMARY AND FUTURE WORK

The prospect of the proposed sequential algorithms in making a nonparametric statistical modeling method following the consistency trace for use in adaptive dynamic programming framework looks promising. The proposed sequential algorithm exploits the propensity of MARS modeling to follow the consistency trace whereas the MARS modeling algorithm provides an explicit complexity measure (the number of basis functions) that can be controlled directly during the model building stage.

This promising prospect of the proposed sequential algorithm to follow a consistency trace using MARS modeling has tremendous value in the context of adaptive dynamic programming. The first and foremost implication is that it provides a way for minimal exploration of the state space without affecting the quality of the SDP solution. In the context of ADP, approximating future value functions for any stage involves as many optimization sub-problems to solve as the number of state points explored. In the proposed sequential algorithms, we stop exploring the state space when the generalization ability of the approximation model levels off, and this results in possible reduction in the state space exploration requirement without affecting the quality of the approximation model, and this reduction in state space exploration directly translates into 'statistical parsimony' and 'computational cost saving'.

The next logical step of this research would be to extend the proposed MARS-based sequential algorithm to use TreeMARS modeling methodology presented in the Chapter 3. The use of TreeMARS will allow handling of a mix of categorical and continuous state variables. It will be interesting to see how the sequential algorithm will need to be modified to make the TreeMARS models to follow the consistency trace.

In MARS based Algorithm-II, there is a model complexity parameter. In this research, we have used the default and recommended value of 3 (Friedman [26]). A large penalty is likely to perform well when the size of the true model is small or the true model has a parsimonious representation, and is likely to perform poorly otherwise. In future, research can be done to study the role of the model complexity parameter in the sequential framework to fine tune the model.

In the MARS based Algorithm-III, a very small threshold on the automatic stopping rule was employed. It will be interesting to study further how to select this threshold. Regardless, both Algorithm-II and Algorithm-III have good potential for adaptive dynamic programming.

REFERENCES

1. William P. Alexander and Scott D. Grimshaw (1996), “Treed Regression”.*Journal of Computational and Graphical Statistics* Vol. 5, No. 2 (Jun., 1996), pp. 156-175
2. Bai, Z.D., Rao, C.R. and Wu, Y. (1999). Model selection with data-oriented penalty. *Journal of Statistical Planning and Inference*, 77: 103-117.
3. Barto, A., Sutton, R. and Brouwer, P. (1981), `Associative search network: A reinforcement learning associative memory', Biological cybernetics 40, 201-211.
4. Bellman, R. E and Kalaba, R. (1959), `On adaptive control processes', IRE Transactions on Automatic Control 4, 19.
5. Bellman, R. E. (1957). *Dynamic Programming*. Princeton: Princeton University
6. Berry, Michael J. A., and Linoff, Gordon,(1997), Data Mining Techniques, John Wiley and Sons
7. Bertsekas, D. (2005), `Dynamic programming and suboptimal control: A survey from ADP to MPC', European Journal of Control 11, 310{334.
8. Bertsekas, D. and Tsitsiklis, J. (1996), Neuro-dynamic programming, Athena Scientific, Belmont, MA.
9. Bertsekas, D. P. (2007). Dynamic Programming and Optimal Control, Vol. II, Athena Scientific, Belmont, MA.
10. Birge, J. and Louveaux, F. (1997), Introduction to Stochastic Programming, Springer-Verlag, New York.
11. Boutilier, C., Dean, T. and Hanks, S. (1999), `Decision-theoretic planning: Structural assumptions and computational leverage', Journal of Artificial Intelligence Research 11, 1-94.

12. Cervellera, C., A. Wen, and V. C. P. Chen (2007). Neural network and regression spline value function approximations for stochastic dynamic programming." *Computers and Operations Research*, 34, pp. 70-90.
13. Cervellera, C., D. Macciò, and M. Muselli (2010). "Functional Optimization Through Semilocal Approximization Minimization." Operations Research, 58(5), pp. 1491-1504
14. Cervellera, C. and D. Macciò (2011). "A Comparison of Global and Semi-local Approximization in T-stage stochastic optimization." European Journal of Operational Research, 208(2), pp. 109-118
15. Chen, V. C. P., D. Ruppert, and C. A. Shoemaker (1999). Applying Experimental Design and Regression Splines to High-Dimensional Continuous-State Stochastic Dynamic Programming." *Operations Research*, 47, pp. 38-53.
16. Chen, V. C. P., K. L. Tsui, R. R. Barton, and J. K. Allen (2003). A review of design and modeling in computer experiments." In *Handbook of Statistics: Statistics in Industry*, (R. Khattree and C. R. Rao, eds.), 22, Elsevier Science, Amsterdam, pp. 231-261.
17. Chen, V.C.P. (1999). Application of MARS and Orthogonal Arrays to Inventory Forecasting Stochastic Dynamic Programs. *Computational Statistics and Data Analysis* 30, 317–341.
18. Chi, H.., and O. K. Ersoy (2002). "Determining the optimum number of hidden nodes in constructive functional-link networks using validation and VC dimension." ECE Technical Reports, TR-ECE 02-06, Purdue University (Available in: http://docs.lib.purdue.edu/ecetr/166).
19. Corder, G.W. & Foreman, D.I (2009), "Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach", Wiley (2009)
20. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill
21. de Boor, C (1978), "A practical guide to splines", Springer-Verlag, New York

22. Fan, Huiyuan. (2008). Sequential Frameworks For Statistics-based Value Function Representation In Approximate Dynamic Programming, PhD Dissertation, University of Texas at Arlington
23. Foufoula-Georgiou, E. and P. K. Kitanidis (1988). Gradient Dynamic Programming for Stochastic Optimal Control of Multidimensional Water Resources Systems. *Water Resources Research*, 24, pp. 1345{1359.
24. Francis, Louise (2001), “Neural Networks Demystified”, Casualty Actuarial Society Forum, Winter 2001, pp 253 - 320.
25. Friedman, J. and Fisher, N. (1999), "Bump hunting in high dimensional data" Statistics and Computing: 123-143
26. Friedman, J. H (1991): “Multivariate Adaptive Regression Splines” (with discussion), *Annals of Statistics*, 19, pp. 1–67.
27. Geman,S., Bienenstock,E., and Doursat,R. (1992). Neural networks and the bias/variance dilemma. Neural Computation 4, 1–58.
28. Halton, J. (1964), "Algorithm 247: Radical-inverse quasi-random point sequence.", ACM, p. 701
29. Hastie, T., Tibshirani, R. and Friedman, J. (2002). *The Elements of Statistical Learning*, Springer, New York.
30. Henri Faure,(1992), "Good permutations for extreme discrepancy", Journal of Number Theory, Volume 42, 1992, pages 47-56.
31. Jekabsons G., ARESLab: Adaptive Regression Splines toolbox for Matlab/Octave, 2010, available at http://www.cs.rtu.lv/jekabsons/
32. Johnson, S. A., J. R. Stedinger, C. A. Shoemaker, Y. Li, and J. A. Tejada-Guibert (1993). Numerical Solution of Continuous-State Dynamic Programs Using Linear and Spline Interpolation." *Operations Research*, 41, pp. 484{500.

33. Jordan, M. and Jacobs, R. (1994), "Hierarchical mixtures of experts and the EM algorithm". *Neural Computation*: 181-214.
34. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone (1984): *Classification and Regression Trees*, Chapman & Hall/CRC publication.
35. Lin, Y., V. C. P. Chen, K. L. Tsui, F. Mistree, and J. K. Allen (2004). A sequential exploratory experimental design method: development of appropriate empirical models in design." ASME Design Engineering Technical Conference, Salt Lake City, Utah, Paper No. DETC2004-57527.
36. Mendel, J. M. and McLaren, R. W. (1970), Reinforcement learning control and pattern recognition systems, Vol. 66, Academic Press, New York, pp. 287-318.
37. Minsky, M. L. (1954), Theory of neural-analog reinforcement systems and its application to the brain-model problem, PhD thesis.
38. Minsky, M. L. (1961), `Steps Toward Artificial Intelligence', Proceedings of the Institute of Radio Engineers 49, 8-30.
39. Nemhauser, G. L. Introduction to Dynamic Programming. Wiley, New York, 1966.
40. Niederreiter, H. (1992). "Random Number Generation and Quasi-Monte CarloMethods.", SIAM, PA.
41. Paul Bratley; Bennett Fox; and Harald Niederreiter (1992). "Implementation and Tests of Low Discrepancy Sequences", ACM Transactions on Modeling and Computer Simulation, Volume 2, Number 3, July 1992, pages 195-213.
42. Powell, W.B. (2007). Approximate dynamic programming: Solving the curses of dimensionality, Wiley, New York.
43. Press, W. H.; Flannery, B. P.; Teukolsky, S. A.; and Vetterling, W. T.(1992) "Quasi- (that is, Sub-) Random Sequences." §7.7 in *Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed.* Cambridge, England: Cambridge University Press, pp. 299-306.
44. Puterman, M.L. (1994) Markov decision processes,Wiley, NewYork, 1994.

45. Rao, C.R. and Wu, Y. (1989). A strongly consistent procedure for model selection in a regreesion problem. *Biometrika*, 76: 369-374.
46. Rao, J.S. (1999). Bootstrap choice of cost complexity for better subset selection. *Statistica Sinica*, 9: 273-287
47. Rao, J.S. and Tibshirani, R. (1997). Discussion to "an asympototic theory for model selection" by Jun Shao. *Statistica Sinica*, 7: 249-252.
48. Shabbir Ahmed, Alan J. King, and Gyana Parija. (2001) A multi-stage stochastic integer programming approach for capacity expansion under uncertainty. Stochastic Programming E-Print Series, http://www.speps.org, 2001.
49. Sharon A. Johnson, Jery R. Stedinger, Christine A. Shoemaker, Ying Li, José Alberto Tejada-Guibert (1993): Numerical Solution of Continuous-State Dynamic Programs Using Linear and Spline Interpolation; *Operations Research* Vol. 41, No. 3, May-June 1993, pp. 484-500 DOI: 10.1287/opre.41.3.484
50. Shen, X. and Ye, J. (2002). Adaptive model selection. *Journal of the American Statistical Association*, 97: 210-221.
51. Si, J., Barto, A. G., Powell, W. B. and Wunsch, D. (2004), `Handbook of learning and approximate dynamic programming', Wiley-IEEE Press.
52. Sobol, I. M. (1967). "The distribution of points in a cube and the approximate evaluation of integrals." *USSR Computational Mathematics and Mathematical Physics*, 7, pp. 784-802.
53. Sutton, R. and Barto, A. (1998), Reinforcement Learning, Vol. 35, MIT Press, Cambridge, MA.
54. T. Hastie, R. Tibshirani , J. H. Friedman (2001): *The Elements of Statistical Learning;* Springer Publication.
55. Trandafir, Aurel and Weisstein, Eric W.(2009). "Quasirandom Sequence." From *MathWorld*--A Wolfram Web Resource.

56. Tsai, J. C. C., and V. C. P. Chen (2005). Flexible and robust implementations of multivariate adaptive regression splines within a wastewater treatment stochastic dynamic program. *Quality and Reliability Engineering International*, 21, pp. 689-699.
57. Victoria C. P. Chen, David Ruppert and Christine A. Shoemaker (1999): Applying Experimental Design and Regression Splines to High-Dimensional Continuous-State Stochastic Dynamic Programming; *Operations Research*, Vol. 47, No. 1 (Jan. - Feb., 1999), pp. 38-53.
58. W. P. Alexander, S. D. Grimshaw (1996): "Treed Regression," *Journal of Computational and Graphical Statistics*, 5, pp. 156–175.
59. Werbos, P. J. (1974), Beyond regression: new tools for prediction and analysis in the behavioral sciences, PhD thesis.
60. Werbos, P. J. (1989), `Back propagation and neurocontrol: A review and prospectus', Neural Networks pp. 209-216.
61. White, D. A. and Sofge, D. A. (1992), Handbook of intelligent control: Neural, fuzzy, and adaptive approaches, Van Nostrand Reinhold Company.
62. Whitt, Ward. (1978) Approximations of Dynamic Programs, Mathematics of Operations Research, Vol. 3, No. 3, August 1978
63. Yang, Z., V. C. P. Chen, M. E. Chang, M. L. Sattler, and A. Wen (2009): "A Decision-Making Framework for Ozone Pollution Control," *Operations Research*, to appear.
64. Yang, Z., V. C. P. Chen, M. E. Chang, T. E. Murphy, and J. C. C. Tsai (2007): "Mining and Modeling for a Metropolitan Atlanta Ozone Pollution Decision-Making Framework," *IIE Transactions*, Special Issue on Data Mining, 39, pp. 607–615.
65. Ye, J. (1998). On measuring and correcting the effects of data mining and model selection. *Journal of the American Statistical Association*, 93: 120-131.

66. Zhang, B. T. (1994). An incremental learning algorithm that optimizes network size and sample size in one trial. Procedings of Conference on Neural networks (ICNN-94), pp. 215-220.
67. Zhang, J., and A. J. Morris (1998). A sequential learning approach for single hidden layer neural networks. Neural Networks, 11(1), pp. 65{80.

BIOGRAPHICAL INFORMATION

Subrat Sahu was born in a small town of Cuttack in the state of Orissa in the eastern part of India. He received his B.Tech degree in Mechanical Engineering from the Orissa University of Agriculture and Technology, Bhubaneswar, India in 2002. He received his M.S degree in Industrial Engineering from the University of Texas at Arlington, USA. At the University of Texas at Arlington, he has been the recipient of the prestigious 'University Scholar' award for two consecutive years at the Annual President's Convocation for Academic Excellence for the year 2009 and year 2010. He is an active member of INFORMS, ASA, and IEE. He has served as the Vice-President of the National Engineering Honor Society, Tau Beta Pi-Texas Eta chapter. During his doctoral studies at UTA, he has worked as a Graduate Research Associate with Dr. Victoria Chen in the National Science Foundation (NSF) funded project on statistically parsimonious adaptive dynamic programming which culminated in his doctoral dissertation. His wide ranging research interest covers the broad areas of operations research, optimization, statistical modeling, and data mining. In future, he intends to engage himself in research activity in the areas of application of statistical modeling and data mining in marketing research and finance.