HIGH-DIMENSIONAL ADAPTIVE DYNAMIC PROGRAMMING WITH MIXED INTEGER LINEAR PROGRAMMING

by

ZIRUN ZHANG

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

May 2015

Acknowledgements

I would never have been able to accomplish my Ph.D. study and dissertation without the guidance of my committee members, and support from my family and friends.

I would like to express my deepest gratitude to my advisors, Dr. Victoria Chen and Dr. Jay Rosenberger, for their guidance, caring, patience, and providing me with very interesting projects for conducting research. I would like to thank Dr. Don Liles, who offered me the opportunity to finance my graduate study.

I would like to thank John Dickson, Nadia Martinez, Benjamin Wilson and Sandy Sung-Ning Wang for the help on my dissertation writing.

I would also like to thank my parents, the O'Neal family, John Dickson, and Sandy Sung-Ning Wang. They are always supporting me and encouraging me with their best wishes.

March 17, 2015

Abstract

HIGH-DIMENSIONAL ADAPTIVE DYNAMIC PROGRAMMING WITH MIXED INTEGER LINEAR PROGRAMMING

Zirun Zhang, PhD

The University of Texas at Arlington, 2015

Supervising Professors: Victoria C.P. Chen, Jay M. Rosenberger

Dynamic programming (DP, Bellman 1957) is a classic mathematical programming approach to solve multistage decision problems. The “Bellman equation” uses a recursive concept that includes both the current contribution and future contribution in the objective function of an optimization. The method has potential to represent dynamic decision-making systems, but an exact DP solution algorithm is limited to small problems with restrictions, such as problems with linear transitions and problems without uncertainty. Approximate dynamic programming (ADP) is a modern branch of DP that seeks to achieve numerical solutions via approximation. It is can be applied to real-world DP problems, but there are still challenges for high dimensions. This dissertation focuses on ADP value function approximation for a continuous-state space using the statistical perspective (Chen et al. 1999). Two directions of ADP methodology are developed: a sequential algorithm to explore the state space, and a sequential algorithm using mixed integer linear programming (MILP) and regression trees.

The first component addresses exploration of the state space. A sequential state space exploration (SSSE) algorithm (Fan 2008) was developed using neural networks. Here it is considered the use of multivariate adaptive regression splines (Friedman 1991) in place of neural networks. In ADP, the value function approximation is defined over a specified state space region. In the real world, the relevant state space region is

unknown. In particular, the ADP approach employed in this dissertation uses a statistical perspective that is analogous to design and analysis of computer experiments (DACE, Chen et al. 2006). In DACE, an experimental design is used to discretize the input space region, which is the state space region in ADP. Since the ADP state space region is unknown, SSSE uses the stochastic trajectories to sample future states and identify the potential range of system state. By reducing iterations without impacting solution quality, SSSE using MARS demonstrates improved efficiency over SSSE with neural networks.

The second component of this dissertation addresses the optimization of a real world, complex, dynamic system. This work is motivated by a case study on the environmental impact of aircraft deicing activities at the Dallas-Fort Worth (D/FW) International Airport. For this case study, the state transitions are nonlinear, the objective function is nonconvex, and the decision (action) space is high-dimensional and discrete. To overcome these complexities, an ADP method is introduced using a piecewise linear value function approximation and MILP. The piecewise linear structure can be transformed for MILP by introducing binary variables. The treed regression value function approximation is flexible enough to approximate the nonlinear structure of the data and structured to be easily formulated in MILP. The proposed ADP approach is compared with a reinforcement learning (Murphy 2005) approach.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

1.1 Research Background

In the area of mathematical optimization, dynamic programming (DP) is a problem solving approach that breaks down a large decision problem into a multistage decision process over time (Bellman 1957). The DP approach is applicable when sub-problems can be nested recursively inside a larger problem. Developed by Richard Bellman (Bellman 1957), the simplifying step in DP is to define a sequence of value functions that break down a large decision making problem into smaller sub-problems. In each sub-problem, an optimization problem is formulated to maximize/minimize the outcome from current and future.

As a significant tool in the mathematical programming field, DP has been widely applied in the areas of engineering, economics, social science and many other fields (e.g., Cervellera et al. 2006, Robichek et al. 1971, Delisi 1974, Denardo et al. 1979, Kennedy 1986). For example, in engineering, DP has been applied to solve network optimization problems such as water reservoir network optimization problems (Cervellera et al. 2006). In the area of computational finance, the DP approach has been applied to formulate and solve problems, such as bond refinancing decision making problem (Robichek et al. 1971). DP is also a very popular programming method for solving problems in the bioinformatics area, such as problems in bio-sequence analysis and gene recognition (Delisi 1974).

Approximate dynamic programming (ADP) is an approach that uses approximation to represent the unknown functions in DP. Important functions in DP are the value function, which represents the bet outcome in DP for any given state, the control function, which represents the optimal decision for any given state, and the Q

function, which represents the outcome given a state and a decision. Value function approximation is a common feature of ADP (Powell 2007, Johnson et al. 1993). There are many fields of related research devoted to the study of ADP. The artificial intelligence community uses reinforcement learning tools in ADP to approximate the Q function (Murphy 2005, Kaelbling et al. 1996). The control theory community often refers to ADP as neuro-dynamic programming (Bertsekas 2007). In the operations research area, ADP methodologies include all DP methods that use approximation in the sub-problem. Different approaches have been applied to obtain value function approximations: problem approximations use value function from a related but simpler problem (Powell 2007); parametric future value approximations use a suitable form of functions with certain parameters (Powell 2007); a rollout approach (Bertsekas and Castanon 1999) uses future values calculated either analytically or by simulation.

"The curse of dimensionality" in the DP approach means its computational requirements grow exponentially with the number of state variables (Powell 2005, Chen et al. 1999). One way to mitigate "the curse of dimensionality" is using the method of design and analysis of computer experiments (DACE) to conduct state space discretization and value function approximation (Chen et al. 1999). DACE was originally developed to replace time-consuming and expensive physical experiments (Dean and Voss 1999, Fan et al. 2013). The DACE approach for ADP uses design of experiments (DoE) to discretize the state space. An optimization tool is required to solve the DP sub-problem at each discretization point. Sequential quadratic algorithms and interior point methods have been applied to solve the sub-problem in DACE based ADP methods (Chen et al. 1999, Fan et al. 2013). However, optimal solutions found by those methods are not always guaranteed to be globally optimal (Boyd and Vandenberghe 2004). DACE based ADP also uses statistical learning tools to conduct the value function

approximation. Popular statistical learning tools in DACE based ADP include artificial neural network (NN), multivariate adaptive regression splines (MARS), and regression tree (Fan 2008, Fan et al. 2013, Sahu 2011).

In this dissertation, the deicing activities system at the Dallas-Fort Worth (DFW) International Airport Deicing problem is a perfect multistage decision problem and ADP methodology is applied. The objective of this DFW Airport case study is to minimize the current and future impact of deicing activities on dissolved oxygen in DFW Airport receiving waters by assigning aircraft to the DFW Airport deicing pad locations over hourly stages. It is well known that the applications of aircraft deicing and anti-icing activities (ADAF) are essential to assure safe aircraft operations in certain weather conditions because icing might directly cause aircraft crashes during takeoff (Valarezo et al. 1993). The worldwide way to perform aircraft deicing is to spray a mixture of hot water and aircraft deicing fluids that contain ethylene or propylene glycol. However, excess use of ADAF has the potential to cause serious adverse environmental issues because the glycol is a nutrient for bacteria in receiving water which leads to a depletion of dissolved oxygen (DO). In 1999, a significant amount of deicing fluid was discharged into Trigg Lake (a local reservoir for DFW Airport runoff) and caused a majority of the fish to die in the lake (Corsi et al. 2006). Given the necessary operations of ADAF, a balance between costly operations and environmental issues must be found (Zhang et al. 2013). The environmental system of DFW Airport evolves over time, and the water quality of receiving waters changes hourly.

The DFW Airport deicing problem has some characteristics that build up a uniquely complex problem. First, it is stochastic because the amount of glycol usage is determined by pilot, not the DFW Airport operations department (Zhang et al. 2013). Density functions of the random variables, such as glycol usage and whether anti-icing

activities are executed or not, are estimated from historical data (Zhang et al. 2013). Second, the ADP problem has a finite horizon since the problem is formulated in 24 hour window; this gives a possible 24-stage scope. Third, due to the flight schedule, changing meteorology, and effects of aerators in a key body of water, the state transitions of the Markov decision processes are non-stationary. The objective functions in the Bellman equation of the DFW Airport deicing ADP problem are non-convex functions where the shape of the future value function is difficult to model to modeling a convex function. Last and most important, the state space has very high dimensionality. There is a potential for over 7000 state variables before applying any variable selection and data mining techniques (Chen et al. 2011). Therefore, besides solving a high dimensional ADP problem, the DFW Airport deicing ADP study also extends the existing research to adaptive value function approximation, non-convex optimization and efficient exploration of state and random spaces.

1.2 Study Objectives

There are practical and methodological objectives for studying this DFW Airport safety operation. First, this study aims to solve the DFW Airport deicing ADP problem with dimensionality higher than existing work. The most recent work (Yang et al. 2007, 2009) handles a 524-dimensionwal state SDP problem in a decision framework to address ozone pollution reduction in Atlanta. The DFW Airport deicing problem has potentially 7000 state variables.

Second, an adaptive statistical modeling tool/algorithm is necessary to conduct the approximation given the characteristics of value functions and state transition functions. Given the non-stationary state transition functions, value function approximation method development is significant when the problem has several requirements. These requirements include being non-convex, avoiding over-fitting and

under-fitting, and being capable of approximating the optimal shape with the desired accuracy.

Third, this study aims to solve each sub-problem to the global optimum by employing piecewise linear value function approximation and mixed integer linear programming (MILP). The shape of value function approximation strongly depends on the future values that are provided by solving the sub-problem. A value function approximation with a more accurate shape would provide a better solution. It is well known that certain non-convex optimization methods such as a branch-and-bound algorithm (Land and Doig 1960) would solve the sub-problem (if formulated as MILP problem) to the global optimum. Other methods, such as heuristic optimization, have many intensive searching approaches that find a high quality solution without a guaranteed global optimum (Russell and Norvig 1995). In this study, an ADP method employing a treed regression model and a branch-and-bound algorithm are expected to significantly improve the solution quality.

Last, a sequential state space exploration (SSSE) algorithm using MARS is developed based on an existing SSSE framework. The state space region is usually assumed to be known, but is actually unknown in many real-world cases (Fan 2008). A sequential algorithm using MARS is developed and applied to an inventory forecasting problem to search for the appropriate state space region. Knowing the appropriate state space region will improve the future value function approximation.

The challenge in this ADP study draws on four directions in the statistics or data mining fields: (1) exploration of the high dimensional state space; (2) accuracy of statistical modeling in approximation; (3) correct shape and flexibility of value function approximations; and (4) lowering computational cost for solving the sub-problem. In next chapter (Chapter 2), the classic DP method and DACE based ADP methodologies are

reviewed. Existing algorithms which explore state and decision spaces are presented. A sequential state space exploration (SSSE) method is presented in Chapter 3. An ADP method employing treed regression and MILP is presented with a case study in Chapter 4. Finally, conclusions and future work are discussed in Chapter 5.

Chapter 2

Dynamic Programming and Approximate Dynamic Programming

Dynamic programming (DP) is one of the potential approaches to solve Markov decision process (MDP) problems (White and White 1989). In this chapter, the framework of MDPs is introduced and then the classic DP methodology is presented. The modern development of approximate dynamic programming (ADP) and several directions of ADP research are also discussed. Those methods include reinforcement learning based ADP and DACE based ADP. First, DACE based ADP is described before discussing statistical learning methods. Then statistical learning methods for value function approximation in ADP methodology are discussed. Last, two existing adaptive value function approximation algorithms which determine the appropriate state space sampling size and state space region are presented.

2.1 Markov Decision Processes and Dynamic Programming

Named after the mathematician Andrey Markov, a Markov decision process is a discrete time stochastic control process. It provides a mathematical modeling framework for complex decision making problems (Bellman 1957). Extending from a Markov chain, which describes a system that transitions from one state to another, MDPs have the "memoryless" Markov property. This property specifies the state of the next time stage is determined by the current state, the decision made at the current time stage, and the randomness that evolves in the state transitions. The decision maker would also receive a corresponding reward or pay a corresponding cost. The difference between MDPs and Markov chains is that MDP have decisions and rewards/costs, but Markov chains do not (Powell 2007).

2.1.1 Finite and Infinite Horizon MDPs

There are two types of MDPs: finite horizon and infinite horizon problems. In finite horizon problems, there is either a very specific horizon, such as a time *T*, or there is a specific goal to reach, such as the travelling salesman problem (Powell 2007, Malandraki and Dial 1996). In infinite horizon problems, it is assumed that the parameters of the contribution function, transition function, and uncertainty or stochastic processes remain the same over time (Berteskas 2005); i.e. the system is stationary. The objective function of an the infinite horizon problem can be rewritten as

$$V(s) = \max E\{\gamma^t C_t(S_t, X_t(S_t))\},$$

where s is the starting state, t represents the time stage, $V(s)$ is the maximum total rewards with starting state s, C_t is the contribution of the stage t, S_t is the state of the time stage, X_t represents the decision at stage t, and γ is the discount factor, which represents the difference in importance between future rewards and present rewards.

There are several algorithmic strategies for solving infinite horizon problems such as value iteration, policy iteration, and linear programming structured from value functions. This study focuses only on studying finite horizon DP problems from both a computational and practical perspective.

2.1.2 Modeling Framework

As discussed before, DP is an approach to break down a multi-stage optimization problem into a sequence of smaller problems. Each smaller sub-problem is indexed by stage number. There are several basic elements in modeling a DP problem such as stage, state, policy, transition function, and value function. Stages can have several meanings in various problems. An optimization problem for a one year term can be broken down into twelve month steps. The stage is then defined by the month index of

the year. In some problems, such as the travelling salesman problem, stages can be defined to be the locations.

The state variables in the DP problem are the information at the beginning of the stage (Powell 2007). The state variables are needed to make decisions for the current stage and describe how the system evolves over time. State variables in dynamic programs can be either discrete or continuous. When solving DP problems with continuous state variables, state discretization is employed because an analytical solution does not exist. Decisions are made based on a given state in the current stage and affect the state of the next stage. Decision variables can also be continuous or discrete. The policy of a dynamic program represents an ordered set of decisions by stage and state.

The transition function is another important element of a DP problem. It describes the evolving process of a state from stage to stage based on the decision made at a given state in current stage. The contribution function, determines the costs incurred or rewards received during the current stage. The contribution function, also called the stage return or reward function, describes the cost or reward generated in each stage. The future value function is the optimal return of system between a given stage and the final stage, given the states of the stage corresponding to the optimal sub-policy.

There are various types of DP based on the five basic elements such as discrete or continuous stages, finite or infinite time horizon, deterministic or stochastic transitions, discrete or continuous (or mixed) states, discrete or continuous (or mixed) decisions (Fan 2008). This dissertation focuses on modeling and solving finite horizon, DP problems with near-continuous states and mixed integer decision variables.

2.1.3 Finite Horizon, Continuous-state Space DP and The Optimality Equations

Given a finite horizon DP problem, the objective function can be written as

$$\max \mathrm{E}\left\{\sum_{t=0}^{T} C_t\left(S_t, X_t(S_t)\right)\right\},$$

where t represents the time stage, C_t is the contribution of the current time stage, S_t is the state of the time stage, X_t represents the decision, and T represents the time horizon. Solving dynamic programs with the objective function above is typically computationally intractable. The essential strategy is to solve the objective function stage by stage. By defining the value function $V_{t+1}(S_{t+1})$ in the state S_{t+1} at the stage $t+1$, and the transition function $S_{t+1}(S_t, x_t)$, the objective function is broken down into a set of optimization problems for each stage. At time stage t, each feasible decision x_t is evaluated and then chosen to maximize/minimize the contribution at time stage t plus the value at time stage $t+1$, as given by

$$V_t(S_t) = \max_{x_t \in X_t}\left(C_t(S_t, x_t) + V_{t+1}\big(S_{t+1}(S_t, x_t)\big)\right),$$

which is called the *optimality equation* for a deterministic problem in stage t.

The transition from S_t to S_{t+1} is defined by

$$S_{t+1} = g(S_t, x_t).$$

For a finite-state DP, the state variable, S_t, takes on a finite or countable-infinite number of values. The transition function is also called a *transition matrix* in finite-state DP methodology. The transition matrix represents the transition from one stage to the next. In this case, regardless of whether the decision is finite or continuous, the transition from S_t to S_{t+1} can only lead to points in the finite set of state space. This means $V_t(S_t)$ only needs to be solved at a finite number of points.

For continuous-state space DP, S_t is continuous and $g(S_t, x_t)$ is a continuous function. It is impossible to directly solve $V_t(S_t)$ for all values of S_t like finite-state space DP. However, a solution may be approximated. The general idea is limiting the

continuous-state space to a finite number of points within the limits of state space. The next step is to solve $V_t(S_t)$ at each of these points and then approximating the surfaces of $(S_t,\ V_t(S_t))$. Let $\hat{V}_{t+1}$ be the approximations for stage *t*+1 in a DP model. For each stage, the problem can be written as

$$\tilde{V}_{t+1}(S_t) = \max_{x_t \in X_t} \left(C_t(S_t, x_t) + \hat{V}_{t+1}\left(S_{t+1}(S_t, x_t)\right) \right)$$

for a finite set of state value S_t in continuous-state space.

When solving a stochastic problem, uncertainty would be involved. For finite-state stochastic DP, we define the probabilities of state at $t+1$ given state at t and decision x_t as $\mathrm{P}(S_{t+1}|S_t, x_t)$. Then the optimal equation for the stochastic problem at stage *t* would be

$$V_t(S_t) = \max_{x_t \in X_t} \left(C_t(S_t, x_t) + \sum_{s' \in S} \mathrm{P}(S_{t+1} = s'|S_t, x_t)\, V_{t+1}(s') \right),$$

or

$$V_t(S_t) = \max_{x_t \in X_t} \left(C_t(S_t, x_t) + E\left\{ V_{t+1}\left(S_{t+1}(S_t, x_t)\right) \middle| S_t \right\} \right),$$

by adding the expectation instead of summing over probabilities. This is defined as the expectation form of the Bellman's equation (Powell 2007), which is

$$V_t(S_t) = \max_{x_t \in X_t} (C_t(S_t, x_t) + E\{V_{t+1}(S_{t+1})|S_t\}),$$

where the transition from current stage to next time stage is implicit. When the contribution function is not deterministic, it is determined by state, action, and the information/uncertainty that arrives/involves in the current stage. In that case, the contribution function is written as $\mathrm{C_t}(\mathrm{S_t}, \mathrm{x_t}, \varepsilon_\mathrm{t})$. The expectation would be derived from optimality equation, given by

$$V_t(S_t) = \max_{x_t \in X_t} E\{C_t(S_t, x_t, \varepsilon_t) + V_{t+1}(S_{t+1})|S_t\},$$

where ε_t is the uncertainty/information involving/arriving at the current time stage.

For a continuous-state, finite horizon stochastic DP model, again, S_t is continuous and the transition function $g(S_t, x_t, \epsilon_t)$ is a continuous function with the random variable ϵ_t. The solution procedure for a continuous-state, finite horizon stochastic DP model is similar to continuous-state, finite horizon deterministic DP, except that an expectation must be calculated. The problem can be written as

$$\tilde{V}_{t+1}(S_t) = \max_{x_t \in X_t} E\left(C_t(S_t, x_t, \varepsilon_t) + \hat{V}_{t+1}(S_{t+1})\right),$$

$$S_{t+1} = g(S_t, x_t, \epsilon_t),$$

where ϵ_t represents the random variable. The expectation with respect to the random variable ϵ_t can be computed for given state S_t and decision x_t. If ϵ_t is a discrete random variable, the expectation is a summation. If ϵ_t is continuous, an additional approximation/estimation would be required to estimate the expectation by sampling the probably distribution of ϵ_t.

2.2 Approximate Dynamic Programming

2.2.1 Basic Idea of ADP

Approximate dynamic programming collectively describes a set of methods to solve multistage decision problems to near optimality by approximating the unknown functions, such as the value function, which represents the best outcome in DP for any given state, the control function, which represents the optimal decision for any given state, and the Q function, which represents the outcome given a state and a decision. There are various approaches of ADP, since many fields of research study solving multi-stage optimization problems by approximation. Reinforcement learning (RL) based ADP describes methods that "learn how to make good decisions by observing their own behavior, and use built-in mechanisms for improving their actions through a reinforcement mechanism" (Sutton and Barto 1998, Lewis and Vrabie 2009). Researchers in the field of control theory study ADP problems under neuro-dynamic

programming (Bertsekas 2007, 2005). The term ADP is frequently used in the operations research field to refer to the value function approximation approach to approximate the optimal value functions (Hua et al. 2006). However, all these problem solving approaches share some common themes under different notation and terminology.

The use of value functions is central to DP in multi-stage decision making. The basic idea is to approximate value functions at each stage and then use value function approximation in the maximization/minimization step of DP method. When solving a continuous-state DP problem by stepping in time, it is necessary to find the value function

$$\tilde{V}_{t+1}(S_t) = \max_{x_t \in X_t} E\left(C_t(S_t, x_t, \varepsilon_t) + \hat{V}_{t+1}(S_{t+1})\right),$$

$$S_{t+1} = g(S_t, x_t, \epsilon_t),$$

for a finite set of state value S_t in continuous-state space. As we discussed in Section 2.1.3, in these problems, classic *backward approach* cannot be used to solve the maximization step because it requires looping over every possible state and every possible decision for stages. As a result, there is replacement of $V_{t+1}(S_{t+1})$ with its approximation $\hat{V}_{t+1}(S_{t+1})$. With value function approximations for all stages, we step forward from the first stage to generate the optimal decision and total returns. With decisions made through time by value functions approximation and simulation, the outcomes of such decisions can be returned and can be used to evaluate the policy quality. From the perspective of policy evaluation, the learning method for value function approximation is important in ADP given that a variety of policies are generated in the backward ADP solution process.

2.2.2 ADP Methods for Continuous-state Space DP

As we discussed in Section 2.1.3, it is impossible to use classic *backward approach* to solve a DP problem when the state variables include any continuous variables. Continuous-state DP problems approximate solutions by two important steps:

the state space discretization and the value functions approximation (Tejada-Guibert et al. 1995, Chen et al. 1999, Cervellera et al. 2006, 2007). The most basic technique to is to form a finite grid of discretization points in the state space, then approximate the value function by some interpolation or statistics methods such as tensor-product cubic spline interpolation (Johnson et al. 1993).

Grid discretization also faces “the curse of dimensionality”. The size of the state space grows exponentially with the finer grid and higher dimensionality of the state space (Cervellera et al. 2006). To mitigate “the curse of dimensionality”, Chen et al. (1999) introduced a statistical modeling approach using Orthogonal Array (OA) experimental designs to discretize the state space and Multivariate Adaptive Regression Splines to approximate the value functions. Design and analysis of computer experiments (DACE) based ADP is a major direction of research to mitigate “the curse of dimensionality”. Further discussion of DACE based ADP will be presented in Section 2.4.

2.2.3 Uncertainty and Approximating Expectation

In stochastic dynamic programming (SDP) problems, random variables could be involved in both contribution functions and transition functions (Cervellera et al. 2006). The general way is to approximate the expectation in the value function calculation by sampling random variables and assigning each outcome a probability. Therefore, using an ADP algorithm to solve multi-stage decision problems requires a sequence of sample realizations of the random variables. There are three ways to obtain random samples such as real world data, computer simulation, and sampling from a known distribution. The first method requires a large amount of real-world data. The second method requires extensive calculations. The third method is the simplest way to sample random variables. There are many tools to generate random observations at high speed (Fan 2008).

2.3 Reinforcement Learning Based ADP

Reinforcement learning (RL) is an area of machine learning. The environment of RL is usually formulated as a Markov decision process. It has been applied successfully to various problems, such as robot control, power system control, and telecommunications (Kalmár et al. 1998, Sutton and Barto 1998, Ernst et al. 2004, Geurts et al. 2006). Reinforcement learning techniques are also highly valued in the ADP area. The basic reinforcement learning model consists of a state space, an action space, a transition probability matrix for a finite state space, and immediate reward and value functions. Common learning algorithms for ADP include one-step temporal-difference learning (Tesauro 1995, Sutton and Barto 1998), Q-learning (Sutton and Barto 1998, Murphy 2005), case based myopic reinforcement learning (CMRL, Kwol et al. 2008) and state action reward state action (Sarsa, Sutton and Barto 1998, Rummery and Niranjan 1994). Note that the restriction to finite state and action spaces limits application to complex problems.

2.3.1 Temporal-difference Learning

Temporal-difference learning (Tsitsiklis and Van Roy 1997, Tesauro 1995, Sutton and Barto 1998) is a method for approximating long-term future cost as a function of the current state. Its algorithm is recursive, efficient, and simple to implement. The long-term future cost, as a mapping from the current state, is represented by an approximate function. The parameters of value functions are updated by each observation of state transition and cost value. Therefore, the approximations of long-term cost would be improved as the number of observed state transitions increases. The algorithm of temporal-difference learning looks simple and elegant; however, a comprehensive analysis of its convergence requires significant sophistication (Sutton and Barto 1998).

Assuming an irreducible aperiodic Markov chain with a finite or countable state space, the action space of temporal-difference learning is considered as a countable subset of Euclidean space. The dynamics of the Markov chain are described by a transition probability matrix *P* and a discount factor *r*. The cost-to-go function associated with the Markov chain is defined by

$$J^*(i) \triangleq E\left[\sum_{t=0}^{\infty} \alpha^t g(i_t, i_{t+1}) | i_0 = i\right].$$

J^* is approximated with a function $\hat{J}$. Given a sequence of states generated by the state transitions, the temporal-difference from state i_t to i_{t+1} is defined by

$$d_t = g(i_t, i_{t+1}) + \alpha \hat{J}(i_{t+1}, r_t) - \hat{J}(i_t, r_t).$$

The parameter vector is initialized to some arbitrary vector and it is updated according to

$$r_{t+1} = r_t + \gamma_t d_t \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \nabla \hat{J}(i_t, r_t),$$

where λ is a parameter between 0 and 1 and the gradient is the vector of partial derivatives with respect to the element of parameter vector. Linear function approximations are a special case of cost-to-go function approximations. In the case of linear function approximations, the TD function is obtained by a sequence of eligibility vectors defined by

$$z_t = \sum_{k=0}^{t} (\alpha\lambda)^{t-k} \nabla \hat{J}(i_t, r_t).$$

The TD updates and eligibility vectors are updated by

$$r_{t+1} = r_t + \gamma_t d_t z_t,$$
$$z_{t+1} = \alpha\lambda z_t + \hat{J}(i_{t+1}, r_t).$$

2.3.2 Q-learning

The Q function of an finite-state space DP is defined as

$$Q(s,a) = R(s) + \sum_{s'} P(s'|s,a)V(s'),$$

where the $R(s)$ represents the immediate reward, $P(s'|s,a)$is the finite-state probability transition given state s and action a, and $V(s')$ represents the cost to go at state s'. By replacing the value function in the way of the Q function, the value iteration of the DP is shown by the Bellman equation in terms of the Q function as follows:

$$\forall s,a: \quad Q(s,a) \leftarrow R(s) + \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a').$$

The ϵ greedy policy can be used to ensure every state is visited infinitely:

$$with\ probability\ \epsilon: \ choose\ an\ action\ at\ random,$$

$$with\ probabilty\ 1-\epsilon: a = \operatorname*{argmax}_a Q(s,a).$$

The Q-learning value iterations algorithm can be described as follows:

Initialize $Q(s,a)$ arbitrary ($\forall s,a$).

For *t=0,1,2,…*

Choose the action a_t for the current states_t. (by ϵ greedy policy)

Take the action, and observe $R(s_t)$, s_{t+1}.

Update the Q function by
$Q(s_t,a_t) \leftarrow (1-\alpha_k)Q(s_t,a_t) + \alpha_k\left[R(s) + \max_a Q(s_{t+1},a_t)\right]$.

A related learning algorithm for MDP, State action reward state action (Sarsa, Rummery and Niranjan 1994, Sutton and Barto 1998). Sarsa is almost identical to Q-learning with the only difference being that the action taken at *t*+1 stage; it is not necessarily to find $\underset{a}{argmax}\, Q(s,a)$. It could be found by the ϵ greedy policy. The updating procedure follows in terms of the Q function:

$$Q(s_t,a_t) \leftarrow (1-\alpha_k)Q(s_t,a_t) + \alpha_k[R(s) + Q(s_{t+1},a_{t+1})].$$

For finite horizon, continuous-state space DP problems, the Q-learning usually utilizes random samples for both state space and action space (Murphy 2005). This method is called fitted Q iterations. In this method, outcome/cost is calculated for each combination of sampled state and action, and the Q function is then fitted as function of state variables and action variables. Let *t* be the time stage. Let $\mathrm{x_t}$ and $\mathrm{u_t}$ represent the state and decision at stage *t* respectively. Let $\mathrm{C_t}$ represent the contribution function and $\mathrm{g_t}$ is the transition function. Let ε_t represent the random variable. Let $\widehat{\mathrm{Q}}_{\mathrm{t+1}}$ represent the Q function at stage *t*+1. The problem can be written as

$$\widetilde{\mathrm{Q}}_{\mathrm{t}}(\mathrm{x_t}, \mathrm{u_t}) = \underset{\varepsilon_t}{\mathrm{E}}\left\{\mathrm{C_t}(\mathrm{x_t}, \mathrm{u_t}, \varepsilon_t) + \min_{\mathrm{u_{t+1}}} \widehat{\mathrm{Q}}_{\mathrm{t+1}}(\mathrm{x_{t+1}})\right\},$$

$$\mathrm{x_{t+1}} = \mathrm{g_t}(\mathrm{x_t}, \mathrm{u_t}, \varepsilon_t),$$

for a sampled set of state value $\mathrm{x_t}$ in continuous-state space.

The algorithm to approximate Q functions for continuous-state space DPs can be described as below:

Backward ADP solution process to approximate Q functions:

1. Randomly sample N points in the state space for the stage t, $t = 0,1, \dots, T$. x_{jt} is the state space sample point.
2. On stage T,

   2a for each point x_{jT}, $j = 1, \dots, N$, sample decision $\boldsymbol{u_{jT}}$, ε_j is random variable, $\mathrm{C_T}$ is the contribution function, E represents the expectation function, solve

   $$Q_{\mathrm{T}}(\mathrm{x_{jT}}, \mathrm{u_{jT}}) = \underset{\varepsilon}{\mathrm{E}}\{\mathrm{C_T}(\mathrm{x_{jT}}, \mathrm{u_{jT}}, \varepsilon)\}.$$

   2b approximate $Q_{\mathrm{T}}(\mathrm{x_{jT}}, \mathrm{u_{jT}})$ with $\hat{Q}_T(x_{jT}, u_{jT})$.
3. For $t = T-1, \dots, 0$,

   3a for each point x_{jt}, $j = 1, \dots, N$, ε_j is random variable, sample a decision u_{jt}, $\mathrm{C_t}$ is the contribution function, g_t is the state transition function, solve

   $$\mathrm{Q_t}(\mathrm{x_{jt}}, \mathrm{u_{jt}}) = \underset{\varepsilon}{\mathrm{E}}\left\{\mathrm{C_t}(\mathrm{x_{jt}}, \mathrm{u_{jt}}, \varepsilon) + \min_{\mathrm{u_{j,t+1}}} \widehat{\mathrm{Q}}_{\mathrm{t+1}}\left(\mathrm{g_t}(\mathrm{x_{jt}}, \mathrm{u_{jt}}, \varepsilon)\right)\right\}.$$

   3b approximate $\tilde{Q}_t(x_{jt}, u_{jt})$ with $\hat{Q}_t(x_{jt}, u_{jt})$.

2.3.3 Other RL Algorithms

There are several others RL algorithms including Actor-Critic methods (Nakamura et al. 2007, Konda et al. 1999, Sutton and Barto 1998) and the R-learning algorithm (Watkins et al. 1992). Actor-critic methods are based on temporal-difference learning and have two separate memories for both the policy (actor) and the value functions (critic). Actor-Critic methods are always on-policy because the critique takes the form of a TD error, which is about the policy currently being taken by the actor. The advantages of Actor-Critic methods are 1) those methods that requires less computational effort to select actions; 2) those methods that can learn the optimal probabilities of selecting various actions.

The other common RL method is R-learning (Watkins et al. 1992, Sutton and Barto 1998). In the R-learning algorithm, the objective is to obtain the maximum reward per time step. The value functions for a policy are defined by the average expected reward per time step

$$\rho^{\pi} = \lim_{n\to\infty} \frac{1}{n} \sum_{t=1}^{n} E_{\pi}\{r_t\}.$$

Most RL algorithms have similar assumptions, such as infinite horizon, the state space is countable or finite, finite action space, and a stationary Markov decision process. Some researchers have conducted learning algorithms for dynamic programs with non-stationary processes and finite horizon based on Q-learning and R-learning algorithms (Garcia and Ndiaye 1998). A value function approximation approach (Li and Littman 2005) is also developed for dynamic programs with non-stationary processes and finite horizon Markov decision process.

2.4 DACE Based DP

2.4.1 An Algorithm of Solving Continuous DP problem in Statistical Perspective

Numerically, solving a continuous-state DP problem requires discretizing the state space. A general algorithm for solving finite DP problems with continuous or near continuous states (Johnson et al. 1993, Cervellera et al. 2006, 2007, Chen et al. 1999, Yang et al. 2009) in the perspective of statistics is stated as follow:

Step 1. Using design of experiments, obtain *N* discretization points in the state space $\{x_{jt}\}_{j=1}^{N}$ for the *t*th stage, $t = 1, \ldots, T$, and $x_{jt} \epsilon R^n$.

Step 2. In the last stage *T*,

2a optimization: for each discretization points x_{jT}, j = 1,…, *N*, ε_j is random variable, solve

$$\tilde{V}_T(x_{jT}) = \min_{u_{jT}} E\{C_T(x_{jT}, u_{jT,}\varepsilon_j)\}.$$

2b approximation: approximate $V_T(x_{jT})$ with $\hat{V}_T(x_{jT})$, for all $x_T \in R^n$, by fitting a statistical model to the data for V_T from step 2a.

Step 3. For stage $t = T\text{-}1, .., 1$,

3a for each discretization point x_{jt}, $j = 1,\ldots, N$, ε_j is random variable, solve

$$\tilde{V}_t(x_{jt}) = \min_{u_{jt}} E\left\{C_t(x_{jt}, u_{jt,}\varepsilon_j) + \hat{V}_{t+1}\left(g_t(x_{jt}, u_{jt}, \varepsilon_j)\right)\right\}.$$

3b approximation: approximate $\tilde{V}_t(x_{jt})$ with $\hat{V}_t(x_{jt})$, for all $x_t \in R^n$, as in step 2b.

The reason for DP not being used to solve real-world problem is called “the curse of dimensionality” (Powell 2007, Bertsekas 2007, Chen et al. 1999). The size of the state space increases exponentially with the dimension of states. Hence the computational effort needed to solve a problem is also increasing exponentially with the dimension of the states. Since the first uniform grid discretization method, (Bellman 1957), studies have focused on developing methods to reduce computational efforts. For example, a gradient DP algorithm (Georgiou and Kitanidis 1988) employs cubic Hermite polynomials to approximate the future value function. Johnson et al. (1993) compared several numerical methods using multi-linear, Hermite gradient DP, and tensor-product cubic

spline interpolation by solving a four-reservoir problem, and the results showed that cubic splines method required less grid levels in each dimension, reducing the computational time. However, in the perspective of state space discretization, the above methods are all based on a full grid points, referred to as full factorial design in statistical design of experiments (Dean and Voss 1999). The number of points is still growing exponentially with the state space dimensionality.

The first truly practical numerical solution approach for a high-dimensional ADP problem was presented by Chen et al. (1999) by utilizing experimental designs (orthogonal arrays, OAs) and MARS. In their method, OAs are applied to discretize the state space as a special subset of full factorial experimental designs. Since OAs grow only polynomial with the dimensions. This was a breakthrough in mitigating the "curse of dimensionality" for the state space. An alternative experimental design, was employed with artificial neural networks (NN) approximations and compared (Cerevellera et al. 2007) with OAs and MARS on a nine-dimensional inventory forecasting problem and eight-dimensional water reservoir problem. Number-theoretic methods (NTMs) were also implemented (Cervellera et al. 2006) to successfully solve a thirty-dimensional water reservoir problem.

2.4.2 Sequential Design and Analysis of Computer Experiments

Design and analysis of computer experiments (DACE) have been applied in DP problems to conduct state discretization (Chen et al. 2006). DACE was originally developed to replace time-consuming expensive physical experiments. It uses design of experiments (DoE) to explore the sample space and statistical learning methods to conduct meta-modeling of the output from a computer model (Chen et al. 2006). The computer model is usually a simulation model.

Since the traditional full grid discretization, which is equivalent with the full factorial design, can be too large in practice for problems with high dimensions, the state space discretization based on efficient design of experiments requires fewer design points. In general, a “space-filling” sampling method is appropriate for DACE based ADP. Example “space-filling” designs include OAs and number-theoretic methods (NTMs) (Chen et al. 2006).

As the most suitable methods for sequential space-filling sampling, NTMs that are low-discrepancy sequence are ideal (Niederreiter 1992). The resulting sequence fills the feature space more uniformly than random points generated by computer. There are many types of quasi-random or low-discrepancy sequences, such as the Faure sequence (Faure 1982), Hammersley sequence (Hammersley 1960), Niederreiter sequence (Niederreiter 1992) and Sobol’ sequence (Sobol 1967). These sequences are useful for approximating functions in high-dimensional feature spaces and numerical integration algorithms based on such sequences are claimed to have superior convergence because low discrepancy sequences tend to sample the feature space "more uniformly" than computer-generated random numbers (Niederreiter 1992).

2.4.3 Statistical Modeling Methods

Besides experimental design selection, meta-modeling is an important task in DACE. Many statistical modeling methods can be applied for meta-modeling in conducting computer experiments. The meta-model can be used to optimize a complex system via a simulation model. Meta-modeling is used in many fields, such as electrical engineering, chemical engineering, mechanical engineering, and operations research (Simpson et al. 1997, Chen et al. 2006, Li and Littman 2005). Five general approaches are discussed by Chen et al. (2006): namely, polynomial models, spatial correlation models, MARS, regression trees and boosting, artificial neural networks. There have

been mixed results from empirical studies; therefore, the choice of method depends on the particular application (Chen et al. 2006). The general task for every statistical modeling method in DACE is to estimate the relationship between a response (dependent) variable and several predictor (independent or explanatory) variables over the experimental region. The mathematical relationship between the response and predictors can be written as

$$E[Y] = \eta = g(x),$$

where Y is a random variable representing the response, η is the mean response, $g(\cdot)$ is an unknown function, and *x* is the vector of input variables. Given the model structure of the approximation function, the unknown $g(x)$ is approximated by a function $g(x;\theta)$ and the primary task then becomes one of estimating the coefficients.

Box and Wilson (1951) developed response surface methodology by using first and second order polynomials to approximate the true function $g(x)$. The purpose of their study was to find the value of predictors that optimize the mean response over the experimental region. In the response surface methodology, they used numerical results of a small experimental sub-region and steepest ascent method to locate an experimental sub-region closer to the optimum. First order models are employed initially and are replaced by second order models when the new experimental sub-region is close to the optimum. The general form of a response surface model is given by

$$\hat{g}(x;\beta) = \beta_0 + \sum_j \beta_j x_j + \sum_j \sum_{k>j} \beta_{jk} x_j x_k + \sum_j \beta_{jj} x_j^2$$

$$+ \sum_j \sum_{k>j} \sum_{l>k} \beta_{jkl} x_j x_k x_l + \cdots + \sum_j \beta_{jj,\ldots,j} x_j^d.$$

The drawback of response surface methodology is that the model structure might not be flexible enough to represent the true response surface (Box and Draper 1987, Myers et al. 2009). Another type of statistical modeling method common in meta-

modeling is the spatial correlation model, also known as “kriging”, that assumes some form of spatial correlation between points from the multiple dimensional input space to predict the response values based on this correlation. However, “kriging” requires a nonconvex optimization routine to fit the parameters (Cressie 1990). This means sometimes the global optimum is not found and it is computationally intensive (Boyd and Vandenberghe 2004).

Multivariate adaptive regression splines (MARS, Friedman 1991) has been widely and well applied in many fields. It is essentially a linear model built up by a forward stepwise algorithm to select model terms (basis functions) and a backward ADP solution process to prune the model untill lack of fit is attained. The MARS model is given by

$$\hat{g}_M(x;\beta) = \beta_0 + \sum_{m=1}^{M} \beta_m B_m(x),$$

where $B_m(x)$ is a basis function, *M* is the number of linearly independent basis functions, and β_m is the an unknown coefficient. There are two types of basis functions in MARS models namely univariate basis functions and interaction basis functions. The univariate basis functions have the form

$$b^+(x;k) = [+(x-k)]_+,$$

$$b^-(x;k) = [-(x-k)]_+,$$

where $[q]_+ = \max\{0, q\}$ and k is a univariate knot. Eligible knots are selected separately for each input variable to coincide with the input levels. In the forward stepwise algorithm, interaction basis functions are products of an existing basis function and a truncated linear function with a new variable. The general form of a MARS basis function is given by

$$B_m(x) = \prod_{l=1}^{L_m} \left[s_{l,m} \cdot \left(x_{v(l,m)} - k_{l,m} \right) \right]_+,$$

where L_m is the number of truncated linear functions in multiplication, $x_{v(l,m)}$ is the input variable of the l^{th} truncated linear function in the m^{th} basis function, and k and s are corresponding knot value and direction. Since MARS is flexible and easily implemented (Chen et al. 1999, Sahu 2011), it has contributed a lot to the DP field as a value function approximation. The computational efforts of applying MARS primarily depends on the number of basis functions added to the model.

A popular statistical modeling tool in reinforcement learning area is artificial neural networks (NN) (Haykin 2004). NN has formed a family of machine learning methods, and various types of NN models that have been used for regression, classification and optimal control (Fan et al. 2013, Manry et al. 1996). Multilayer feed-forward NNs are widely used for approximation tasks (Fan et al. 2013). There are multiple layers of variables such as the input layer where the nodes are input variables, the output layer where responses variables are represented by the nodes, and at least one "hidden" layer with a specified number of hidden nodes. The transformations between layers are defined by activation functions. The number of nodes on hidden layer defines the complexity of the feed-forward NN model with one hidden layer. Therefore, ANNs models are flexible and can produce good directions. However, this family of models is known to produce results that are hard to interpret given the complex model structure (Sahu 2011).

As a precursor to Friedman's MARS model, classification and regression tree (CART) (Friedman et al. 1984) were developed and applied widely in both machine learning and operations research fields. Evolving from recursive partitioning (Breiman et al. 1984), the CART algorithm has two processes, a forward stepwise procedure and a backward procedure for pruning. The forward process partitions the feature space into disjoint hyper-rectangles by indicator functions given by

$$b^{+}(x;k) = 1\{x > k\},$$

$$b^{-}(x;k) = 1\{x \leq k\},$$

where k is the split point. The resulting basis functions are products of indicator functions given by

$$B_m(x) = \prod_{l=1}^{L_m} \left[b^{s_{l,m}} \cdot \left(x_{v(l,m)}; k_{l,m} \right) \right]_+.$$

CART has a piecewise constant form. Modern statisticians also refers to CART as a piecewise constant for piecewise linear function (Loh 2002) since it is very flexible to apply linear regression models on the leaf of the tree. The benefits are that a less complex tree would be built, and the linear regression models would provide easier interpretation (Alexander and Grimshaw 1996). In the forward stepwise procedure, a binary partition is selected at each stage to minimize the total sum of the squared errors, and the splitting would stop when the stopping criteria is satisfied. The widely used greedy search algorithm would stop the partitioning when the sample size on terminal nodes is too small or the fractional decrease in total sum of the squared errors is less than a pre-specified value (Friedman et al. 1984). However, the forward stepwise algorithm usually builds up a very complex tree (maximum tree) and has the issue of over-fitting. Alexander and Grimshaw (1996) applied a direct stopping rule to stop the partitioning in order to determine the optimal tree. A sequence of subtrees would be generated, and an N-fold cross validation would be applied to obtain the prediction mean square error (PMSE) of each subtree. Among those subtrees, the one with smallest PMSE is called 0-SE tree, and the 1-SE tree is the subtree with the smallest size and the PMSE within one standard error of the smallest PMSE (Loh 2002).

It is common to apply linear models on the leaf of a tree learning method to generate a piecewise linear model. Alexander and Grimshaw (1996) initially developed a

treed regression method that creates a binary tree with simple linear regression function at each leaf. It is proven that these types of models are more parsimonious than the CART models because fewer splits are generated and the final tree would be less complicated for interpretation. Loh (2002) developed a regression tree algorithm called GUIDE, which controls the variable selection bias by employing Chi-square residual analysis and detects two-variable interactions directly. GUIDE enables complex modeling on the terminal nodes such as polynomial or simple linear regression models.

The existing continuous-state DP technique handles categorical variables by coding them as discrete variables and then treating them as continuous variables. This motivates research to develop TreeMARS and CATreeMARS, which are based on the decision tree structure (Sahu 2011), handle a mix of continuous and categorical or qualitative variables for value function approximation. Essentially both algorithms combine the CART and MARS methods by applying recursive partitioning to build up a tree model and then applying the MARS algorithm on the terminal nodes. The difference between TreeMARS and CATreeMARS is that TreeMARS has both quantitative and categorical variables for recursive partitioning, while CATreeMARS would only use categorical variables to build up the tree part. A tree based algorithm, called extremely randomized trees, was developed by Geurts et al. (2006) consists of randomizing strongly both attribute and cut-point choice while splitting a tree node. The main strength of the extremely randomized trees algorithm is computational efficiency (Geurts et al. 2006).

One of the most critical components of modern DP methodology, the value function approximation, requires a model based statistical modeling method. Artificial neural networks (NN) have been applied much in reinforcement learning based ADP (Ernst et al. 2004). A power system oscillation damping control problem (Ernst et al.

2004) was solved by a Q-learning based ADP with extremely randomized trees as learning function. MARS and OA were utilized by Chen et al. (1999) in several DACE based ADP problems such as the inventory problem and Water Reservoir problem. ANNs were used as value function approximations in a nine dimension ADP problem by Fan (2006), in his value function adaptive algorithms and sequential state space exploration algorithm.

Good statistical models and learning algorithms are able to handle value function approximation with high dimension, with the help of experimental of design or sequential sampling of state space (Fan et al. 2013). This is the advantage of DACE based ADP over other simulation based ADP method or reinforcement learning based ADP method. The benefits would be clearer and more significant when solving real world high-dimensional, large scale and non-convex multi-stage decision problem (Chen et al. 1999).

2.4.4 Adaptive Value Function Approximation and State Space Exploration

Two sequential algorithms were developed within the DACE based ADP structure by (Fan et al. 2013). The first one is the adaptive value function approximation (AVFA), which uses a sequential concept based on consistency theory. AVFA is essentially a trade-off between variance and bias by increasing training data untill the testing error levels off. In statistical modeling methods, the “generalization error” is defined as the expected loss or expected prediction error over an independent test sample. The “generalization error” can be decomposed into bias and variance, which is not true for the case of training error. Therefore, while the training error decreases monotonically with the increasing model complexity, the testing error will begin to increase when the model becomes overly complex (Sahu 2011).

Because the state space region is are usually unknown for real world continuous-state dynamic programs. A sequential state space exploration (SSSE) framework was proposed by Fan (2008) to identify the state space region forward through the stages. The combination of SSSE and AVFA requires several iterations of backward and forward procedures to identify the state space region and perfect modeling complexity. This is described further in Chapter 3

2.5 Research Focus

Besides reinforcement learning based ADP methods and DACE based ADP methods, there are some other ADP methods that are used in certain situations. Among those ADP methods, DACE based ADP algorithms can handle problems with higher dimension and a finite time horizon (Yang et al. 2009). Reinforcement learning ADP algorithms are simpler to implement, and many of them have provably convergence (Sutton and Barto 1998) for infinite horizon problems. Both of them have some heuristic methods such as sample average approximation or stochastic approximation handling uncertainty.

This dissertation focuses on studying multistage decision problems with finite horizon, non-stationary transition functions, near continuous and high-dimensional state spaces, and mixed discrete-continuous and high-dimensional action spaces. A high-dimensional uncertainty space is also involved in the application study. Both DACE based ADP methods and reinforcement learning based ADP methods are developed. Besides solving a highly complex real world problem, this study also compares different methodologies and considers directors for improvement.

Chapter 3

Sequential Algorithms of ADP Using MARS

In most dynamic programs, the state space is assumed to be known and state space sampling is conducted within the assumed region (Fan 2008). Unfortunately, this is not the case in many real-world continuous-state space DP problems. A sequential framework to conduct state space exploration (SSSE, Fan 2008) is designed to identify an appropriate specification of the state space for a continuous-state space DP application. The adaptive value function approximation (AVFA) algorithms (Fan et al. 2013) are also designed to identify the trade-off between variance and bias by increasing training data until the testing error levels off. In this work, forward sequential state space exploration (SSSE) is an iterative forward-backward framework that combines NN with AVFA in finite-horizon continuous-state DP problems. This study adapts the existing SSSE algorithms to another statistical learning tool, MARS, which adaptively determines the model structure (Sahu 2011). Simulation experiments show that the SSSE-AVFA sequential ADP algorithm using MARS generates better solutions that the version using NN. The version with MARS achieves smaller variances and converges to the state space region in fewer iterations. In addition, some variations of the new SSSE framework are studied and experimental results show that these variations can significantly reduce the required number of iterations without compromising the state space exploration results and DP solution quality.

3.1 SSSE Algorithms Using MARS

3.1.1 SSSE Framework Using MARS

In sequential ADP algorithms, AVFA identifies the tradeoff between variance and bias. For a given model form, it determines an appropriate statistical model structure for approximating the value functions by sequentially increasing model complexity and

training data in the approximation step. To sequentially sample the continuous-state space, a low discrepancy sequence is used for state space discretization. The sample points with corresponding optimal objective values are used to approximate the value function. A nonparametric statistical learning model is adjusted to increase or decrease its structures in the sequential framework. For example, the number of hidden nodes in the hidden layer of an NN model is the adjustable parameter within the AVFA framework by NNs. Figure 3-1 shows the process of AVFA with NNs (Fan et al. 2013). The parameters of this process include the initial training data size, the values of adjustable parameters to train the NN value function models, and stopping rules of the sequential process.

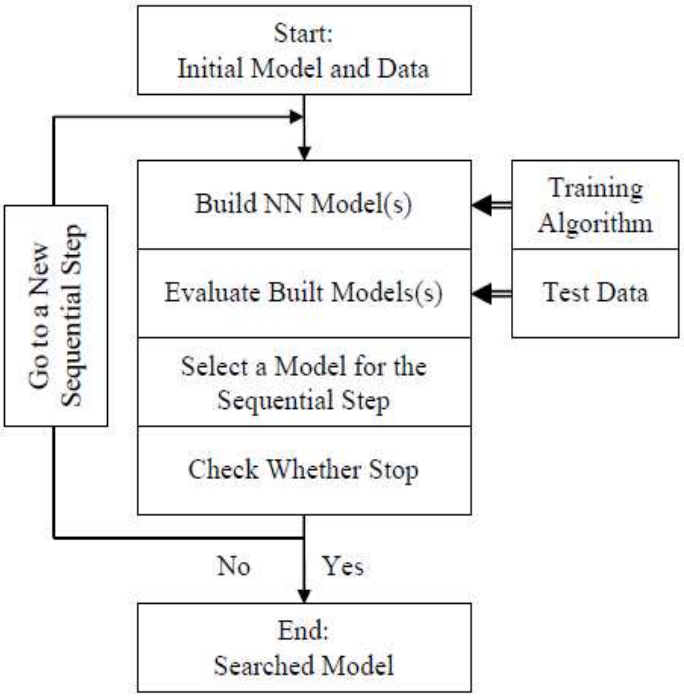


Figure 3-1 AVFA Sequential ADP Framework with NNs (original figure from Fan 2008)

MARS holds some important advantages over NNs. The structure of MARS lends itself naturally to control for the complexity of the model building stage which helps

to control the consistency (Sahu 2011). In addition, MARS can capture complex nonlinearity and the underlying interactions in the data efficiently. It can be used to approximate any arbitrary regression function (Friedman 1991). Therefore, this research uses MARS as the nonparametric statistical model of choice for use in the ADP AVFA framework. Unlike specifying the number of hidden nodes in the NN hidden layers, the model complexity selection is built into the model building adaptive process. The AVFA sequential ADP framework by MARS is similar to the framework of NN except that the model building process can adaptively adjust the complexity parameter by itself.

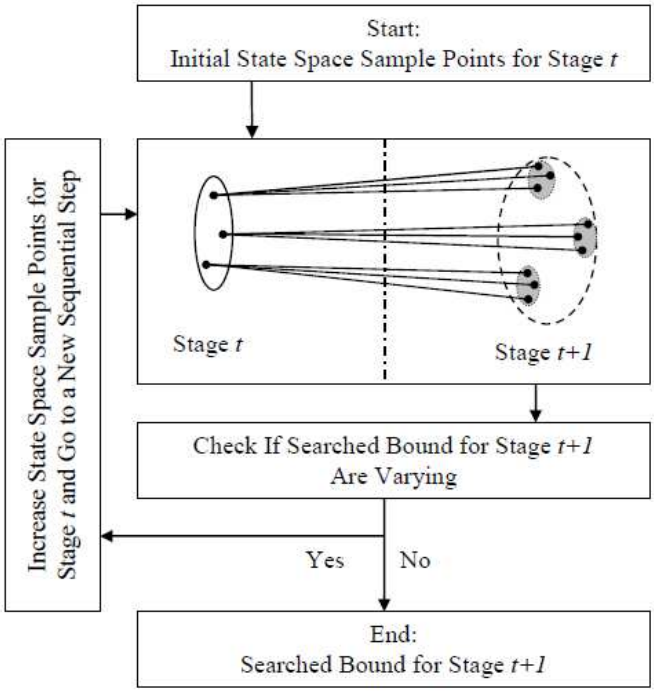


Figure 3-2 SSSE Forward Exploration Flowchart (original figure from Fan 2008)

Based on the advantages of MARS, we propose an SSSE framework using MARS. Similar to Fan's SSSE framework, our study has an forward exploration step. In the forward exploration step, the state space sample points are transitioned forward from stage *t* to the future state in stage *t*+1, and the resulting encompassing limits that specify

the region of the future state points are tracked until certain stopping criteria are met, indicating little change in the stage *t*+1. There are two types of forward explorations steps. The first type is used before the value functions have not been approximated. In this case, a decision at each stage on each state design point is sampled multiple times from the action space to determine the state of the next stage. Once a value function approximation exists, the Bellman equation (Chapter 2) may be used to solve for the optimal decisions at the current stage to obtain the state of the future stage. Since the future value function is based on the optimal decision, the second type should generate a better representation of the future state space region. Figure 3-3 shows the sequential state space exploration framework using MARS.

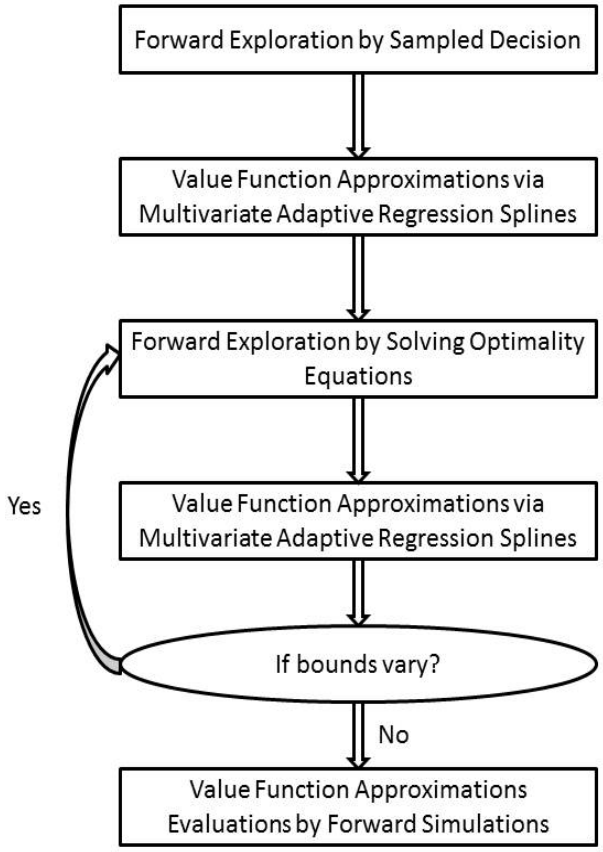


Figure 3-3 SSSE by MARS

3.1.2 Algorithms

The algorithm of SSSE using MARS can be described by the following steps (each step will be described in a pseudo code later):

Step 1. Given limits for the state space region of the first stage, use forward exploration to sample the future state space region.
Step 2. Given the state space region explored in Step 1, apply AVFA using MARS to get value function approximations.
Step 3. Given updated value function approximations, use forward exploration to sample the future state space region by solving optimality equations. Use sampled points to update the limits of future state space region. If it is at least the second occurrence of Step 3, and the update of state space region results in no change, then move to Step 5. Otherwise, go to Step 4.
Step 4. Given identified limits of the state space region in Step 3, apply AVFA via MARS. Go to Step 3.
Step 5. Apply forward re-optimization simulation to evaluate the value function approximations with the latest limits of state space region.

Let i represents the stage number and j represents the forward exploration iteration number within Steps 1 and 3. The notation in the algorithms of Steps 1 to 4 is as follows:

N — number of stages
$DF_{i,j}$ – state space sample set at stage i and forward exploration iteration j
$VF_{i,j}$ – state points projected by state transition
f — state transition
$NF_{i,j}$ – size of the set $DF_{i,j}$
N_d – size of action space sample for each state space sample point
N_e -- size of random space sample for each state and action sample combination
LB_i, UB_i – lower limit and upper limit of stage i
$LB_{i,j}, UB_{i,j}$ -- lower and upper limit of forward exploration iteration j at stage i
PF – sample random path
NPF – size of sample random paths

Steps 1-4 are the SSSE framework. Step 5 is the evaluation step for the obtained limits of state space region and value function approximations. The algorithms in Steps 2 and 4 employ the AVFA algorithms using MARS. Each of the five steps can be described in pseudo code as follows:

Step 1:

For $i = 0\ to\ N-1$:

Initial state space sample $DF_{i,0}$ by low discrepancy sequences;

For j = $1\ to\ N_I$:

sample state $DF_{i,j}$, sample N_d action points per state, and sample N_e random points per state-decision combination,

$$VF_{i,j} = f(s_{i,j}, x, \varepsilon_{i,j}),$$

$$UB_{i+1,j} = \max(VF_{i,j}),\ LB_{i+1,j} = \min(VF_{i,j});$$

update LB_{i+1}, UB_{i+1}:

if $j = 0$, $LB_{i+1} = LB_{i+1,0}$, $UB_{i+1} = UB_{i+1,0}$,

otherwise

$$LB_{i+1} = \min(LB_{i+1}, LB_{i+1,j}),$$

$$UB_{i+1} = \max(UB_{i+1}, UB_{i+1,j}).$$

Stop if the stopping criterion is satisfied.

Step 2, 4:

For i = $N\ to\ 1$:

Initial state space sample $DF_{i,N}$;

For j = $1\ to\ N_I$:

sample state $DF_{i,j}$;

solving optimality equation

$$V_t(S_t) = \min_{x_t \in X_t} E\{C_t(S_t, x_t, \varepsilon_t) + \hat{V}_{t+1}(S_{t+1}) | S_t\}.$$

Approximate value function using MARS based on the sequentially added training dataset $VF_{i,j}$;

Stop the sequential steps if the mean square error from testing dataset is not varying within certain sequential steps.

Step 3:

For $i = 0\ to\ N-1$:

Initial state space sample $DF_{i,0}$ by low discrepancy sequences;

For j = 1 to N_I:

sample state $DF_{i,j}$,

solve optimality equation

$$V_t(S_t) = \min_{x_t \in X_t} E\{C_t(S_t, x_t, \varepsilon_t) + \hat{V}_{t+1}(S_{t+1})\};$$

sample N_e random points per state-decision, transit to next stage;

$$VF_{i,j} = f(s_{i,j}, x_{i,j}, \varepsilon_{i,j}),$$

$$UB_{i+1,j} = \max(VF_{i,j}),\ LB_{i+1,j} = \min(VF_{i,j}).$$

update LB_{i+1}, UB_{i+1}:

if $j = 0$,

$$LB_{i+1} = LB_{i+1,0},\ UB_{i+1} = UB_{i+1,0},$$

otherwise

$$LB_{i+1} = \min(LB_{i+1}, LB_{i+1,j}),$$

$$UB_{i+1} = \max(UB_{i+1}, UB_{i+1,j}).$$

Stop if the stopping criterion is satisfied.

Step 5:

For $i = 1\ to\ N_i$:

Sample random path PF_i

For $j = 1\ to\ NPF_i$:

For stage from 0 to *N*

Solve optimality equation

$$V_t(S_t) = \min_{x_t \in X_t} E\{C_t(s_t, x_t, \varepsilon_t) + \hat{V}_{t+1}(S_{t+1})\};$$

Obtain $C_{t,i}(s_t, x_t^*)$, s_{t+1}, based on sample path PF_i;

$$S_i = \frac{1}{NPF_i} \sum_{i=0}^{NPF_i} \sum_{t=0}^{N} C_{t,i}(s_t),$$

which is the *simulation cost* of initial state i.

The procedure in Step 5 is to compare value function approximations by simulation. This step is also called *re-optimization* simulation. To compare the value function approximations by different models, same set of initial states are used to evaluate value function approximations. The simulation costs of various value function approximations are compared in terms of *absolute error*. The *absolute error* is defined as the difference between current solution and the best known solution. For each initial

state, the best known solution with the lowest simulation cost is selected. The absolute error of any other method is the difference between the simulation cost of this method and the best know solution. Usually absolute error of methods are compared in boxplots.

There are three types of stopping criteria in the SSSE framework using MARS. The first stopping criterion is implemented in the forward exploration Steps 1 and 3. The first stopping criterion is the distance between identified state space limits is less than 0.01 for several consecutive iterations within forward exploration. The second type of stopping criterion implemented with the AVFA algorithms is when the moving average difference between the testing mean square errors of several iterations is less than 0.01. This rule is implemented in Steps 2 and 4. The last type of rule implemented as stopping criterion after each occurrence of Step 4 is distance between *LB*s and *UB*s for consecutive rounds of forward exploration is less than 0.01. This rule is met to stop the SSSE iterations and proceed to Step 5.

3.2 Application in an Inventory Forecasting Problem

3.2.1 Inventory Forecasting Problem

Inventory forecasting problem is a typical stochastic DP problem with a continuous-state space (Chen et al. 1999, Fan 2008, Sahu 2011). To illustrate the SSSE-AVFA ADP algorithms, a nine dimensional inventory forecasting problem is chosen. This problem has been frequently studied (Chen et al. 1999, Cervellera et al. 2007, Wen et al. 2005) for DACE based ADP.

The state variables for the inventory forecasting problem consist of the inventory levels and demand forecasts for each item. For the chosen nine-dimensional problem, three items and two forecasts for each item are considered. At the beginning of stage t, let $I_t^{(i)}$ be the inventory level for item i at the beginning of the current stage. Let $D_{t,t}^{(i)}$ be the forecast made at the beginning of the current stage for the demand of item i occurring

over the stage. Similarly, let $D_{t,t+1}^{(i)}$ be the forecast made at the beginning of the current stage for the demand of item i occurring over the next stage. Therefore, the state vector at the beginning of stage t is:

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

The decision variables of the inventory problem are the order quantities for the items. Let $u_t^{(i)}$ be the amount of item i ordered at the beginning of stage t.

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

We assume that orders arrive instantly. The transition function from period t to period $t+1$ is

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

where D is the forecasted random demand vector modeled in transition functions by the multiplicative Martingale model of forecast evolution (Chen et al. 1999).

The constraints for inventory forecasting are placed on the amount ordered in the form of capacity constraints, and on the state variables, in the form of the state space region. In this study, the capacity constraints are chosen to be restrictive, forcing interactions between the state variables. The objective function consists of inventory holding costs and backorder costs. The typical V-shaped cost function in inventory modeling is

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

where h_i is the holding cost per unit for item i, and π_i is the backorder cost per unit for item i. The smoothed version of the objective function (Chen et al. 1999) is employed.

3.2.2 Implementation Setup

The experiments were conducted on a Dual 2.6 GHz Linux workstation. The SSSE-AVFA algorithms were implemented using the MATLAB software system. The

Areslab toolbox (Jekabsons 2010) was used to build up the MARS models. The original version of MARS was modified to be implemented with the maximum number of basis functions = $\left\lfloor \frac{2n+c}{2+c} \right\rfloor$ (Jekabsons 2010), 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 cap is the maximum number of basis functions a model can have after the forward selection algorithm. To compare the numerical results, this study also used the Neural Network toolbox of MATLAB to build value function approximations.. The training algorithm of the NN toolbox is based on the back propagation process (Lippmann 1987). For all of the NN models, the activation functions for the hidden nodes and for the output nodes were sigmoid and linear, respectively. A training process was limited to 100 network weight updates. In this study, two value functions of the second stage and the third (last) stage are required to be approximated. For both stage 2 and stage 3, training data used to build the value function approximations are sampled by a nine-dimensional Sobol' sequence. The test data are taken as 500 design points by a nine-dimensional Halton sequence.

3.2.3 Value Function Approximation Accuracy

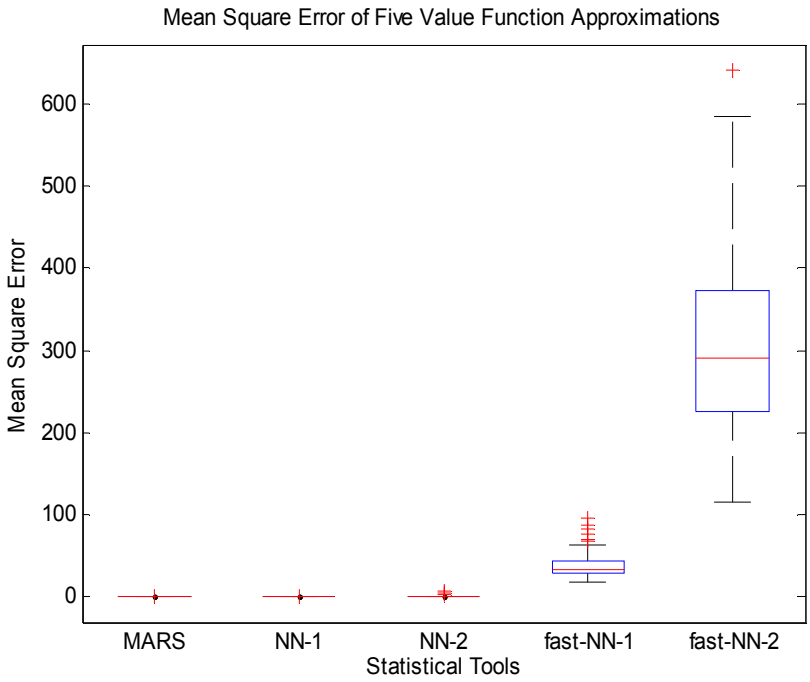


Figure 3-4 Approximation Accuracy of Several Regression Models

To compare the approximation accuracy of three potential tools for value function approximation, this study uses 2000 data points from the training dataset and 500 separate data points to evaluate the trained models. Figures 3-4 and 3-5 show that MARS and NN perform well in terms of mean square error of value function approximations. The columns 4, 5, and 6 in Figure 3-4 and column 3 in Figure 3-5 are results from experiments with a faster version of NN (Narasimha et al. 2008). MARS and NN perform much better than the faster version NN model (Narasimha et al. 2008) in terms of approximation accuracy.

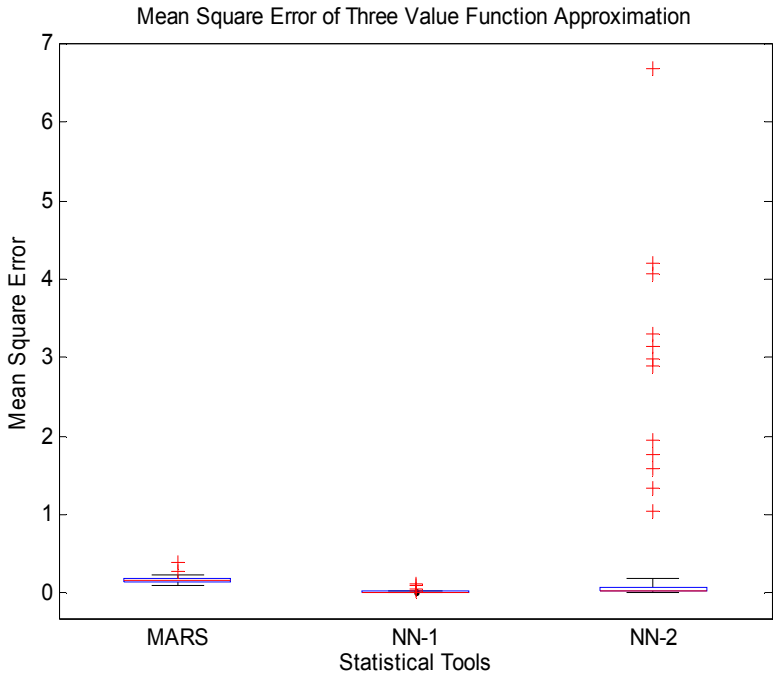


Figure 3-5 Approximation Accuracy in terms of R-square of Several Models

3.3 Experimental Results and Discussions

3.3.1 Observations on Searching Limits

The experimental results of this study are presented from Figure 3-6 to Figure 3-11. In each of those figures, there are nine subplots representing nine state variables on each stage. The vertical axis represents the upper and lower limits of this state variable. The horizontal axis represents the iteration number of the SSSE experiment. Figures 3-6 and 3-7 show the limits of the evolving state space regions of the nine state variables in Step 1. In Step 1, decisions are sampled within the action space to explore the state space of the next stage.

In this inventory forecasting problem, Step 1 required 17 iterations to conduct the limit search. In stage 2, $D_{t,t}^{(2)}, D_{t,t}^{(3)}$ show an unstable upper limit. In stage 3, $D_{t,t}^{(1)}$, $D_{t,t}^{(2)}$ and $D_{t,t}^{(3)}$ show unstable upper limits in the first several iterations. The limits always get wider

as the SSSE iterations increment. In addition, the limits are relatively wider than the originally assumed limits. Figures 3-8 and 3-9 show the limits of the evolving state space of the nine state variables after each occurrence of Step 3.

In the forward exploration of Step 3, value function approximations are used to search the state space of the next stage. In this inventory forecasting problem, the Steps 3 and 4 procedures took 12 iterations using SSSE with MARS. It took 6 and 8 iterations for stage 2 and stage 3 to finalize their limits within Step 3, respectively. In stage 2, the third inventory variable (c) shows unstable upper and lower limits. The first demand forecast variable (d) shows unstable upper limit. In stage 3, the first and third inventory variables (a, c) show unstable upper and lower limits before the Steps 3 and 4 loops. The third and fourth (f, g) show unstable lower limits before several Steps 3 and 4 loops. The identified state space region is also relatively wider than the originally assumed regions.

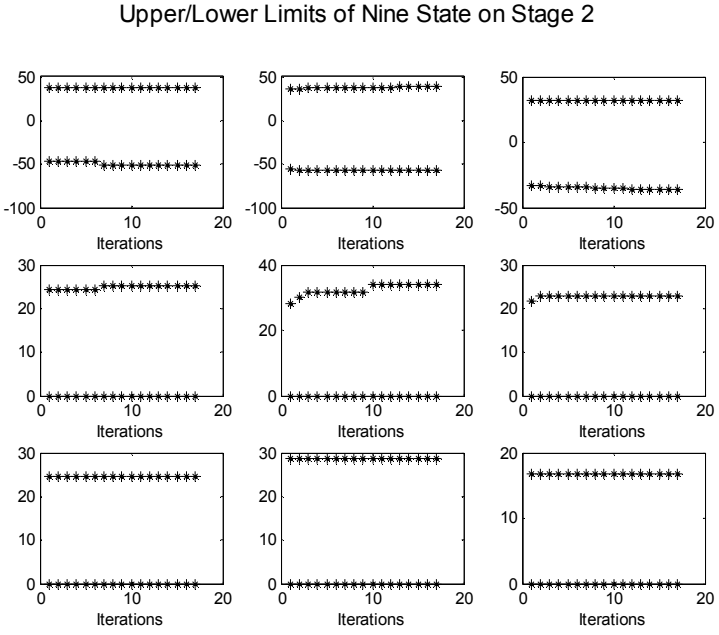


Figure 3-6 Observations of Limits on Stage 2 after Sequential Step 1

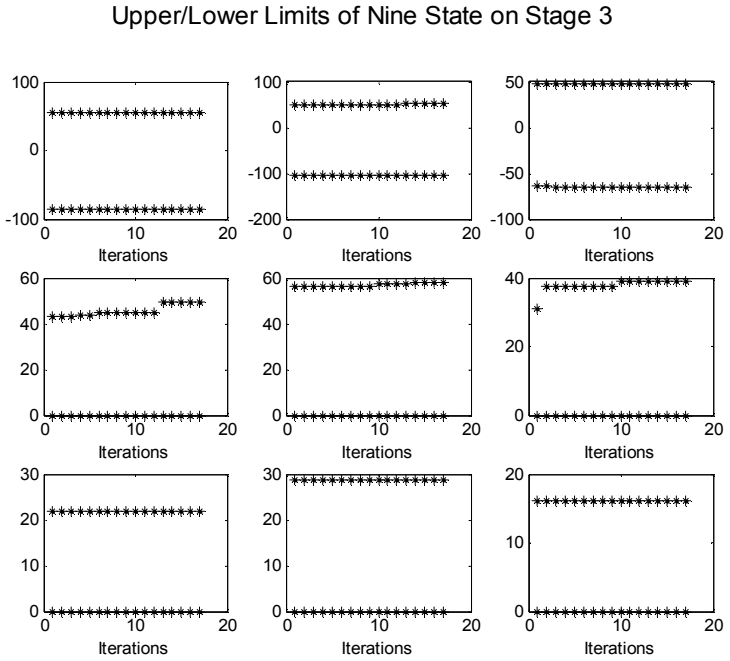


Figure 3-7 Observations of Limits on Stage 3 after Sequential Step 1

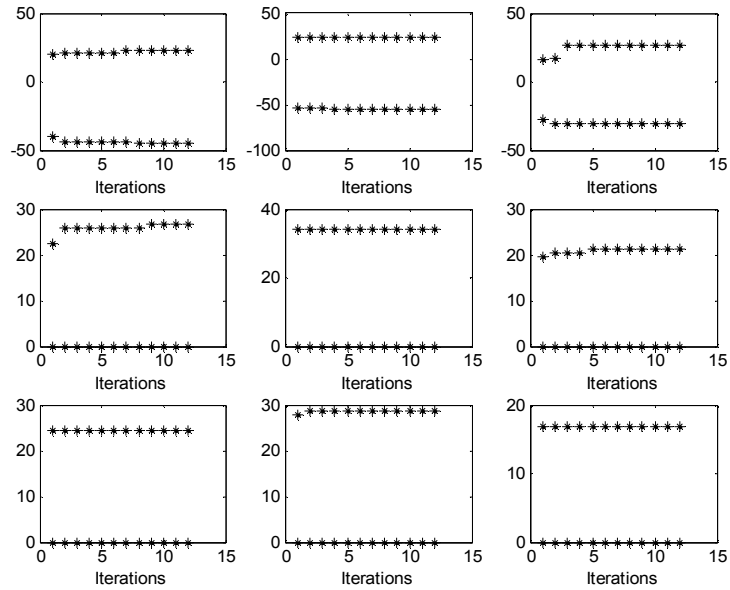


Figure 3-8 Observations of Limits on Stage 2 after Sequential Step 3 via SSSE with MARS

Figures 3-10 and 3-11 show the limits of evolving state space of the nine state variables in Step 3 using Neural Network toolbox (Fan 2008). We compare the performance of SSSE framework using MARS and the NN toolbox by the numbers of Steps 3 and 4 iterations before meeting stopping criteria. The framework using the NN toolbox takes 25 iterations to find stable limits. It is more than the number of iterations required for the SSSE framework using MARS. It is also observed that the state space region is more stable in the SSSE framework via MARS than with NNs. In stage 2, the first inventory variable shows an unstable lower limit in the first several iterations. The first, second, fourth and sixth demand forecast variables show unstable upper limits in the first 20 iterations. Figures 3-8 and 3-9 show the state space region after every occurrence of Step 3. The second inventory variable in stage 3 shows an unstable lower limit. The third inventory variable shows unstable upper and lower limits. The six demand forecast variables show unstable upper limits except for the fifth demand forecast variable.

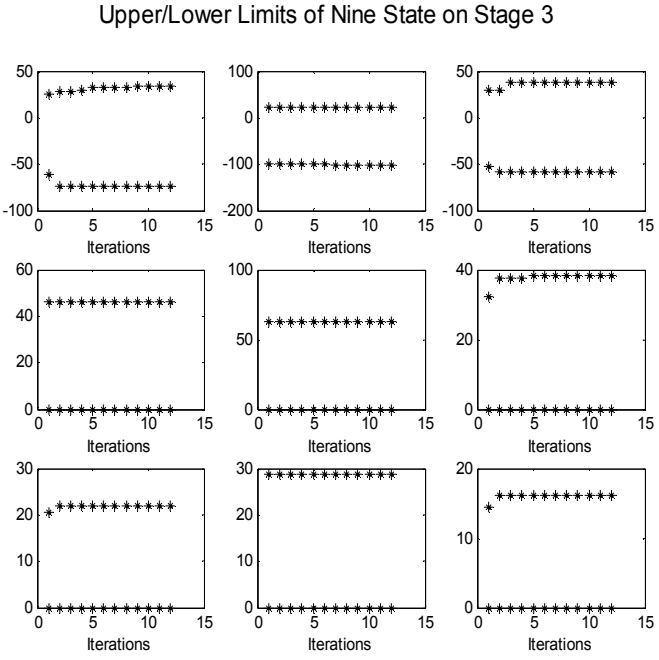


Figure 3-9 Observations of Limits on Stage 3 after Sequential Step 3 via MARS

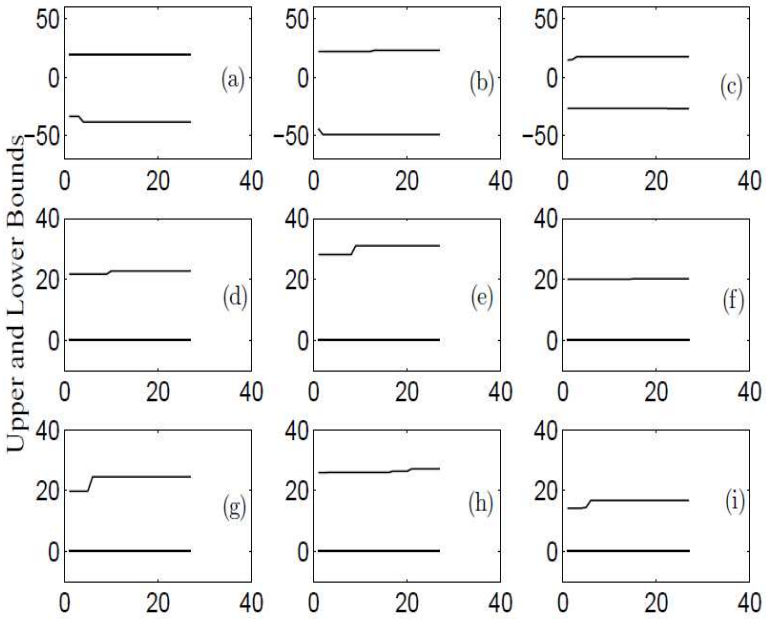


Figure 3-10 Observations of Limits on Stage 2 after Step 3 via NNs

(original figure from Fan 2008)

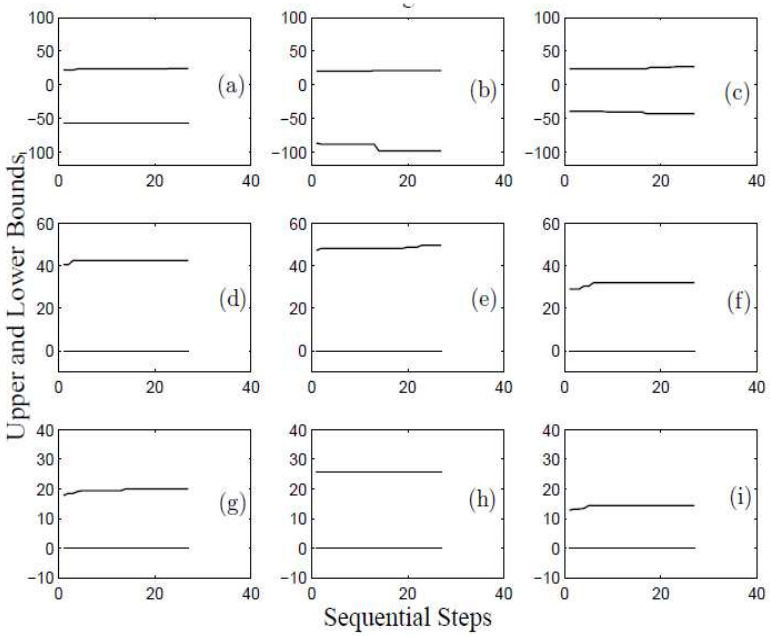


Figure 3-11 Observations of Limits on Stage 3 after Step 3 via NNs

(original figure from Fan 2008)

3.3.2 Simulation Comparison

Figure 3-13 shows the final state space region of SSSE via MARS. Figure 3-14 shows the final state space region of SSSE via NN. In both figures, the lowest point of each column in the lower limits figure represents the final lower limit for each of the nine state variables. Similarly, the highest point of each column in the upper limits figure represents the final upper limit for each of the nine state variables. The final limits of the state space region by both methods are very close. As described in Section 3.3.1, the SSSE framework by MARS shows more stable limits and takes fewer iterations than by NN. It takes about 10 more iterations of Steps 3 and 4 for SSSE via NN to reach the stopping criteria. Figure 3-12 shows the simulations results for SSSE Step 5. The results show that SSSE via MARS and NN generate close results in terms of simulation cost.

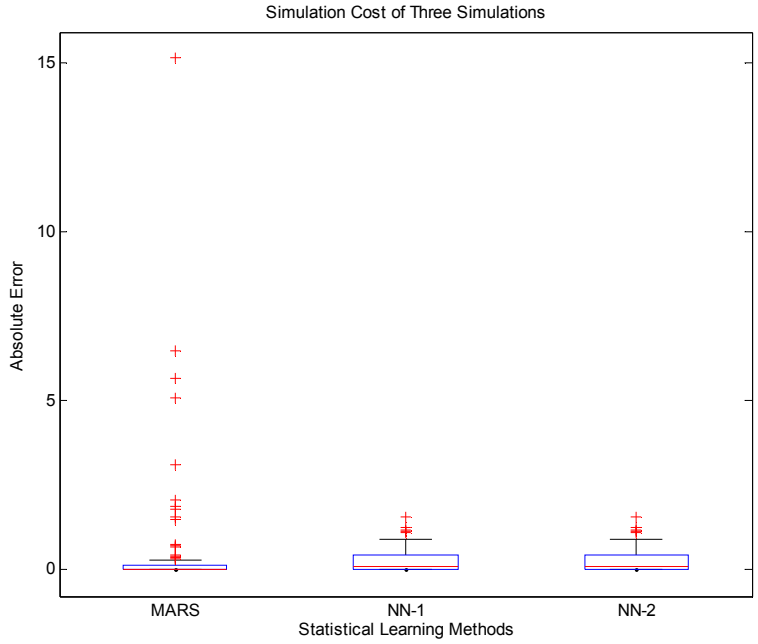


Figure 3-12 Simulation Results of Algorithms (Step 5 of Sequential Algorithms)

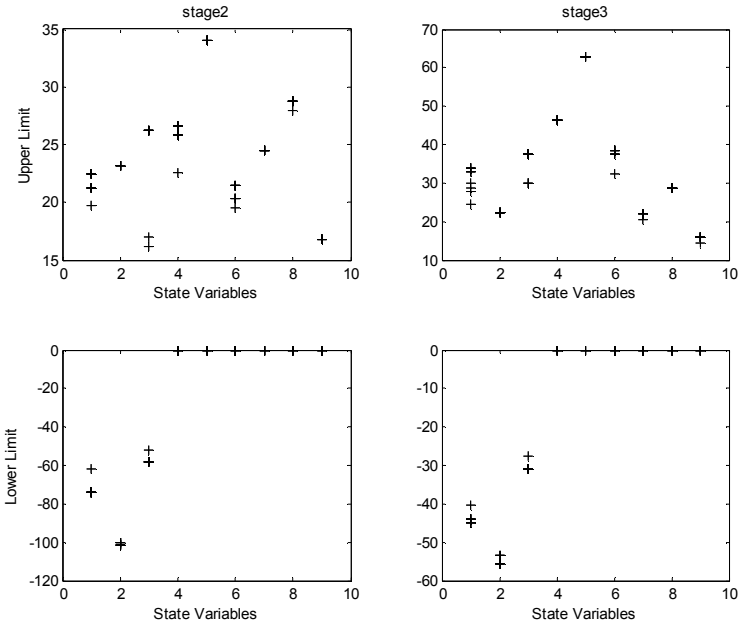


Figure 3-13 Identified Limits via MARS

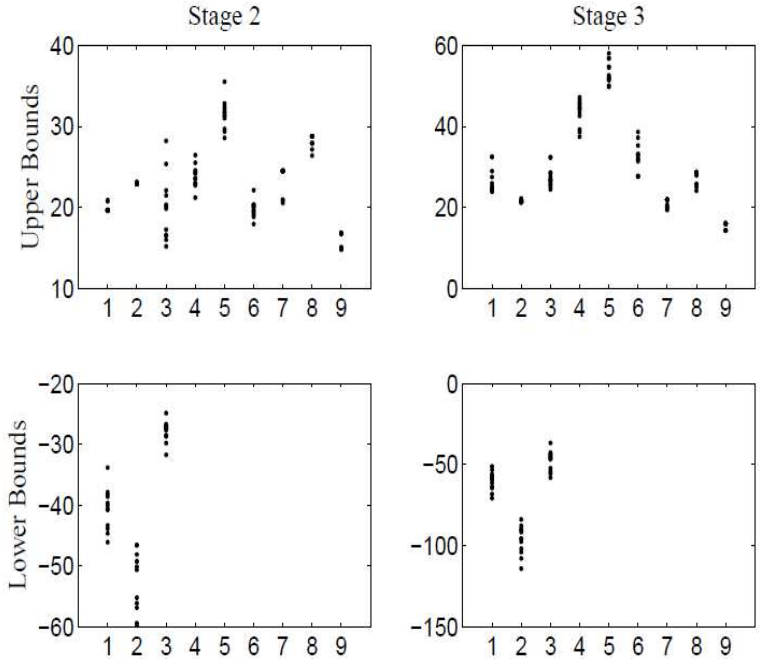


Figure 3-14 Identified Limits via NNs (original figure from Fan 2008)

3.3.3 Improving SSSE Limits Search by Percentile Rules

In the SSSE framework via MARS described in Section 3.1.2, the sequential algorithm updates the upper/lower limits by tracking maximum/minimum of the sampled future state space points (after state transition) in Step 1 and Step 3. In this section, we experiment with the percentile rule instead of maximum/minimum (MIN/MAX) rule. The *percentile rule* is defined as the upper/lower limit of the next stage updated by the 95th/5th or 99th/1st percentile of the sampled future state values (after state transition). The reason for this experiment is to determine if the SSSE using MIN/MAX rules consider a very small number of extreme cases in each state space exploration. These extreme cases require that the algorithms take additional iterations because limits continue to change. Avoiding these cases might be helpful in reducing SSSE computational effort.

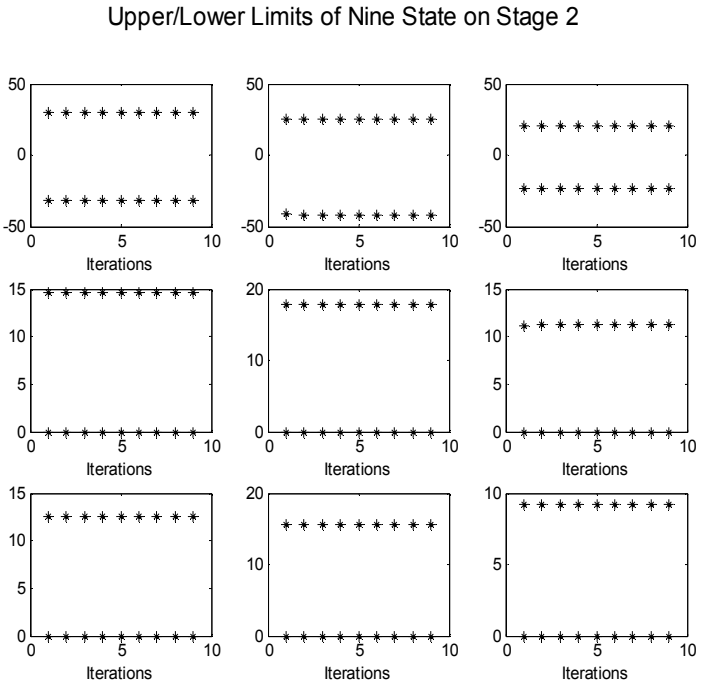


Figure 3-15 99th/1st Percentile Limits Observed on Stage 2 after Step 1 via SSSE with MARS

We implement the percentile rule instead of the MIN/MAX rule in Step 1 and Step 3 in the SSSE algorithms. By using imputing percentile in the forward exploration algorithms, we expect fewer iterations to meet stopping criteria as well as more stable upper and lower limits. In the meantime, the simulation (policy evaluation) results would not change significantly by implementing these percentile rules.

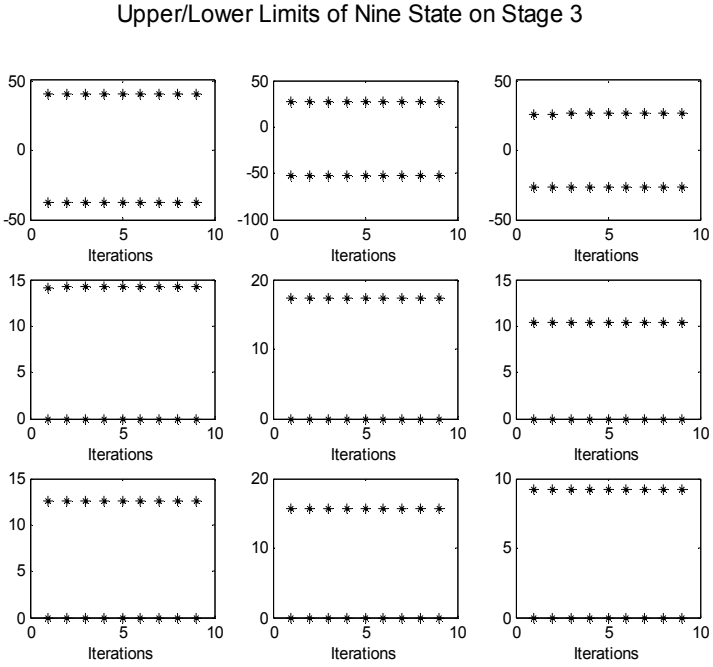


Figure 3-16 99th/1st Percentile Limits Observed on Stage 3 after Step 1 via MARS

Figures 3-15 to 3-18 show the SSSE results by the 99th/1st percentile rule. Figures 3-15 and 3-16 show the state space region search process on stage 2 and stage 3 in Step 1. Figures 3-17 and 3-18 show the state space region search process on stage 2 and stage 3 in the loop of Step 3 and Step 4. From these figures, it is observed that the 99th/1st percentile rule meets the stopping rules faster than the MIN/MAX rule in the SSSE framework. Step 1 takes 9 iterations on both stage 2 and stage 3 with the 99th/1st percentile rule. This is 10 iterations fewer than the number of iterations for Step 1 by the

MIN/MAX rule. Besides the number of iterations, the second demand forecast variable shows a more stable search by the 99th/1st percentile rule than the MIN/MAX rule because the dotted line is more flat.

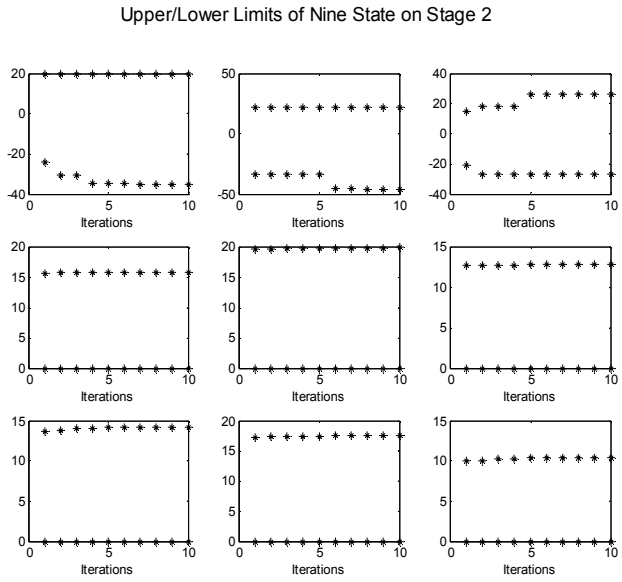


Figure 3-17 99th/1st Percentile Limits on Stage 2 after Step 3 via SSSE with MARS

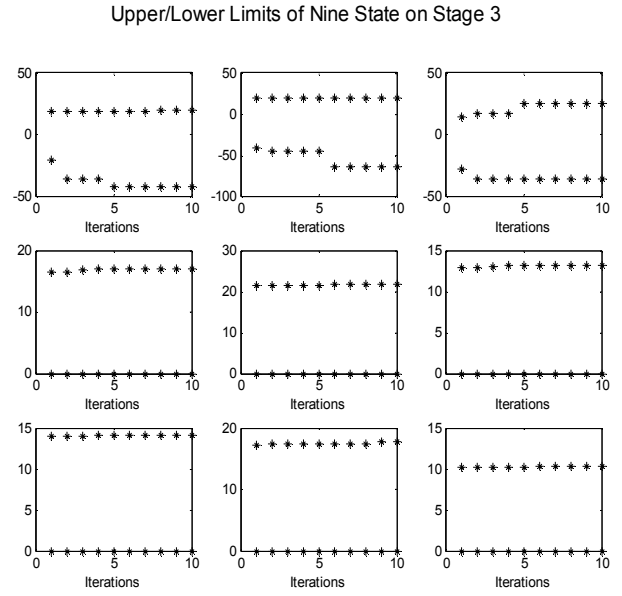


Figure 3-18 99th/1st Percentile Limits on Stage 3 after Step 3 via SSSE with MARS

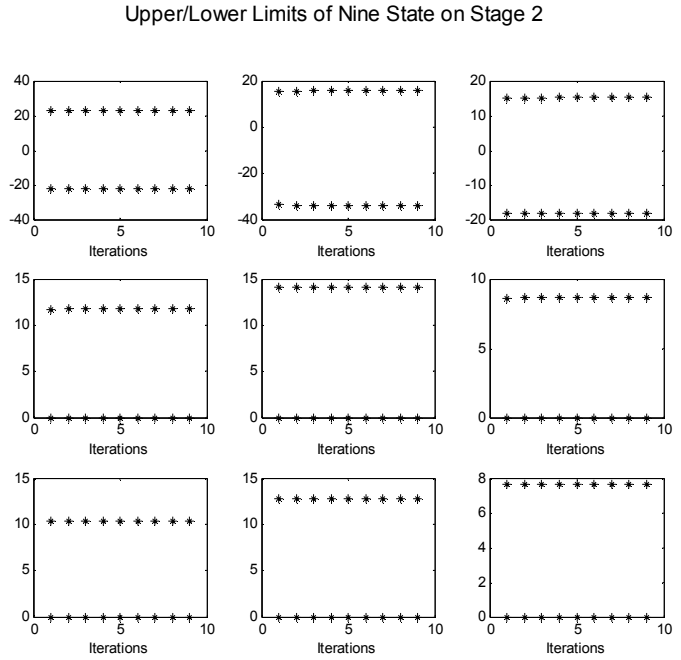


Figure 3-19 SSSE 95th/5th Percentile Limits Observed on Stage 2 after Step 1 via MARS

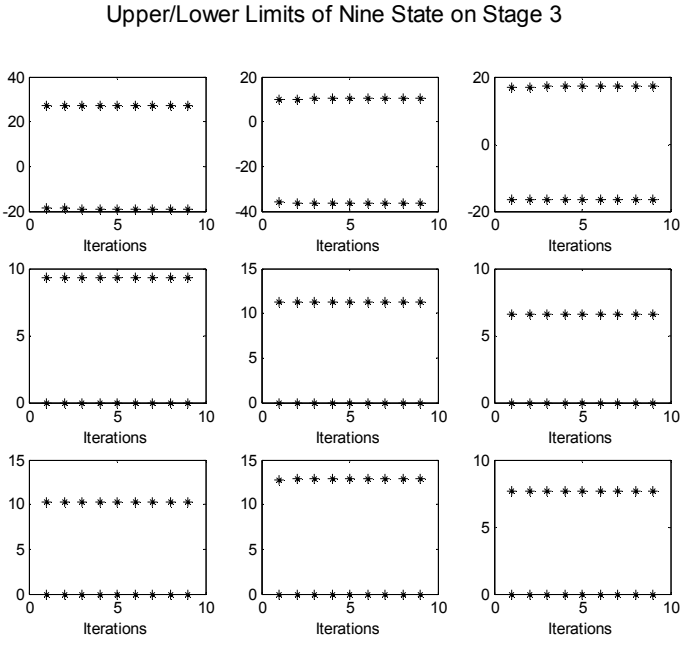


Figure 3-20 SSSE 95th/5th Percentile Limits Observed on Stage 3 after Step 1 via MARS

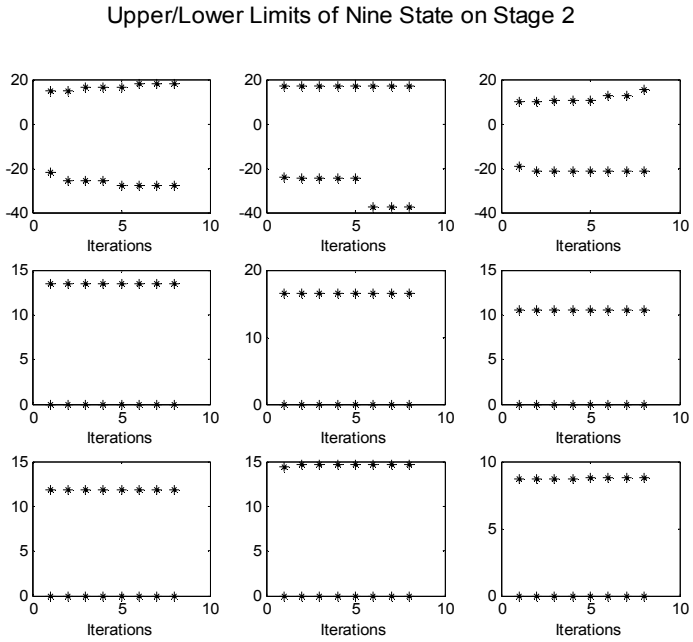


Figure 3-21 SSSE 95th/5th Percentile Limits Observed on Stage 2 after Step 3 via MARS

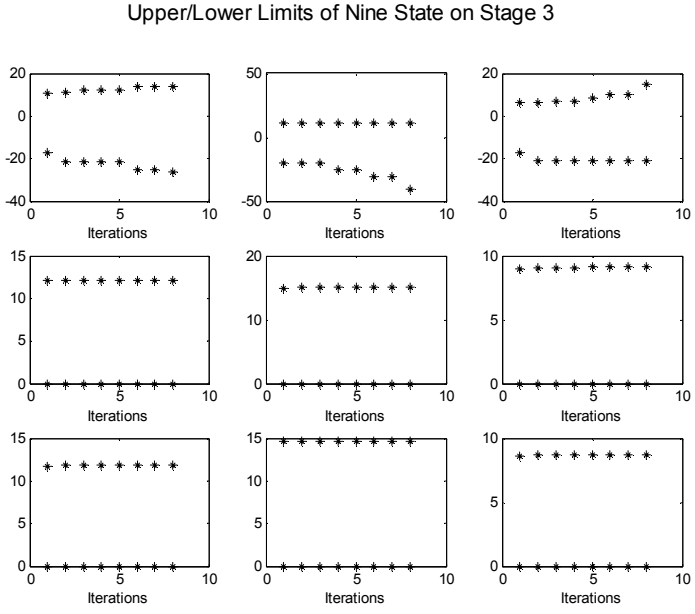


Figure 3-22 SSSE 95th/5th Percentile Limits Observed on Stage 3 after Step 3 via MARS

. Figures 3-19 to 3-22 show the SSSE results by 95th/5th percentile rule. Figures 3-19 and 3-20 show the state space exploration process in stage 2 and stage 3 for Step 1. Figures 3-21 and 3-22 show the state space region evolution on stage 2 and stage 3 in the loop of Step 3 and Step 4. From experimental results, it is observed that the stable limit searching by the 95th/5th percentile rule meets the stopping rules even faster than the 99th/1st percentile rule. Step 1 takes 8 iterations on stage 2 and stage 3 before meeting the stopping criteria. It takes one fewer iteration than the Step 1 by the 99th/1st percentile rule. The loop of Step 3 and Step 4 also takes 1 fewer iteration by the 95th/5th percentile rule than the 99th/1st percentile rule. However, more instabilities are observed from the 95th/5th percentile forward exploration than the 99th/1st percentile rule forward exploration of the inventory variables. In Figures 3-21 and 3-22, the three inventory variables are less stable in the first several iterations before finally meeting the stopping rules.

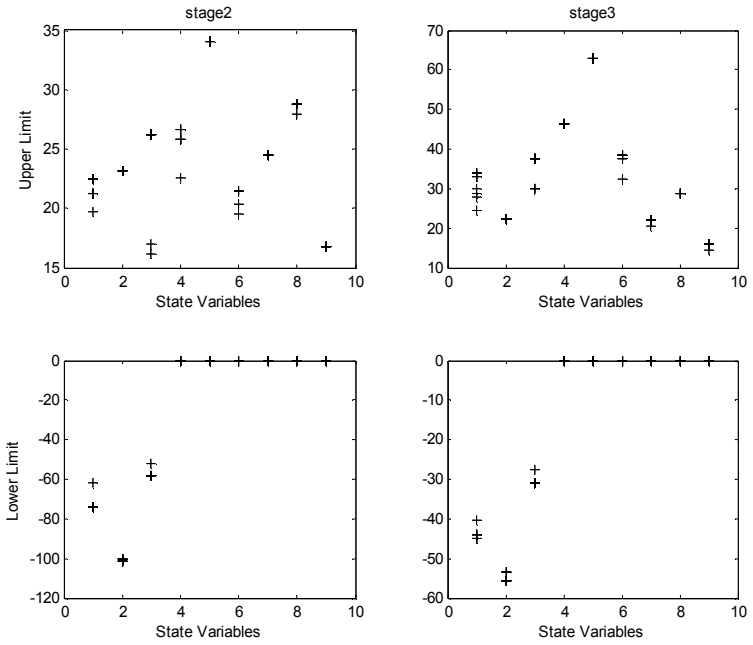


Figure 3-23 Searched Limits via MARS with MIN/MAX Rule

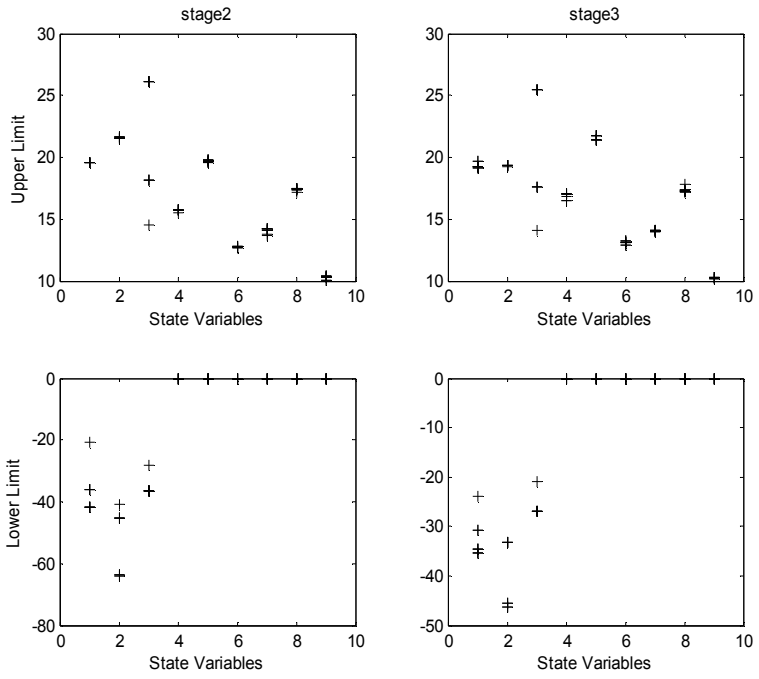


Figure 3-24 Searched Limits via MARS with 99th/1st Percentile Rule

Figures 3-23, 3-24, and 3-25 show the final state space region by three rules. The lowest point of each column on the lower limit figures represents the final lower limit of each state variable. The highest point of each column on the upper limit figures represents the final upper limit of each state variable. Figure 3-28 illustrates the Step 5 simulation results by three different sets of final limits that are the results of the three rules. Given different final limits, SSSE by MARS framework shows similar results for three methods of forward state space exploration. The 99th/1st percentile limits search gives much narrower limits than the MIN/MAX rule possibly because a few extreme cases widen the state space in the MIN/MAX search.

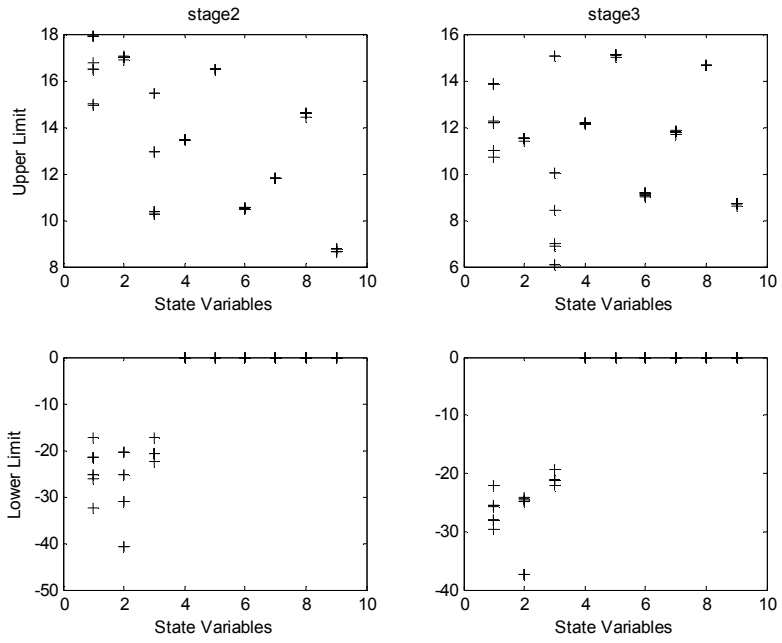


Figure 3-25 Searched Limits via MARS with 95th/5th Percentile Rule

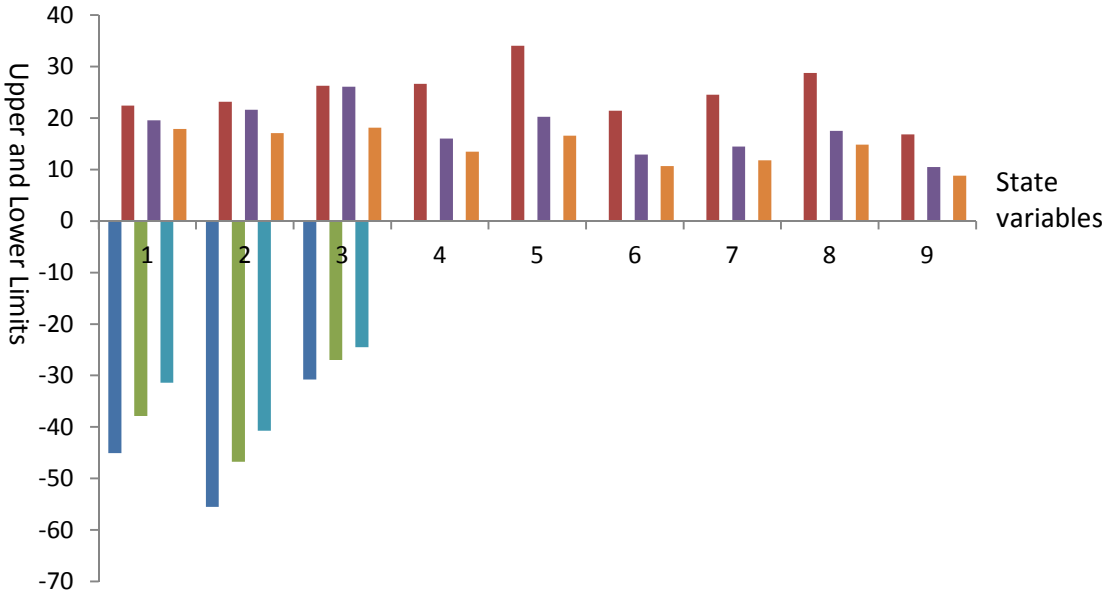


Figure 3-26 Searched Limits of Stage 2 via MARS by Various Rules

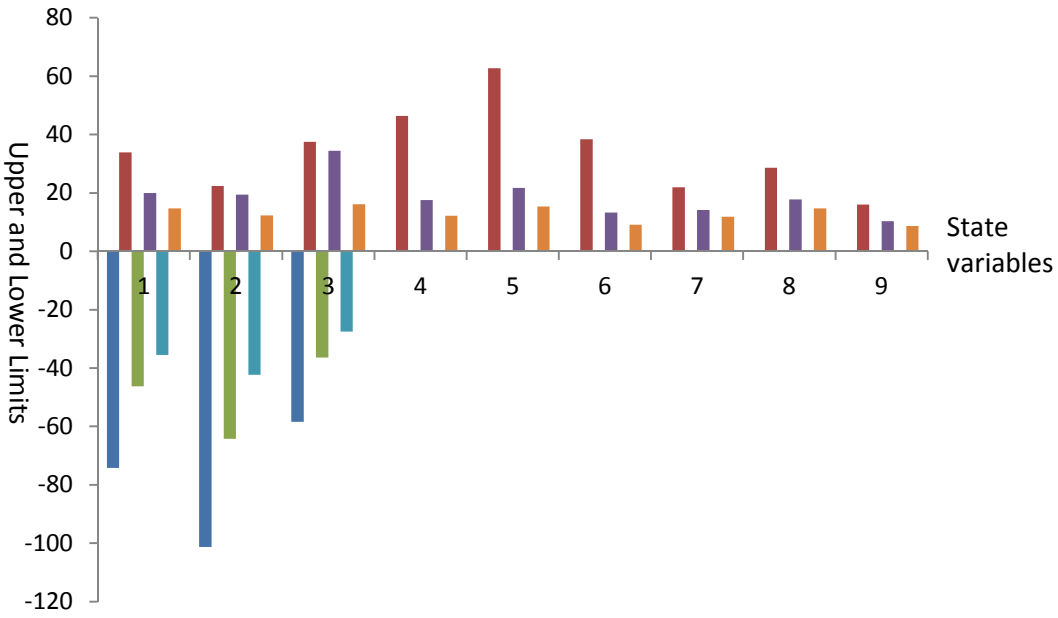


Figure 3-27 Searched Limits of Stage 3 via MARS by Various Rules

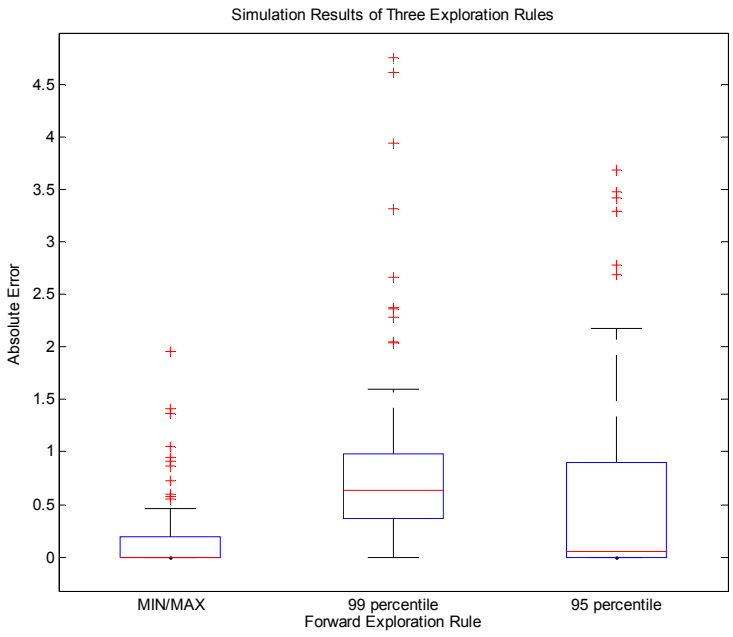


Figure 3-28 Simulation Cost Comparisons of SSSE via Various Rules

The width of the identified state space region by the 99th/1st percentile rule is 25 percent less than the MIN/MAX rule on stage 2. The width of the limits by the 99th/1st percentile rule is 50 percent less than the MIN/MAX rule on stage 3. In addition, 95th/5th percentile limits search decreases another 20 percent on the width of limits. However, there is an increasing number of points outside of the limits (extrapolation) by the state transitions. In the simulation, there are about 1.5 percent of the state transitions are extrapolations caused with the 99th/1st percentile limits. By the 95th/5th percentile rules, almost 10 percent of transitions are extrapolations. It is concluded that the 99th/1st percentile limits search in the forward exploration performs better than both the MIN/MAX rule and the 95th/5th percentile rule in terms of the convergence of the state space region, the stability of state variables, and the simulation results.

Chapter 4

Sequential ADP with Treed Regression and MILP

Solving optimality equations is an important step in DACE based ADP method. This chapter aims to improve the ADP solution quality by proposing a method based on the treed regression model (Friedman 1991, Alexander and Grimshaw 1996) and the branch-and-bound algorithm (Land and Doig 1960). A real-world DFW Airport deicing problem is studied by the proposed method with several experiments and benchmark studies.

4.1 Motivation

In DACE based ADP, various statistical learning tools such as MARS, CART and NN have been used to approximate future value functions (Chen et al. 1999, Fan 2008, Sahu 2011). These advanced approximation tools can adapt the model structures given the training data by selecting the most significant variables (Sahu 2011), adjusting the number of nodes on the hidden layer (Fan 2008), varying the parameters estimated in active functions (Fan 2008), and backward pruning to choose the best number of basis functions (Sahu 2011).

With the advanced approximation tools in ADP such as MARS and NN, the optimization objective functions for the optimality equations usually have nonconvex, nonlinear model structures. MARS models have higher order basis functions and interaction terms (Friedman 1991). Classic multiplayer NN models with a backpropagation algorithm consist of sets of adaptive weights that are tuned by the learning algorithm (Fan et al. 2013, Lippmann 1987). These models or algorithms have been used in solving various ADP problems (Chen et al. 1999, Fan et al. 2013, Yang et al. 2007) and are able to make good approximations. In previous studies, sequential quadratic algorithms and interior point methods have been applied to solve optimality

equations with NN or regression spline value function approximations (Chen et al. 1999, Fan et al. 2013). The optimal value and solutions found by those methods are not always guaranteed to be globally optimal (Boyd and Vandenberghe 2004).

In this chapter, we propose an ADP method with treed regression that aims to solve the optimality equation to the global optimum. Our proposed method is expected to significantly improve the solution quality by employing treed regression and the branch-and-bound optimization algorithm. In this study, a treed regression modeling procedure combining regression tree and stepwise linear regression is employed to construct a piecewise linear value function approximation in ADP. It can be described by two phases. The first phase uses CART with a pre-pruning procedure to incorporate all state variables and to restrict the minimum size of training data assigned to terminal leaf. The second phase refines the approximation from the first phase using multiple stepwise linear regression models based on all variables and the training data distributed to each terminal leaf.

The advantages of this procedure for approximating value functions include: 1) Piecewise linear functions can be used to approximate nonlinear and non-convex value functions. 2) Interaction between independent variables would be captured by the nested tree structure. 3) Only the significant variables would be selected. Both regression tree and stepwise linear regression are capable of selecting significant variables. 4) An optimization problem with a piecewise linear regression function can be transformed into a mixed integer linear programming (MILP) formulation. To illustrate this characteristic, a simple example is given by assuming a piecewise linear regression function given by

$$y = \begin{cases} a_1 x + k_1, & x < bp, \\ a_2 x + k_2, & x \geq bp, \end{cases}$$

where y represents the objective function, a_1, k_1, a_2, k_2 represent estimated coefficients, x represents an explanatory variable and bp is the breakpoint. The function can be transformed to

$$y = d_1(a_1 x + k_1) + d_2(a_2 x + k_2),$$

where d_1 and d_2 are binary variables. d_1=1 represents $x < bp$ and d_2=1 represents $x \geq bp$. These conditions can be linearized to

$$d_1 + d_2 = 1,$$

$$d_1 * M \geq bp - x,$$

$$d_2 * M > x - bp,$$

where d_1 and d_2 are binary variables.

The rest of this chapter is organized as follows. Section 2 discusses the background of the DFW Airport deicing and anti-icing activities. Section 3 presents the proposed the ADP framework and algorithms. The DFW Airport deicing ADP application is also presented in this section. Section 4 illustrates various MILP formulations that are included in solving this real-world problem. Section 5 includes case descriptions, implementation strategy, optimal policy discussion, value function approximation accuracy study and corresponding solution quality discussion. In Section 6, a fitted Q-learning method is presented as a benchmark study. Several factors such as state space sampling, optimal value functions and Q functions are also studied in this section.

4.2 DFW Airport Deicing and Anti-icing Activities

The Dallas/Fort Worth (DFW) International Airport experienced an ecological accident in 1999 when a large number of fish died in Trigg Lake, a manmade waterway on the perimeter of the airport. Deicing and anti-icing activities on airplanes caused this incident due to significant amounts of glycol runoff into local waterways. Deicing and anti-icing airplanes prevent buildup of frozen water on airplanes at high altitudes. At an

airport, humidity in the air can condensate on the cold wings and the resulting accumulation of frost or ice and snow on airplanes, can greatly decrease an airplane's aerodynamic performance (FAA Report 1996, Valarezo et al. 1993). It has also been proven by Valarezo et al. (1993) that ice accumulation of 0.005 inches on the leading edge of a wing results in reductions in maximum lift capacity of 20% at the critical span-wise stations on the wing or tail of a 200-seat transport. Furthermore, ice buildup on the fuselage can break off, fall into a rear engine, and damage it. Ice buildup on the rear side of engine fan blades can result in serious damage as the airplane begins its take-off (Kroes et al. 1993). Due to the severity of this problem, the Federal Aviation Administration (FAA) safety regulations require airplanes at DFW Airport to be deiced and anti-iced every winter season to remove ice and snow from control surfaces. This ultimately leads to the fish dying in Trigg Lake in 1999.

Use of glycol-based aircraft deicing and anti-icing fluids (ADAF) has been the worldwide standard for airplane deicing/anti-icing. An airplane is deiced by spraying a mixture of heated deicing fluids and hot water. The deicing fluid contains a minimum of 80% ethylene or propylene glycol by weight and a mixture of water, buffers, wetting agents, and oxidation inhibitors. To prevent further ice formation, anti-icing is applied to aircraft surfaces by spraying anti-icing fluid, which is a mixture that contains ethylene or propylene glycol and thickening agents that enable the fluid to adhere to the aircraft for extended protection and holdover times. Deicing and anti-icing fluids become an environmental problem when they fall or shear off airplanes, collect on the ground, and flow into waterways. The aqueous solubility of these substances can result in high chemical oxygen demand loadings and create serious toxicity problems in receiving waters (Corsi et al., 2006).

The DFW International Airport implemented several measures to avoid recurrence of any more environmental accidents. Eight deicing source isolation pads were constructed where airplanes are deiced and anti-iced to capture ADAF runoff and channel it into a reverse osmosis wastewater treatment system. However, this “drip and shear” runoff may run into local receiving waters, which can weaken water quality and life. To alleviate the “drip and shear” problem in Trigg Lake, 17 aerators were installed to maintain proper dissolved oxygen (DO) levels. To monitor DO levels in waterways surrounding the airport, water quality data was collected at nine sites in collaboration with the United States Geological Survey (USGS).

An ADP method that uses treed regression and MILP is proposed in this study to solve the DFW Airport deicing problem. There is little related research on this ADP application topic in the literature. Z. Yang and V. Chen (2007) developed a Decision-Making Framework (DMF) that searches for dynamic control strategies to reduce ozone pollution in the metropolitan Atlanta region. Data mining methods such as autocorrelation analysis and regression tree have been used for DO analysis at the DFW Airport water quality monitoring sites (H. Fan et al., 2011). The state transition models built (Chen et al. 2011) will be used to track water quality hourly and transition the state of the system. The dynamic program elements of this problem are discussed in Section 4.3.2. Solution of this problem by applying the proposed method is discussed in Section 4.5.

4.3 Proposed ADP Method

4.3.1 Framework and Algorithms

The treed regression based ADP method for finite horizon problem includes two processes, namely: a backward ADP solution process to approximate value functions and a forward simulation process to evaluate value functions approximations. Figures 4-1 and 4-2 illustrate those two processes.

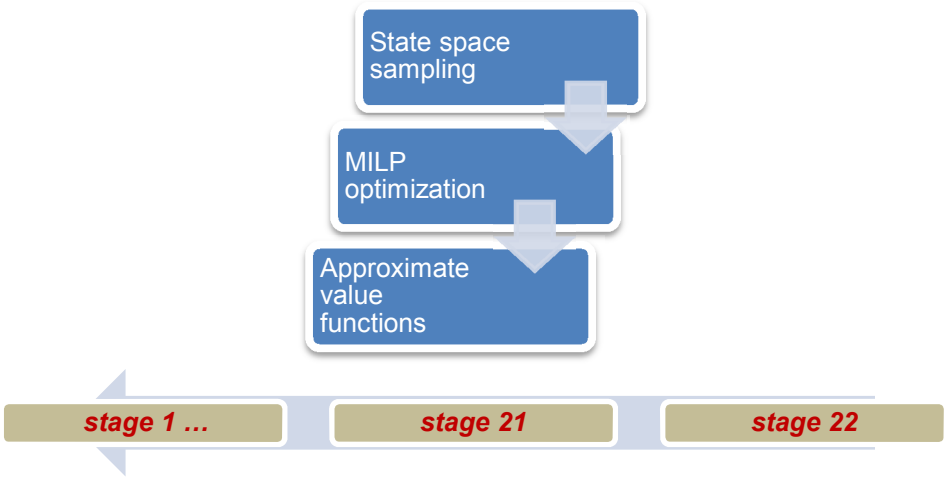


Figure 4-1 SDP Backward ADP Solution Process with MILP in Finite Horizon

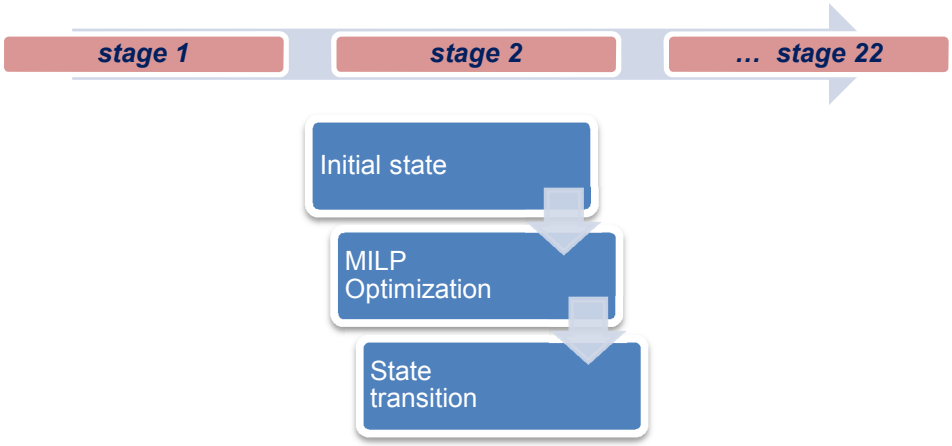


Figure 4-2 SDP Forward Simulation Process with MILP in Finite Horizon

In Figure 4-1, on each stage of the backward ADP solution process, there are three steps namely state space sampling, solving the optimization equation for each sample state, and value function approximation. For state space sampling, the Sobol' sequence (Sobol 1967) is chosen to sequentially sample the state space. The quasi-random sequence has a better space filling property than the ordinary pseudorandom number generation (Morokoff et al. 1994). On each design point, a non-convex optimization problem to solve the optimality equation is formulated as a MILP and solved

with a branch-and-bound algorithm. Then value function is approximated using the two phases' treed regression with AVFA algorithms.

The forward simulation process is to evaluate the value function approximations. For a stochastic system, a large number of random paths are generated given an initial system state. For each initial design point and random path, a decision is made at each stage within a number of scenarios generated for random variables. The objective function is the average of objective functions over the total number of scenarios. Constraints for each scenario are incorporated within the optimization problem. After forward re-optimization on every random path on the initial state, the mean value is used to approximate the actual cumulative cost of the policy generated within this framework.

The framework of the backward ADP solution process and the forward simulation process can be algorithmically presented as follows:

Backward ADP solution process for value function approximation:

1. Using design of experiments, choose/sample N discretization points in the state space, $x_{jt}, j = 1, \dots, N$ for stage $t = 1, \dots, T$.
2. In the last stage T,

2a optimization: for each discretization points x_{jT}, j = 1,…, N, ε_j is random variable, C_t is the contribution function, solve

$$V_T(x_{jT}) = \max_{u_{jT}} \underset{\varepsilon}{E}\{C_T(x_{jT}, u_{jT,}\varepsilon)\}.$$

2b approximation: approximate $V_T(x_{jT})$ with $\hat{V}_T(x_{jT})$, by fitting a statistical model to the data for V_T from step 2a.

3. For stage $t = T - 1, \dots, 1$,

3a for each discretization point $x_{jt}, j = 1, \dots, N$, ε_j is random variable, g_t is the state transition function, solve

$$\tilde{V}_t(x_{jt}) = \max_{u_{jt}} \underset{\varepsilon}{E}\left\{C_t(x_{jt}, u_{jt,}\varepsilon) + \hat{V}_{t+1}\left(g_t(x_{jt}, u_{jt}, \varepsilon)\right)\right\}.$$

3b approximation: approximate $\tilde{V}_t(x_{jt})$ with $\hat{V}_t(x_{jt})$, as step 2b.

Forward simulation:

1. Sample N_s points in the state space of first stage $x_{j0},\ j = 1, \dots, N_s$; generate sample paths for random variables ε_r, $r = 1, \dots, N_r$, N_r is the number of sample paths.
2. For $t = 1, \dots, T$ and for sample path $r = 1, \dots, N_r$

   2a if $t < T$, solve

$$\max_{u_{jt}} \ \underset{\varepsilon}{E}\left\{C_t(x_{jt}, u_{jt,}\varepsilon) + \hat{V}_{t+1}\left(g_t(x_{jt}, u_{jt}, \varepsilon)\right)\right\},$$

   obtain cost/contribution and state transitions

$$C_t(x_{jt},\ u_{jt}, \varepsilon_r),$$

$$x_{j,\,t+1} = g_t(x_{jt}, u_{jt}, \varepsilon_r);$$

   otherwise, solve

$$\max_{u_{jT}} \ \underset{\varepsilon}{E}\{C_T(x_{jT}, u_{jT,}\varepsilon)\}.$$

   2b obtain cumulative cost over stages $\sum_{t=1}^{N} C_{t,r}$

3. For each initial state, average total cost among scenarios

$$\sum_{r=1}^{N_r}\sum_{t=1}^{N} C_{t,r},$$

which is the expected total return for each initial state point.

4.3.2 Dynamic Program for Airport Deicing

After state transition development of the DFW Airport environmental system (Fan 2006, Chen et al. 2011), the initial system is transformed into a Markov decision process. At each stage, state variables are divided into two types. The first type of state variable influences the state of the next stage directly by combining with the effect of the decision made. The second type of state variables influences a state beyond next stage. These are referred to as “carried over” state variables. This set of state variables would be identically brought into the set of state variables in the next stage. The associated transition for this type of state variable is called “identity transition” (Yang et al. 2009). This identity transitions are necessary to maintain the “memory-less” property so that the state in the next stage only depends on the current state and decision.

In this DFW Airport environmental system, the time horizon is limited by 24 hours. Each hour is considered as one stage. Departures are not scheduled before 5am (Chen et al. 2011). State transitions are not needed last two hours of the day. This effectively decreased the time horizon to 18 stages. Table 4.1 describes the dimensions of the ADP framework. The dimension of each stage varies from 67 to 150. Table 4.2 shows the set of state variables at stage 20.

In Table 4.2, there are three columns of variables under both non-carried over and carried over sets. Column DO has state variables that show dissolved oxygen concentrations measured at three water monitoring sites. For example, $O^{(OF)}{}_{(16)}$ represents the dissolved oxygen concentration measured at site OF in stage 16. Column pad glycol represents the glycol usage at any deicing pad location. There are two types of glycol usage such as deicing glycol usage and anti-icing glycol usage. For example, $G^{(D)(SE)}{}_{(17)}$ represents deicing glycol usages on the deicing pad location SE in stage 17 and $G^{(A)(WK)}{}_{(16)}$ represents anti-icing glycol usage on the deicing pad location WK in stage 16. The third column of state variables are runway glycol usage that includes glycol usage of aircraft by using the runway. For example, $G^{(D)(36R)}{}_{(17)}$ represents deicing glycol usage on aircraft that will take off on runway 36R.

In a real case study, the dimensions of each state would be decreased because of the runway selection. Wind direction determines the runway selections. It is a fact that the airplane should take off in the direction opposite to the wind direction (Chen et al. 2011). Since DFW Airport is located in a north-south direction, only the north-to-south and the south-to-north wind directions are considered while assigning a runway. There are 8 deicing pad locations and 16 runways (including both directions). If the wind direction is from north to south on one day, only 8 out of 16 runways would be selected. For example, on a Jan 31st of 2010, the wind direction is from south to north. Only the

variables related to runway 17R and 18L are considered useful in the columns of runway glycol. Other variables related to runway 36R, 35L, 31L and 35C are zero in this case.

Table 4-1 State Dimension of DFW Airport Deicing ADP Framework

stage	0	1	2	3	4	5	6	7	8	9	10	11
dimension	106	108	109	111	115	119	123	128	134	139	144	150
stage	12	13	14	15	16	17	18	19	20	21	22	23
dimension	143	141	138	137	137	137	124	117	110	98	84	67

Table 4-2 State Variables of Stage 20 of DFW Airport Deicing ADP Framework

stage	non-carried over			carried over		
	DO	pad	runway glycol	DO	pad	runway glycol
20	$O^{(OF)}_{(16)}$ $O^{(IN)}_{(16)}$ $O^{(IN)}_{(19)}$ $O^{(OUT)}_{(14)}$ $O^{(OUT)}_{(18)}$	$G^{(D)(SE)}_{(8,13)}$ $G^{(D)(SE)}_{(17)}$ $G^{(D)(SW)}_{(14)}$ $G^{(D)(NE)}_{(14)}$ $G^{(A)(SW)}_{(16)}$ $G^{(A)(WK)}_{(16)}$	$G^{(D)(36R)}_{(17)}$ $G^{(D)(36R)}_{(19)}$ $G^{(D)(17R)}_{(18)}$ $G^{(D)(35C)}_{(-4,7)}$ $G^{(D)(18L)}_{(-4,7)}$ $G^{(A)(35L)}_{(8,13)}$ $G^{(A)(36R)}_{(8,13)}$ $G^{(A)(31L)}_{(-4,7)}$	$O^{(OF)}_{(17,19)}$ $O^{(IN)}_{(17,18)}$ $O^{(OUT)}_{(15,17)}$ $O^{(OUT)}_{(19)}$	$G^{(D)(SE)}_{(14,16)}$ $G^{(D)(SE)}_{(18,19)}$ $G^{(D)(SW)}_{(15,17)}$ $G^{(D)(NE)}_{(15,17)}$ $G^{(A)(SW)}_{(17,19)}$ $G^{(A)(WK)}_{(17,19)}$	$G^{(D)(36R)}_{(18)}$ $G^{(D)(17R)}_{(19)}$ $G^{(D)(35C)}_{(8,10)}$ $G^{(D)(18L)}_{(8,10)}$ $G^{(A)(35L)}_{(14,16)}$ $G^{(A)(36R)}_{(14,16)}$ $G^{(A)(31L)}_{(8,10)}$

Table 4-3 Deicing Variables of Stage 0 to 5 of DFW Airport Deicing ADP Framework

stage	runway	carried over							
0	$G^{(A)(18L)}_{(0)}$	$N^{(NB)(HY)}_{(0)}$	$G^{(D)(EK)}_{(0)}$	$G^{(D)(SE)}_{(0)}$	$G^{(D)(35C)}_{(0)}$	$G^{(D)(18L)}_{(0)}$	$G^{(A)(35C)}_{(0)}$	$G^{(A)(31L)}_{(0)}$	
1	$G^{(A)(18L)}_{(1)}$	$G^{(D)(EK)}_{(1)}$	$G^{(D)(SE)}_{(1)}$	$G^{(D)(35C)}_{(1)}$	$G^{(D)(18L)}_{(1)}$	$G^{(A)(35C)}_{(1)}$	$G^{(A)(31L)}_{(1)}$		
2	$G^{(A)(18L)}_{(2)}$	$G^{(D)(EK)}_{(2)}$	$G^{(D)(SE)}_{(2)}$	$G^{(D)(35C)}_{(2)}$	$G^{(D)(18L)}_{(2)}$	$G^{(A)(35C)}_{(2)}$	$G^{(A)(31L)}_{(2)}$		
3	$G^{(A)(18L)}_{(3)}$	$G^{(A)(SW)}_{(3)}$	$G^{(D)(EK)}_{(3)}$	$G^{(D)(SE)}_{(3)}$	$G^{(D)(35C)}_{(3)}$	$G^{(D)(18L)}_{(3)}$	$G^{(A)(35C)}_{(3)}$	$G^{(A)(31L)}_{(3)}$	$G^{(A)(17C)}_{(3)}$
4	$G^{(A)(18L)}_{(4)}$	$G^{(A)(SW)}_{(4)}$	$G^{(D)(EK)}_{(4)}$	$G^{(D)(SE)}_{(4)}$	$G^{(D)(35C)}_{(4)}$	$G^{(D)(18L)}_{(4)}$	$G^{(A)(35C)}_{(4)}$	$G^{(A)(31L)}_{(4)}$	$G^{(A)(17C)}_{(4)}$
5	$G^{(A)(18L)}_{(5)}$	$G^{(A)(SW)}_{(5)}$	$G^{(D)(EK)}_{(5)}$	$G^{(D)(SE)}_{(5)}$	$G^{(D)(35C)}_{(5)}$	$G^{(D)(18L)}_{(5)}$	$G^{(A)(31L)}_{(5)}$	$G^{(A)(17C)}_{(5)}$	

Table 4-4 Definition of Deicing Pad Capacity

1. Taxiway C - up to 3 any type aircraft at one time.
2. Taxiway HY - 2 NBs or 1 WB or 3 RJs.
3. Taxiway WK - 2 RJs & 1 WB.
4. SW Hold Pad - 8 NBs and 2 WBs.
5. NE Hold Pad - 1 WB aircraft, 3 RJs and 1 NB can be deiced at one time.
6. Taxiway EK - 2 NBs or 1 WB.
7. SE Hold Pad - 7 NBs and 2 WBs.

The decision variables of this dynamic program are the assignments of aircraft in each time slot to specific deicing pad locations. The decision variables are linked to state transitions using a set of variables called deicing variables. In addition, there are also two sets of deicing variables: carried over and non-carried over. Table 4.3 shows the deicing variables of stages 0 through stage 5. The deicing variables measure the deicing and anti-icing glycol usage in the current stage and the state variables are transitioned from past stages. The deicing variables are directly related to the aircraft assignment decision in the current stage. They also interact with the state variables and the random variables to determine the outcome of the current stage. Via the state transition equations, those variables affect the future state of the dynamic system. The notation of deicing variables is same as that of the state variables. Decision variables have to satisfy the capacity constraints of the deicing pad locations. The capacities are described in Table 4.4.

In this DFW Airport environmental system, there are several groups of random variables. Binary variables of whether to conduct anti-icing, glycol usage per airplane and runway selections are considered as random variables in the ADP model. The distributions are estimated from historical data. Within this non-stationary finite horizon system, state transitions have been developed from historical data (Chen et al. 2011).

4.4 MILP Formulations

In this section, two types of MILP formulations are presented that represent most of the optimality equations over the stages. The first MILP formulation is the optimality equation on the last stage. The value function is not involved in this formulation. The second MILP formulation is the optimality equation of the stage before last stage. Similar to the optimality equation of previous stage, value function approximations are involved as part of objective function.

4.4.1 MILP Formulation on Last Stage

On a “medium” deicing activity day, for example February 1st, 2010, the wind direction was from south to north at the beginning of the day. Airplanes would only depart from Runways 13R, 18R, 18L, 17R, 17C or 17L. Given the number of airplanes of each size, the objective is to find a feasible assignment that minimizes the cost of low dissolved oxygen at three water monitoring sites (OUT, IN and OF19) plus the cost of the deicing time for more than one hour. There are two levels’ of penalties, 100 and 1, for partial amounts of DO which are less than 3 mg/l and 7 mg/l, respectively. The penalty coefficient for a partial deicing time that is greater than 60 minutes for each stage is 5. The deicing pad location assignment problem is subject to following constraints:

- Capacity: At each deicing pad location, all possible combinations of airplanes are considered within the capacity of that location.
- Cover: Each airplane has to be deiced at one and only one deicing pad location.
- State Transition: State variables on the 23rd stage such as the DO amount at each site in the 22nd stage are either calculated by the state transition functions or directly carried over by identity transitions.

The following assumptions are taken to model this problem: The time for deicing and anti-icing is deterministic and the same for each airplane. It is assumed to be 13.5 minutes in this case. Assuming the deicing time for any airplane is one unit, the deicing

time of a deicing pad location on one stage is generated from the total frequencies of all possible combinations. Table 4-5 is an example of all possible airplane combinations in one unit deicing time at HY. For example, if in stage 23, the deicing pad location HY deices airplanes in the 6th combination twice and the 4th combination once, then the deicing time at HY is 3*13.5=40.5 minutes. Glycol usage amounts of deicing and anti-icing are ordered by pilots. They are modeled as random variables. This variable of each airplane on each deicing pad location follows a Lognormal or a Weibull distribution, in which the parameters are estimated from historical data (Chen et al. 2011).

Table 4-5 All Airplane Combinations in a Deicing Time Unit at Location HY

airplanes combination	1	2	3	4	5	6	7
WB	1	0	0	0	0	0	0
NB	0	2	1	1	0	0	0
RJ	0	0	0	1	3	2	1

The decision variables in this problem can be separated into two groups: u_{ijt}^{m} and r_{ij}^{m}.

$$u_{ijt}^{m} = \begin{cases} 1, & \text{if the airplane } i \text{ of the type } j \text{ is assigned to be } \text{deicing pad location } t \text{ in stage } m \\ 0, & \text{otherwise} \end{cases}$$

where $m = 22, 23$, j=1,2,3, t=1,2,…,7, $i = 1,2, \dots, N_j^m$. N_j^m is the number of airplane of type *j* on stage *m*. Let r_{ij}^{m} be the frequency of combination *i* at deicing pad location *j* on stage *m*.

In the set of parameters of the model, a represents the penalty coefficients for dissolvable oxygen less than 7 (a_1 =1) and less than 3 (a_2 =100). Let b be penalty coefficients for deicing time more than one hour at each deicing pad location. Let $pen\_DO_{ijk}^{m}$ represent the partial amount of DO less than 7 (*j*=1) or partial amount of DO less than 3 (*j*=2) at monitoring site i in stage m with the k^{th} realization of the random glycol amount variables and the anti-icing actions. Let DO_{ik}^{m} be the amount of DO at monitoring site *i* in stage m with the k^{th} realization of the random glycol amount variables and anti-

icing action. Let $pen\_T_i^m$ be the amount of deicing time more than one hour at deicing pad location *i* in the stage *m* and t_i^m be the total amount of deicing time of the deicing pad location *i* in stage m. Let X^m be the set of state variables in stage *m* and. Let f_j be the j^{th} state transition function from the 22nd hour to the 23rd hour. Let s_k^m be the binary variable indicating whether $RW18L\_A_k^m$ is less than or equal to 10 ($s_k^m = 1$), or greater than 10 ($s_k^m = 0$). The total anti-icing glycol amount used on aircraft that choose runway 18L on the stage m is $RW18L\_A_k^m$. It is calculated by the k^{th} realization of the random glycol amount variables and the anti-icing action. The possibility of type airplane j choosing runway 18L from deicing pad location pad *t* is p_{jt}^{18L}. Let ua_{ijtk}^m be the binary random parameter that represents the k^{th} sample value of the anti-icing action of the airplane i of type j at location t on the stage m. The parameter ga_{ijtk}^m is the glycol amount that represents the k^{th} sample value of the anti-icing glycol used by the airplane i of type j at location t on the stage m. Number of type airplane j in the v^{th} combination at deicing pad location *t* is $\mathrm{na}_{\mathrm{tvj}}^{\mathrm{m}}$. Let m_j be the number of combinations at deicing pad locations j. The number of airplanes of type j is n_j. Expected total deicing and anti-icing time for each airplane is T. Given the parameters, the optimality equation can be formulated as:

$$\text{Minimize } g = \frac{1}{10}\sum_{i=1}^{3}\sum_{j=1}^{2}\sum_{k=1}^{10} a_j\, pen\_DO_{ijk}^{23} + \frac{1}{10}\sum_{i=1}^{3}\sum_{j=1}^{2}\sum_{k=1}^{10} a_j\, pen\_DO_{ijk}^{22} + b\sum_{i=1}^{7} pen\_T_i^{23} + b\sum_{i=1}^{7} pen\_T_i^{22} \quad (1)$$

Subject to

$$pen\_DO_{i1k}^m \geq 7 - DO_{ik}^m \qquad i = 1,2,3,\ k = 1,....,10,\ m = 22,23 \quad (2)$$

$$pen\_DO_{i2k}^m \geq 3 - DO_{ik}^m \qquad i = 1,2,3,\ k = 1,....,10,\ m = 22,23 \quad (3)$$

$$pen\_T_i^m \geq t_i^m - 60 \qquad i = 1,...,7,\ m = 22,23 \quad (4)$$

$$t_i^m = \sum_{j=1}^{m_i} r_{ij}^m \cdot T \qquad i = 1,...,7,\ m = 22,23 \quad (5)$$

$$DO_{1k}^m = f_1\left(X^m\right) \qquad k = 1,....,10,\ m = 22,23 \quad (6)$$

$$DO_{2k}^m = f_2\left(X^m\right) \qquad k = 1,....,10,\ m = 22,23 \quad (7)$$

$$DO_{3k}^m = s_k^m f_3\left(X^m\right) + \left(1 - s_k^m\right) f_4\left(X^m\right) \qquad k = 1,....,10,\ m = 22,23 \quad (8)$$

$$RW18L\_A_k^m \le 10 + s_k^m \cdot M \qquad k = 1,....,10,\ m = 22,23 \quad (9)$$

$$RW18L\_A_k^m > 10 + (s_k^m - 1) \cdot M \qquad k = 1,....,10,\ m = 22,23 \quad (10)$$

$$RW18L\_A_k^m = \sum_{t=1}^{7} \sum_{j}^{3} \sum_{i}^{n_j} u_{ijt}^m p_{jt}^{18L} ua_{ijtk}^m ga_{ijtk}^m \qquad k = 1,....,10,\ m = 22,23 \quad (11)$$

$$\sum_{t=1}^{7} u_{ijt}^m = 1 \qquad j = 1,2,3,\ i = 1,...,n_j,\ m = 22,23 \quad (12)$$

$$\sum_{i=1}^{n_j} u_{ijt}^m = \sum_{v=1}^{m_t} r_{tv}^m na_{tvj} \qquad j = 1,2,3,\ t = 1,...,7,\ m = 22,23 \quad (13)$$

$$X^{23} = F\left(X^{22}\right) \quad (14)$$

$$pen\_DO_{ijk}^m \ge 0 \qquad i = 1,2,3,\ k = 1,....,10,\ m = 22,23,\ j = 1,2 \quad (15)$$

$$pen\_T_i^m \ge 0 \qquad i = 1,...,7,\ m = 22,23 \quad (16)$$

$$RW18L\_A_k^m \ge 0 \qquad k = 1,....,10,\ m = 22,23 \quad (17)$$

$$u_{ijt}^m \in \{0,1\} \qquad j = 1,2,3,\ i = 1,...,n_j,\ t = 1,...,7,\ m = 22,23 \quad (18)$$

$$s_k^m \in \{0,1\} \qquad t = 1,...,7,\ m = 22,23 \quad (19)$$

$$r_{tv}^m \in \mathrm{Z}_+ \qquad t = 1,...,7,\ v = 1,...,m_t,\ m = 22,23 \quad (20)$$

The objective (1) is to minimize expected penalty from low DO and deicing time over one hour. Constraints (2), (3), and (15) are equivalent to truncated functions:

$$pen\_DO_{i1k}^m = \begin{cases} 7 - DO_{ik}^m, & if \quad DO_{ik}^m < 7 \\ 0, & otherwise \end{cases} \qquad i = 1,2,3,\ k = 1,...,10,\ m = 22,23 \quad (21)$$

Constraints (4) and (16) are equivalent to truncated functions:

$$pen\_T_i^m = \begin{cases} t_i^m - 60, & if \;\; t_i^m > 60 \\ 0, & otherwise \end{cases} \qquad i = 1,2,3\,,\; m = 22,23 \quad (22)$$

Constraint (5) calculates the deicing time needed at each deicing pad location. DO amounts at the three monitoring sites are modeled in state transition functions realized in constraints (6), (7), and (8). Constraints (8), (9), and (10) are state transition functions at the OUT site, which have "if then" constraints. Decision variables are involved in bifurcating the tree regression models, which are shown in the constraints set (11). They are equivalent to:

$$DO_{3k}^m = \begin{cases} f_3\left(X^m\right), \; if \;\; \sum_{t=1}^{7}\sum_{j}^{3}\sum_{i}^{n_j} u_{ijt}^m p_{jt}^{18L} ua_{ijtk}^m ga_{ijtk}^m \le 10 \\ f_4\left(X^m\right), \; otherwise \end{cases} \qquad k = 1,\ldots,10\,,\; m = 22,23 \quad (23)$$

Covering constraint set (12) ensures that each airplane is assigned to only one deicing pad location, and constraint set (13) combines the assignment of each airplane with the frequency of airplane combinations at each deicing pad location.

4.4.2 MILP Formulations in Stage 21

In stage 21, optimality equation includes value function approximation as part of the objective function. The treed regression based piecewise linear value function approximation is transformed into MILP by adding binary variables. These binary represents branching and node selections. Since variables and parameters are already defined for the optimality equation without a value function approximation in Section 4.4.1, additional variables and parameters include: FV_k^{21} is the future value at stage 21 from scenario *k* in stage 21. X^{21} and $X^{22}$are the state variables in stage 21 and 22. ua_{ijtk}^{21} is the binary value that represents the kth sampled value of the anti-icing action of the airplane i of type j at location t in the stage 21. ga_{ijtk}^{21} is the glycol amount that is the kth

sampled value of the anti-icing glycol used by the airplane i of type j at location t on the stage 21. Let gd_{ijtk}^{21} be the glycol amount of scenario k by airplane i of type j at the location t on stage 21. Let $TNode_{kq}^{21}$ be the value of linear regression function at terminal q in scenario k. Let $Term_{kq}^{21}$ be the binary variable that represents whether a value function is assigned to a terminal node or not. NTD^{21} is the number of terminal nodes in the value function approximation.

$$\text{minimize} \quad g' = g + \frac{1}{10}\sum_{k=1}^{10} FV_k^{21} \tag{24}$$

subject to

$$X_k^{22} = F\left(X^{21}, u_{ijt}^{21}, ua_{ijtk}^{21}, ga_{ijtk}^{21}, gd_{ijtk}^{21}\right) \quad k = 1,....,10\,,\ j = 1,2,3\,,\ i = 1,2,3\,,\ t = 1,...,7 \tag{25}$$

$$FV_k^{21} = \sum_{q=1}^{NTD} TNode_{kq}^{21} \qquad k = 1,....,10 \tag{26}$$

$$\sum_{q=1}^{NTD} term_{kq}^{21} = 1 \qquad k = 1,....,10 \tag{27}$$

$$term_{kq}^{21} \in \{0,1\} \qquad k = 1,....,10\,,\ q = 1,....,NTD \tag{28}$$

where g is the objective function in (1). The constraints also need include constraints of the optimality equation without value function approximations in Section 4.4.1.

In stage 21, the objective function includes two parts, the contribution function and the value function. The first part is essentially the same with the objective function of the last stage. The second part of the objective function is the average of the value function approximations over sampled scenarios. Constraints from (2) to (20) from the last stage are given to this optimization model as a subset of constraints. Constraints (23) are considered as state values of the next stage that are transitioned from (21). Decision variables and random variables (glycol usages) are involved in these sets of constraints. Constraints (24), (25), and (26) represent the future values by the terminal nodes of the

tree model. Let $term_{kq}^{21}$ be a binary variable that represents whether a value function is assigned to the terminal node q or not. The summation of those binary variables for each scenario is one because only one terminal node would be realized in each scenario. Moreover, another set of constraints is necessary to determine the relation between those binary variables and branching rules. Similar to the case described in Section 4.1, the realization of terminal node t is determined by branching rules, form example $x_1 > c_1$, $x_2 < c_2$, and $x_3 = c_3$. Constraints that connect the branching rules and variable term should be

$$x_1 - c_1 > M * (term - 1),$$
$$x_2 - c_2 < M * (1 - term),$$
$$x_3 - c_3 < M * (1 - term),$$
$$x_3 - c_3 > M * (term - 1),$$

where *M* is a big number and the *term* is a binary variable indicating selection of the selection of the terminal node.

4.5 Solution Method

We choose a "medium" deicing activity day, February 1st, 2010, for this case study. Wind direction was from south to north at the beginning of the day. Airplanes would only depart from runways 13R, 18R, 18L, 17R, 17C or 17L. Table 4.7 shows the three types of aircraft to be deiced. Table 4.6 shows the number of patterns of choice for each time slot per deicing pad location. These patterns are extracted from capacity information of each location.

Table 4-6 Patterns of Aircraft Combinations for Each Time Slot within Capacity

Location	TxC	TxHY	TxWK	SW	NE	TxEK	SE
Number of Patterns	19	7	9	165	23	6	135

Table 4-7 Number of Aircraft Deiced on Feb 1st, 2010

	Wide Body	Narrow Body	Regional Jet
5am-6am	0	10	6
6am-7am	1	16	13
7am-8am	1	20	11
8am-9am	0	18	5
9am-10am	4	0	15
10am-11am	11	1	14
11am-12pm	7	1	15
12pm-1pm	3	21	4
1pm-2pm	13	4	10
2pm-3pm	3	11	8
3pm-4pm	2	24	15
4pm-5pm	1	23	10
5pm-6pm	0	27	9
6pm-7pm	1	16	9
7pm-8pm	3	14	9
8pm-9pm	0	16	1
9pm-10pm	1	8	2
10pm-11pm	1	5	0
11pm-12pm	0	1	3

4.5.1 Implementation Strategy

As a continuous-state space DP problem, the DFW Airport deicing problem uses the Sobol' sequence for state space sampling. The limits of the state space region are estimated from historical data by using the maximum and minimum value (Chen et al. 2011). When solving the optimality equations, ten scenarios are generated based on the distribution that is estimated from historical data. An IBM CPLEX 12.5 mixed integer programing solver is used to solve each optimality equation per design point. Considering the number of design points and number of stages, a computational time limit of 3600 seconds is set for every run of the branch-and-bound algorithm. The computational time required for each state is usually less than five seconds. For some extreme cases, it

required more than 1200 seconds to find the global optimum. In the objective function of the optimality equation, a piecewise linear penalty is applied to the low dissolved oxygen in the receiving waters. There are two types of penalty functions in the sensitivity analysis. The first type has 100 penalty coefficients for DO under 3mg/L and 1 penalty coefficient for DO under 7 mg/L. The second type of penalty function is more restrictive than the first type; it has 100 penalty coefficients for DO under 7 mg/L and 1 penalty coefficient for DO under 10mg/L.

An adaptive value function approximation (AVFA, Fan et al. 2013) algorithm is implemented in ADP method for this problem. Similar to the inventory forecasting problem in Section 3.1.1, 200 initial design points are used, and 50 design points are added in each iteration. Sequential state space sampling and value function approximation are conducted iteratively untill the stopping criteria of approximation accuracy is satisfied. The approximation accuracy is measured in terms of mean square error from an independent test data set. To approximate the value function, different settings are studied to determine the complexity of the tree. In this case study, there is no post-pruning. Pre-pruning is conducted by determining the minimum number of observations assigned to each terminal leaf. Considering the number of state variables, there are sets of experiments with 20, 50 and 100 minimum observations on a terminal leaf. In the forward simulation, the same ten scenarios are used for each random variable to make decisions. For policy evaluation, 100 initial state and 100 sample paths are used. These sample paths are dynamically generated to prove the robustness of the optimal policy.

4.5.2 Solution Discussion

Tables 4-8 and 4-9 show a forward simulation solution of for initial state. In this solution, the deicing NE location is heavily used from 7am to 12pm. It demonstrates that

NE has a greater influence on reducing the environment impact than the other deicing pad locations under this circumstance. From 6am to 7am, aircraft are almost evenly assigned to the seven locations. During 12pm to 6pm, TxHY and TxWK are used less often than the other deicing pad locations. None of the wide body aircraft are assigned to these two locations. This most likely occurs because they have a much smaller capacity than the other deicing pad locations. The deicing pad locations SW and SE have much higher capacities. However, their usages are just slightly higher than other locations. This might be because the aircraft assigned to those two locations have a slightly stronger impact on the environment.

Table 4-11 show the usage efficiency of each deicing pad location. It is calculated as the number of assigned aircraft divided by maximum capacity. Between 6pm and 12am, deicing pad locations TxC, TxHY, TxWK and TxEK are used more heavily than the other three deicing pad locations. In the last three hours, very few aircraft are assigned to SW, NE and SE, even though the capacity of SW and SE are significantly greater than the other locations. This is most likely due to the fact that any deicing activity at pad SW, NE and SE have a greater environmental impact than the other four locations. Overall, some shift of assignment is observed over stages. These shifts are probably due to the current and future environmental impacts that have a different balance over the hours. Table 4.10 shows the contribution by the optimal policy for each stage. In this case study, most of the contribution comes from the first six hours.

Table 4-8 Solution by ADP with MILP of an Initial State (per Aircraft Type per Hour)

	TxC			TxHY			TxWK			SW			NE			TxEK			SE		
	WB	NB	RJ	WB	NB	RJ	WB	NB	RJ	WB	NB	RJ	WB	NB	RJ	WB	NB	RJ	WB	NB	RJ
5am-6am		4	1									5		5			1				
6am-7am		2	3		1	3		4		1	4	1		4	2			4		1	
7am-8am																			1	20	11
8am-9am																				18	5
9am-10am																			4		15
10am-11am													3						8	1	14
11am-12pm																			7	1	15
12pm-1pm		3	2		4			3	1		2		2	2	1		6		1	1	
1pm-2pm			5		2	3	4						4			3	2		2		2
2pm-3pm		1			4				4	1					1	1	2	1	1	4	2
3pm-4pm		3	4		4			2	2		6	4	2	1	1		2	2		6	2
4pm-5pm		2	2		2			4			3	5	1	3			6	2		3	1
5pm-6pm		9	1		2	2		1	4		3	1		1	1		4			7	
6pm-7pm		3	1			5		2		1		1		4			3	2		4	
7pm-8pm		2	2	2	2			4				1		3	5	1	2	1		1	
8pm-9pm		5			3	1		4						1			1			2	
9pm-10pm		2	2	1	3			1			2										
10pm-11pm				1				3			1						1				
11pm-12pm									1								1	1			1

Table 4-9 Aircraft Assignment by ADP with MILP per Hour per Deicing Pad Location

	TxC	TxHY	TxWK	SW	NE	TxEK	SE
5am-6am	5			5	5	1	
6am-7am	5	4	4	6	6	4	1
7am-8am							32
8am-9am							23
9am-10am							19
10am-11am					3		23
11am-12pm							23
12pm-1pm	5	4	4	2	5	6	2
1pm-2pm	5	5	4		4	5	4
2pm-3pm	1	4	4	1	1	4	7
3pm-4pm	7	4	4	10	4	4	8
4pm-5pm	4	2	4	8	4	8	4
5pm-6pm	10	4	5	4	2	4	7
6pm-7pm	4	5	2	2	4	5	4
7pm-8pm	4	4	4	1	8	4	1
8pm-9pm	5	4	4		1	1	2
9pm-10pm	4	4	1	2			
10pm-11pm		1	3	1		1	
11pm-12m			1			2	1

Table 4-10 Contribution of Each Hour of an Initial State

Start Time	Hourly Contribution	Start Time	Hourly Contribution	Start Time	Hourly Contribution
5am	63.07	12pm	0.00	7pm	0.00
6am	57.87	1pm	0.00	8pm	0.00
7am	68.80	2pm	0.00	9pm	0.00
8am	82.08	3pm	0.00	10pm	0.00
9am	3.61	4pm	0.00	11pm	0.00
10am	3.66	5pm	0.00		
11am	0.00	6pm	0.00		

Table 4-11 Average Deicing Pad Locations Using Percentage (Usage/Capacity)

	TxC	TxHY	TxWK	SW	NE	TxEK	SE
5am-6am	0.06	0	0	0.01	0.02	0.03	0
6am-12pm	0.09	0.07	0.07	0.03	0.1	0.11	0.75
12pm-6pm	0.59	0.43	0.46	0.14	0.22	0.86	0.2
6pm-12am	0.31	0.33	0.28	0.03	0.14	0.36	0.05

4.5.4 Sensitivity Analysis

Two sensitivity analyses are conducted with this ADP problem. In the first analysis, two types of penalty functions are applied. The stricter penalty function has coefficients 100 for $DO < 10$ and 1 for $DO < 7$. The other penalty function has penalty coefficients 100 on $DO < 7$ and 1 on $DO < 3$. The second sensitivity analysis studied different numbers of randomness realizations when approximating the expectation.

4.5.4.1 Penalty Functions

We studied four groups of ADP policies using different value function approximations and different penalty functions. Group 1 has 50 minimum observations on the terminal leaf and penalty functions with 7 and 3 break points. Group 2 has 100 minimum observations on the terminal leaf and penalty functions with 7 and 3 break points. Group 3 has 50 minimum observations on the terminal leaf and penalty functions with 15 and 7 break points. Group 4 has 100 minimum observations on the terminal leaf and penalty functions with 15 and 7 break points. In groups 1 and 3, the value function approximations are expected to generate a larger tree than group 2 and group 4 based on the parameters. In groups 1 and 3, the tree structured piecewise linear functions would have more breaking points. It would require more computational effort when solving MILP. Since the system evolves over time and DO amount is affected by the state of previous stages, it is also expected that stricter penalty functions would yield a higher dependency among stages.

Figure 4-3 shows the simulation results of four groups of experiments from left to right. Each boxplot represents absolute errors for 100 initial states. The *absolute error* is defined as the difference between the current solution and the best known solution. Groups 1 and 2 use the same type of penalty function. Group 1 has relatively larger tree model than group 2. The cost boxplot in Figure 4-3 shows that group 1 generates a better optimal policy than group 2 in terms of mean and variance of expected costs. This also applies for groups 3 and 4 with the same penalty function. The difference between groups 3 and 4 is the tree models for value function approximations. Group 3 generates an optimal policy with lower cost compared to group 4 because group 3 has a larger tree structured value function approximation. Besides value function complexity, penalty functions also affect the solutions. The experimental results show that group 1 has a lower mean cost and more variability than group 3. Group 2 has a lower mean cost and more variability compared to group 4.

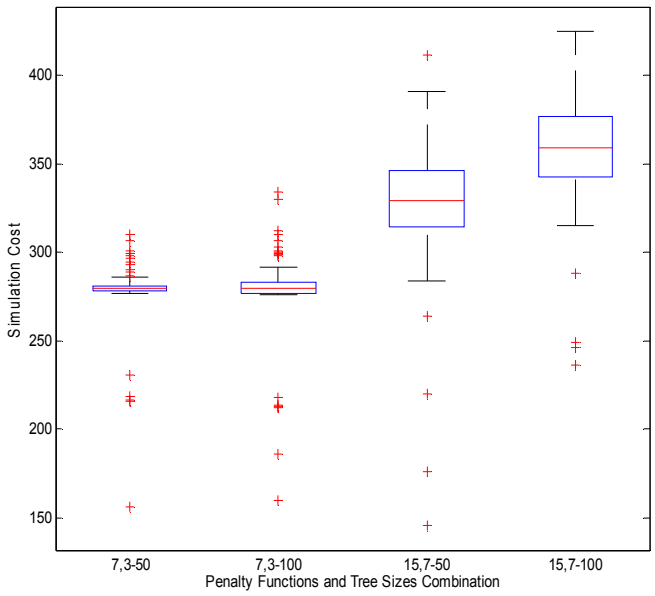


Figure 4-3 Simulation Results by Different Penalty Functions and Tree Sizes

Tables 4.12 and 4.13 show the simulation results generated by the stricter penalty function from the same initial state used to the generate the optimal policy in Section 4.4.2. From the comparison of deicing pad location usage efficiency (actual usage divided by capacity), some obvious shift of aircraft assignments can be observed. In the two time sections from start to 12pm, deicing pad locations NE and TxEK are used more often by the ADP policy using the stricter penalty function. Location SE is used less often before 12pm. Under the penalty with the smaller break value, the assignments made to the deicing pad locations NE and TxEK keep the DO amount of receiving waters at a reasonable level. However, the same assignment would not be able to keep the DO amount under more restrictive constraints (stricter penalty function).

Table 4-10 Solution of ADP Stricter Penalty Function (Compared with Table 4.9)

	TxC	TxHY	TxWK	SW	NE	TxEK	SE
5am	4				7	5	
6am	3	3	4	2	8	8	1
7am	11	1	2	1	8	8	1
8am	2	4	2	4	5	6	
9am				3	10	4	2
10am				6	16	4	0
11am	2				16	2	3
12pm	4	4	4	4	4	5	3
1pm	5	8	4	2	3	3	2
2pm	4	1	3	6		4	4
3pm	9	4	5	12	3	4	4
4pm	4	2	4	8	4	8	4
5pm	5	2	1	19	4	1	2
6pm	2	4	4	3	7	4	2
7pm	2	2	2	3	11	3	3
8pm	4	3	4		3	1	2
9pm	4	4	1	2			
10pm		1	3	1		1	
11pm		1	1	2			

Table 4-11 Approximate Average Hourly Usage Efficiency of Deicing Pad Locations by Stricter Penalty Functions (Compared with Table 4.10)

	TxC	TxHY	TxWK	SW	NE	TxEK	SE
5am-6am	0.44	0	0	0	0.47	0.83	0
6am-12pm	0.33	0.15	0.15	0.09	0.7	0.89	0.04
12pm-6pm	0.57	0.39	0.39	0.28	0.2	0.69	0.12
6pm-12pm	0.22	0.28	0.28	0.06	0.23	0.25	0.04

4.5.4.2 Study of Stochastic Optimization

When solving the optimality equations, scenarios are sampled from the estimated distributions to approximate the expectation of objective functions. A number of scenarios might affect the solution quality and approximation quality (Dupačová et al. 2003). The effect of the number of scenarios is studied in this section. In this study, if more than 15 scenarios are generated to solve each optimality equation, the average computational time for each optimization would be greater than 15 seconds. This would bring up the total computational time to 30 hours for the backward ADP solution process only. It would also increase the computational time of the forward simulation to 41.7 hours approximately. The total of 71.7 hours is much longer than the DP time horizon of 18 hours. In this study, the experiments are performed in three groups, namely 1, 5 and 10 sampled scenarios for decision making. The scenarios are sampled before the experiments. These scenarios are used in both forward simulation and backward ADP solution processes. Figure 4-4 shows the simulation cost of the three groups. The absolute error used in the boxplot is defined in Section 3.1. Group 3 possessing 10 scenarios for making decisions generates an optimal policy that has lower total cost than the other two groups. In terms of variance, group 3 does well since the box is flatter than the other two groups whereas group 2 generates a lower cost by 5 scenarios than the first group. Group 1 is essentially a deterministic problem and it generates the optimal policy with highest costs among the three groups.

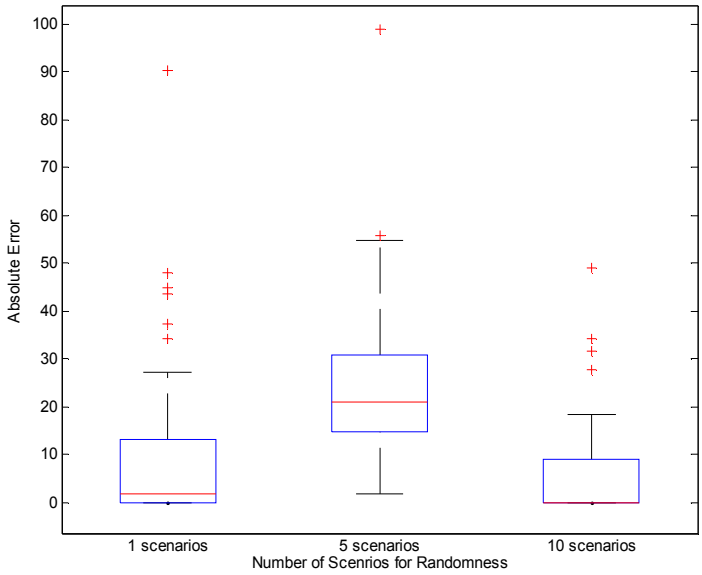
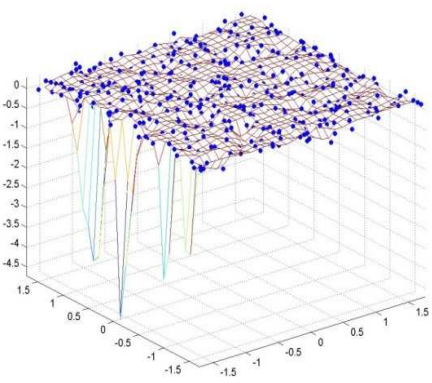


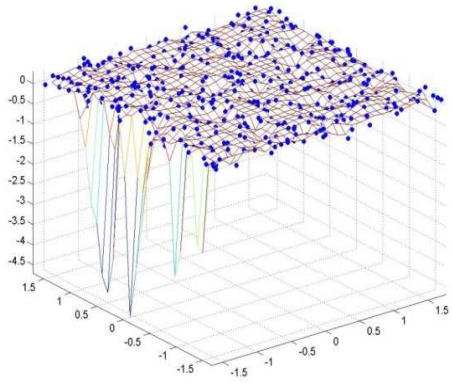
Figure 4-4 Simulation Costs Compared by Three Groups of Randomness Realizations

4.5.4.3 Surface Plots of Value Functions

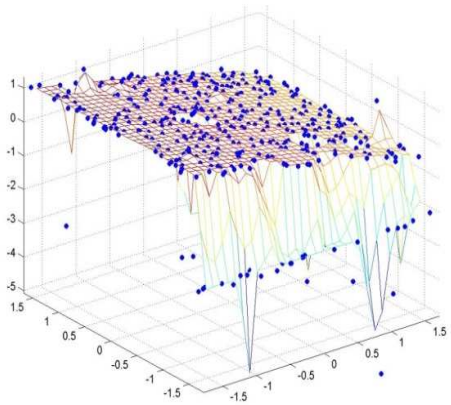
Figures 4-5 to 4-7 are surface plots of value functions in stage 6, stage 12 and stage 18. They are generated in this way. The two most significant state variables that contribute to value function are selected by p-values. Then the future values are plotted over these two state variables. Many spikes observed on these plots represent the smoothness of the surface of the value functions. A surface plot of stage 18 has more spikes than the plots of stage 6 and stage 12. It can be observed that the penalty function has an impact on the surface plot of value function. In stage 12, more spikes are observed on the surface plots with the breaking points of 15 and 7. Many spikes can be observed on groups 3 and 4 in stage 18. It can also be observed that the tree's complexity in the value function approximation also impact the shape of those surface plots. In stage 18, value function approximations with a larger tree create more spikes in the shape of the surface plot.



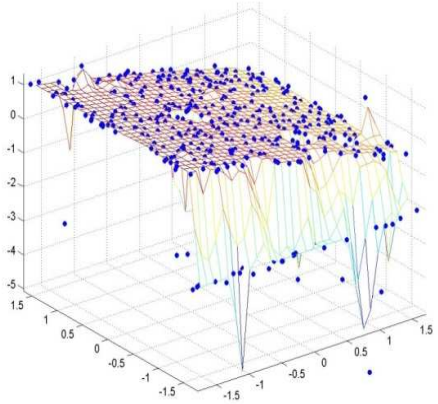
Group 1



Group 2



Group 3



Group 4

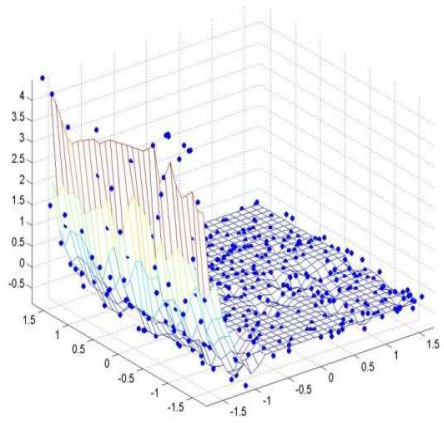
Figure 4-5 Surface Plots of Value Functions on Stage 6

group 1: 50 minimum observations on leaf and 7, 3 penalty breaking points

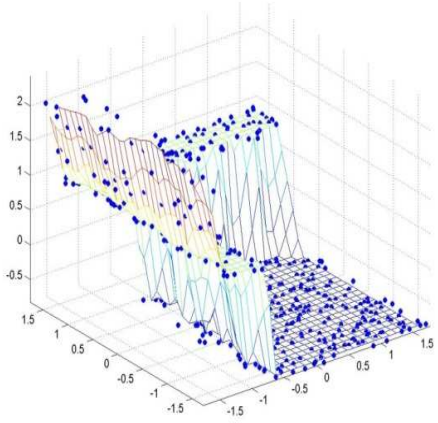
group 2: 100 minimum observations on leaf and 7, 3 penalty breaking points

group 3: 50 minimum observations on leaf and 15, 7 penalty breaking points

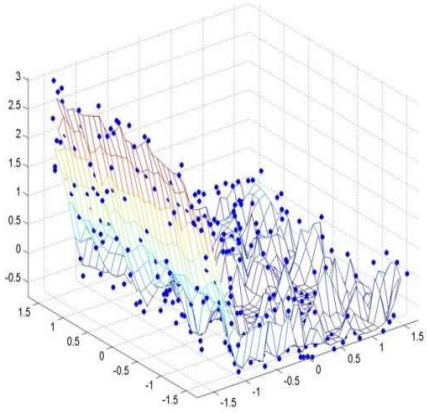
group 4: 100 minimum observations on leaf and 15, 7 penalty breaking points



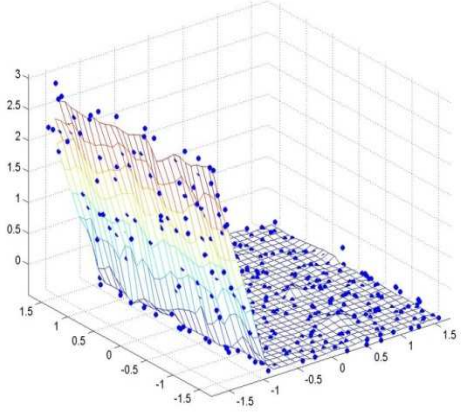
Group 1 stage 12



Group 2 stage 12



Group 3 stage 12



Group 4 stage 12

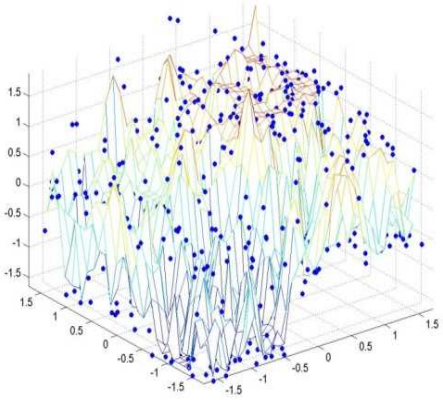
Figure 4-6 Surface Plots of Value Functions on Stage 12

group 1: 50 minimum observations on leaf and 7, 3 penalty breaking points

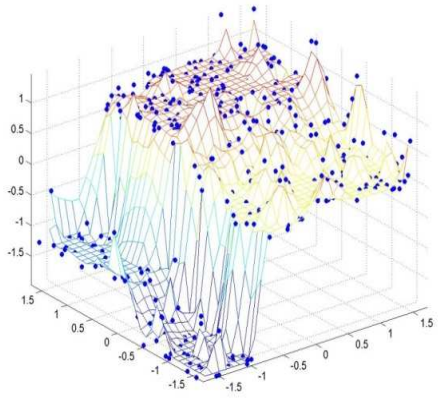
group 2: 100 minimum observations on leaf and 7, 3 penalty breaking points

group 3: 50 minimum observations on leaf and 15, 7 penalty breaking points

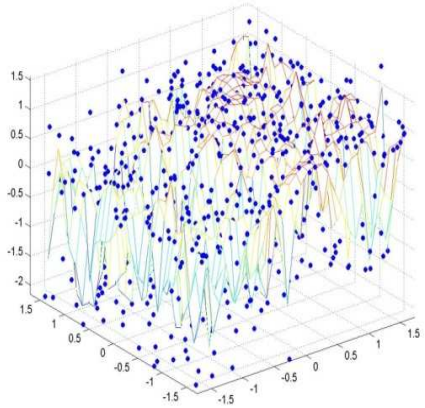
group 4: 100 minimum observations on leaf and 15, 7 penalty breaking points



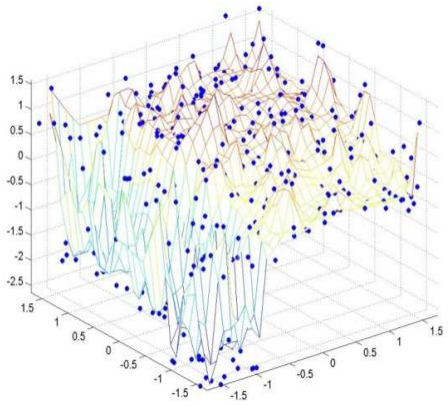
Group 1 stage 18



Group 2 stage 18



Group 3 stage 18



Group 4 stage 18

Figure 4-7 Surface Plots of Value Functions on Stage 18

group 1: 50 minimum observations on leaf and 7, 3 penalty breaking points

group 2: 100 minimum observations on leaf and 7, 3 penalty breaking points

group 3: 50 minimum observations on leaf and 15, 7 penalty breaking points

group 4: 100 minimum observations on leaf and 15, 7 penalty breaking points

4.5.5 Study of AVFA

We implement AVFA in this study. The AVFA algorithm is similar to the algorithm presented in Section 3.1.1. Six groups of experiments are conducted. They are grouped by minimum observations on the terminal leaves (20, 50 and 100) and piecewise linear penalty functions (breaking points 15, 7 and breaking points 7, 3). Table 4.14 describes these six groups of experiments. The motivation of the AVFA algorithm is to identify the trade-off between variance and bias by increasing training data until the testing error levels off (Fan et al. 2013). Each experiment starts with 200 design points, and 50 design points are added iteratively until the moving average errors from a test data set meet the stopping criteria. There are 200 independent design points used as the testing dataset. Three accuracy measurements, the mean average error (MAE), the mean square error (MSE) and the adjusted R^2, are generated from the test dataset. Approximation accuracy in stages 6, 12 and 18 are used to show the approximation processes within the AVFA algorithms. Figures 4-8 to 4-13 show the evolving processes of MAE, MSE, adjusted R^2 on both the training and testing datasets over the iterations.

Table 4-12 Six Groups of Experiments and Experiments' Settings

	tree structure	penalty function
group 1	20 minimum observations on leaf	7, 3 breaking points
group 2	50 minimum observations on leaf	7, 3 breaking points
group 3	100 minimum observations on leaf	7, 3 breaking points
group 4	20 minimum observations on leaf	15, 7 breaking points
group 5	50 minimum observations on leaf	15, 7 breaking points
group 6	100 minimum observations on leaf	15, 7 breaking points

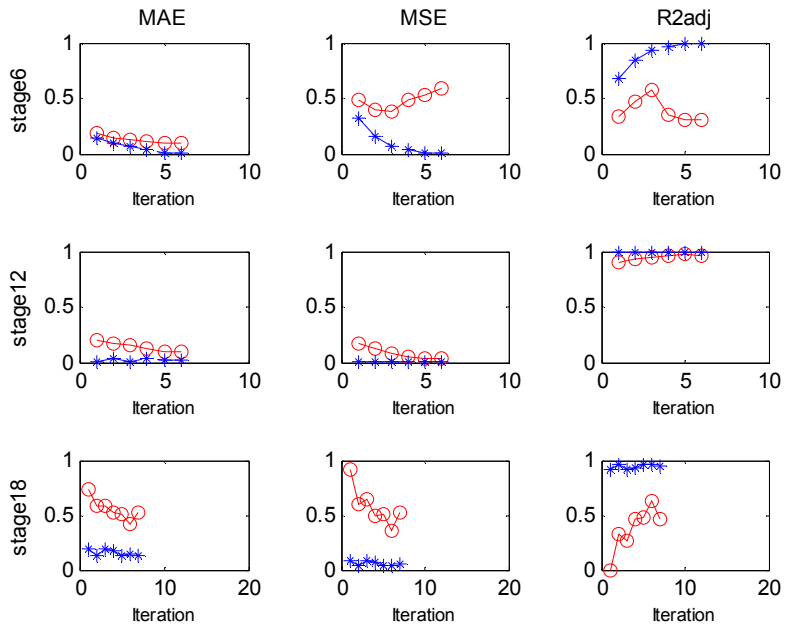


Figure 4-8 MAE, MSE, Adjust R-square of Experiment Group 1 within AVFA

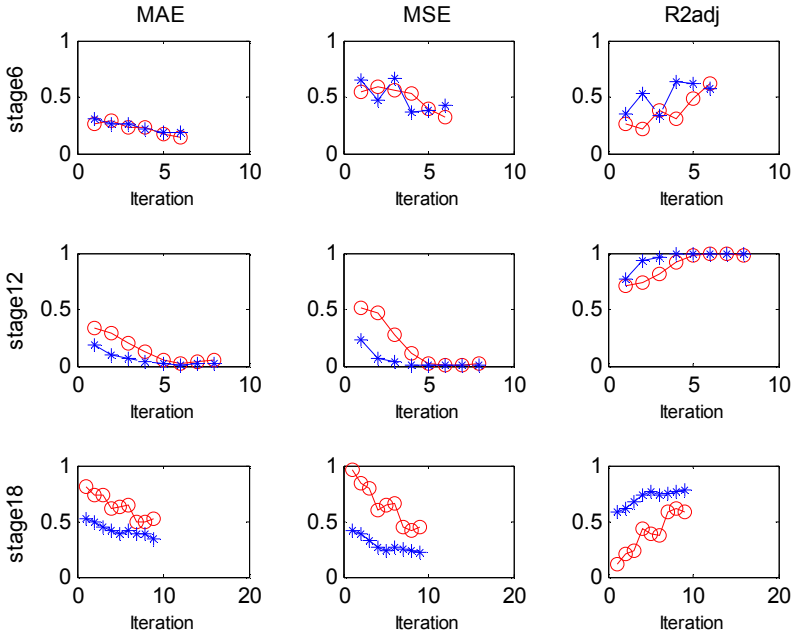


Figure 4-9 MAE, MSE, Adjust R-square of Experiment Group 2 within AVFA

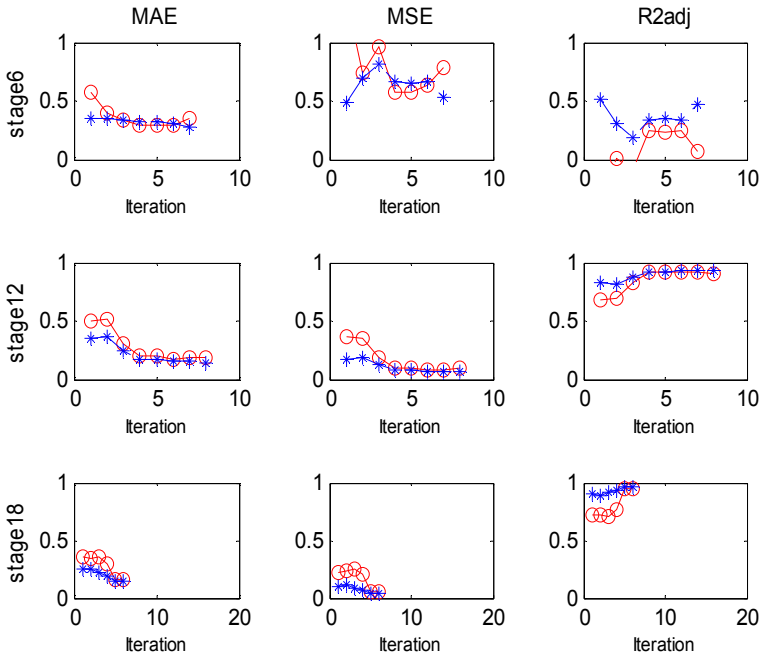


Figure 4-10 MAE, MSE, Adjust R-square of Experiment Group 3 within AVFA

Figures 4-8, 4-9 and 4-10 show the error for the value function approximation over the iterations in the AVFA algorithm for ADP with a penalty function of break points 7 and 3. The results show that group 3 took more iterations to reach the stopping criteria than other groups. In stage 6, the testing error in group 3 is much larger than groups 1 and 2. The distance between the training error and the testing error is smallest in group 3 on stage 18. It is also observed that group 2 shows that the testing error is closer to the training error in stage 6. This might be because the approximation model in group 1 is being over-fitted by the larger tree based models.

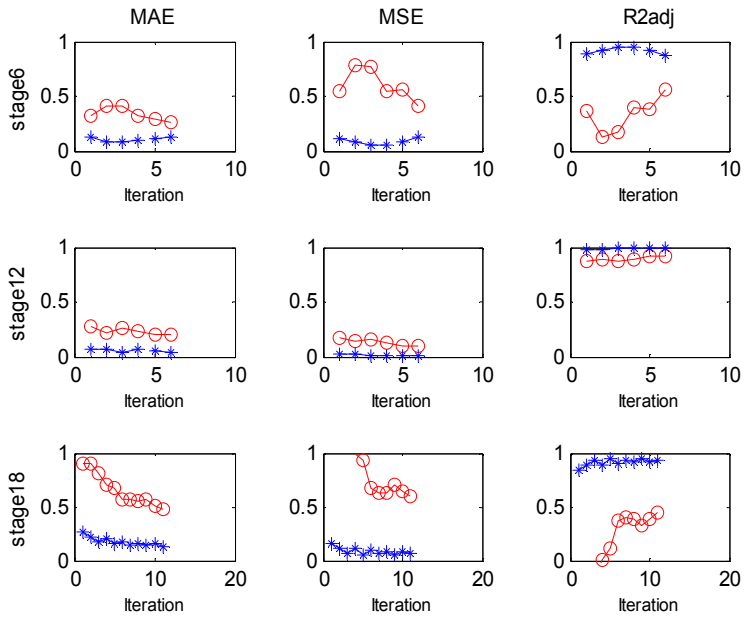


Figure 4-11 MAE, MSE, Adjust R-square of Experiment Group 4 within AVFA

Figures 4-11, 4-12 and 4-13 show errors in the approximation over the iterations for a penalty function with break points 15 and 7. The penalty function on low DO is stricter in this case. A stronger dependency over stages is expected. Based on the parameter setting of penalty functions and value function approximation, it is also expected that 100 minimum observations at the leaf would possibly generate an under-fitting tree for value function approximations. If the parameter is 20 minimum points on the leaf, over-fitting could be a possibility. The observations meet these expectations since group 6 performs worse than groups 4 and 5 in terms of approximation accuracy. In stage 6, group 4 shows a testing error much bigger than groups 5 and 6. Distance between the testing data and the training data in group 4 is larger than the other groups. In stage 18, group 2 performs the best in terms of both distance between the training error, testing error, and approximation accuracy. It is possibly due to the fact that 100

minimum observations at the terminal leaf in group 6 generate an under-fitting tree based model in stage 18.

Figures 4-14 and 4-15 show simulation results of the six groups' experiments. Boxplots of simulation cost in groups 1, 2 and 3 are put together in one figure because they use the same penalty function. The figures show that the ADP policy of group 2 generates a lower average simulation cost than groups 1 and 3. This is consistent with the results in the approximation accuracy study that group 2 has a better approximation accuracy than groups 1 and 3. Similar explanations can be applied to the simulation study of groups 4, 5 and 6. Group 5 generates a lower average simulation cost than groups 4 and 6.

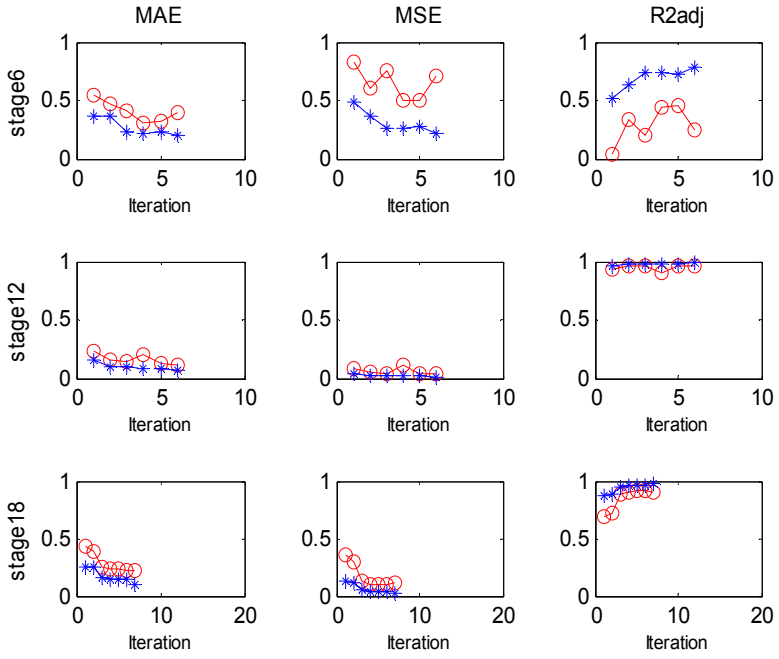


Figure 4-12 MAE, MSE and Adjust R-square of Experiment Group 5 within AVFA

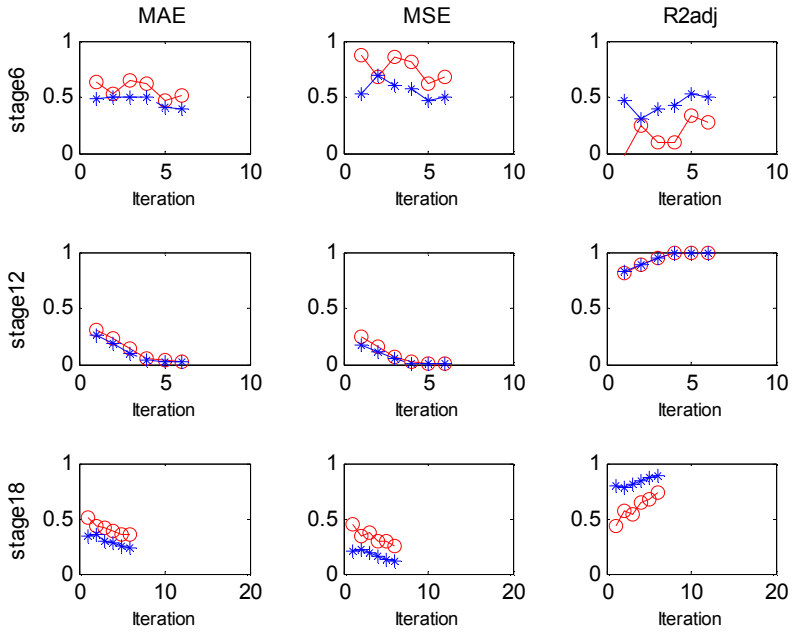


Figure 4-13 MAE, MSE and Adjust R-square of Experiment Group 6 within AVFA

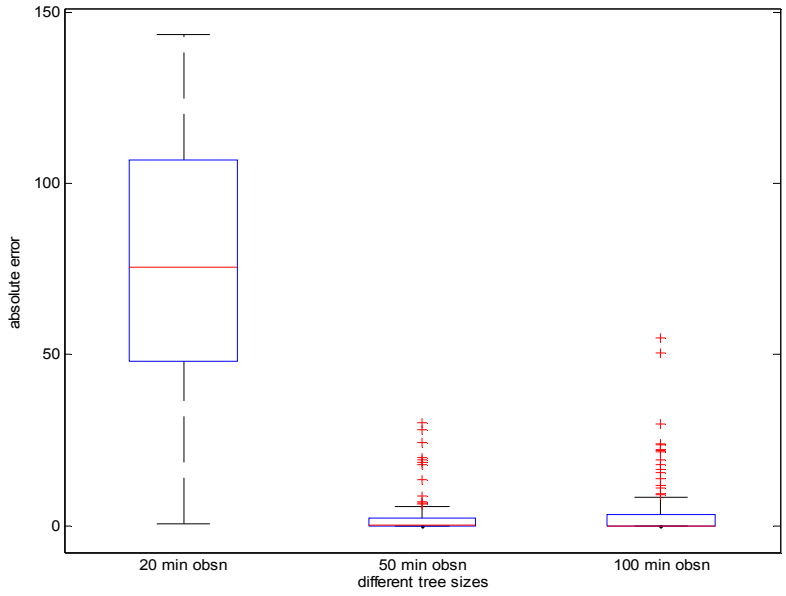


Figure 4-14 Simulation Comparison of Group 1, 2 and 3

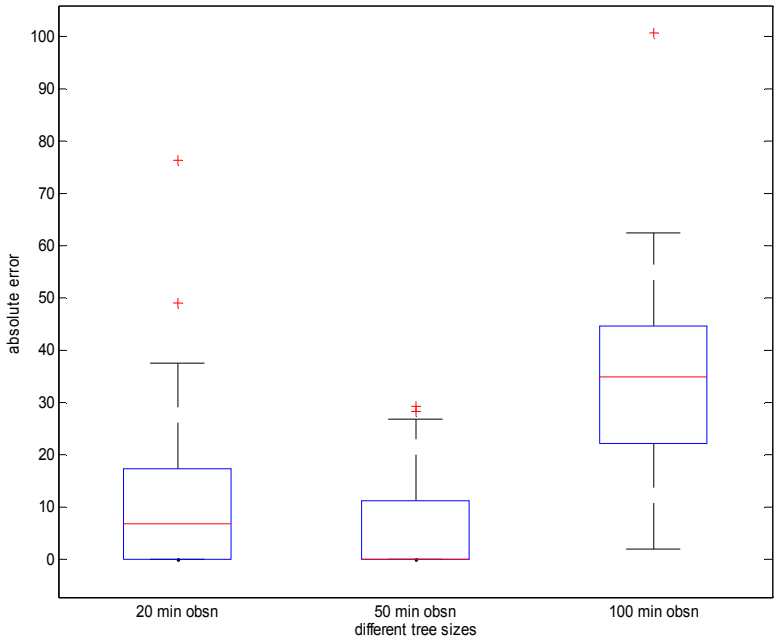


Figure 4-15 Simulation Comparison of Group 4, 5 and 6

4.5.6 Benchmarking Against Decision Making without Value Functions

In this section, the solutions of our ADP method are compared with the optimization solution of decision making without value functions. The implementation of optimization without value function is simple in that the value function approximations are simply removed from objective functions in the forward simulation. Then the objective values of all stages are summed and compared with the ADP simulation costs. Figure 4-16 shows the ratio of the cumulative costs over stages. The *ratio* is defined as a cumulative cost of the ADP simulation divided by a cumulative cost of optimization without the value function at each stage. Three simulations of ADP for the same case are used in this comparison. In Figure 4-16, IP represents optimizations without value function approximations. Every simulation starts with same initial state. Results shows

that the ADP simulation costs are always lower than the optimization without value function approximations since the ratios are always less than 1.

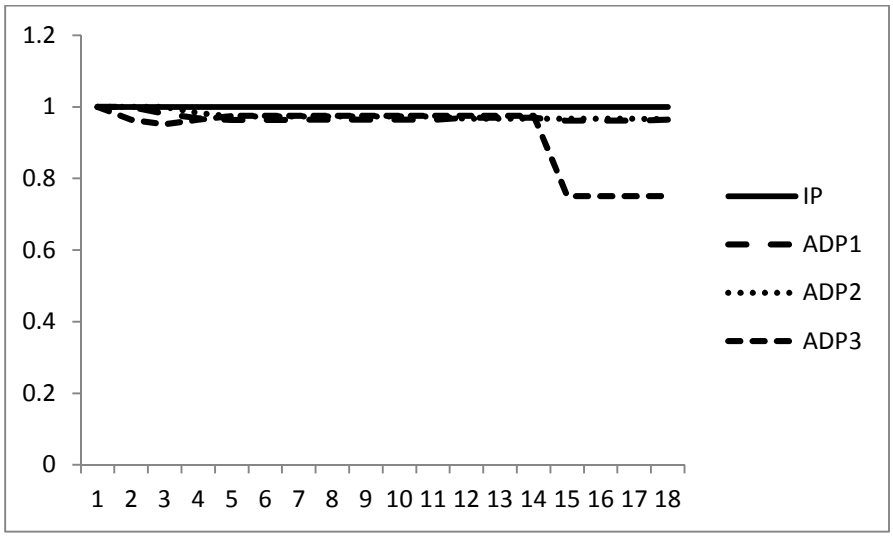


Figure 4-16 Ratios of Simulation Costs

4.6 Benchmarking Against Reinforcement Learning based ADP Model

Reinforcement learning (RL) is another modern methodology used in ADP to solve multistage decision making problems. It also uses the idea of the optimality equation (Sutton and Barto 1998). Reinforcement learning methods are usually developed to solve problems within an infinite time horizon (Sutton and Barto 1998). There is very limited study on reinforcement learning with a finite horizon problems (Li and Littman 2005). Q-learning with finite horizon is called fitted Q iterations (Murphy 2005), which is different from DACE based DP. The action space is always sampled at each stage in Q-learning. A value function is the function of both state space and action in Q-learning. In this study, a comparison of our ADP method and Q-learning is conducted in the context of the DFW Airport deicing problem.

Besides the value function, the other difference between RL based ADP and DACE based ADP is the action space. RL based ADP is usually used with a countable

action space (Sutton and Barto 1998). To apply Q-learning to solve the DFW Airport deicing problem, we select limited decisions within the full action space. Simulation costs show that the RL based ADP with the limited action space is also able to generate a solution with a low simulation cost. Designed for an uncountable action space, one of the advantages of our ADP method is that it enables the uncountable discrete action space search by employing MILP and the branch-and-bound algorithm to solve the optimality equation. In rest of this section, a Q-learning algorithm is proposed to solve DFW Airport deicing problem. The implementation strategy will be discussed as well as the numerical results. Two state space sampling methods such pseudorandom number generation and low discrepancy sequences are studied in Section 4.6.3. This study shows that the optimal value function of our ADP method is able to generate better optimal policy than Q-learning but is more computationally expensive. At the end of this section, a new method is proposed to quickly evaluate the optimal policy with the optimal value function. The method is then applied to solve the DFW Airport deicing problem.

4.6.1 Fitted Q-learning Algorithm

Similar to finite horizon DACE based ADP, fitted Q-learning involves two steps: a backward ADP solution process of approximating the Q function, and a forward simulation process of optimal policy evaluation. The Q-learning backward ADP solution process usually utilizes random samples for both state and action spaces (Sutton and Barto 1998). For each combination of state points and action samples, a set of scenarios is generated for the random variables. The decision that gives the lowest average return is selected over the scenarios as the optimal solution. The Q function is then fitted as function of the state variables and action variables. The forward simulation process evaluates the Q function by selecting an action that gives the lowest value of Q functions

for a given state. The backward ADP solution process and the forward simulation process of fitted Q-learning are described below, respectively, to solve this deicing problem.

Backward ADP solution process to approximate Q functions:

1. Randomly sample N points in the state space x_{jt} for the stage t, $t = 0,1, \dots, T$.

2. In stage T,

   2a for each point x_{jT}, $j = 1, \dots, N$, sample decision $\boldsymbol{u_{jT}}$, ε_j is random variable, $\mathrm{C_T}$ is the contribution function, solve

$$Q_{\mathrm{T}}(\mathrm{x_{jT}}, \mathrm{u_{jT}}) = \underset{\varepsilon}{\mathrm{E}}\{\mathrm{C_T}(\mathrm{x_{jT}}, \mathrm{u_{jT,}}\varepsilon)\};$$

   2b approximate $Q_{\mathrm{T}}(\mathrm{x_{jT}}, \mathrm{u_{jT}})$ with $\hat{Q}_T(x_{jT},\ u_{jT})$.

3. For $t = T - 1, \dots, 0$,

   3a for each point x_{jt}, $j = 1, \dots, N$, ε_j is random variable, sample a decision u_{jt}, $\mathrm{C_t}$ is the contribution function, g_t is the state transition function, solve

$$\mathrm{Q_t}(\mathrm{x_{jt}}, \mathrm{u_{jt}}) = \underset{\varepsilon}{\mathrm{E}}\left\{\mathrm{C_t}(\mathrm{x_{jt}}, \mathrm{u_{jt,}}\varepsilon) + \min_{\mathrm{u_{j,t+1}}} \widehat{\mathrm{Q}}_{\mathrm{t+1}}\left(\mathrm{g_t}(\mathrm{x_{jt}}, \mathrm{u_{jt}}, \varepsilon)\right)\right\};$$

   3b approximate $\tilde{Q}_t(x_{jt}, u_{jt})$ with $\hat{Q}_t(x_{jt},\ u_{jt})$.

Forward simulation process to evaluate Q functions:

Sample N_s points in the state space of first stage, x_{j0}, $j = 1, .., N_S$

For $j = 1$ to N_S

   For $r = 1$ to N_r

      For stage $t = 0$ to T , solve

$$\max_{\mathrm{u_{jt}}}\ \underset{\varepsilon}{\mathrm{E}}\{\widehat{\mathrm{Q}}_{\mathrm{t}}(\mathrm{x_{jt}},\ \mathrm{u_{jt}},\ \varepsilon)\},$$

      generate sample path ε_r, obtain cost/contribution and state transitions

$$C_t(x_{jt},\ u_{jt}, \varepsilon_r),$$

$$x_{j,\,t+1} = g_t(x_{jt}, u_{jt}, \varepsilon_r),$$

      simulation cost for initial state j and sample path r

$$\sum_{t=0}^{T} C_t(x_{jt},\ u_{jt}, \varepsilon_r).$$

   For each initial state, simulation cost is

$$\frac{1}{N_r}\sum_{r=0}^{N_r}\sum_{t=0}^{T} C_t(x_{jt},\ u_{jt}, \varepsilon_r).$$

4.6.2 Comparisons of Optimal Value Function and Q Function

In this section, three ADP methods are compared. DACE based ADP that uses treed regression is the first method. The second ADP method uses Q-learning and a limited action space. Both action and state spaces are sampled in the backward ADP solution process. The third method mixes the optimal value function of DACE based ADP and the limited action space of Q-learning. Each of the three ADP methods is tested in experiments to solve the DFW Airport deicing problem. Results of three sets of experiments show that the optimal value functions perform better than the Q function with lower simulation costs. Three sets of experimental results are illustrated in Figures 4-17, 4-18 and 4-19. Boxplots of absolute errors are used to compare the three ADP methods. The first group represents the simulation cost of 100 initial states by ADP with the optimal value functions and the full action space. The second boxplot represents the simulation cost of same initial states by ADP with Q functions. The third boxplot represents the simulation cost of same initial states by ADP with the optimal value functions and the limited action space. Each of three experiment sets shows that the ADP with optimal value functions generates the lowest simulation cost. It can also be observed that the third ADP method generates lower simulation costs than the ADP with Q functions. The second and third ADP methods share the same action space. Therefore, it is seen that by employing MILP and treed regression, DACE based ADP is able to explore the full action space and perform better than the other methods. In addition, the optimal value function shows an advantage over Q functions in terms of simulation cost.

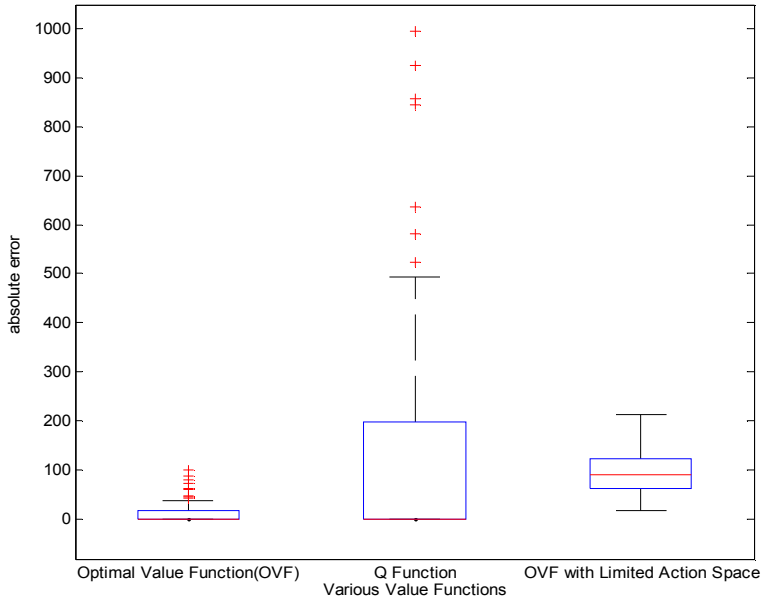


Figure 4-17 First Case of Comparison

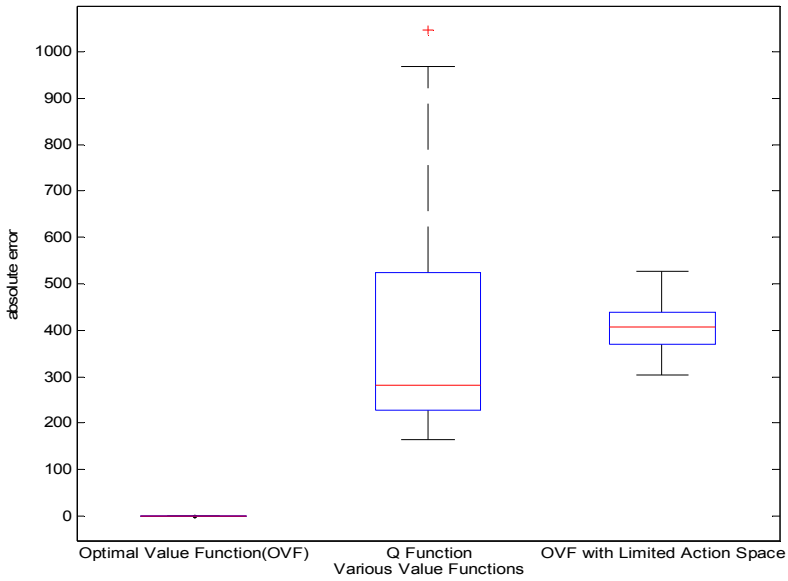


Figure 4-18 Second Case of Comparison

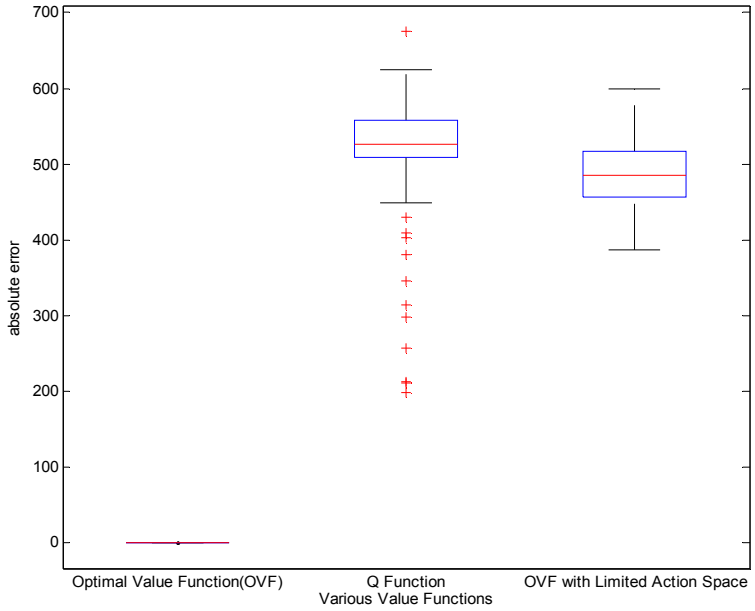


Figure 4-19 Third Case of Comparison

4.6.3 Study of State Space Sampling

Low discrepancy sequences such as the Sobol' sequence have been proven to converge faster than Monte-Carlo simulation by pseudorandom number generation (Morokoff et al. 1994). Sobol' sequence is considered to have a better "space filling" property (Pronzato and Müller 2012). In this study, both Sobol' sequence and pseudorandom number generation are applied to solve the DFW Airport deicing ADP problem. Figures 4-20, 4-21 and 4-22 show simulation results of three sets of comparisons study. In each figure, the first boxplot represents the simulation result by using Sobol' sequence. The second and third boxplots represent simulation results by using pseudorandom number generation. It is observed that the first boxplot of each figure has lower cost and a smaller box compared to the second and third boxplots.

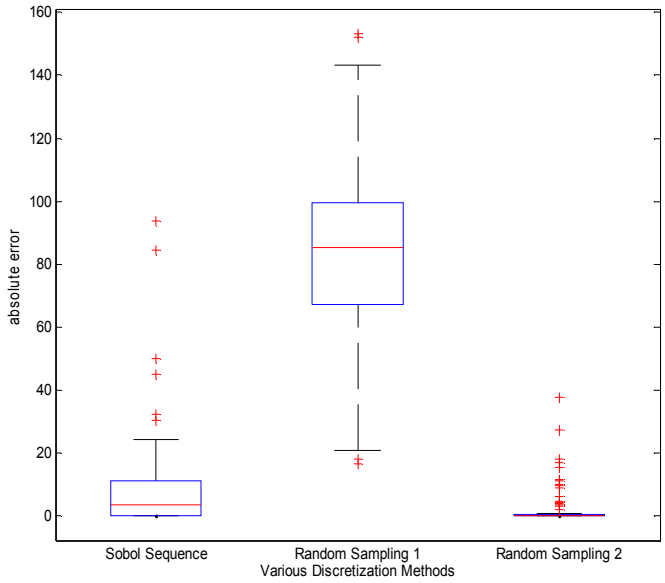


Figure 4-20 Simulation Results of State Sampling Methods
by the ADP with Optimal Value Functions

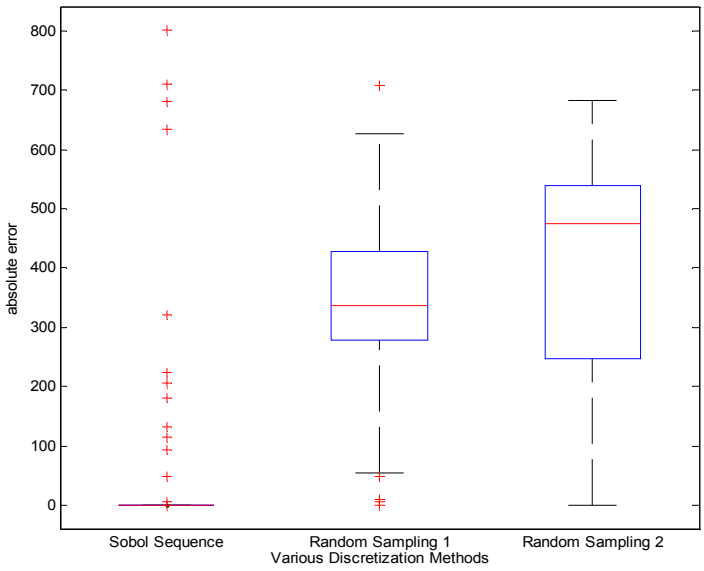


Figure 4-21 Simulation Results of State Space Sampling Methods
by the ADP with Q functions

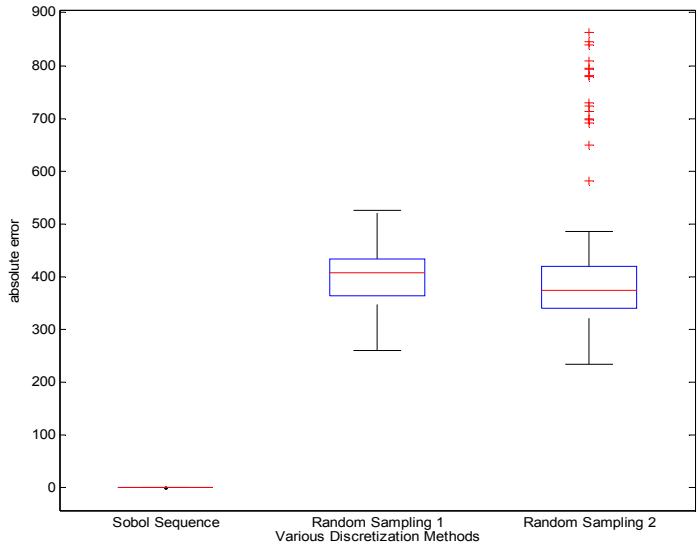


Figure 4-22 Simulation Results of State Space Sampling Methods
by the ADP with Limited Action Space

4.6.4 An Extended Method to Faster Generate Optimal Policy

In Section 4.6.2, it is observed that the optimal value function is able to generate a solution with a lower simulation cost. ADP with optimal value function is able explore an uncountable discrete action space. However, this action space exploration can be time consuming. A limited action space needs less computational effort since enumeration is sufficient for making decisions. In this section, an approach is proposed to approximate value functions with the full action space and use a limited action space to evaluate the optimal policy. If the limited action space is representative of good solutions in the full action space, the proposed method would be able to tradeoff solution quality and computational effort for policy evaluation. This method is tried and compared with other ADP methods in this section. Figures 4-23, 4-24, and 4-25 show simulation results for three ADP methods such as ADP with optimal value function and full action space, ADP with optimal value functions and limited action space, and the proposed ADP method.

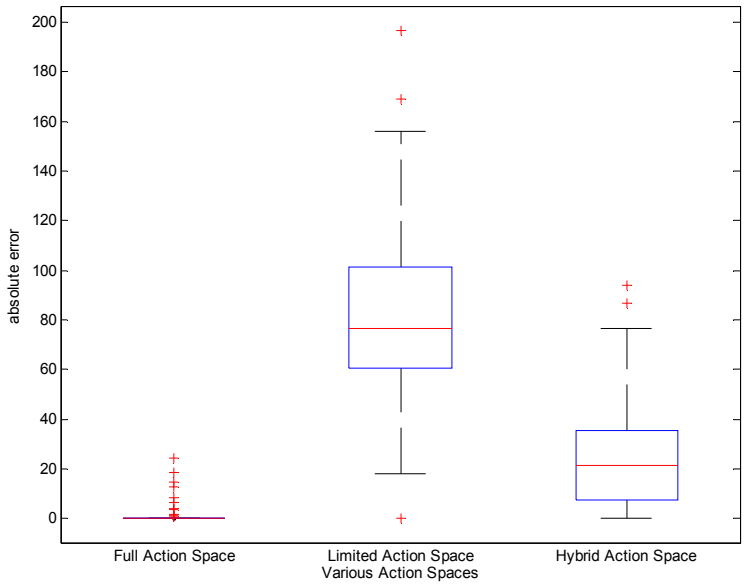


Figure 4-23 Simulation Costs Comparison 1

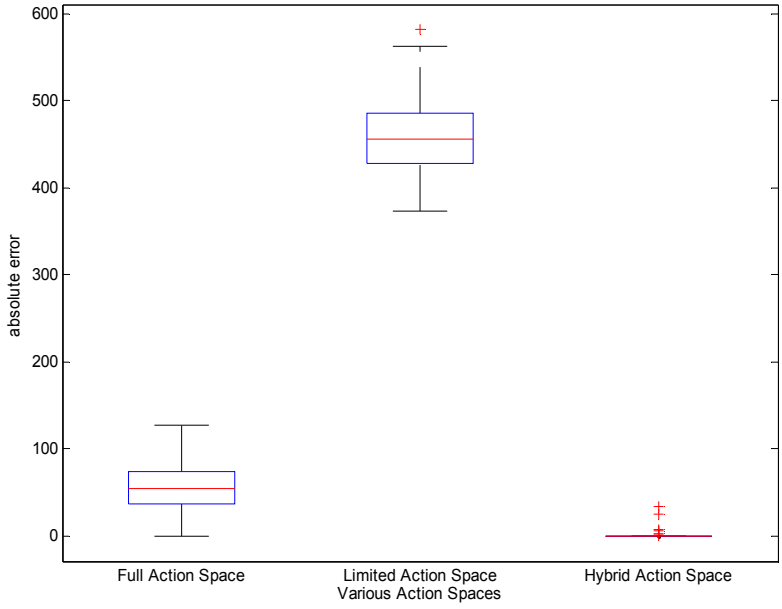


Figure 4-24 Simulation Costs Comparison 2

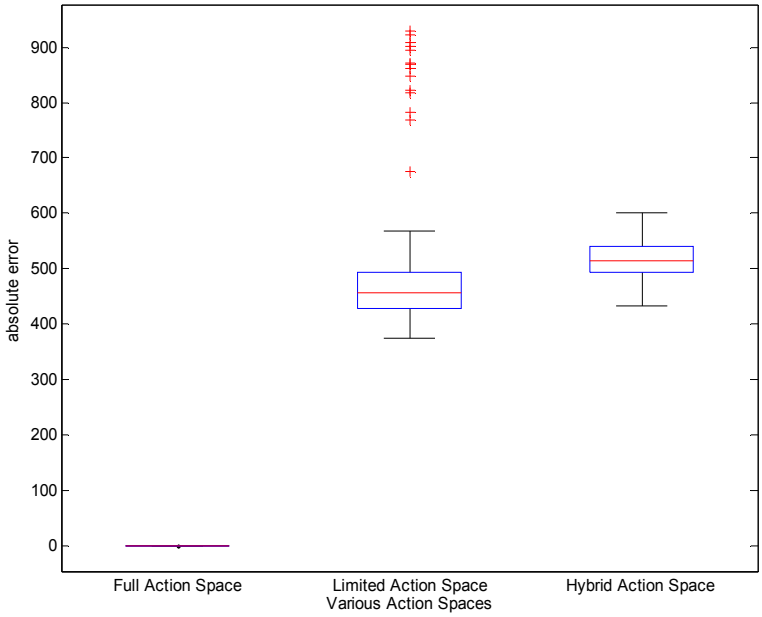


Figure 4-25 Simulation Costs of Comparison 3

Figures 4-23, 4-24 and 4-25 demonstrate that the solution generated by the proposed ADP method is at least as good as the ADP method with limited action space. In some cases, the proposed ADP method can generate a policy as good as the ADP method with the full action space. In terms of computational time, the proposed method evaluates the ADP policy much faster than the other methods. Figures 4-26 and 4-27 are scatter plots of the solution quality over computational time for a simulation of 100 initial state. Solution quality is measured by both mean and variance of the simulation costs in this study. This proposed method uses the least amount of computational time. It also generates good solutions in terms of both mean and variance of simulation costs.

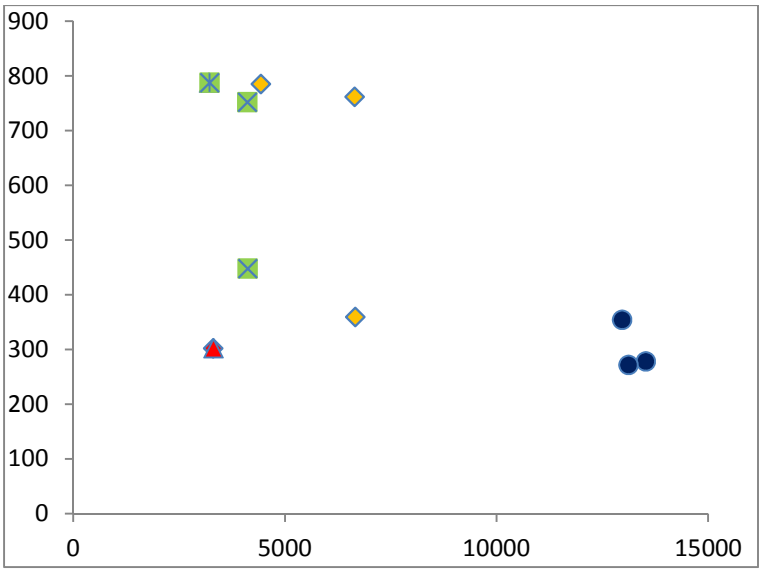


Figure 4-26 Mean Values of Simulation Costs vs. Computational Time

(red: fast algorithm to generate optimal policy; yellow: ADP with optimal value function and limited action space; blue: ADP with optimal value function and full action space; green: ADP with Q function and limited action space.)

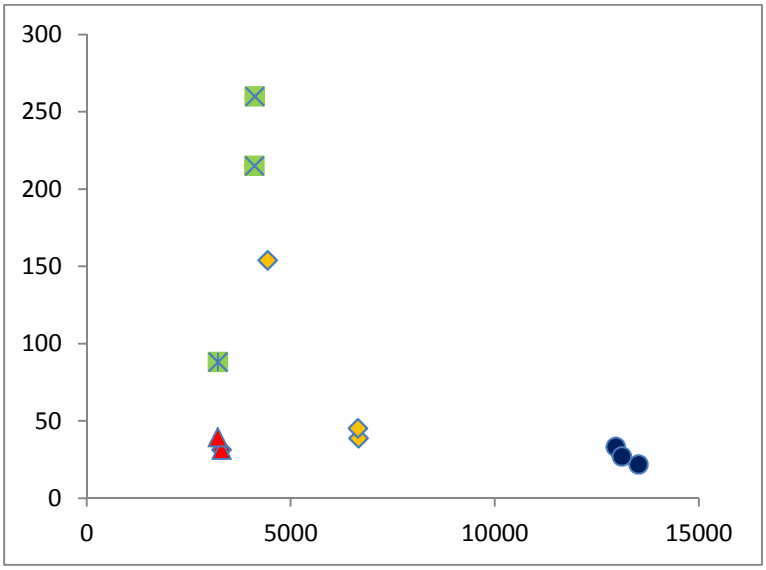


Figure 4-27 Standard Deviation of Simulation Costs vs. Computational Time

Chapter 5

Conclusion and Future Study

This dissertation developed novel methods in the approximate dynamic programming (ADP) area from both statistical and optimization perspectives. There are four major parts of the study: 1) Sequential state space exploration (SSSE) algorithms are extended from neural networks (NN) to multivariate adaptive regression splines (MARS). Some improvements of SSSE algorithms are studied for an inventory forecasting DP problem. 2) A novel ADP algorithm based on treed regression and mixed integer linear programming (MILP) is invented to improve the solution quality by solving the optimality equations to the global optimum at each state discretization point on each stage. This idea takes advantage of the flexibility of treed regression and also demonstrates that solving the optimality equations to the global optimum is important for non-convex, non-stationary and high0dimensional DP problems. 3) A high dimensional real world multi-stage decision making problem is solved by this method. The DFW Airport deicing problem is a non-stationary, a non-convex and multistage decision problem with high dimensional continuous-state space and uncountable discrete action space. Our method solves this complex problem with high solution quality. 4) By solving the problem with a reinforcement learning (RL) based ADP method with some simplifications and generalization, some benchmark studies are built up by applying a finite horizon Q-learning ADP method to the DFW Airport deicing problem. The results show the advantages of the optimal value function over RL using the Q function. It is concluded that low-discrepancy sequences show more stable results than random sampling by pseudo-random number generation. By combining the advantages of these two methods, the extended method is able to quickly generate optimal policies with good solution quality.

The future work will focus on ADP method development with the following directions: 1) Extending ADP with MILP to other statistical learning methods such as piecewise linear regression splines by taking advantages of both flexible statistical learning methods and solving optimality equations to global optimality; 2) Extending ADP with MILP to problems with infinite horizon. In the real world, there are many ADP problems that are the infinite horizon because the actual future impact is considered within an infinite time horizon. This would also extend the study of the ADP method with MILP to stationary complex systems. 3) Solving genetic large scale integer programming problems such as large scale vehicle routing problems using ADP methods. These research directions have potential in improving solution quality and computational time of large scale integer programming problems.

References

Alexander, W. P., & Grimshaw, S. D. (1996). Treed regression. *Journal of Computational and Graphical Statistics*, *5*(2), 156-175.

Bellman, R. (1957). E. 1957. dynamic programming. *Princeton University Press. Bellman Dynamic programming1957*.

Bertsekas, D. P. (2007). Separable dynamic programming and approximate decomposition methods. *Automatic Control, IEEE Transactions on*, *52*(5), 911-916.

Bertsekas, D. P., & Castanon, D. A. (1999). Rollout algorithms for stochastic scheduling problems. Journal of Heuristics, 5(1), 89-108.

Bertsekas, D. P. (2005). Dynamic Programming and Suboptimal Control: A Survey from ADP to MPC*. *European Journal of Control*, *11*(4), 310-334.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Box, G. E., & Wilson, K. B. (1951). On the experimental attainment of optimum conditions. *Journal of the Royal Statistical Society. Series B (Methodological)*,*13*(1), 1-45.

Box, G. E., & Draper, N. R. (1987). *Empirical model-building and response surfaces*. John Wiley & Sons.

Boyd, S., & Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Cervellera, C., Chen, V. C., & Wen, A. (2006). Optimization of a large-scale water reservoir network by stochastic dynamic programming with efficient state space discretization. *European Journal of Operational Research*, *171*(3), 1139-1151.

Cervellera, C., Wen, A., & Chen, V. C. (2007). Neural network and regression spline value function approximations for stochastic dynamic programming. *Computers & operations research*, *34*(1), 70-90.

Chen, V. C., Ruppert, D., & Shoemaker, C. A. (1999). Applying experimental design and regression splines to high-dimensional continuous-state stochastic dynamic programming. *Operations Research*, *47*(1), 38-53.

Chen, V. C., Günther, D., & Johnson, E. L. (2003). Solving for an optimal airline yield management policy via statistical learning. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *52*(1), 19-30.

Chen, V. C., Tsui, K. L., Barton, R. R., & Meckesheimer, M. (2006). A review on design, modeling and applications of computer experiments. *IIE transactions*, *38*(4), 273-291.

Chen, V.C., Huff, B., Rosenberger. J, Zeng, L., Jen, C., Ariyajunya, B., LeBoulluec, A., Sahu, S., Martinez, N., Goswami, A., Kudale, S., "GOALI: Statistically Parsimonious Adaptive Dynamic Programming for Minimizing the Environmental Impact of Airport Deicing Activities," Technical Report to D/FW International Airport, 2011.

Corsi, S. R., Harwell, G. R., Geis, S. W., & Bergman, D. (2006). Impacts of aircraft deicer and anti-icer runoff on receiving waters from Dallas/Fort worth International Airport, Texas, USA. *Environmental toxicology and chemistry*,*25*(11), 2890-2900.

Cressie, N. (1990). The origins of kriging. *Mathematical geology*, *22*(3), 239-252.

Dean, A., & Voss, D. (1999). Design and analysis of experiments. Speringer-Verlag New York.

Delisi, C. (1974). A theory of precipitation and agglutination reactions in immunological systems. *Journal of theoretical biology*, *45*(2), 555-575.

Denardo, E. V., & Fox, B. L. (1979). Shortest-route methods: 1. Reaching, pruning, and buckets. *Operations Research*, *27*(1), 161-186.

Dupačová, J., Gröwe-Kuska, N., & Römisch, W. (2003). Scenario reduction in stochastic programming. *Mathematical programming*, *95*(3), 493-511.

Ernst, D., Glavic, M., & Wehenkel, L. (2004). Power systems stability control: reinforcement learning framework. *Power Systems, IEEE Transactions on*,*19*(1), 427-435.

FAA Report, 1996. Report on FAA Deicing Program at Guardia and O'Hare Airport. Report Number E5-FA-7-002, Office of Inspector General, Department of Transportation, Federal Aviation Regulation. (http://ntl.bts.gov/lib/1000/1500/1543/e5-fa-7-002.pdf)

Fan, H. (2008). *Sequential frameworks for statistics-based value function representation in approximate dynamic programming*. ProQuest.

Fan, H., Tarun, P. K., & Chen, V. C. (2013). Adaptive value function approximation for continuous-state stochastic dynamic programming. *Computers & Operations Research*, *40*(4), 1076-1084.

Fan, H., Tarun, P. K., Shih, D. T., Kim, S. B., Chen, V. C., Rosenberger, J. M., & Bergman, D. (2011). Data mining modeling on the environmental impact of airport deicing activities. *Expert Systems with Applications*, *38*(12), 14899-14906.

Faure, H. (1982). Discrépance de suites associées à un système de numération (en dimension s). *Acta Arithmetica*, *41*(4), 337-351.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The annals of statistics*, 1-67.

Foufoula-Georgiou, E., & Kitanidis, P. K. (1988). Gradient dynamic programming for stochastic optimal control of multidimensional water resources systems. *Water resources research*, *24*(8), 1345-1359.

Garcia, F., & Ndiaye, S. M. (1998). A learning rate analysis of reinforcement learning algorithms in finite-horizon. In *Proceedings of the 15th International Conference on Machine Learning (ML-98)*.

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, *63*(1), 3-42.

Hammersley, J. M. (1960). Monte Carlo methods for solving multivariable problems. *Annals of the New York Academy of Sciences*, *86*(3), 844-874.

Haykin, S., & Network, N. (2004). A comprehensive foundation. *Neural Networks*, *2*(2004).

Hua, Z., Zhang, B., & Liang, L. (2006). An approximate dynamic programming approach to convex quadratic knapsack problems. *Computers & operations research*, *33*(3), 660-673.

Jekabsons, G. (2010). ARESLab: adaptive regression splines toolbox for Matlab. *Institute of Applied Computer Systems Riga Technical University, Meza*, *1*(3).

Jeong, K., Kim, S., & Bandeira, N. (2012). False discovery rates in spectral identification. BMC bioinformatics, 13(Suppl 16), S2.

Johnson, S. A., Stedinger, J. R., Shoemaker, C. A., Li, Y., & Tejada-Guibert, J. A. (1993). Numerical solution of continuous-state dynamic programs using linear and spline interpolation. *Operations Research*, *41*(3), 484-500.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 237-285.

Kalmár, Z., Szepesvári, C., & Lörincz, A. (1998). Module-based reinforcement learning: Experiments with a real robot. *Machine Learning*, *31*(1-3), 55-85.

Kennedy, J. O. (1986). Dynamic programming applications to agriculture and natural resources. School of Economics, La Trobe Univ., Melbourne.

Kim, H., Loh, W. Y., Shih, Y. S., & Chaudhuri, P. (2007). Visualizable and interpretable regression models with good prediction power. IIE Transactions, 39(6), 565-579.

Kim, S.B., Chen, V.C., Rosenberger, J., Fan, H., Tarun, P.K., ... "Data-Driven Optimization for Minimizing the Environmental Impact of Airport Deicing Activities," Technical Report to D/FW International Airport, 2007.

Kim, S. B., Rosenberger, J. M., Chen, V. C., Fan, H., Shih, D. T., and Bergman, D. (2007). "Statistically mining the environmental impact of airport deicing activities." In Proc. 2007 Industrial Engineering Research Conference, Nashville, TN, USA, May.

Konda, V. R., & Tsitsiklis, J. N. (1999, November). Actor-Critic Algorithms. In*NIPS* (Vol. 13, pp. 1008-1014).

Kroes, M.J., Watkins, W.A., and Delp, F. (1993) Aircraft maintenance and repair, 6th Edition, Mc-Graw-Hill.

Kwon, I. H., Kim, C. O., Jun, J., & Lee, J. H. (2008). Case-based myopic reinforcement learning for satisfying target service level in supply chain. *Expert Systems with Applications*, *35*(1), 389-397.

Land, A. H., & Doig, A. G. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, 497-520.

Lewis, F. L., & Vrabie, D. (2009). Reinforcement learning and adaptive dynamic programming for feedback control. *Circuits and Systems Magazine, IEEE*, *9*(3), 32-50.

Li, L., & Littman, M. L. (2005, July). Lazy approximation for solving continuous finite-horizon MDPs. In *AAAI* (Vol. 5, pp. 1175-1180).

Li, R., & Sudjianto, A. (2005). Analysis of computer experiments using penalized likelihood in gaussian kriging models. *Technometrics*, *47*(2).

Lippmann, R. P. (1987). An introduction to computing with neural nets. *ASSP Magazine, IEEE*, *4*(2), 4-22.

Loh, W. Y. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, *12*(2), 361-386.

Malandraki, C., & Dial, R. B. (1996). A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research*, *90*(1), 45-55.

Manry, M. T., Hsieh, C. H., Dawson, M. S., Fung, A. K., & Apollo, S. J. (1996). Cramer rao maximum a-posteriori bounds on neural network training error for non-gaussian signals and parameters. *International Journal of Intelligent Control and Systems*, *1*(3), 381-391.

Morokoff, W. J., & Caflisch, R. E. (1994). Quasi-random sequences and their discrepancies. *SIAM Journal on Scientific Computing*, *15*(6), 1251-1279.

Murphy, S. A. (2005). A generalization error for Q-learning. *Journal of machine learning research: JMLR*, *6*, 1073.

Myers, R. H., Montgomery, D. C., & Anderson-Cook, C. M. (2009). *Response surface methodology: process and product optimization using designed experiments* (Vol. 705). John Wiley & Sons.

Nakamura, Y., Mori, T., Sato, M. A., & Ishii, S. (2007). Reinforcement learning for a biped robot based on a CPG-actor-critic method. *Neural Networks*, *20*(6), 723-735.

Narasimha, P. L., Delashmit, W. H., Manry, M. T., Li, J., & Maldonado, F. (2008). An integrated growing-pruning method for feedforward network training.*Neurocomputing*, *71*(13), 2831-2847.

Niederreiter, H., & NSF-CBMS Regional Conference on Random Number Generation. (1992). *Random number generation and quasi-Monte Carlo methods* (Vol. 63). Philadelphia: Society for Industrial and Applied mathematics.

Pronzato, L., & Müller, W. G. (2012). Design of computer experiments: space filling and beyond. *Statistics and Computing*, *22*(3), 681-701.

Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality* (Vol. 703). John Wiley & Sons.

Russell, S., Norvig, P., & Intelligence, A. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, *25*.

Robichek, A. A., Elton, E. J., & Gruber, M. J. (1971). Dynamic programming applications in finance. *The Journal of Finance*, *26*(2), 473-506.

Rummery, G. A., & Niranjan, M. (1994). On-line Q-learning using connectionist systems.

Sacks, J., Schiller, S. B., & Welch, W. J. (1989). Designs for computer experiments. *Technometrics*, *31*(1), 41-47.

Simpson, T. W., Peplinski, J., Koch, P. N., & Allen, J. K. (1997). On the use of statistics in design and the implications for deterministic computer experiments. *Design Theory and Methodology-DTM'97*, 14-17.

Sobol, I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational mathematics and mathematical physics*, (7), 86-112.

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning*. MIT Press.

Tejada-Guibert, J. A., Johnson, S. A., & Stedinger, J. R. (1995). The value of hydrologic information in stochastic dynamic programming models of a multireservoir system. *Water resources research*, *31*(10), 2571-2579.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58-68.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*,*42*(5), 674-690.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3-4), 279-292.

Wahba, G., & Wendelberger, J. (1980). Some new mathematical methods for variational objective analysis using splines and cross validation. *Monthly weather review*, *108*(8), 1122-1143.

White, C. C., & White, D. J. (1989). Markov decision processes. European Journal of Operational Research, 39(1), 1-16.

Valarezo, W. O., Lynch, F. T., & McGhee, R. J. (1993). Aerodynamic performance effects due to small leading-edge ice (roughness) on wings and tails. *Journal of aircraft*, *30*(6), 807-812.

Zhang, Z, Rosenberger, J.M., Chen, V.C., Zeng, L & Bergman, D 2013, 'Adaptive dynamic programming for airport deicing'. in *IIE Annual Conference and Expo 2013*. Institute of Industrial Engineers, pp. 3670-3678, IIE Annual Conference and Expo 2013, San Juan, 18-22 May.

Biographical Information

Zirun Zhang was born in Shucheng, Anhui, China, in 1988. He received his B.S. degree in Mathematics from North China Electric Power University and he also received his masters' degree in Management Science and Engineering from Shanghai Jiao Tong University. He began his Ph.D. study in the field of Operations Research at the University of Texas at Arlington in Aug, 2010. His research interests include operations research, mathematical programming, optimization algorithms, data mining, statistics and machine learning methods.