Algorithms for Partially Observable Markov Decision Processes

by

Hsien-Te Cheng

B.Sc., National Taiwan University, 1975

M.Sc., National Taiwan University, 1977

M.Sc., University of Idaho, 1979

A Thesis Submitted in Partial Fulfillment of

the Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate Studies

Faculty of Commerce and Business Administration

We accept this thesis as conforming

to the required standard

The University of British Columbia

August 1988

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Commerce

The University of British Columbia
Vancouver, Canada

Date December 21, 1988

ABSTRACT

The thesis develops methods to solve discrete–time finite–state partially observable Markov decision processes. For the infinite horizon problem, only discounted reward case is considered. Several new algorithms for the finite horizon and the infinite horizon problems are developed.

For the finite horizon problem, two new algorithms are developed. The first algorithm is called the relaxed region algorithm. For each support in the value function, this algorithm determines a region not smaller than its support region and modifies it implicitly in later steps until the exact support region is found. The second algorithm, called linear support algorithm, systematically approximates the value function until all supports in the value function are found. The most important feature of this algorithm is that it can be modified to find an approximate value function. The number of regions determined explicitly by both algorithms is the same as the number of supports in the value function, which is much less than the number of regions generated by the one–pass algorithm. Since the vertices of each region have to be found, these two algorithms are more efficient than the one–pass algorithm. The limited numerical examples also show that both methods are more efficient than the existing algorithms.

For the infinite horizon problem, it is first shown that the approximation version of linear support algorithm can be used to substitute the policy improvement step in a standard successive approximation method to obtain an ϵ–optimal value function. Next, an iterative discretization procedure is developed which uses a small number of states

to find new supports and improve the value function between two policy improvement steps. Since only a finite number of states are chosen in this process, some techniques developed for finite MDP can be applied here. Finally, we prove that the policy improvement step in iterative discretization procedure can be replaced by the approximation version of linear support algorithm.

The last part of the thesis deals with problems with continuous signals. We first show that if the signal processes are uniformly distributed, then the problem can be reformulated as a problem with finite number of signals. Then the result is extended to where the signal processes are step functions. Since step functions can be easily used to approximate most of the probability distributions, this method can be used to approximate most of the problems with continuous signals. Finally, we present some conditions which guarantee that the linear support can be computed for any given state, then the methods developed for finite signal cases can be easily modified and applied to problems for which the conditions hold.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I am indebted to Professor Shelby Brumelle for introducing me to the subject of partially observable Markov decision processes and for guiding this research. His suggestions, advice, patience, and financial support have been invaluable to this work. I am deeply grateful.

I am grateful to Professor Martin L. Puterman for introducing me to the computational aspect of dynamic programming and Markov decision processes. His comments on these topics were always useful.

I wish to thank Professor John Hughes and Carl Walters, who reviewed earlier drafts and offered many comments that resulted in improved clarity of presentation in the final draft of the thesis.

I am grateful to Dr. Sondik for providing the computer code for the one-pass algorithm, and Professor Rubin of the University of North Carolina for providing the computer code for the Mattheiss algorithm.

I also want to thank my friends for their encouragement during this work. Special thanks go to Mei-Jean Goh and Margaret Judd for proof-reading my thesis.

I would also like to thank my family for their encouragement and financial support during my study years. The greatest debt of gratitude goes to my wife Lily for willingly sharing the frustrations, and for the encouragement and limitless patience exhibited during this work.

CHAPTER 1

INTRODUCTION

After Bellman (1957) and Howard (1960) introduced the dynamic programming and Markov decision process models, the Markov Decision Process (MDP) has received much attention in operations research. It has been applied to a wide range of problems.

A common situation is that a problem can be formulated as a the Markov decision process in all respects except that the states of the system are not fully observable and significant costs may be incurred to get this information. The following maintenance and repair problem, similar to the one discussed in Smallwood and Sondik (1973), is illustration of this fact.

A machine is used to produce a particular product. Only one product is produced at each time period. This machine has two identical components, each of which must operate once upon the product before it is finished. The life of each component has an exponential distribution. The lifetime of one component is independent of the other. There is a positive probability that an operational component will break down in the process of manufacturing a product. If both components function well, then there is only a small chance that this machine produces a defective product. However, if either component fails, there is a higher probability that the machine will produce a defective product.

Since both components are identical, we can model the dynamics of the machine with a three–state discrete time Markov process. The three states correspond to zero,

one, or two components having failed. Generally, we do not know whether a component is in a good or compromised condition.

In order to know the state of the machine, we may stop its operation for inspection. However, most of the time, we can expect the machine to be functioning well, and hence such an action will increase the costs unnecessarily. Alternatively, the decision maker may choose to continue production or to examine the final products. Continuing production will not give us any information about the state of the system. Examining the final product, though, will give us the probability distribution of the state of the system by Bayes' rule. These two options will not tell us the exact state of the system under study. Therefore, even if we indeed have a Markov decision process, we may not know the system state when we choose an action.

A problem that can be formulated using the MDP but which suffers from an imperfect state observation is usually referred to as a *Partially Observable Markov Decision Process* (POMDP). More precisely, a partially observable Markov decision process is a generalized Markov decision process which allows an imperfect observation of the system states.

As a descriptive model, the POMDP offers many advantages over MDP. For example, the POMDP allows an imperfect state observation, which happens often in the real world. The observation process may be non–Markovian. The POMDP model also forces the model builder to make a clear distinction between real systems and observations. Moreover, similar to what seems to occur in practice, an action or decision taken in

POMDP may affect the quality of future observations. Hence, the POMDP has been used to model a wide range of problems which will be presented later.

As a prescriptive tool, however, the POMDP is awkward. The decision maker may be forced to make decisions based upon the entire history of the system, a string of past decisions and observations. The POMDP is usually converted into an equivalent completely observable MDP, where the state space is the conditional probability distribution of the system state given the history of the system (Åstorm 1965). This conversion facilitates analysis of the problem, but does not overcome computational difficulties introduced by the state space being continuous and not finite.

The aim of this thesis is to develop some efficient solution methods for POMDP's. The development of the POMDP will be reviewed in Section I of this chapter, while the plan and major results of this thesis will be summarized in Section II.

I. Development of POMDP

The Partially Observable Markov Decision Process (POMDP) is a natural extension of the MDP. Research on the POMDP began in the early sixties, just a few years after Howard's work (1960). Drake (1962) was the first person who developed the explicit POMDP model. About the same time, Åstorm (1965, 1969) formulated the finite horizon POMDP.

Several researchers have studied the theory of POMDP. Sawaragi and Yoshikawa

(1970) studied the POMDP with an uncountable action space and a countable system states. Rhenius (1974) considered POMDP problems with both action and system state spaces being Borel spaces. White and Harrington (1980) studied the relationship between the value functions, observation quality, and suboptimal decisions. Platzman (1980) developed the conditions under which the undiscounted infinite horizon POMDP is well defined.

Some researchers studied conditions that ensure the optimal policies have certain structural characteristics. Albright (1979) presented conditions under which the optimal policy of a two system state POMDP is monotone on the probability distribution of the system states. White (1980) gave conditions which yield monotone optimal policies for finite horizon POMDP that is either completely observable or nonobservable. Recently, Lovejoy (1987) provided sufficient conditions for the optimal value in a discrete time, finite POMDP to be monotone on the space of state probability vectors ordered by likelihood ratios.

The computational difficulties associated with POMDP's have been under study since the mid–sixties. Kakalik (1965) divided the space of state probability vectors into equal area grids and considered each grid as a state. The problem was then transformed into a finite MDP problem. Satia and Lave (1973) developed an implicit enumeration algorithm for computing an ϵ–optimal value function for the finite horizon POMDP. Smallwood and Sondik (1973) and Sondik (1971) developed a so–called one–pass algorithm for the finite horizon POMDP, and discovered that POMDP's are not as computationally intractable as general nondenumberable state MDP's. Sondik (1971,

1978) developed a policy iteration type algorithm for the infinite horizon POMDP. Brumelle and Sawaki (1978) presented a partition method for finite horizon POMDP and a modified policy iteration algorithm for the infinite horizon POMDP. Platzman (1981) discussed a finite memory algorithm to find a ϵ–optimal policy for infinite horizon POMDP. Recently, White and Scherer (1986) developed a reward revision algorithm to solve infinite horizon problems.

There are some algorithms developed only for special cases of the POMDP. Wang (1976, 1977) considered a two action replacement problems. Buckman and Miller (1979) reformulated an investigation problem as a regenerative stopping problem. Hughes (1980) reformulated a two–action, two–state sequential quality control model as a renewal problem. We will discuss some of the algorithms in more detail later in this thesis.

POMDP have been used to model a wide range of problems (Monahan 1982). One of the major applications is the machine replacement and quality control problems. Eckles (1968), Ohnishi et al. (1984, 1986), Ross (1971), Wang (1976, 1977), White (1977, 1978, 1979a, 1979b) applied POMDP to different settings of machine replacement and quality control problems. Kaplan (1969) applied the results of the machine inspection and replacement problem to a cost control problem in accounting. Hughes (1977) modeled the internal control of a corporate control system as a POMDP. Smallwood (1971) developed a two state POMDP model of optimal teaching strategies. Smallwood et al. (1971) used POMDP concepts in the development of methodology for the analysis of health care system. White (1976) applied the theory of POMDP to design question-

naires in situations where responses may not be truthful. Hsu and Marcus (1980) used the POMDP as a tool to solve the decentralized control of finite Markov processes. Eagle (1984) used the POMDP to study searching for the moving target. Recently, Lovejoy (1983) and Lane (1986a, 1986b) have applied the POMDP to fishery problems.

II. Summary of Results and Plan of the Thesis

The major results of this thesis can be divided into three parts: algorithms for the finite horizon POMDP, algorithms for the infinite horizon POMDP, and methods for solving the POMDP with continuous signals. Each of these sets of results is discussed below:

1. Algorithms for Finite Horizon POMDP:

Two new algorithms are developed for solving finite horizon POMDP problems. The first algorithm, called the relaxed region algorithm, is a modification of Sondik's one–pass algorithm. Instead of finding an exact support region in the state space which corresponds to a linear support for the value function, a larger relaxed region which contains this exact region is found. In later steps, this relaxed region is modified. At the end of the procedures, the regions corresponding to each of the linear supports in the value function are found exactly. Unlike Sondik's one–pass algorithm, the number of regions produced by this method is exactly the same as the number of linear supports in the value function. In other words, the number of regions produced by this method is much smaller than that of Sondik's one–pass algorithm. Since fewer regions are

produced, this method is much more efficient than the Sondik method.

The second new algorithm is a linear support algorithm. This algorithm can be viewed as a special type of relaxed region algorithm, although it is different from the relaxed region algorithm discussed above. This algorithm uses only the convexity and piecewise–linearity of POMDP. This method systematically approximates the value function until all linear supports in the value function are found. One of the most important features of this algorithm is that it can be used to find an ϵ–optimal value function. Although this algorithm may not be the only algorithm which can find an approximate value function, it might be the only algorithm which can be used for finding both exact and approximate value functions. The numerical examples show that this is a very efficient algorithm for finding an exact value function. More importantly, when there are a large number of linear supports to form a value function, this algorithm requires only a fraction of CPU time to find an approximate value function which is very close to the optimal one. Because of its ability to find an approximate value function efficiently, this algorithm also is used in the development of several methods in the infinite horizon POMDP.

2. Algorithms for Infinite Horizon POMDP:

Several new algorithms are developed in this part.

In regular successive approximation method, the exact value function is found in each iteration. As we know, when a large number of linear supports are required to form a value function, it is difficult to find an exact value function, no matter which

algorithm is used. A natural resolution to this difficulty is to apply an approximate value function for each policy improvement and hope an ϵ–optimal value function for infinite horizon POMDP can be obtained. We will prove that this approach is workable in Chapter 4; i.e., an ϵ–optimal value function can be obtained by repeatedly applying the approximate policy improvement step. This is the first result in this part.

Another possible method for overcoming the difficulty of the standard successive approximation approach is to reduce the number of iterations of policy improvement. The method developed under this category introduces a discrete phase between two policy improvement steps. In each of the discrete phases, only a small set of states are considered. In each iteration in a discrete phase, linear supports corresponding to these selected states are computed. The maximal value of newly computed linear supports and the linear supports in the previous value function form a new value function. By performing iterations in a discrete phase, the value function can be improved without complicating computations. We called this method the *iterative discretization procedure.*

Since only a finite number of states are considered in a discrete phase, some techniques developed for the finite MDP can be applied, at lease in concept, to iterative discretization procedures to accelerate the convergence. Three methods are considered. They are the Gauss–Seidel method, the action elimination method, and the modified policy iteration method.

In the iterative discretization procedure introduced here, the exact value function is required for each iteration of policy improvement. A natural extension is to replace

this exact value function by an approximate value function since an approximate value function is easier to compute and numerically more stable. It is shown that the iterative discretization procedure can work with an approximate policy improvement.

3. Algorithms for POMDP with Continuous Signal Space:

In this part, the assumption of only a finite number of signals is relaxed. It is assumed that there is a probability distribution of signals for each system state and action.

There is little research in this area, especially with respect to a general purpose algorithm. The major difficulty for developing such an algorithm is that the property of piecewise linearity of the value function which is available in the finite signal setting is not preserved in the setting. This feature raises computational difficulties.

It is first proven that, if all signal processes are uniformly distributed, the problem can be reformulated as a finite signal problem. As a result, all the algorithms for the finite signal problems can then be applied.

This result is then extended to step functions; that is, if the signal processes are step functions, the problem can be reformulated as a finite signals problem. Although there may not be many problems in which signal processes are step functions, step functions can be easily applied to approximate any distribution. Therefore, we can use this approach to solve most problems.

The assumption of finite signals is only used to guarantee that it is possible to find

the linear support for any given state in the algorithms discussed in Part 1 and Part 2. If there is a method which guarantees that linear supports can be obtained for the given states, then the algorithms can be applied to problems with continuous signals without major changes. Some conditions are developed that guarantee the existence of a linear support for any given state. If these conditions are met, then the methods developed in previous two parts can be used in here too.

The plan for the remainder of the thesis follows.

Chapter 2: Problem Formulation and Preliminary Results

The formal problem setting for a POMDP is introduced. Next the problem is reformulated as a completely observable MDP with continuous state space. The properties of this newly formulated MDP, which will be used in later chapters, are then discussed.

Chapter 3: Algorithms for Finite Horizon POMDP

This chapter discusses the algorithms for finite horizon POMDP. Major existing algorithms, the partition method, Sondik's one–pass algorithm, and the Monahan method, are presented and reviewed. Then two new algorithms, the relaxed region algorithm and the linear support algorithm, are developed. Some computational results are used to compare the efficiency of these algorithms.

Chapter 4: Algorithms for Infinite Horizon POMDP

This chapter discusses the computational methods for infinite horizon POMDP problems and is organized as follows. First, the major existing algorithms are reviewed.

Next a successive approximation method with approximate policy improvement to obtain an ϵ–optimal value function and an iterative discretization procedure are developed. Then various methods to accelerate convergence are considered. Then some of the previous ideas are combined to apply approximate policy improvement with iterative successive approximation. Finally, a numerical comparison of algorithms is presented.

Chapter 5: POMDP with Continuous Signal Distribution;

This chapter first considers the refomulation of a problem with uniformly distributed signal processes to a problem with a finite number of signals. Then this procedure is extended to signal processes with distributions which are step functions. Some conditions which guarantee that a linear support to the value function at any given state can be constructed are stated. Finally, the algorithms developed in the previous two chapters are adapted to this setting.

CHAPTER 2

PROBLEM FORMULATION AND PRELIMINARY RESULTS

In this chapter, the formal problem setting for a POMDP will be introduced. Then the problem is transformed to a completely observable MDP. Lastly, the properties to be used in later chapters are discussed.

I. The Partially Observable Markov Decision Processes

Underlying a POMDP is assumed to be a discrete–time finite–state Markov chain. This process has N stage–invariant states labelled $1, 2, \ldots, N$. In order to distinguish states for the underlying process from the states in the decision problem discussed later, the states in the underlying process are called system states. Time is divided into discrete periods labelled by nonnegative integers t. At any time epoch t the system state is denoted by X_t.

At each decision time epoch on a stage, the decision maker has to choose an action from an action space D. The action chosen at time t is denoted by Y_t. In this thesis, the action space is assumed finite.† It is also assumed that the action space D is stage–invariant for an infinite horizon stage problem.

† The assumption of a finite action space is not necessary for the model development. Sawaragi and Yoshikawa (1970) developed the theory of POMDP's with an uncountable action space and a countable system state space. However, in order to develop efficient computational algorithms, we limit our attention to a finite action space.

At decision epoch t, if the process is in system state i and the decision maker chooses action d, then a set of transition probabilities $\{p_{ij}^d(t);\ j = 1, 2, \ldots, N\}$ ($\sum_j p_{ij}^d(t) = 1$ for any system state i) is established to describe the probability of moving to system state j at the next time epoch; that is, $p_{ij}^d(t) = \Pr[X_{t+1} = j \mid X_t = i, Y_t = d]$. If the transition probability is independent of time t or it is clear from content, the index t is omitted. For an infinite horizon problem, stationary transition probabilities are assumed. It is also assumed that the decision maker knows the transition probability function.

A finite reward $r^d(i)$ is obtained whenever the system is in system state i and action d is chosen. The reward is stage–invariant. The notation r^d is a N–dimensional column vector with $r^d(i)$ as its i–th element.

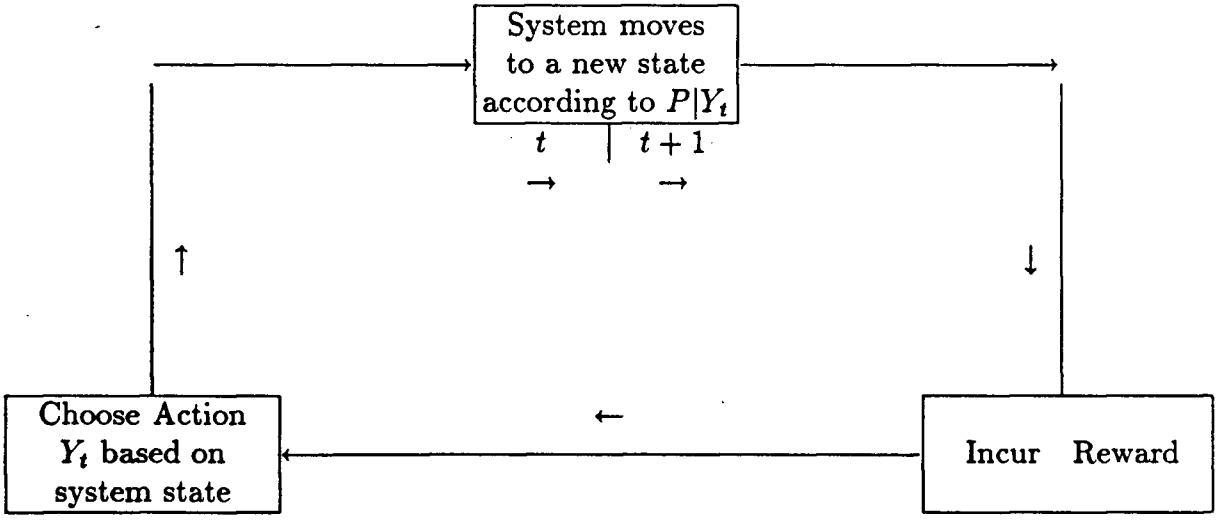

Fig. 2.1: Decision diagram for completely observable Markov decision processes

If the decision maker knows the current system state before choosing an action,

the decision problem is a regular completely observable MDP. The decision maker can, then, choose an action based on the current system state to maximize the total expected (discounted) reward. To make the decision procedure clear, consider the block diagram of Fig. 2.1.

POMDP's have the characteristic that the state of the system cannot be observed directly. Instead, the decision maker receives a random signal from the system. In Chapter 2 to 4, the signal space Θ is assumed finite. This assumption will be relaxed in Chapter 5.

Let the signal received at time t be denoted by Z_t. Although the signal is random, the decision maker knows the conditional probability of getting signal θ when the system state and action are known. Assuming that this conditional probability is stage invariant, it can be denoted by a matrix $Q^d = [q^d_{i\theta}]$, where

$$q^d_{i\theta} = \Pr[Z_t = \theta \mid X_t = i, Y_{t-1} = d] \qquad \forall t.$$

Assume that the objective of the decision model is to maximize the total expected (discounted) reward. Since the decision maker does not know the exact system state of the process when choosing an action, he/she may choose an action based on the entire history of the process. (This will be discussed in next section.) To contrast Fig. 2.1, consider the block diagram for POMDP in Fig. 2.2.

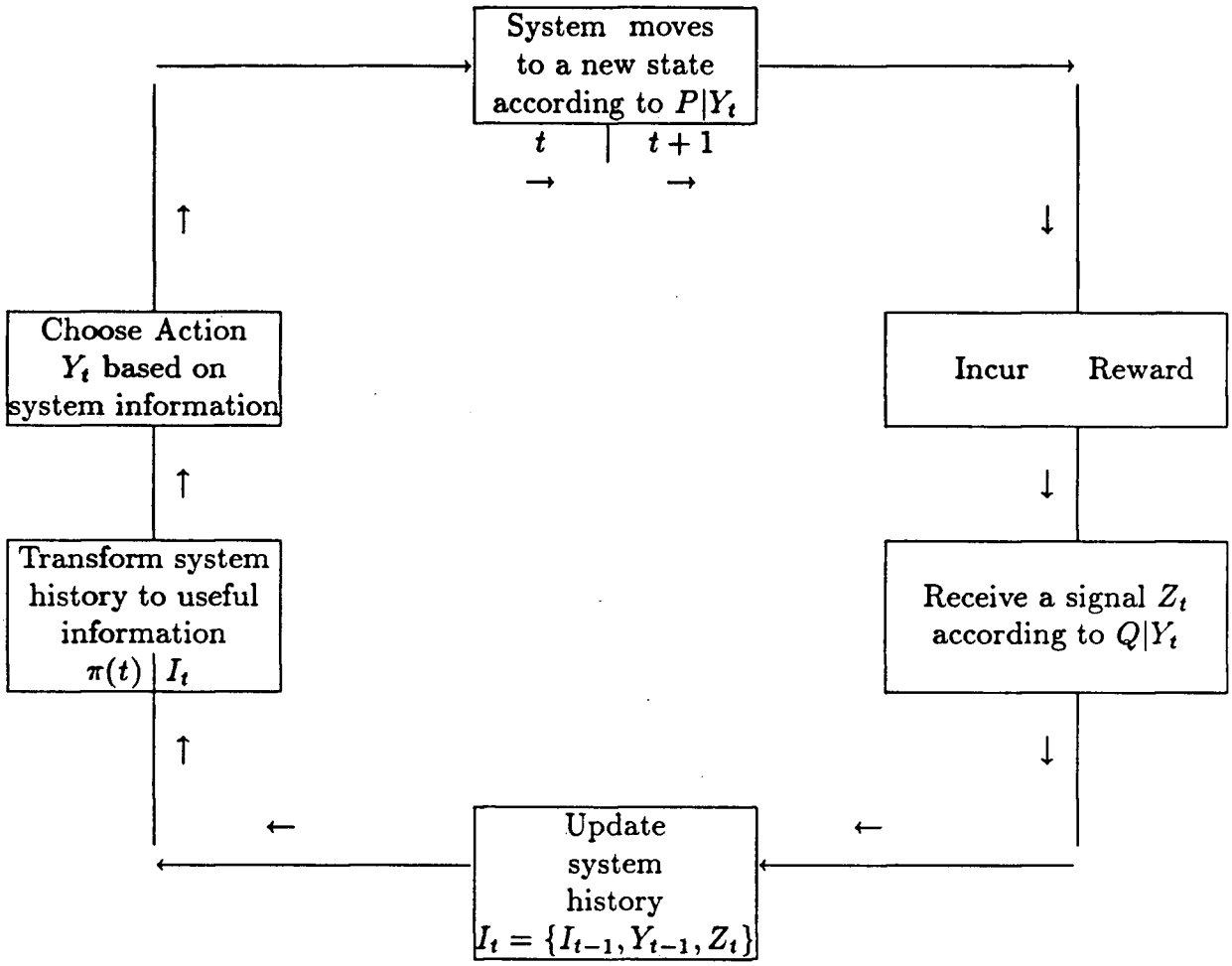

Fig. 2.2: Decision diagram for partially observable Markov decision processes.

II. Problem Formulation

For a POMDP problem, the system state is usually unknown to the decision maker at the time of choosing an action. Moreover, since the signal process itself need not be a Markov process, it may be necessary to base decisions on more than just the signal received in epoch t. In this section, new formulations will be considered to resolve this difficulty.

Let $\pi(0) = [\pi_1(0), \pi_2(0), \ldots, \pi_N(0)]$ be the probability distribution of the initial system states where $\pi_i(0) = \Pr\{X_0 = i\}$ for $1 = 1, 2, \ldots, N$. Assume $\pi(0)$ is known by the decision maker. The history of the process up to time t is denoted as I_t where

$$I_0 = [\pi(0), Z_0]$$

$$I_t = [\pi(0), Z_0, Y_0, Z_1, Y_1, \ldots, Z_{t-1}, Y_{t-1}, Z_t]$$

$$\text{and} \quad I_{t+1} = [I_t, Y_t, Z_{t+1}].$$

We remark that $\{I_t,\ t = 0, 1, 2, \ldots\}$ is a Markov decision process. Let $v_t(\cdot)$ denote the optimal total expected reward from time t to the end of the decision horizon. The dynamic programming recursive equation can now be written as

$$\begin{aligned} v_t(I_t) &= \max_{Y_t \in D} \mathrm{E}\{r^{Y_t}(X_t) + \beta \cdot v_{t+1}(I_{t+1}) \mid I_t, Y_t\} \\ &= \max_{Y_t \in D} \{\mathrm{E}(r^{Y_t}(X_t) \mid I_t, Y_t) + \beta \cdot \mathrm{E}(v_{t+1}(I_{t+1}) \mid I_t, Y_t)\} \\ &= \max_{Y_t \in D} \{\sum_{i=1}^{N} \Pr(X_t = i \mid I_t) \cdot r^{Y_t}(i) \\ &\qquad + \beta \cdot \sum_{\theta \in \Theta} \Pr(Z_{t+1} = \theta \mid I_t, Y_t) \cdot v_{t+1}([I_t, Y_t, Z_{t+1} = \theta])\} \end{aligned} \qquad (2-1)$$

where β is a discount factor. We assume that $0 < \beta < 1$ for infinite horizon problems and $0 < \beta$ for finite horizon problems. Note that the state variables are known exactly in equation (2–1). By Bayes' rule and induction, it is also possible to find the value of $\Pr(Z_{t+1} = \theta \mid I_t, Y_t)$ for any given θ, I_t and action. Therefore, we have a completely observable Markov decision process.

In order to solve the recursive equation (2–1), the value functions in period t must be evaluated for each possible history I_t. When there are a large number of possible actions

and/or signals or the decision horizon is long, then the number of possible histories I_t is large (or infinite) and grows linearly in t. The computational requirements of this dynamic programming algorithm can be truly prohibitive. (The details of this problem are discussed in Section 4.1 of Bertsekas (1976).) Therefore, we should consider other possible state variables.

Let $\pi_i(t) = \Pr(X_t = i \mid I_t)$ and $\pi(t) = [\pi_1(t), \pi_2(t), \ldots, \pi_N(t)]$ where $\sum_{i=1}^{N} \pi_i(t) = 1$ and $0 \leq \pi_i(t) \leq 1$ for $i = 1, 2, \ldots, N$. It is easy to show

$$\Pr(X_{t+1} = i \mid I_t, Y_t, Z_{t+1}) = \Pr(X_{t+1} = i \mid \pi(t), Y_t, Z_{t+1})$$

$$\text{and} \quad \Pr(Z_{t+1} = \theta \mid I_t, Y_t) = \Pr(Z_{t+1} = \theta \mid \pi(t), Y_t).$$

Therefore, given I_t, $Y_t = d$, and $Z_{t+1} = \theta$, by Bayes' rule,

$$\begin{aligned}
\pi_j(t+1) &= \Pr(X_{t+1} = j \mid I_t, Y_t = d, Z_{t+1} = \theta) \\
&= \Pr(X_{t+1} = j \mid \pi(t), Y_t = d, Z_{t+1} = \theta) \\
&= \frac{\Pr(X_{t+1} = j, Z_{t+1} = \theta \mid \pi(t), Y_t = d)}{\Pr(Z_{t+1} = \theta \mid \pi(t), Y_t = d)} \\
&= \frac{\sum_{i=1}^{N} \pi_i(t) \cdot p_{i,j}^d \cdot q_{j,\theta}^d}{\sum_{k=1}^{N} \sum_{i=1}^{N} \pi_i(t) \cdot p_{i,k}^d \cdot q_{k,\theta}^d}.
\end{aligned} \tag{2-2}$$

Or, in matrix form,

$$\begin{aligned}
T(\pi(t), d, \theta) &\equiv \Pr(X_{t+1} \mid I_t, Y_t = d, Z_{t+1} = \theta) \\
&= \Pr(X_{t+1} \mid \pi(t), Y_t = d, Z_{t+1} = \theta) \\
&= \frac{\pi(t) \cdot P^d \cdot Q_\theta^d}{\pi(t) \cdot P^d \cdot Q_\theta^d \cdot \mathbf{1}}
\end{aligned} \tag{2-3}$$

where Q_θ^d is a diagonal matrix with $q_{j,\theta}^d$ as its diagonal elements and $\mathbf{1}$ is an N-dimensional column vector with all elements being 1.

It is well known that $\pi(t)$ is a sufficient statistic for I_t when choosing an action at time epoch t. More precisely, $\pi(t)$ summarizes all of the necessary information of the history of the process for choosing an action at time t (Bertsekas 1976, Monahan 1982, Sondik 1971, Striebel 1965). Let $\Pi \equiv \{\pi \in R^N : \sum_{i=1}^{N} \pi_i = 1 \text{ and } \pi_i \geq 0 \text{ for } i = 1, 2, \ldots, N\}$. The following result is also readily established (Åstrom 1965, Monahan 1982, Rhenius 1974, Sawaragi and Yoshikawa 1970, and Sondik 1971): for any fixed sequence of actions $Y_1, Y_2, \ldots, Y_t \in D$, the sequence of probabilities $\{\pi(k)\}$ where $k = 0, 1, \ldots, t$ is a Markov process; that is, if $\Gamma \subset \Pi$, then

$$\Pr(\pi(t+1) \in \Gamma \mid \pi(0), \pi(1), \ldots, \pi(t), Y_t) = \Pr(\pi(t+1) \in \Gamma \mid \pi(t), Y_t).$$

Therefore, consider using the probability distribution of the system states as the state variables; that is, using Π as the new state space. Then, for $\pi \in \Pi$,

$$\begin{aligned} v_t(\pi) &= \max_{d \in D} \; \mathrm{E}\{r^d(X_t) + \beta \cdot v_{t+1}(T(\pi, d, \theta)) \mid \pi, d\} \\ &= \max_{d \in D} \{\sum_{i=1}^{N} \pi_i \cdot r^d(i) + \beta \cdot \sum_{\theta \in \Theta} \Pr(Z_{t+1} = \theta \mid \pi, d) \cdot v_{t+1}(T(\pi, d, \theta))\}. \end{aligned} \tag{2-4}$$

Note that, in this representation, the state variable is known exactly; and hence (2–4) is the recursive equation of a completely observable Markov decision process. Thus, the POMDP can be converted into an equivalent (completely observable) Markov decision process.

When using I_t as the state variables, the state space is different in each stage. In contrast, using $\{pi(t), t = 0, 1, \ldots\}$ as the state variables, the state space is always

the same, Π. The stage invariant state space aids the problem analysis; however, to find an optimal policy is still not an easy task because the state space Π is continuous and not finite. In general, it is not easy to solve a Markov decision process with a continuous state space. Fortunately, POMDP's possess some special properties which will be discussed later in this chapter. These properties will help us to develop some useful computational algorithms to find an optimal policy.

III. Notations and Operators

In this section, some notation and operators useful for later chapters are introduced.

As mentioned in previous sections, Π is the *state space* and D the *action space*. A *policy* is a function which maps the state space into the action space; i.e., if δ is a policy, then $\delta : \Pi \rightarrow D$ where $\delta(\pi)$ is the action taken in state $\pi \in \Pi$. Let the *policy space* Δ be the set of all stationary policies. Let B be the set of all bounded real-valued functions on Π. In this thesis, unless otherwise stated, the norm $\|\cdot\|$ is the supreme norm; for example, if $v \in B$, then $\|v\| = \sup\{|v(\pi)| : \ \pi \in \Pi\}$.

For convenience, define the *local income function* h which assigns a real number to each triple (π, d, v) with $\pi \in \Pi$, $d \in D$, and $v \in B$. In Chapter 2 to 4, h is defined as

$$h(\pi, d, v) = \pi \cdot r^d + \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, d) \cdot v(T(\pi, d, \theta)).$$

Note that with this notation Equation (2–4) can be rewritten as

$$v_t(\pi) = \max_{d \in D} h(\pi, d, v_{t+1}).$$

The local income function can, then, generate a *return operator* H_δ on B for each $\delta \in \Delta$; i.e., $[H_\delta(v)](\pi) = h(\pi, \delta(\pi), v)$. If $\delta(\pi) = d$ for all $\pi \in \Pi$, then H_d instead of H_δ is used. Finally an *optimal operator* $H : B \to Bv$ is defined as $Hv = \max_{\delta \in \Delta}[H_\delta v]$.

IV. Major Properties

In the previous sections, the POMDP was converted into an equivalent completely observable Markov decision process. In this section, the more important properties of this new Markov decision process are considered.

The first three properties, boundedness, monotonicity, and contraction, are commonly assumed in dynamic programming. It is shown that these properties are also present in a POMDP. Then, two further properties of the POMDP, i.e., piecewise linearity and convexity of the value functions, are discussed. Finally, the existence of an optimal value function and policy is also discussed.

1. Boundedness:

The usual assumption of boundedness is as follows: *There exist numbers* K_1 *and* K_2 *such that* $\|H_\delta v\| \leq K_1 + K_2 \cdot \|v\|$ *for all* $v \in B$ *and* $\delta \in \Delta$ (Whitt 1978).

Let $r = \max_{d \in D}\{|r^d(i)| : \ i = 1, 2, \ldots, N\}$, then for arbitrary $\pi \in \Pi$, $\delta \in \Delta$, and $v \in B$

$$|[H_\delta v](\pi)| = |\pi \cdot r^d + \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot v(T(\pi, \delta(\pi), \theta))|$$

$$
\begin{aligned}
&\leq |\pi \cdot r^d| + \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot |v(T(\pi, \delta(\pi), \theta))| \\
&\leq r + \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot \|v\| \\
&= r + \beta \cdot \|v\|
\end{aligned}
$$

Therefore, $\|H_\delta v\| \leq r + \beta \|v\|$ for all $v \in B$ and $\delta \in \Delta$; that is, both H_δ for all $\delta \in \Delta$ and H satisfy the boundedness assumption.

2. Monotonicity:

The monotonicity assumption is as follows: *If $v \geq u$ in B, i.e., if $v(\pi) \geq u(\pi)$ for all $\pi \in \Pi$, then $H_\delta v \geq H_\delta u$ in B for all $\delta \in \Delta$* (Whitt 1978). Now assume $v \geq u$ in B, then for arbitrary $\delta \in \Delta$ and $\pi \in \Pi$,

$$
\begin{aligned}
(H_\delta v)(\pi) - (H_\delta u)(\pi) &= \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot v(T(\pi, \delta(\pi), \theta)) \\
&\quad - \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot u(T(\pi, \delta(\pi), \theta)) \\
&= \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot \{v(T(\pi, \delta(\pi), \theta)) - u(T(\pi, \delta(\pi), \theta))\} \\
&\geq 0
\end{aligned}
$$

Therefore, $(H_\delta v)(\pi) \geq (H_\delta u)(\pi)$ or $H_\delta v \geq H_\delta u$ for all $\pi \in \Pi$ and $\delta \in \Delta$; that is, both H_δ for all $\delta \in \Delta$ and H satisfy the monotonicity assumption.

3. Contraction:

The contraction property is one of the most important properties in a problem with an infinite horizon. This property can be stated thus: *For some fixed β, $0 \leq \beta < 1$, then $\|H_\delta v - H_\delta u\| \leq \beta \cdot \|v - u\|$ for all $u, v \in B$ and $\delta \in \Delta$* (Whitt 1978).

For arbitrary $u, v \in B$ and $\pi \in \Pi$,

$$\begin{aligned}|(H_\delta v)(\pi) - (H_\delta u)(\pi)| &\leq \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot |v(T(\pi, \delta(\pi), \theta)) - u(T(\pi, \delta(\pi), \theta))| \\ &\leq \beta \cdot \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, \delta(\pi)) \cdot \|v - u\| \\ &= \beta \cdot \|v - u\|\end{aligned}$$

Hence $\|H_\delta v - H_\delta u\| \leq \beta \cdot \|v - u\|$ for all $u, v \in B$ and $\delta \in \Delta$; that is, both H_δ for all $\delta \in \Delta$ and H satisfy the contraction mapping assumption.

4. Piecewise Linearity of the Value Function:

The piecewise linearity and convexity of the value function are the two of the most important properties of a POMDP. Piecewise linearity was first discovered by Åstrom (1965), but was formally introduced by Sondik (1971). Let us briefly describe the piecewise linearity of a POMDP.

The following is Lemma 2.1 of Sondik (1971):

Lemma 2.1:

$T(\pi, d, \theta)$ preserves straight lines. That is, if $0 \leq \rho \leq 1$, and $\pi^1, \pi^2 \in \Pi$, then $\rho \cdot \pi^1 + (1-\rho) \cdot \pi^2$ is a straight line in Π with end points π^1, π^2. If the transformation of this line for a fixed signal θ and action d is considered, then $T(\rho \cdot \pi^1 + (1-\rho) \cdot \pi^2, a, \theta) = \mu \cdot T(\pi^1, d, \theta) + (1-\mu) \cdot T(\pi^2, d, \theta)$, where, as ρ ranges between 0 and 1, μ ranges between 0 and 1. Hence the image of a straight line under $T(\cdot, d, \theta)$ is a straight line, specifically,

$$\mu = \frac{\rho \cdot \Pr(\theta \mid \pi^1, d)}{\rho \cdot \Pr(\theta \mid \pi^1, d) + (1-\rho) \cdot \Pr(\theta \mid \pi^2, d)}.$$

A function v is called piecewise linear and convex if there is a finite set of N–dimensional column vectors, A, such that $v(\pi) = \max_{i=1,2,\ldots,k}\{\pi \cdot \alpha^i : \text{ where } \alpha^i \in A\}$. For $\pi \in \Pi$, $d \in D$, and $\theta \in \Theta$, define $A_{\pi,d,\theta} = \{\hat{\alpha} \in A : \ T(\pi,d,\theta)\cdot\hat{\alpha} \geq T(\pi,d,\theta)\cdot \alpha \ \forall\ \alpha \in A\}$. Then $v(T(\pi,d,\theta)) = T(\pi,d,\theta)\cdot\alpha$ for all $\alpha \in A_{\pi,d,\theta}$. If $\alpha_{\pi,d,\theta}$ is a vector in $A_{\pi,d,\theta}$, then, by (2–3) and (2–4), $Hv(\pi)$ can be written as:

$$\begin{aligned} H_d v(\pi) &= \pi \cdot r^d + \beta \cdot \sum_{\theta\in\Theta} \Pr(\theta \mid \pi, d)\cdot v(T(\pi,d,\theta)) \\ &= \pi \cdot r^d + \beta \cdot \sum_{\theta\in\Theta} (\pi \cdot P^d \cdot Q^d_\theta \cdot \mathbf{1}) \cdot \frac{\pi \cdot P^d \cdot Q^d_\theta}{\pi \cdot P^d \cdot Q^d_\theta \cdot \mathbf{1}} \cdot \alpha_{\pi,d,\theta} \\ &= \pi \cdot [r^d + \beta \cdot \sum_{\theta\in\Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\pi,d,\theta}]; \end{aligned} \tag{2-5}$$

Since $r^d + \beta \cdot \sum_{\theta\in\Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\pi,d,\theta}$ is an N–dimensional vector, (2–5) can be simplified as $H_d v(\pi) = \pi \cdot \alpha_{\pi,d}$ where $\alpha_{\pi,d} = r^d + \beta \cdot \sum_{\theta\in\Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\pi,d,\theta}$. Moreover, $Hv(\pi) = \max_{d\in D} H_d v(\pi)$, then

$$\begin{aligned} Hv(\pi) &= \max_{d\in D}\{H_d v(\pi)\} \\ &= \max_{d\in D}\{\pi \cdot \alpha_{\pi,d}\} \\ &= \pi \cdot \alpha_\pi \end{aligned} \tag{2-6}$$

where $\alpha_\pi = r^{\hat{d}} + \beta \cdot \sum_{\theta\in\Theta} P^{\hat{d}} \cdot Q^{\hat{d}}_\theta \cdot \alpha_{\pi,\hat{d},\theta}$ if $\hat{d}$ is an optimal action for state π.

As described in Lemma 2.1, $T(\cdot,d,\theta)$ preserves straight lines. As a result of this property and the assumption of the piecewise linearity of v with a finite number of linear segments, for any given action d in D and signal θ in Θ, the state space Π can be partitioned into a finite number of regions such that $A_{\pi,d,\theta}$ is constant for all states π in the interior of each region. Moreover, if there are only finite numbers of actions

in D and signals in Θ, a finer partition with a finite number of cells can be found such that $A_{\pi,d,\theta} = A_{\pi',d,\theta}$ for all $d \in D$, and $\theta \in \Theta$, given π and π' are in the same region. Therefore, if $Hv(\hat{\pi}) = \hat{\pi} \cdot \alpha_{\hat{\pi}}$, then $Hv(\pi) = \pi \cdot \alpha_{\hat{\pi}}$ for all π in the same region as $\hat{\pi}$ in the finer partition. That is, Hv is a piecewise linear function.

Note that, if the number of signals is not finite, the state space Π might be partitioned into an infinite number of regions. In this case, Hv might not be a piecewise linear function with a finite number of linear segments. However, the convexity is preserved even in this case. We will discuss this type of problem in Chapter 5.

5. Convexity of the Value Function:

The convexity of the value function for a system with a finite number of system states, actions, and signals was first presented and proven formally by Åstrom (1969). His theorem can be simply described as: *Given a convex function v, Hv is also convex.* Note that this theorem can only be applied directly to the optimal operator. For an arbitrarily given policy $\delta \in \Delta$, $H_\delta v$ is not necessarily convex even though v is a convex function. However, in the proof of this theorem, the fact that $H_d v$ is convex for all $d \in D$ is proven. This is conceptually important for developing an algorithm for a finite horizon problem.

White and Harrington (1980) extended Åstrom's results to a more general setting where the signal space is not necessarily finite. In Lemma 3.1 of their paper, they showed that *$H_d v$ is convex whenever v is, even if the signal space is not finite.* Since the maximal value of a set of convex functions is convex function, Hv is also a convex

function.

Therefore, if v is piecewise linear and convex, so is Hv. Moreover, from piecewise linearity, $Hv(\pi) = \pi \cdot \alpha^i$ for π in region i. Suppose there are k' regions and let $A_H = \{\alpha^1, \alpha^2, \ldots, \alpha^{k'}\}$. Since Hv is piecewise linear and convex, the function value of Hv can be written as

$$Hv(\pi) = \max\{\pi \cdot \alpha^i : \ \alpha^i \in A_H\} \quad \forall \pi \in \Pi. \tag{2-7}$$

That is, if the vectors in A_H are known, then, by piecewise linearity and convexity of Hv, the function value of Hv can be easily obtained by (2–7). It is not necessary to know the area of each region explicitly.

6. Optimal Value Function and Policy:

Since the operators H_δ and H are contraction mappings, for each operator, there exists an unique fixed point in B, denoted by v_δ and v^*, respectively, such that $H_\delta v_\delta = v_\delta$ and $Hv^* = v^*$ (Él'gol'c 1964). This implies that, if the initial state is $\pi \in \Pi$ and policy $\delta \in \Delta$ is used over the infinite horizon, the total expected discount reward will be $v_\delta(\pi)$. Then, by definition of H, $v^*(\pi) = \max_{\delta \in \Delta}\{v_\delta(\pi)\}$ for all π in Π. Hence, v^* is the optimal value function.

A policy $\delta \in \Delta$ which attains $H_\delta v^* = Hv^*$ is called an *optimal policy.* Sawaragi and Yoshikawa (1970) have shown that if the action space D is finite, there exists an optimal stationary policy. A policy δ is said to be ϵ–optimal if $v_\delta \leq v^*$ and $\|v^* - v_\delta\| \leq \epsilon$. For any given $\epsilon > 0$, there exists an ϵ–optimal stationary policy even when the action space D is not finite (Sawaragi and Yoshikawa 1970).

Since a POMDP has a continuous state space, it is usually very difficult to find the v_δ for $\delta \in \Delta$ or v^*. Therefore, bounds for these values have been sought by several researchers.

Åstrom (1965) studied the finite horizon problem in an optimal control setting. He found that the value function for a partially observable control system is always better than that of an unobservable control system (or open–looped system). In contrast, the value function of a partially observable system is no better than that of a completely observable system.

White and Harrington (1980) extended Åstrom's theorem to compare the results of different qualities of measurement. They concluded that if the quality of the observation is improved, then the value function will also be improved. This result is true even when the optimal policy for the lower quality system is applied to the better measurement system.

CHAPTER 3

ALGORITHMS FOR FINITE HORIZON POMDP

Finite horizon algorithms are important not only to solve finite horizon problems but also for use as the policy improvement step for an infinite horizon problem. Hence an efficient algorithm is desired.

If a one–stage POMDP problem can be solved, then by induction, a finite horizon problem can be solved. Therefore, in this chapter, the main focus will be on how to compute Hv for a given piecewise linear and convex function v. We will start with a characterization of v in term of its supports, and then compute a set of supports for Hv.

Let v be a piecewise linear and convex function and A a given support set containing a finite number of N–dimensional column vectors such that

$$v(\pi) \geq \pi \cdot \alpha \quad \forall \alpha \in A \text{ and } \pi \in \Pi$$

$$\text{and} \qquad v(\pi) = \max\{\pi \cdot \alpha : \ \alpha \in A\} \quad \forall \pi \in \Pi.$$

Therefore, every vector in A is a support of v. The support sets discussed in this dissertation usually satisfy the following condition: for every α in a support set A, there exist a state π in Π such that $v(\pi) = \pi \cdot \alpha$. The supports in a support set A which do not satisfy this condition can be deleted from A without changing the value function. In this dissertation, this condition is not a necessary one in the development of algorithms. However, delete the unnecessary supports will improve the efficiency of the algorithms.

A region, $R(\alpha, A)$, is called a *support region* for a support $\alpha \in A$ if

$$R(\alpha, A) = \{\pi \in \Pi : \ \pi \cdot \alpha \geq \pi \cdot \tilde{\alpha} \quad \forall \tilde{\alpha} \in A\}.$$

The support regions have the following two properties: (1) $\cup_{\alpha \in A} R(\alpha, A) = \Pi$; and (2) $\text{int}R(\alpha, A) \cap \text{int}R(\hat{\alpha}, A) = \emptyset$ when $\alpha \neq \hat{\alpha}$, where intR(α, A) is the the interior of $R(\alpha, A)$. Hence, $v(\pi) = \pi \cdot \alpha$ if $\pi \in R(\alpha, A)$.

As discussed in Chapter 2, Hv is also piecewise linear and convex. To represent Hv easily, a support set, A_H, is required such that

$$Hv(\pi) \geq \pi \cdot \alpha \quad \forall \alpha \in A_H \text{ and } \pi \in \Pi$$

$$\text{and} \qquad Hv(\pi) = \max\{\pi \cdot \alpha : \ \alpha \in A_H\} \quad \forall \pi \in \Pi.$$

There are an infinite number of support sets which satisfy these two conditions. Among them, the smallest set, denoted as A_H^*, is the one for which the interior of the support region for each support in A_H^* is not empty. By definition of piecewise linearity, A_H^* is a finite set. The smallest set is desired since it will simplify the calculation of $Hv(\pi)$ for any given $\pi \in \Pi$.

Although the smallest set is desired, it might be difficult to find since identifying an empty interior of a region is a time–consuming task. Each of the algorithms discussed below will generate an alternative set of supports. Recall that, for given $\pi \in \Pi$, $d \in D$ and $\theta \in \Theta$, $A_{\pi,d,\theta} = \{\hat{\alpha} : T(\pi, d, \theta) \cdot \hat{\alpha} \geq T(\pi, d, \theta) \cdot \alpha \quad \forall \alpha \in A\}$. Define A_H† as

$$A_H = \{\alpha : \ \exists \pi \in \Pi, d \in D \text{ and a selection of } \alpha_{\pi,d,\theta} \in A_{\pi,d,\theta} \text{ for each } \theta \in \Theta$$

† A_H depends on v and A. For simplification, the dependence will not be shown on the notation of A_H.

$$\text{such that } \alpha = r^d + \beta \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \cdot \alpha_{\pi,d,\theta} \text{ and } Hv(\pi) = \pi \cdot \alpha\}.$$

Note that A_H is a finite set and A_H^* is a subset of A_H. A_H is equal to A_H^* most of the time; however, occasionally some unnecessary supports will be included in A_H. These unnecessary supports affect the operations of the algorithms by imposing additional computations that are time–consuming to detect. However, it may be inefficient to delete these vectors from A_H.

The main purpose of this chapter is to discuss two methods, the relaxed region algorithm and the linear support algorithm, for computing Hv and A_H from given v and A. These two algorithms are closely related to existing algorithms which will also be discussed. A partition method will be presented in Section I. Then the two well known algorithms, the one–pass algorithm and the Monahan algorithm, will be reviewed in Sections II and III. The relaxed region algorithm will be discussed in Section IV, and, in Section V, the linear support algorithm is developed. Some numerical examples will be presented in Section VI for comparing the efficiency of these algorithms.

I. Partition Method

The partition method was introduced by Brumelle and Sawaki (1978) for general piecewise linear dynamic programming and by Sawaki (1983) for the POMDP. When being used for the POMDP, this method is similar to Sondik's one–pass algorithm (Sondik 1971, Smallwood and Sondik 1973).

A collection $B = \{B_1, B_2, \cdots, B_n\}$ of subsets of Π is a partition of state space Π

if B_i is a convex polyhedron, $\cup_{i=1}^{n} B_i = \Pi$ and $\text{int}(B_i) \cap \text{int}(B_j) = \emptyset$ where $i \neq j$ and $\text{int}(B_i)$ is the interior of B_i. The product of two partitions C_1 and C_2 is $C_1 \bigotimes C_2 = \{B_i \cap D_j : \ B_i \in C_1 \text{ and } D_j \in C_2\}$. The product of $C_1, C_2, \cdots, C_m$ is defined inductively by $\bigotimes_{i=1}^{m} C_i = C_m \bigotimes \bigotimes_{i=1}^{m-1} C_i$.

If d is an action in D, θ is a signal in Θ, and $\hat{\alpha}$ is a support in A, define $S_{d,\theta,\hat{\alpha}}$ as

$$S_{d,\theta,\hat{\alpha}} = \{\pi \in \Pi : \ T(\pi, d, \theta) \cdot \hat{\alpha} \geq T(\pi, d, \theta) \cdot \alpha \quad \forall \alpha \in A\}.$$

Note that $S_{d,\theta,\hat{\alpha}}$ is a convex polyhedron and might be empty. Then $S_{d,\theta} = \{S_{d,\theta,\alpha} : \forall \alpha \in A\}$ is a partition of the state space Π. A product partition can be formed from these partitions $S_{d,\theta}$ for all $\theta \in \Theta$; i.e., $S^d = \bigotimes_{\theta \in \Theta} S_{d,\theta}$. Denote the cells in S^d by $S^d = \{S_1^d, S_2^d, \ldots, S_l^d\}$.

Recall that $H_d v$ is the value of using action d for one period for any $\pi \in \Pi$ and terminating with reward v. Now we will construct a set of supports, A_d, such that there is a support ξ^{di} for each cell S_i^d in S^d. By (2–5),

$$\begin{aligned} H_d v(\pi) &= \pi \cdot r^d + \beta \sum_{\theta \in \Theta} \Pr(\theta \mid \pi, d) \cdot v(T(\pi, d, \theta)) \\ &= \pi \cdot [r^d + \beta \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \cdot \alpha_{\pi,d,\theta}] \end{aligned} \tag{3-1}$$

where $\alpha_{\pi,d,\theta} \in A_{\pi,d,\theta}$. If π is in the interior of $S_{d,\theta,\hat{\alpha}}$, then $A_{\pi,d,\theta} = \hat{\alpha}$. Since for any given signal θ the mapping $A_{\cdot,d,\theta} : \pi \to A_{\pi,d,\theta}$ is constant on the interior of S_j^d, $r^d + \beta \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \cdot \alpha_{\pi,d,\theta}$ will be the same for the states in the interior of S_j^d. Therefore, (3–1) can be rewritten as

$$H_d v(\pi) = \pi \cdot \xi^{di} \qquad \text{for} \quad \pi \in \text{int}(S_i^d),$$

where $\xi^{di} = r^d + \beta \sum_{\theta \in \Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\pi,d,\theta}$ and is an N–dimensional column vector. Define $A_d = \{\xi^{di} : \ i = 1, 2, \ldots, l\}$. It is possible that a $\pi \in \Pi$ is on the boundaries of several regions. For example, let $\pi \in S^d_i \cap S^d_j$. Since π is a state in S^d_i, $\alpha_{\tilde{\pi},d,\theta} \in A_{\pi,d,\theta}$ where $\tilde{\pi} \in \text{int}(S^d_i)$ and $\alpha_{\tilde{\pi},d,\theta}$ is the constant vector for the states in the interior of S^d_i. Then, by the continuity of $H_d v$, $H_d v(\pi) = \pi \cdot \xi^{di}$. Similarly, $H_d v(\pi) = \pi \cdot \xi^{dj}$. Hence, no more new vectors are generated from these states on the boundaries. If A_d is known, then the region S^d_i can be simply represented as $S^d_i = \{\pi \in \Pi : \ \pi \cdot \xi^{di} \geq \pi \cdot \xi \quad \forall \xi \in A_d\}$ and $H_d v(\pi) = \max\{\pi \cdot \alpha : \ \forall \alpha \in A_d\}$.

It is possible to find the partition S^d for each d in D. A product partition can be formed from these partitions S^d; i.e., $S = \bigotimes_{d \in D} S^d$. Since there are only a finite number of cells in S^d for every $d \in D$ and a finite number of actions in D, S contains a finite number of cells. Denote the cells in S as $S = \{B_1, B_2, \ldots, B_n\}$. For each action d, S is finer than S^d. Let $\alpha^{di} = \xi^{dj}$ if B_i is a subset of S^d_j, then $H_d v(\pi) = \pi \cdot \alpha^{di}$ for all $\pi \in B_i$ where $i = 1, 2, \ldots, n$. Note that if both B_i and B_j are subsets of S^d_m, then α^{di} equals α^{dj}.

To find $Hv = \max_{d \in D} H_d v$, define the convex polyhedrons G_{di} for $i = 1, 2, \ldots, n$ and $d \in D$ by

$$G_{di} = \{\pi \in B_i : \quad \pi \cdot \alpha^{di} \geq \pi \cdot \alpha^{ai} \quad \forall \ a \in D\}.$$

Note that G_{di} might be empty. If G_{di} is not empty, then for $\pi \in G_{di}$ the action d is optimal and

$$Hv(\pi) = \pi \cdot \alpha^{di}.$$

Therefore, α^{di} is a support of Hv and should be included in A_H if G_{di} is not empty. Let $U_i = \{G_{di}: \quad d \in D\}$. Then U_i is a partition of B_i and $U = \bigcup_{i=1}^{n}(U_i \setminus \emptyset)$ is a partition of Π. The sets in U can be rearranged as $U = \{R_1, R_2, \ldots, R_m\}$. Define $A_H = \{\alpha^{di}: d \in D, i = 1,2,\ldots,n, \text{ and } G_{di} \neq \emptyset\}$. Then, $Hv(\pi) = \max\{\pi \cdot \alpha: \quad \alpha \in A_H\}$ and we have finished the construction of the supports of Hv.

It is worth mentioning that the number of supports in A_H is usually less than the number of cells in U. This can be easily seen from the following example. Assume $R_a = G_{di} \subseteq B_i$ and $R_b = G_{dl} \subseteq B_l$; moreover, assume both B_i and B_l are subsets of S_j^d. Then $\alpha^{di} = \alpha^{dl} = \xi^{dj}$; that is, the same vector corresponds to states in R_a and R_b. Let $R(\alpha^{di}, A_H)$ be the support region for the vector α^{di}, i.e., $R(\alpha^{di}, A_H) = \{\pi \in \Pi: \pi \cdot \alpha^{di} \geq \pi \cdot \alpha \ \forall \alpha \in A_H\}$, then $R_a \cup R_b \subseteq R(\alpha^{di}, A_H)$.

II. One–Pass Algorithm

In this section, we will study Sondik's one–pass algorithm. The one–pass algorithm will produce the same partition U and support set A_H as the partition method discussed in Section I; however, the approaches of these two methods are different.

In the previous section, U is a partition of the state space Π. Now let us focus on a cell $G_{\hat{d}i}$ in U. Recall that

$$G_{\hat{d}i} = \{\pi \in B_i: \ \pi \cdot \alpha^{\hat{d}i} \geq \pi \cdot \alpha^{di} \quad \forall d \in D\}.$$

That is, a state π is in $G_{\hat{d},i}$ if and only if the following two conditions are satisfied:

(i) $\pi \cdot \alpha^{\hat{d}i} \geq \pi \cdot \alpha^{di}$ for all $d \in D$;

(ii) $\pi \in B_i$.

The first condition implies that $\hat{d}$ is the optimal action for the states in $G_{\hat{d},i}$ and $\alpha^{\hat{d}i}$ is a vector in A_H. Since

$$B_i = \bigcap_{d \in D} \{\pi \in \Pi : \quad \pi \cdot \alpha^{di} \geq \pi \cdot \xi \quad \forall \xi \in A_d\},$$

the second condition is similar to finding $\pi \in \Pi$ satisfying the following set of constraints

$$\pi \cdot \alpha^{di} \geq \pi \cdot \xi^d \quad \forall \ \xi^d \in A_d \quad \text{and} \ d \in D.$$

Sondik (1971) and Smallwood and Sondik (1973) discussed a simple method to represent these two conditions. Let

$$A_{\pi,d} = \{\alpha : \ \alpha = r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\pi,d,\theta} \quad \text{where} \quad \alpha_{\pi,d,\theta} \in A_{\pi,d,\theta}\}.$$

Then $H_d v(\pi) = \pi \cdot \alpha$ for all $\alpha \in A_{\pi,d}$. Let $\hat{\pi}$ be an arbitrary state in $G_{\hat{d},i}$, then $\alpha^{di} \in A_{\hat{\pi},d}$. Since $\hat{\pi} \cdot \alpha$ is a scalar and the values $\hat{\pi} \cdot \alpha$ are the same for all $\alpha \in A_{\hat{\pi}d}$, the optimal action(s) for the state $\hat{\pi}$ are

$$D_{\hat{\pi}} = \{\tilde{d} : \quad \tilde{d} = \arg\max_{d \in D} \{\hat{\pi} \cdot \alpha^{di}\}\}.$$

Note that $D_{\hat{\pi}}$ might contain more than one action; however, $\hat{d}$ is in $D_{\hat{\pi}}$. Moreover, it is possible that $A_{\hat{\pi},\hat{d}}$ contains more than one vector, but $\alpha^{\hat{d}i}$ is in $A_{\hat{\pi},\hat{d}}$. First, assume that there is only one action, $\hat{d}$, in $D_{\hat{\pi}}$ and one vector, $\alpha^{\hat{d}i}$, in $A_{\hat{\pi},\hat{d}}$. This assumption will be relaxed later.

Now consider the second condition. The constraint set to represent the second condition might not seem easy to set up since all the vectors in A_d for all $d \in D$ have to be known even though only the region $G_{\hat{d},i}$ is considered. Fortunately, a simple representation can be developed. For simplification, assume that there is only one vector in $A_{\hat{\pi},d}$ for each $d \in D$. This assumption will be relaxed later.

Let $\alpha^{di} = r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\hat{\pi},d,\theta}$. Since $\{\alpha_{\hat{\pi},d,\theta}\} = A_{\pi,d,\theta}$ for all $\pi \in \text{int}(\text{B}_\text{i})$, then by definition of $A_{\pi,d,\theta}$, for all $\pi \in B_i$,

$$T(\pi,d,\theta) \cdot \alpha_{\hat{\pi},d,\theta} \geq T(\pi,d,\theta) \cdot \alpha \qquad \forall \alpha \in A,$$

$$\text{or} \qquad \pi \cdot P^d \cdot Q^d_\theta \cdot \alpha_{\hat{\pi},d,\theta} \geq \pi \cdot P^d \cdot Q^d_\theta \cdot \alpha \qquad \forall \alpha \in A. \qquad (3-2)$$

Therefore, if π is in B_i, then (3–2) holds for all $\theta \in \Theta$ and $d \in D$.

Conversely, if (3–2) holds for all $\theta \in \Theta$ and $d \in D$, then, for $\hat{\pi} \in B_i$,

$$\pi \cdot P^d \cdot Q^d_\theta \cdot \alpha_{\hat{\pi},d,\theta} \geq \pi \cdot P^d \cdot Q^d_\theta \cdot \alpha_{\tilde{\pi},d,\theta} \qquad \forall \theta \in \Theta$$

where $\tilde{\pi}$ is an arbitrary state in Π but not in B_i. Summing over all $\theta \in \Theta$, we obtain

$$\sum_{\theta \in \Theta} \pi \cdot P^d \cdot Q^d_\theta \cdot \alpha_{\hat{\pi},d,\theta} \geq \sum_{\theta \in \Theta} \pi \cdot P^d \cdot Q^d_\theta \cdot \alpha_{\tilde{\pi},d,\theta}$$

$$\text{or} \qquad \pi \cdot [r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\hat{\pi},d,\theta}] \geq \pi \cdot [r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\tilde{\pi},d,\theta}].$$

Since $r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q^d_\theta \cdot \alpha_{\tilde{\pi},d,\theta}$ is a vector in A_d, hence

$$\pi \cdot \alpha^{di} \geq \pi \cdot \xi^d \qquad \forall \xi^d \in A_d.$$

Therefore, $\pi \in \Pi$ is in B_i if and only if (3–2) holds for for all $\theta \in \Theta$ and $d \in D$;† that is, $B_i = \bigcap_{d \in D}\{\pi \in \Pi : \ \pi \cdot P^d \cdot Q^d_\theta \cdot (\alpha_{\hat{\pi},d,\theta} - \alpha) \geq 0 \quad \forall\ \theta \in \Theta \text{ and } \alpha \in A\}$. The region $G_{\hat{d},i}$ can now be represented as

$$G_{\hat{d},i} = \{\pi \in \Pi : \quad \pi \cdot \alpha^{\hat{d}i} \geq \pi \cdot \alpha^{di} \text{ and } \pi \cdot P^d \cdot Q^d_\theta \cdot (\alpha_{\hat{\pi},d,\theta} - \alpha) \geq 0 \quad \forall\ d \in D,\ \theta \in \Theta,\ \text{and } \alpha \in A\}. \qquad (3-3)$$

If there is more than one vector in $A_{\hat{\pi},d,\theta}$ for some $d \in D$ and $\theta \in \Theta$, then consider all possible selections of one vector, say $\alpha_{\hat{\pi},d,\theta}$, from each $A_{\hat{\pi},d,\theta}$. If there is only one optimal action for $\hat{\pi}$, then, by using (3–3), each selection $\{\alpha_{\hat{\pi},d,\theta} : \ d \in D, \theta \in \Theta\}$ can be used to determine a region. Note that $G_{\hat{d},i}$ is one of the generated regions, and all of these regions contain $\hat{\pi}$. Moreover, the intersection of the interior of any two of these generated regions is empty.

If $D_{\hat{\pi}}$ has i actions, i.e. there are i optimal actions corresponding to the state $\hat{\pi}$, then, for each selection of $\{\alpha_{\hat{\pi},d,\theta} : \ d \in D, \theta \in \Theta\}$, by changing the optimal action, i regions can be determined. Therefore, if there are j ways of selecting $\{\alpha_{\hat{\pi},d,\theta}\}$, then there are $i \cdot j$ regions and $G_{\hat{d},i}$ is one of the regions.

Although this method may be inefficient for determining a particular region $G_{\hat{d},i}$, all these regions generated are also cells in U. Having discovered this property, Sondik

† In fact, as discussed in Smallwood and Sondik (1973), since $T(\cdot, d, \theta)$ preserves straight lines, only those α's whose support regions have common boundaries with the support region for $\alpha_{\hat{\pi},d,\theta}$ have to be considered.

(1971) and Smallwood and Sondik (1973) developed a systematic method, the one–pass algorithm, to find all vectors in A_H.

The one–pass algorithm uses an arbitrary state as an initial point. With this initial point, an optimal action (or actions) and an optimal vector (or vectors) for this point are found. Using (3–3), a region (or regions) can be obtained. All vertices of these generated region(s) are then determined. Similar to the procedure discussed above, each generated vertex is used to determine new regions and their corresponding vertices. Each vertex is used to determine regions once and the regions which generate this vertex need not be determined again. There are only a finite number of regions in U and each region only has a finite number of vertices. Hence, the algorithm will terminate in a finite number of iterations. Since the states in the interior of any region belong to exactly one region, the algorithm is called one–pass algorithm.

As with the partition method, the one–pass algorithm has the major disadvantage of dividing the support regions into several subregions. As mentioned in Smallwood and Sondik (1973), most of the computational time in the one–pass algorithm is spent in solving linear programs to determine the vertices of the regions. The computational efficiency can be improved by not partitioning the support regions. This is the motivation for the development of the relaxed region algorithm which will be discussed in Section IV.

III. The Monahan Algorithm

Monahan (1982) proposed an algorithm which he called Sondik's one–pass algorithm. Since the basic idea of this algorithm is different from the one–pass algorithm discussed above, we will consider his algorithm separately.

As described in Section I, if $\pi \in S_i^d$, then $H_d v(\pi) = \pi \cdot \xi^{di}$ where $\xi^{di} = r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \cdot \alpha_{\pi,d,\theta}$. Therefore, in order to find ξ^{di}, we need to know $\alpha_{\pi,d,\theta}$ for each θ in Θ and a state π in S_i^d. As discussed in Section II, to find the region S_i^d is not an easy task. Let us consider an alternative approach. Instead of finding $\alpha_{\pi,d,\theta}$ for some π, an arbitrary α_θ from A is chosen for each $\theta \in \Theta$ and the vector $\tilde{\alpha} = r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \cdot \alpha_\theta$. is calculated. If the vectors $\{\alpha_\theta, \theta \in \Theta\}$ are chosen so that $\alpha_\theta = \alpha_{\pi,d,\theta}$, then $\tilde{\alpha}$ equals ξ^{di}. Let $\tilde{A}_d$ be the set of all possible $\tilde{\alpha}$'s; i.e.,

$$\tilde{A}_d = \{\tilde{\alpha} : \ \tilde{\alpha} = r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \cdot \alpha_\theta \quad \text{where } \alpha_\theta \in A\}.$$

Therefore, $\xi^{di} \in \tilde{A}_d$ and $A_d \subseteq \tilde{A}_d$. This implies that $\tilde{A}_d$ might contain some unnecessary supports of $H_d v$, and for all $\pi \in \Pi$

$$\begin{aligned} H_d v(\pi) &= \max\{\pi \cdot \alpha : \quad \alpha \in A_d\} \\ &= \max\{\pi \cdot \alpha : \quad \alpha \in \tilde{A}_d\}. \end{aligned}$$

Note that, if there are K actions in D, L signals in Θ, and M supports in A, then at most there are M^L vectors in $\tilde{A}_d$ and $K \cdot M^L$ vectors in $\bigcup_{d \in D} \tilde{A}_d$.

Since $A_H \subseteq \bigcup_{d \in D} A_d$, then $A_H \subseteq \bigcup_{d \in D} \tilde{A}_d$. A support α in $\bigcup_{d \in D} \tilde{A}_d$ is in A_H if and only if the support region of α is not empty. Therefore, to find whether if a vector

α in $\bigcup_{d\in D}\tilde{A}_d$ is also in A_H, the support region $R(\alpha, \bigcup_{d\in D}\tilde{A}_d) = \{\pi \in \Pi : \quad \pi\cdot\alpha \geq \pi\cdot\tilde{\alpha} \quad \text{for each } \tilde{\alpha} \in \bigcup_{d\in D}\tilde{A}_d\}$ is determined. Note that $R(\alpha, \bigcup_{d\in D}\tilde{A}_d)$ is the same as $R(\alpha, A_H)$ if α is a support in A_H and empty otherwise. Hence, if $R(\alpha, \bigcup_{d\in D}\tilde{A}_d)$ is not empty, then α is a support in A_H. A slight modification by Eagle(1984) which eliminate some unnecessary vectors by dominance arguments can determine the vectors in A_H more efficiently.

The Monahan algorithm is simple to use. However, when the number of actions, signals, or supports in A is large, the number of vectors in $\bigcup_{d\in D}\tilde{A}_d$ is also large. In this case, it is time-consuming to determine the supports in A_H.

IV. Relaxed Region Algorithm

As discussed in Sections I and II, the major disadvantage of the partition method and the one-pass algorithm is that a support region for a support in A_H is usually divided into several subregions. The aim of this section is to develop a relaxed region algorithm which can reduce the number of generated regions. In fact, the number of regions found by this relaxed region algorithm is exactly the same as the number of supports in A_H.

Let $\hat{\pi}$ be an arbitrary state in Π. As discussed in Section II, $A_{\hat{\pi},d}$ for each $d \in D$ and $D_{\hat{\pi}}$ can be easily found. Now choose an arbitrary $\hat{d} \in D_{\hat{\pi}}$, and $\hat{\alpha} \in A_{\hat{\pi},\hat{d}}$. Then, by definition, the support region for $\hat{\alpha}$, $R(\hat{\alpha}, A_H)$, is

$$R(\hat{\alpha}, A_H) = \{\pi \in \Pi : \quad \pi\cdot\hat{\alpha} \geq \pi\cdot\alpha \quad \text{for each} \quad \alpha \in A_H\}$$

$$= \{\pi \in \Pi : \quad \pi \cdot \hat{\alpha} \geq \pi \cdot \alpha^d \quad \text{for each} \quad \alpha^d \in A_d \quad \text{and} \quad d \in D\}.$$

Therefore, a state $\pi \in \Pi$ is in $R(\hat{\alpha}, A_H)$ if and only if π satisfies the following three sets of constraints:

(i) $\pi \cdot \hat{\alpha} \geq \pi \cdot \alpha^d$ for all $\alpha^d \in A_{\hat{\pi},d}$ and $d \in D$;

(ii) $\pi \cdot \hat{\alpha} \geq \pi \cdot \xi^{\hat{d}}$ for all $\xi^{\hat{d}} \in A_{\hat{d}}$;

(iii) $\pi \cdot \hat{\alpha} \geq \pi \cdot \xi^d$ for all $\xi^d \in A_d$ and $d \in D$, but $d \neq \hat{d}$.

Since $A_{\hat{\pi},d}$ for each $d \in D$ has already been found, the first set of constraints is easy to set up. By the construction of $A_{\hat{\pi},d}$, a selection of $\alpha_{\hat{\pi},\hat{d},\theta}$ for all θ in Θ is known such that $\hat{\alpha} = r^{\hat{d}} + \beta \cdot \sum_{\theta \in \Theta} P^{\hat{d}} \cdot Q_\theta^{\hat{d}} \cdot \alpha_{\hat{\pi},\hat{d},\theta}$ for some special $\alpha_{\hat{\pi},\hat{d},\theta} \in A_{\hat{\pi},\hat{d},\theta} \subseteq A$. As discussed in Section II, the second set of constraints can be rewritten as

$$\pi \cdot P^{\hat{d}} \cdot Q_\theta^{\hat{d}} \cdot \alpha_{\hat{\pi},\hat{d},\theta} \geq \pi \cdot P^{\hat{d}} \cdot Q_\theta^{\hat{d}} \cdot \alpha \qquad \forall \alpha \in A \quad \text{and} \quad \theta \in \Theta.$$

Therefore, the second set of constraints can also be set up.†

In order to find the region $R(\hat{\alpha}, A_H)$, the third set of constraints has to be set up too. However, this set of constraints is difficult to set up since it requires the methods discussed in the previous sections to find all vectors in A_d for each $d \in D$. Since this set of constraints is difficult to set up, Sondik's method uses an $\alpha^d \in A_{\hat{\pi},d}$ for each $d \in D$ to substitute for $\hat{\alpha}$; i.e., the third set of constraints is replaced by

$$\pi \cdot \alpha^d \geq \pi \cdot \xi^d \qquad \text{for each } \xi^d \in A_d, \ \alpha^d \in A_{\hat{\pi},d}, \text{ and } d \in D.$$

† Similar to the footnote in Section II, only those α's $\in A$ whose corresponding support regions and the support region corresponding to $\alpha_{\hat{\pi},\hat{d},\theta}$ which have common boundaries should be considered.

Consequently, his method has to divide $R(\hat{\alpha}, A_H)$ into several smaller regions.

Since the third set of constraints is so difficult to set up, an alternative method is proposed here. Consider a relaxed region which is defined by the first two sets of constraints but not the third set of constraints. By definition of $R(\hat{\alpha}, A_H)$, $\pi \cdot \hat{\alpha} \geq \pi \cdot \alpha$ for all $\alpha \in A_H$ and $\pi \in R(\hat{\alpha}, A_H)$. Suppose that we have identified a particular set of A_H and denoted it as $\tilde{A}_H$. In order to fully utilize the available information, these supports in $\tilde{A}_H$ are used to determine a relaxed region for $\hat{\alpha}$; that is, the constraints $\pi \cdot \hat{\alpha} \geq \pi \cdot \alpha$ for each $\alpha \in \tilde{A}_H$ determine this relaxed region. A relaxed region for the support $\hat{\alpha}$, $\tilde{R}_{\hat{\alpha}}$, is defined as:

$$\begin{aligned} \tilde{R}_{\hat{\alpha}} = \{\pi \in \Pi : \quad & \pi \cdot \hat{\alpha} \geq \pi \cdot \alpha^d \quad \forall\ \alpha^d \in A_{\hat{\pi},d} \text{ and } d \in D; \\ & \pi \cdot P^{\hat{d}} \cdot Q_\theta^{\hat{d}} \cdot \alpha_{\hat{\pi},\hat{\alpha},\theta} \geq \pi \cdot P^{\hat{d}} \cdot Q_\theta^{\hat{d}} \cdot \alpha \quad \forall \alpha \in A; \\ & \pi \cdot \hat{\alpha} \geq \pi \cdot \alpha' \quad \forall\ \alpha' \in \tilde{A}_H\} \end{aligned} \qquad (3-4).$$

Since $\tilde{R}_{\hat{\alpha}}$ does not include the third set of constraints which is used to define $R(\hat{\alpha}, A_H)$, it is clear that $R(\hat{\alpha}, A_H) \subseteq \tilde{R}_{\hat{\alpha}}$. Moreover, all states $\pi \in \tilde{R}_{\hat{\alpha}}$ must satisfy the constraints $\pi \cdot \hat{\alpha} \geq \pi \cdot \alpha$ for each $\alpha \in \tilde{A}_H$. Since $\pi \cdot \alpha > \pi \cdot \tilde{\alpha}$ for all $\pi \in \text{int}(\text{R}(\alpha, \text{A}_\text{H}))$ where $\alpha, \tilde{\alpha} \in A_H$, it can be shown that $\tilde{R}_{\hat{\alpha}} \bigcap \text{int}(R(\alpha, A_H)) = \emptyset$ where $\alpha \in \tilde{A}_H$ and $\text{int}(R(\alpha, A_H))$ is the interior of the support region for support α.

Using the relaxed regions discussed above, a relaxed region algorithm can be developed to determine all supports in A_H. This algorithm starts with an arbitrary state $\hat{\pi} \in \Pi$ and an empty set $\tilde{A}_H$. The sets $A_{\hat{\pi},d}$ for each $d \in D$ and $D_{\hat{\pi}}$ are determined. The

supports in $\bigcup_{d\in D_{\hat{\pi}}} A_{\hat{\pi},d}$ are then put into $\tilde{A}_H$. According to (3–4), a relaxed region for each support in $\bigcup_{d\in D_{\hat{\pi}}} A_{\hat{\pi},d}$ is obtained, and its vertices are found. As will be discussed later, these vertices are used to find the supports in A_H and generate new relaxed regions. This procedure is repeated until no more new supports and relaxed regions are generated.

The relaxed region algorithm relies on the vertices of the generated relaxed regions to find supports in A_H. This process works as follows: Consider all vertices of a relaxed region $\tilde{R}_{\hat{\alpha}}$. Let π' be an arbitrary vertex of $\tilde{R}_{\hat{\alpha}}$. Similar to the procedure discussed before, an $A_{\pi',d}$ for each $d \in D$ can be found. Since $R(\hat{\alpha}, A_H) \subseteq \tilde{R}_{\hat{\alpha}}$, π' can either be on the boundary of $R(\hat{\alpha}, A_H)$ or not. If π' is not on the boundary of $R(\hat{\alpha}, A_H)$, then $\hat{\alpha}$ is not in $A_{\pi',d}$ for any $d \in D_{\pi'}$. Suppose $d' \in D_{\pi'}$ and $\alpha' \in A_{\pi',d'}$, then α' is a support in A_H and $\alpha' \neq \hat{\alpha}$.

If π' is on the boundary of $R(\hat{\alpha}, A_H)$ and π' is not a vertex of the state space Π, then there is a support region other than $R(\hat{\alpha}, A_H)$ such that π' is on the common boundary of this region and of $R(\hat{\alpha}, A_H)$. This implies that $\hat{\alpha} \in \bigcup_{d\in D_{\pi'}} A_{\pi',d}$ and there is at least one vector $\alpha' \in \bigcup_{d\in D_{\pi'}} A_{\pi',d}$ where $\alpha' \neq \hat{\alpha}$ and $\pi' \in R(\hat{\alpha}, A_H) \bigcap R(\alpha', A_H)$.

Whether π' is on the boundary of $R(\hat{\alpha}, A_H)$ or not, a support $\alpha' \in A_H$ and $\alpha' \neq \hat{\alpha}$ can be found. If a relaxed region for the support, $\tilde{R}_{\alpha'}$, for α', is not found previously, then a relaxed region $\tilde{R}_{\alpha'}$ can be found using (3–4). Similarly, the vertices of $\tilde{R}_{\alpha'}$ can be used to find supports in A_H. If $\tilde{R}_{\alpha'}$ has been found previously, then π' has reached the boundary of $R(\alpha', A_H)$ since $\tilde{R}_{\hat{\alpha}} \bigcap \text{int}(R(\alpha', A_H)) = \emptyset$. A relaxed region for support

α' need not be found again.

There is only a finite number of supports in A_H, and each support in A_H is used to generate one relaxed region only. Moreover, there are only a finite number of vertices for each relaxed region. The relaxed region algorithm will terminate in a finite number of iterations.

As will be shown in Theorem 3.2, the vertices of the support regions will be included in the vertices of the generated support regions. Although these vertices might not be useful in a finite horizon algorithm, they can play an important role in computing the error bound for an infinite horizon problem. This problem will be discussed in the next chapter.

The procedure for this algorithm is outlined below.

Step 0: Initialize $\tilde{A}_H$, A_H, W, E, and a point search table to empty sets. Put an arbitrary state $\pi \in \Pi$ into the point search table with an unmarked attribute.

Step 1: Proceed to Step 7 if there is no unmarked state in the point search table.

Step 2: Choose an unmarked state from the point search table. Denote this state as $\hat{\pi}$. Mark $\hat{\pi}$.

Step 3: Find $A_{\hat{\pi},d}$ for each $d \in D$ and $D_{\hat{\pi}}$. Let $W = \bigcup_{d \in D_{\hat{\pi}}} A_{\hat{\pi},d}$. If the sum of the number of vectors in W and the number of zero elements in $\hat{\pi}$ is greater than or equal to N, then put $\hat{\pi}$ into E. Put all vectors in W into $\tilde{A}_H$.

Step 4: Return to Step 1 if W is empty. Otherwise, go to Step 5.

Step 5: Choose a vector from W and denote it as $\hat{\alpha}$. Delete $\hat{\alpha}$ from W. If $\hat{\alpha}$ is not in A_H, then put it in A_H and go to Step 6. Else, go to Step 4.

Step 6: Use (3–4) to define a relaxed region $\tilde{R}_{\hat{\alpha}}$. Determine all the vertices of $\tilde{R}_{\hat{\alpha}}$. Put all vertices which do not exist in the point search table into the point search table with an unmarked attribute. Go to Step 4.

Step 7: Stop. $Hv(\pi) = \max\{\pi \cdot \alpha : \quad \alpha \in A_H\}$ for all $\pi \in \Pi$, and E contains all the vertices of the support regions.

The following two system states, three actions and two signals example illustrates the use of this algorithm.

Example:

$$P^1 = \begin{bmatrix} .8 & .2 \\ .5 & .5 \end{bmatrix} \quad Q^1 = \begin{bmatrix} .8 & .2 \\ .6 & .4 \end{bmatrix} \quad r^1 = \begin{bmatrix} -4 \\ 5 \end{bmatrix}$$

$$P^2 = \begin{bmatrix} .5 & .5 \\ .4 & .6 \end{bmatrix} \quad Q^2 = \begin{bmatrix} .9 & .1 \\ .4 & .6 \end{bmatrix} \quad r^2 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$$

$$P^3 = \begin{bmatrix} .6 & .4 \\ .3 & .7 \end{bmatrix} \quad Q^3 = \begin{bmatrix} .9 & .1 \\ .2 & .8 \end{bmatrix} \quad r^3 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$\text{And} \quad A = \{\alpha^1, \alpha^2 : \quad \text{where } \alpha^1 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}, \text{ and } \alpha^2 = \begin{bmatrix} 3 \\ 9 \end{bmatrix}\}.$$

The discount factor β is 1; that is, there is no discounting in reward. Choose $\pi = [0, 1]$ as the initial state and put it into the point search table with an unmarked attribute.

Iteration 1:

Pick $[0,1]$ from the point search table. Mark it. Denote it as $\hat{\pi}$. Then,

$$A_{\hat{\pi},1} = \{\xi^{11}\} = \{\begin{bmatrix} 0.2 \\ 11.0 \end{bmatrix}\}; \quad A_{\hat{\pi},2} = \{\xi^{21}\} = \{\begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix}\}; \quad A_{\hat{\pi},3} = \{\xi^{31}\} = \{\begin{bmatrix} 4.4 \\ 8.2 \end{bmatrix}\}.$$

Since $\hat{\pi} \cdot \xi^{11} = 11.0 > \hat{\pi} \cdot \xi^{21} > \hat{\pi} \cdot \xi^{31}$, $\hat{d} = 1$ is the optimal action for $\hat{\pi}$, and $\xi^{11} = [0.2, 11.0]^T$ is a support in A_H. Put ξ^{11} into A_H and $\tilde{A}_H$. Since $\hat{\pi}$ has a zero element, put $\hat{\pi}$ in E.

The relaxed region, $\tilde{R}_1$, for ξ^{11} can then be determined by the following set of constraints:

(i) $\pi \cdot \xi^{11} \geq \pi \cdot \xi^{21}$

$\pi \cdot \xi^{11} \geq \pi \cdot \xi^{31}$

(ii) Since $\alpha_{\hat{\pi},1,1} = \alpha^2$, and $\alpha_{\hat{\pi},1,2} = \alpha^2$, then,

$\pi \cdot P^1 \cdot Q_1^1 \cdot [\alpha^2 - \alpha^1] \geq 0$

$\pi \cdot P^1 \cdot Q_2^1 \cdot [\alpha^2 - \alpha^1] \geq 0$

(iii) $\pi \in \Pi$.

Now substitute the data into the constraints and rewrite these constraints as:

(i) $\pi \cdot \begin{bmatrix} 0.2 \\ 11.0 \end{bmatrix} \geq \pi \cdot \begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix}$

$\pi \cdot \begin{bmatrix} 0.2 \\ 11.0 \end{bmatrix} \geq \pi \cdot \begin{bmatrix} 4.4 \\ 8.2 \end{bmatrix}$

(ii) $\pi \cdot \begin{bmatrix} .8 & .2 \\ .5 & .5 \end{bmatrix} \cdot \begin{bmatrix} .8 & 0 \\ 0 & .6 \end{bmatrix} \cdot (\begin{bmatrix} 3 \\ 9 \end{bmatrix} - \begin{bmatrix} 4 \\ 5 \end{bmatrix}) \geq 0$

$\pi \cdot \begin{bmatrix} .8 & .2 \\ .5 & .5 \end{bmatrix} \cdot \begin{bmatrix} .2 & 0 \\ 0 & .4 \end{bmatrix} \cdot (\begin{bmatrix} 3 \\ 9 \end{bmatrix} - \begin{bmatrix} 4 \\ 5 \end{bmatrix}) \geq 0$

(iii) $\pi_1 + \pi_2 = 1$ and $\pi_1, \pi_2 \geq 0$

The vertices of $\tilde{R}_1$ are $[0,1]$ and $[0.27,0.73]$. Since $[0,1]$ is already in the point search table, according to the procedure, only $[0.27,0.73]$ has to be put into the point search table with an unmarked attribute.

Iteration 2:

There is only one unmarked state in the point search table, hence $\hat{\pi} = [0.27,0.73]$. Mark $\hat{\pi}$. Then,

$$A_{\hat{\pi},1} = \{\xi^{12}\} = \{\begin{bmatrix} 0.2 \\ 11.0 \end{bmatrix}\}; \quad A_{\hat{\pi},2} = \{\xi^{22}\} = \{\begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix}\}; \quad A_{\hat{\pi},3} = \{\xi^{32}\} = \{\begin{bmatrix} 4.4 \\ 8.2 \end{bmatrix}\}.$$

Since $\hat{\pi} \cdot \xi^{12} = \hat{\pi} \cdot \xi^{22} = 8.09 \geq \hat{\pi} \cdot \xi^{32}$, both ξ^{12} and ξ^{22} are in W and $\hat{\pi}$ should be in E. Since ξ^{12} is already in A_H, only ξ^{22} is put into A_H and $\tilde{A}_H$. The relaxed region, $\tilde{R}_2$, for ξ^{22} can be defined by the following constraints:

$$\text{(i)} \quad \pi \cdot \begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix} \geq \pi \cdot \begin{bmatrix} 0.2 \\ 11.0 \end{bmatrix}$$

$$\pi \cdot \begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix} \geq \pi \cdot \begin{bmatrix} 4.4 \\ 8.2 \end{bmatrix}$$

$$\text{(ii)} \quad \pi \cdot \begin{bmatrix} .5 & .5 \\ .4 & .6 \end{bmatrix} \cdot \begin{bmatrix} .9 & 0 \\ 0 & .4 \end{bmatrix} \cdot (\begin{bmatrix} 3 \\ 9 \end{bmatrix} - \begin{bmatrix} 4 \\ 5 \end{bmatrix}) \geq 0$$

$$\pi \cdot \begin{bmatrix} .5 & .5 \\ .4 & .6 \end{bmatrix} \cdot \begin{bmatrix} .1 & 0 \\ 0 & .6 \end{bmatrix} \cdot (\begin{bmatrix} 3 \\ 9 \end{bmatrix} - \begin{bmatrix} 4 \\ 5 \end{bmatrix}) \geq 0$$

$$\text{(iii)} \quad \pi_1 + \pi_2 = 1 \quad \text{and} \quad \pi_1, \pi_2 \geq 0$$

The vertices of $\tilde{R}_2$ are $[0.27,0.73]$ and $[0.78,0.22]$. Since $]0.27,0.73]$ is already in the point search table, only $[0.78,0.22]$ is put into the point search table with an unmarked attribute.

Iteration 3:

Now $[0.78,0.22]$ is the only unmarked state in the point search table, denote it as

$\hat{\pi}$. Mark $\hat{\pi}$. Then,

$$A_{\hat{\pi},1} = \{\xi^{13}\} = \{\begin{bmatrix} 0.2 \\ 11.0 \end{bmatrix}\}; \quad A_{\hat{\pi},2} = \{\xi^{23}\} = \{\begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix}\}; \quad A_{\hat{\pi},3} = \{\xi^{33}\} = \{\begin{bmatrix} 4.62 \\ 7.91 \end{bmatrix}\}.$$

Since $\hat{\pi} \cdot \xi^{33} = 5.35 > \hat{\pi} \cdot \xi^{23} > \hat{\pi} \cdot \xi^{13}$, only ξ^{33} is included in A_H. Put ξ^{33} into A_H and $\tilde{A}_H$. There is only one support corresponding to $\hat{\pi}$ and none of the elements of $\hat{\pi}$ is zero, so $\hat{\pi}$ should **not** be put into E.

Similar to $\tilde{R}_1$ and $\tilde{R}_2$, the relaxed region, $\tilde{R}_3$, for vector ξ^{33} can be determined. The vertices of $\tilde{R}_3$ are $[0.73, 0.27]$ and $[1, 0]$. Since both vertices are not in the point search table, both are put into the point search table with unmarked attribute.

Iteration 4:

Pick $[0.73, 0.27]$ from the point search table. Denote it as $\hat{\pi}$. Mark $\hat{\pi}$. Then,

$$A_{\hat{\pi},1} = \{\xi^{14}\} = \{\begin{bmatrix} 0.2 \\ 11.0 \end{bmatrix}\}; \quad A_{\hat{\pi},2} = \{\xi^{24}\} = \{\begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix}\}; \quad A_{\hat{\pi},3} = \{\xi^{34}\} = \{\begin{bmatrix} 4.62 \\ 7.91 \end{bmatrix}\}.$$

Since $\hat{\pi} \cdot \xi^{24} = \hat{\pi} \cdot \xi^{34} = 5.50 > \hat{\pi} \cdot \xi^{14}$, then ξ^{24} and ξ^{34} are both supports in A_H. However, since both supports are already in A_H, no relaxed region has to be determined and no new vertex is generated. There are two optimal vectors for $\hat{\pi}$, so $\hat{\pi}$ should be put into E.

Iteration 5:

Now, there is only one unmarked state, $[1, 0]$, in the point search table, denote it as $\hat{\pi}$. Mark it. Then,

$$A_{\hat{\pi},1} = \{\xi^{15}\} = \{\begin{bmatrix} 0.36 \\ 10.2 \end{bmatrix}\}; \quad A_{\hat{\pi},2} = \{\xi^{25}\} = \{\begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix}\}; \quad A_{\hat{\pi},3} = \{\xi^{35}\} = \{\begin{bmatrix} 4.62 \\ 7.91 \end{bmatrix}\}.$$

Since $\hat{\pi} \cdot \xi^{35} = 4.62 > \hat{\pi} \cdot \xi^{25} > \hat{\pi} \cdot \xi^{15}$, then only ξ^{35} is a support in A_H. However, ξ^{35} is equal to ξ^{33} and is in A_H already, no relaxed region or new vertex is generated. There is a zero element in $\hat{\pi}$, so $\hat{\pi}$ should be in E.

Iteration 6:

Finally, since there is no unmarked state in the point search table, the process is now completed. There are three supports, $[0.2, 11.0]^T$, $[4.0, 9.6]^T$, and $[4.62, 7.91]^T$, in A_H, and four vertices, $[0, 1]$, $[0.27, 0.73]$, $[0.73, 0.27]$, and $[1, 0]$, in E. ■

The number of relaxed regions generated from this example is three which is equal to the number of supports in A_H. For comparison, Sondik's one–pass algorithm generated five regions as shown below:

State Used	Corresponding Vector	Vertices Generated
$[0, 1]$	$[0.2, 11.0]^T$	$[0, 1], [0.27, 0.73]$
$[0.27, 0.73]$	$[4.0, 9.6]^T$	$[0.27, 0.73], [0.57, 0.43]$
$[0.57, 0.43]$	$[4.0, 9.6]^T$	$[0.57, 0.43], [0.73, 0.27]$
$[0.73, 0.27]$	$[4.62, 7.91]^T$	$[0.73, 0.27], [0.83, 0.17]$
$[0.83, 0.17]$	$[4.62, 7.91]^T$	$[0.83, 0.17], [1.0, 0]$

As in the relaxed region algorithm, the vertices of each region in Sondik's algorithm have to be identified. Since finding all vertices of a region is the most time–consuming step in both algorithms, the relaxed region algorithm requires less computational time than the one–pass algorithm because fewer regions are generated.

Some of the more salient properties of the relaxed region algorithm are discussed below.

THEOREM 3.1:

All supports in A_H^* are also in the support set A_H which is generated by the relaxed region algorithm.

Proof:

Let α be the support which is in A_H^* but not in the generated A_H.

First assume that the supports whose support regions have common boundaries with the support region $R(\alpha, A_H^*)$ are found and put in $\tilde{A}_H$. Without loss of generality, assume these supports be $\alpha^1, \alpha^2, \ldots, \alpha^k$ where the superscripts stand for the order of generated sequence. Since $\tilde{R}_{\alpha^k} \subseteq R(\alpha^k, A_H)$ and $\tilde{R}_{\alpha^k} \cap \mathrm{int}(\mathrm{R}(\alpha^{\mathrm{i}}, \mathrm{A_H})) = \emptyset$ for $i = 1, 2, \ldots, k-1$, then at least one of the extreme points of $\tilde{R}_{\alpha^k}$ is in $R(\alpha, A_H^*)$. Let this point be $\hat{\pi}$, then $\alpha \in \cup_{d \in D_{\hat{\pi}}} \{A_{\hat{\pi},d}\}$. Contradiction.

If not all supports whose support regions have common boundaries with $R(\alpha, A_H^*)$ are found, then assume that $\hat{R}$ be a set which contains $R(\alpha, A_H^*)$ and the support regions which can form a connected set with $R(\alpha, A_H^*)$ and its support is not found. Similar to the previous proof, let $\alpha^1, \alpha^2, \ldots, \alpha^k$ be a sequence of supports in $\tilde{A}_H$ and whose support regions have common boundaries with $\hat{R}$. Then at least one of the extreme points of $\tilde{R}_{\alpha^k}$ is in $\hat{R}$, then this state can be used to find at least one of the supports which have not be found. Therefore, the procedure cannot terminate. Contradiction. ■

THEOREM 3.2:

The states in E are the vertices of the support regions for supports in A_H. Conversely, all vertices of the support regions for the supports in A_H are in E.

Proof:

Let $\hat{\pi}$ be an arbitrary state in E. There is at least one support, $\hat{\alpha}$, in A_H, such that $Hv(\hat{\pi}) = \hat{\pi} \cdot \hat{\alpha}$. Without loss of generality, assume there are m supports in A_H. Then consider the following linear programming problem:

$$\max \quad \pi \cdot \hat{\alpha}$$

Subject to:

$$(1) \quad \pi \cdot \hat{\alpha} \geq \pi \cdot \alpha \qquad \text{for each } \alpha \in A_H \text{ and } \hat{\alpha} \neq \alpha$$

$$(2) \quad \pi_k \geq 0 \qquad k = 1, 2, \ldots, N$$

$$(3) \quad \sum_{k=1}^{N} \pi_k = 1$$

Note that there are $m+N$ constraints which define the support region for $\hat{\alpha}$. Since $\hat{\pi} \in E$, if $\hat{\pi}_k > 0$ for all $k = 1, 2, \ldots, N$, then there are at least N supports in A_H corresponding to $\hat{\pi}$ including $\hat{\alpha}$. Set the slack variables of the $N-1$ constraints corresponding to these supports other than $\hat{\alpha}$ to zero, then, it can be seen that $\hat{\pi}$ is a basic feasible solution of the LP. Similarly, if some $\hat{\pi}_k = 0$, then set the slack variables of these constraints $\pi_k \geq 0$ to zero. Then $\hat{\pi}$ is also a basic feasible solution of the LP. Therefore, $\hat{\pi}$ is a vertex of the support region for $\hat{\alpha}$. This proves the first part of the theorem.

Now assume $\hat{\pi}$ is a vertex of a support region $R(\hat{\alpha}, A_H)$ where $\hat{\alpha}$ is an arbitrary support in A_H. Then $\hat{\pi}$ is one of the basic feasible solutions of the LP shown above. If n elements of $\hat{\pi}$ are zero where n can be $0, 1, \ldots$ or $N-1$, then at least $N-n-1$ constraints

in (1) should be tight when π is substituted by $\hat{\pi}$. Denote the α's in the tight constraints, including $\hat{\alpha}$, as $\alpha^1, \alpha^2, \ldots, \alpha^{N-n}$ where the superscripts indicate the order of vectors in the set A_H. The vertex $\hat{\pi}$ will be placed in this point search table no later than the vertices of the relaxed region for α^{N-n}. Since $\hat{\pi} \cdot \alpha^1 = \hat{\pi} \cdot \alpha^2 = \cdots = \hat{\pi} \cdot \alpha^{N-n}$, when $\hat{\pi}$ is chosen for finding new supports in A_H, then $\{\alpha^1, \alpha^2, \ldots, \alpha^{N-n}\} \subseteq \bigcup_{d \in D_*} A_{\hat{\pi},d}$; that is, $\hat{\pi}$ is in E. ■

Before closing this section, let us compare the relaxed region algorithm and Sondik's one–pass algorithm.

The biggest difference between the relaxed region algorithm and Sondik's algorithm is in their methods of defining the regions for a support in A_H. As mentioned before, Sondik's algorithm needs to unite several subregions in order to define a support region for a support in A_H. The relaxed region algorithm will always find a region not smaller than its support region. This can be seen in the example presented in this section. There are three supports in A_H. For the first support, $[0.2, 11.0]^T$, the relaxed region algorithm and one–pass algorithm define the same region as its support region. However, for the second support, $[4.0, 9.6]^T$, the relaxed region algorithm finds a larger region initially, but the one–pass algorithm requires unification of two regions to form the support region. For the third support, $[4.62, 7.91]^T$, the relaxed region is exactly the same as its support region; but, the one–pass algorithm again requires unification of two regions to form its support region. The total number of regions generated from one–pass algorithm

is usually more than the number of supports in A_H; however, the number of regions produced in the relaxed region algorithm is always the same as the number of supports in A_H.

From a computational point of view, there is a difference in the number of constraints used to define a region between these two algorithms. In Sondik's algorithm, the constraint set includes all constraints shown in (3–3); however, in the relaxed region algorithm only those constraints in (3–3) which concern the optimal action are considered. As a result, the one–pass algorithm usually has a much larger number of constraints than the relaxed region algorithm for defining a region. It was mentioned in Smallwood and Sondik(1973) that most of the computational time in Sondik's algorithm was spent in linear programming in order to determine the boundaries and vertices of the regions. Since there are fewer constraints in each constraint set and fewer regions to be solved, the relaxed region algorithm requires less computer memory and computational time than Sondik's algorithm does.

It is not clear whether the relaxed region algorithm requires less computational time and computer memory than the Monahan algorithm. However, use of the vertices of the support regions for computing the error bound might be an important consideration for using the relaxed region algorithm as a policy improvement step in an infinite horizon problem.

V. Linear Support Algorithm

In the previous section, a relaxed region algorithm was discussed. Although the constraint sets for determining a region in this algorithm are much simpler than those of the one–pass algorithm, they are still very complicated. The original motivation for the linear support algorithm discussed in this section is to develop an algorithm which does not require complicated constraint sets.

Besides having simpler constraint sets, the linear support algorithm also has a special property which makes it more attractive. If a large number of supports in A_H are required to characterize Hv, then computing Hv and A_H is usually very time consuming regardless of which algorithm is used. In this case, an approximate solution for Hv might be tolerable if the maximal difference between the exact solution and the approximate solution is less than a given error. However, none of the algorithms discussed so far can be modified to find this kind of solution. The linear support algorithm described in this section can provide this kind of approximation; this is the most important feature of this algorithm.

Let $A_\pi = \cup_{d \in D_\pi} \{A_{\pi,d}\}$ for $\pi \in \Pi$. Then A_π is a set of supports for Hv at π. In order to have a finite set of supports which characterize Hv, only those supports in A_π for all $\pi \in \Pi$ are considered. Recall that, if $\hat{\pi}$ is an arbitrary state in Π and $\alpha \in A_{\hat{\pi}}$, then $Hv(\pi) \geq \pi \cdot \alpha$ for all $\pi \in \Pi$ and $Hv(\hat{\pi}) = \hat{\pi} \cdot \alpha$.

The basic idea of the linear support algorithm can be described as follow. Let $\tilde{A}_H$

be a finite set of supports for Hv and define $\tilde{H}v(\pi) = \max\{\pi \cdot \alpha : \alpha \in \tilde{A}_H\}$ for all $\pi \in \Pi$. If $\tilde{H}v$ is used to approximate the value function Hv, then the maximal error for this approximation is $\max_{\pi \in \Pi}\{Hv(\pi) - \tilde{H}v(\pi)\}$. If this error is not zero, then a "proper" support can be chosen and included in $\tilde{A}_H$ to arrive at a better approximation. This procedure is repeated until the approximation is within a tolerable range. If the exact solution is needed, then the procedure can be repeated until the exact function is found. The major problem is how to choose a "proper" support.

The algorithm starts with finding the linear supports corresponding to each of the extreme points of the state space Π. The generated supports are put into $\tilde{A}_H$. In order to know how good this approximation is, a relaxed region for a support $\hat{\alpha}$ in $\tilde{A}_H$ is defined as

$$\tilde{R}_{\hat{\alpha}} = \{\pi \in \Pi : \quad \pi \cdot \hat{\alpha} \geq \pi \cdot \alpha \quad \forall\ \alpha \in \tilde{A}_H\}.$$

Note that the support region $R(\hat{\alpha}, A_H)$ is a subset of $\tilde{R}_{\hat{\alpha}}$. If there are k supports currently in $\tilde{A}_H$, then there will be k relaxed regions. All vertices of these relaxed regions are found.† If $\max_{\alpha \in \tilde{A}_H}\{\pi \cdot \alpha\}$ is used to approximate $Hv(\pi)$ for all $\pi \in \Pi$, then the error of this approximation can be defined as a function g where $g(\pi) = Hv(\pi) - \max_{\alpha \in \tilde{A}_H}\{\pi \cdot \alpha\}$ for all $\pi \in \Pi$. As shown in the following lemma and theorem, the maximal error of this approximation, i.e., the maximal value of g in Π, will be at

† In fact, only the vertices of $k-1$ regions have to be found. The vertices of the k-th region can be found in the vertices of other regions with the exception of the vertex which generates the support corresponding to this relaxed region.

one of the vertices of these relaxed regions.

LEMMA 3.1:

Let $\tilde{R}_{\hat{\alpha}}$ be the relaxed region for a support $\hat{\alpha} \in \tilde{A}_H$. The maximal value of g in $\tilde{R}_{\hat{\alpha}}$ will occur at one of the vertices of $\tilde{R}_{\hat{\alpha}}$.

Proof:

For $\pi \in \tilde{R}_{\hat{\alpha}}$, $\pi \cdot \hat{\alpha} \geq \pi \cdot \alpha$ for all $\alpha \in \tilde{A}_H$. Therefore, the function g in $\tilde{R}_{\hat{\alpha}}$ can be rewritten as $g(\pi) = Hv(\pi) - \pi \cdot \hat{\alpha}$ for all $\pi \in \tilde{R}_{\hat{\alpha}}$. Since Hv is a convex function and $\pi \cdot \hat{\alpha}$ is a linear function, g is a convex function in $\tilde{R}_{\hat{\alpha}}$. Moreover, $\tilde{R}_{\hat{\alpha}}$ is a convex polytope. The maximal value of a convex function in a convex polytope will be at one of the extreme points of the convex polytope. Therefore, the maximal value of g in $\tilde{R}_{\hat{\alpha}}$ will be at one of the vertices of $\tilde{R}_{\hat{\alpha}}$. ■

THEOREM 3.3:

The maximal value of g in Π will be at one of the vertices of these relaxed regions whose corresponding supports are in $\tilde{A}_H$.

Proof:

As shown in Lemma 3.1, the maximal value of g in $\tilde{R}_{\hat{\alpha}}$ is in one of the vertices of $\tilde{R}_{\hat{\alpha}}$. By definition of the relaxed region, the union of all relaxed regions is Π. Therefore, the maximal value of g will be in one of these relaxed regions; then, by Lemma 3.1, the maximal value of g in Π will be at one of the vertices of these relaxed regions. ■

Assume that all vertices for the generated relaxed regions are in a set E. By

Theorem 3.3, the maximal error of this approximation will be at one of the vertices in E. Denote this vertex as $\hat{\pi}$. If $g(\hat{\pi})$ is equal to zero, then there is no error for this approximation; i.e., this is an exact solution. If $g(\hat{\pi})$ is greater than zero, the linear support(s) of Hv at $\hat{\pi}$, $A_{\hat{\pi}}$, can be found. Note that the supports in $A_{\hat{\pi}}$ are not currently in $\tilde{A}_H$ since $Hv(\hat{\pi}) = \hat{\pi} \cdot \hat{\alpha} > \max\{\hat{\pi} \cdot \alpha : \ \alpha \in \tilde{A}_H\}$ where $\hat{\alpha} \in A_{\hat{\pi}}$. Therefore, if one or more supports in $A_{\hat{\pi}}$ are included in $\tilde{A}_H$, a better approximation of Hv can be found.

If the supports in $A_{\hat{\pi}}$ are included in $\tilde{A}_H$, a new approximation for Hv and new relaxed regions for every support in $\tilde{A}_H$ can be determined. By Theorem 3.3, the maximal error of the new approximation is at one of the vertices of the newly generated relaxed regions for the supports currently in $\tilde{A}_H$. However, some of these newly generated relaxed regions are not the same as the relaxed regions before the supports in $A_{\hat{\pi}}$ are included in $\tilde{A}_H$. In order to find the maximal error of the new approximation function, the vertices of all newly generated relaxed regions have to be determined again and this is time–consuming. Fortunately, as will be shown in the next lemma, all of the vertices of these relaxed regions are in the set $E \bigcup C$ where C is the set of all vertices for the relaxed region(s) for the support(s) in $A_{\hat{\pi}}$. This implies only those vertices in the relaxed region(s) for the support(s) in $A_{\hat{\pi}}$ have to be identified.

LEMMA 3.2:

Let $\tilde{A}_H$ be a set of supports as described above and let the relaxed regions defined by the supports in $\tilde{A}_H$ be $\tilde{R}_\alpha = \{\pi \in \Pi : \ \pi \cdot \alpha \geq \pi \cdot \tilde{\alpha}$ where $\tilde{\alpha} \in \tilde{A}_H\}$. Let E be the set of all vertices for the relaxed regions corresponding to supports in $\tilde{A}_H$. Assume $\hat{\pi}$ is a state in E and $A_{\hat{\pi}}$ is the set of supports for Hv at $\hat{\pi}$. Let the relaxed region for a

support $\alpha \in \tilde{A}_H \cup A_{\tilde{\pi}}$ *be* $\tilde{R}'_\alpha = \{\pi \in \Pi : \ \pi \cdot \alpha \geq \pi \cdot \tilde{\alpha} \quad \forall\ \tilde{\alpha} \in \tilde{A}_H \cup A_{\tilde{\pi}}\}$. *Let* C *be the set of all vertices for the relaxed regions* R'_α *for* $\alpha \in A_{\tilde{\pi}}$ *and* E' *be the set of vertices for the relaxed regions* R'_α *for* $\alpha \in \tilde{A}_H \cup A_{\tilde{\pi}}$. *Then* $E' \subseteq E \cup C$.

Proof:

Let $\tilde{\pi}$ be an arbitrary vertex of E'. If $\tilde{\pi}$ is in R'_α for an α in $A_{\tilde{\pi}}$, then $\tilde{\pi} \in C$.

If $\tilde{\pi}$ is not in $\tilde{R}'_\alpha$ for any $\alpha \in A_{\tilde{\pi}}$, then, since $\tilde{\pi} \in \Pi \subset \Re^N$, N equations are required to define $\tilde{\pi}$. If $\tilde{\pi}$ is not on the boundary of Π, then there exists a set of supports $\{\alpha^1, \alpha^2, \ldots, \alpha^N\}$ such that these N equations can be represented as

$$\tilde{\pi} \cdot \alpha^N = \tilde{\pi} \cdot \alpha^i \qquad \text{where } i = 1, 2, \ldots, N-1;$$

$$\text{and} \qquad \sum_{k=1}^{N} \tilde{\pi}_k = 1.$$

Note that $\alpha^i \in \tilde{A}_H$ and $\alpha^i \notin A_{\tilde{\pi}}$ for $i = 1, 2, \ldots, N$. Therefore, $\tilde{\pi}$ is a vertex of $\tilde{R}_{\alpha^N}$; that is, $\tilde{\pi}$ is in E.

Similarly, if $\tilde{\pi}$ is on the boundary of Π, some of the constraints needed to define $\tilde{\pi}$ are the boundary conditions. Following the same argument, $\tilde{\pi}$ should be a vertex in E. Therefore $E' \subseteq E \cup C$. ■

In fact $E \cup C$ might contain some states which are no longer a vertex of any relaxed region. These vertices can be determined by computing the error. Define $g'(\pi) = Hv(\pi) - \max_{\alpha \in \tilde{A}_H \cup A_\pi}\{\pi \cdot \alpha\}$ for all $\pi \in E$.† If $g(\pi) > g'(\pi)$ and π is not a

† A vertex $\pi \in E$ which has been used to find new supports will have $g(\pi) = 0$. Therefore, it is not necessary to compute $g(\cdot)$ for this kind of vertex.

vertex of C, then π is not a vertex of any relaxed region and can be deleted from the set E'.

If C is included in E and the supports in $A_{\hat{\pi}}$ are included in $\tilde{A}_H$, then the maximal error for the new approximation function, by Lemma 3.2, will be at one of the states in E. Determine $g(\pi)$ for all $\pi \in E$ and find the maximal value. This maximal error will not be greater than that of the previous approximation. The support of Hv is found at the state which has the maximal error in the approximation. The procedure discussed above is repeated until no more new vertices and supports are generated. Then the set of supports can be used to form Hv without any error.

If a small error is tolerable, this procedure can be modified slightly to find an approximate solution. When the errors corresponding to the vertices of the region are less than the tolerable error, a more accurate approximation is not necessary for the states in this region. This can be restated more constructively: when the error of a vertex is less than the tolerable error, this vertex does not have to be considered. This modification can guarantee that the maximal error from the resulting approximation will not be more than the tolerable error.

This algorithm can be summarized as follows:

Step 0. Initialize $\tilde{A}_H$, E, $\tilde{E}$ and C to empty sets. Put all vertices of Π, $\pi^1, \pi^2, \ldots, \pi^N$, into E and $\tilde{E}$. Find the supports and the value of Hv at π for each $\pi \in E$, then put the newly generated supports into the set $\tilde{A}_H$. Determine the relaxed regions for each support in $\tilde{A}_H$ and find all vertices of these relaxed regions.

Put these generated vertices into C. If a vertex is in C but not in E, put it in E.

Step 1. Find $Hv(\pi)$ for each $\pi \in C$. Empty C.

Step 2. Compute $g(\pi) = Hv(\pi) - \max_{\alpha \in \tilde{A}_H}\{\pi \cdot \alpha\}$ for $\pi \in E \backslash \tilde{E}$. If $g(\pi)$ is less than the given tolerable error, then put π into $\tilde{E}$. If all $g(\pi)$ are less than the tolerable error, go to Step 6; otherwise, pick a vertex with the largest value of g from E and denote this vertex as $\hat{\pi}$.

Step 3. Empty the set $A_{\hat{\pi}}$. Find the linear support(s) for Hv at $\hat{\pi}$ and put these generated supports in $A_{\hat{\pi}}$. Put the supports in $A_{\hat{\pi}}$ which are not already in $\tilde{A}_H$ into $\tilde{A}_H$. Find the relaxed region for each support in $A_{\hat{\pi}}$. Find all vertices of these newly generated relaxed regions and put into C. If $\hat{\pi}$ is not a vertex in C, delete $\hat{\pi}$ from E; otherwise, put $\hat{\pi}$ into $\tilde{E}$.

Step 4. Compute $g'(\pi) = Hv(\pi) - \max_{\alpha \in A_*} \pi \cdot \alpha$ for all $\pi \in E \backslash \tilde{E}$. If $g'(\pi) < g(\pi)$ and $\pi \notin C$, then delete π from E.

Step 5. Put the vertices in C which are not in E into E. Go to Step 1.

Step 6. Stop. The value of $\max_{\alpha \in \tilde{A}_H}\{\pi \cdot \alpha\}$ is an approximation of $Hv(\pi)$ with maximal error less than the given tolerable error. The set E contains all the vertices of the support regions for the supports in $\tilde{A}_H$.

Note that if the tolerable error in the above algorithm is zero, then an exact solution of Hv can be found, and A_H is equal to $\tilde{A}_H$.

Unlike the one–pass algorithm and the relaxed region algorithm, only the support itself is important in the linear support algorithm. The information concerning which particular selection of $\alpha_{\pi,d,\theta} \in A_{\pi,d,\theta}$ forms a support in A_π is not used in this algorithm. Therefore, all supports in $\cup_{d\in D}\tilde{A}_d$ as defined in Monahan's algorithm can be generated. Hence,

$$Hv(\pi) = \max_{\alpha\in\cup_{d\in D}\tilde{A}_d} \{\pi\cdot\alpha\}$$

$$A_\pi = \{\hat{\alpha}\in\cup_{d\in D}\tilde{A}_d : \ \pi\cdot\hat{\alpha} \geq \pi\cdot\alpha \quad \forall\alpha\in\cup_{d\in D}\tilde{A}_d\}$$

and $\quad \tilde{A}_H \subseteq A_H \subseteq \cup_{d\in D}\tilde{A}_d.$

Example: (continued)

The linear support algorithm is used to solve the same problem as shown in Section IV.

The extreme points of Π, $[0,1]$ and $[1,0]$, are put into E and $\tilde{E}$. The solution procedure starts with finding the linear supports for the extreme points of Π. The linear support for state $[0,1]$ is $[0.2,11.0]^T$, and the linear support for state $[1,0]$ is $[4.62,7.91]^T$. Therefore, $\tilde{A}_H = \{[0.2,11.0]^T,[4.62,7.91]^T\}$. Since there are only two supports in $\tilde{A}_H$, there is only one relaxed region whose vertices have to be found. Let $\tilde{R}_1$ be the relaxed region for the support $[0.2,11.0]^T$; that is, $\tilde{R}_1 = \{\pi\in\Pi : \ \pi_1+\pi_2 = 1$ and $0.2\cdot\pi_1+11\cdot\pi_2 \geq 4.62\cdot\pi_1+7.91\cdot\pi_2\}$. The vertices of $\tilde{R}_1$ are $[0,1]$ and $[0.41,0.59]$. Since the vertex $[0,1]$ is already in E, put $[0.41,0.59]$ in E. The vertex $[0.41,0.59]$ is the only one in $E\backslash\tilde{E}$. Since $g([0.41,0.59]) = 0.74 > 0$, $[0.41,0.59]$ is used to find a new support.

The linear support for state $[0.41, 0.59]$ is $[4.0, 9.6]^T$. Since $[4.0, 9.6]^T$ is a new support, it is put into $\tilde{A}_H$. The vertices of the relaxed region for the support $[4.0, 9.6]^T$ are $[0.27, 0.73]$ and $[0.73, 0.27]$. Put these two vertices into C. Since $[0.41, 0.59]$ is not a vertex in C, delete it from E.

There are no vertices in $E \backslash \tilde{E}$. Put the vertices in C into E. Since these two vertices are not in $\tilde{E}$, $g([0.27, 0.73])$ and $g([0.73, 0.27])$ must be computed. Both values are zero; therefore, they should be in $\tilde{E}$. Now, no vertex in E has function value g greater than 0, and the process is completed. All supports in A_H have been found.

There are three supports, $[0.2, 11.0]^T$, $[4.0, 9.6]^T$, and $[4.62, 7.91]^T$, in A_H, and four states, $[0, 1]$, $[0.27, 0.73]$, $[0.73, 0.27]$, and $[1, 0]$, in E.

Now assume that the tolerable error is 0.75. Since $g([0.41, 0.59])$ is 0.74, which is smaller than the tolerable error, [0.41.0.59] is a vertex in $\tilde{E}$ and not used to find new support. There is no vertex in $E \backslash \tilde{E}$. The process is completed. The result is two supports, $[0.2, 11.0]^T$, and $[4.62, 7.91]^T$, in $\tilde{A}_H$, and three vertices, $[0, 1]$, $[0.41, 0.59]$, and $[1, 0]$, in E with a maximal approximation error of less than 0.75. ■

Before we conclude this section, the following questions are raised: (1) Can this algorithm be terminated within a finite number of iterations? (2) Can this algorithm find all supports in A_H? (3) Does E contain all vertices of the support regions, or, for an approximate solution, does E contain all vertices of the relaxed regions corresponding to the supports in $\tilde{A}_H$? The second half of question 3 has been answered by Lemma 3.2. The remaining questions are answered by the following theorem.

THEOREM 3.4:

(1) The linear support algorithm will terminate in a finite number of iterations.

If the tolerable error is set to be zero, then

(2) All supports in A_H can be found by the linear support algorithm.

(3) E contains all vertices of the support regions.

Proof:

(1) When a vertex is chosen for finding a linear support, the linear support cannot be the same as any of the supports currently in $\tilde{A}_H$. Since there is only a finite number of supports in A_H, there is no more approximation error after all supports in A_H are found. Therefore, no more relaxed regions and vertices are generated; that is, the algorithm will terminate after a finite number of iterations.

(2) Assume that $\tilde{A}_H$ is not the same as A_H and the process is terminated. Since $\tilde{A}_H$ is not the same as A_H, this implies that there is at least one $\pi \in \Pi$ such that $\max_{\alpha \in A_H}\{\pi \cdot \alpha\} - \max_{\alpha \in \tilde{A}_H}\{\pi \cdot \alpha\} > 0$. Then, by Theorem 3.3, the maximal error should occur at one of the vertices of the relaxed region. Therefore, the process should be continued and the algorithm cannot be terminated. Contradiction.

(3) When all supports in A_H are found, the relaxed region corresponding to the supports in $\tilde{A}_H$ are the support regions. Then, by Lemma 3.2, the result follows. ■

Let us now compare the linear support algorithm with the relaxed region algorithm. The linear support algorithm can be viewed as a relaxed region algorithm, but it is not the same as the one discussed in Section IV. The linear support algorithm uses a simpler

constraint set to define a relaxed region than the relaxed region algorithm. The number of relaxed regions whose vertices have to be found is the same as the number of supports in A_H for the relaxed region algorithm; in contrast, there is one relaxed region for which the vertices do not have to be found for the linear support algorithm. Both algorithms can generate all vertices for the support regions. However, the computational time for both algorithms should be very close; although we might expect the computational time for the linear support algorithm to be slightly less. As mentioned before, the most important difference between these two algorithms is that, unlike the relaxed region algorithm, the linear support algorithm can serve as an approximation algorithm.

Although the basic idea and motivation of the linear support algorithm is to make the relaxed region algorithm more efficient, it is similar to an algorithm discussed in Sondik (1971) provided that there are only two system states. Sondik claimed that his algorithm might not be as efficient as the one–pass algorithm if the number of system states or supports in A_H is large. However, the computational requirement for the linear support algorithm is about the same as or less than that for the relaxed region algorithm which has been shown to be more efficient than the one–pass algorithm.

VI. Numerical Examples

In this section, several sets of test data are used to compare the efficiency of the algorithms discussed in this chapter. The basis of comparison is CPU time. All algorithms were implemented as Fortran 77 programs which were run on the Amdahl 5860

with FPU at the University of British Columbia.

The code for the one–pass algorithm is based on the original code provided by Dr. Sondik. The Monahan algorithm is coded with Eagle's modification; that is, if all elements of a generated support are less than or equal to another support, then this support is deleted before linear programming is used to determine the unnecessary supports. Since there are usually more generated supports than system states, the dual formulations are used for linear programming. The IMSL routines are used to solve these linear programming problems to determine the unnecessary supports. In the relaxed region algorithm and the linear support algorithm, all vertices of a relaxed region have to be found. The Mattheiss algorithm is used here for finding all vertices of the convex polytopes. This algorithm is discussed in Mattheiss (1973) and Mattheiss and Rubin (1980).

The test data can be divided into two groups. The first group contains the data for the machine maintenance problem discussed in Smallwood and Sondik (1973). The second group contains several sets of randomly generated data for problems with three, four, and five system states. In order to minimize the effect of the terminal reward, all problems are solved for twenty stages and the terminal reward is set to be zero. For all problems, the discount factor β is 1; i.e., there is no discounting in reward.

1. Machine Maintenance Problem:

The data for this test problem is in Smallwood and Sondik (1973). The CPU times

Number of Periods Left	Number of supports	Sondik's Algorithm	Monahan's Algorithm	Relaxed Region Algorithm	Linear Support Algorithm
1	1	28	0	2	0
2	1	28	0	2	0
3	1	28	0	2	0
4	1	28	1	3	1
5	1	28	0	2	1
6	2	28	2	8	3
7	3	45	6	14	6
8	4	101	11	22	11
9	4	103	15	22	12
10	5	110	17	28	15
11	6	104	27	39	20
12	8	168	37	54	31
13	10	232	57	73	49
14	15	340	74	115	103
15	13	270	146	113	111
16	14	336	156	105	106
17	9	333	123	59	74
18	12	201	74	73	65
19	10	203	99	69	67
20	13	233	92	89	76
Total		2947	937	894	751

(unit : .001 CPU second)

Table 3.1: CPU times of the machine maintenance problem

and the number of supports at each period are recorded in Table 3.1.

From Table 3.1, the CPU time required for every period using Sondik's one–pass algorithm is always longer than that for other algorithms. It is clear that Sondik's algorithm is the least efficient algorithm among these four algorithms. This result is expected as discussed before.

As discussed in Section III, The Monahan algorithm generates $K \cdot M^L$ supports

in each iteration where K is the number of actions, L is the number of signals, and M is the number of supports in the previous iteration. Although a large number of generated supports can be eliminated by simply comparing their elements, there might still be a large number of supports which require linear programming to determine whether or not it is a required support. Therefore, the CPU time required for an iteration of the Monahan algorithm is directly related to the number of supports in the previous iteration. In contrast, the relaxed region algorithm and the linear support algorithm need to identify the same number of relaxed regions as the number of required supports for an iteration. Therefore, the CPU time required for an iteration of the latter algorithms is directly related to the number of necessary supports in the current iteration. The results of this example confirm this phenomenon. For instance, there are only nine supports for the seventeenth iteration, but fourteen supports for the previous iteration. As a result, the Monahan algorithm spent about twice the CPU time to perform this iteration as compared with the relaxed region algorithm and the linear support algorithm. Similar results are also observed for the fifteenth and nineteenth iteration. On the other hand, ten supports are needed at the thirteenth iteration and fifteen supports at the fourteenth iteration. The Monahan algorithm only requires approximately $\frac{2}{3}$ of CPU time used in the relaxed region algorithm and the linear support algorithm.

The total CPU time requirement for the Monahan algorithm is 5% more than that for the relaxed region algorithm and 20% more than that for the linear support algorithm. It is clear that the relaxed region algorithm and the linear support algorithm

are more efficient than the Monahan algorithm for this set of data.

As discussed in Section V, the linear support algorithm can be used to find an approximate solution if a small error is tolerable in each iteration. For comparison purposes, the tolerable error is set to 0.1, 0.01, 0.005, and 0.001, respectively, in 4 case runs. The number of supports and CPU times required for different tolerable errors are shown in Table 3.2. It is easy to see that the CPU times required for these approximate solutions must be less than or equal to that for the linear support algorithm for finding an exact solution. However, the CPU time required to obtain these approximate solutions is surprisingly short. If the tolerable error is set to 0.1 for each iteration, it takes 19% of the CPU time required to find the exact solution by the Monahan method. It requires 57%, 64%, and 77% of the CPU times for the Monahan algorithm if the tolerable errors for each iteration are 0.01, 0.005, and 0.001, respectively. The large reduction of CPU time required is due to fewer supports being generated from each iteration.

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Tolerable Error = 0.1	0.179	4	0.12508
Tolerable Error = 0.01	0.536	9	0.00863
Tolerable Error = 0.005	0.604	10	0.00283
Tolerable Error = 0.001	0.695	13	0

Table 3.2: CPU times of the machine maintenance problem for the approximation method

The error bound for these approximate solutions can also be calculated using the

following formula:

$$\frac{1-\beta^n}{1-\beta} \cdot T.E. \qquad \text{if} \quad 0 < \beta < 1;$$

$$\text{and} \qquad n \cdot T.E. \qquad \text{if} \quad \beta = 1$$

where n is the number of iterations, $T.E.$ is the tolerable error for each iteration, and β is the discount factor. Therefore, the error bounds are 2.0, 0.2, 0.1, and 0.02 for the tolerable errors of 0.1, 0.01, 0.005, and 0.001, respectively. These error bounds might be too large when compared with the maximal value of 10.59079 at the end of the twentieth iteration or when compared with the difference of 3.4025 between one–period maximal and minimal reward. Since the exact solution is known, the maximal error at the end of the twentieth iteration can be computed. The maximal errors are 0.12508, 0.00863, 0.00283, and 0 for the tolerable errors of 0.1, 0.01, 0.005, and 0.001, respectively. Except for the tolerable error of 0.1, the maximal errors are less than the tolerable error in one iteration. Even for a tolerable error of 0.1, the maximal error is just slightly more than the tolerable error in one iteration. The actual error is significantly smaller when compared with the maximal value at the end of the twentieth iteration or with the difference of one–period maximal and minimal reward.

This example shows that if two or more supports have very similar slopes, then the approximation method does not recognize these as distinct supports and treats them as one support. In this way, the number of supports can be reduced at each iteration and, as a result, the CPU times required can also be reduced. Since supports with similar slopes are considered as one and the number of supports are reduced, numerical stability

improves. A problem which cannot be solved by other algorithms might be able to be solved by using this method to get an approximate solution.

2. Randomly Generated Data:

Several sets of data with 3, 4, and 5 system states are generated to compare the efficiency of the algorithms discussed in this chapter. All these data are listed in the Appendix 1. From the discussions in Sections II, IV, and V, and the previous numerical example, it is clear that Sondik's one–pass algorithm is not as efficient as the other algorithms. Therefore, the one–pass algorithm will not be considered further in this thesis.

The first group of randomly generated data consists of five data sets with three states, three actions, and three signals. These data are listed in D3.1 to D3.5 in Appendix 1. The number of supports and the maximal error at the end of the twentieth iteration, and the CPU times are shown in Tables 3.3 to 3.7.

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	0.646	5	0
Relaxed Region Algorithm	0.812	5	0
Linear Support Algorithm T.E.= 0	0.473	5	0
Linear Support Algorithm T.E.= 0.1	0.230	4	0.00279
Linear Support Algorithm T.E.= 0.01	0.231	4	0.00279
Linear Support Algorithm T.E.= 0.005	0.226	4	0.00279
Linear Support Algorithm T.E.= 0.001	0.323	4	0.00015

Table 3.3: Results of the data set D3.1

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	0.882	7	0
Relaxed Region Algorithm	0.663	4	0
Linear Support Algorithm T.E.= 0	0.496	7	0
Linear Support Algorithm T.E.= 0.1	0.225	4	0.07402
Linear Support Algorithm T.E.= 0.01	0.321	5	0.00475
Linear Support Algorithm T.E.= 0.005	0.324	5	0.00475
Linear Support Algorithm T.E.= 0.001	0.417	6	0.00018

Table 3.4: Results of the data set D3.2

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	1.528	7	0
Relaxed Region Algorithm	0.926	5	0
Linear Support Algorithm T.E.= 0	0.818	7	0
Linear Support Algorithm T.E.= 0.1	0.260	4	0.13268
Linear Support Algorithm T.E.= 0.01	0.497	6	0.01228
Linear Support Algorithm T.E.= 0.005	0.481	6	0.01228
Linear Support Algorithm T.E.= 0.001	0.662	7	0.00041

Table 3.5: Results of the data set D3.3

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	4.626	10	0
Relaxed Region Algorithm	1.012	11	0
Linear Support Algorithm T.E.= 0	1.631	10	0
Linear Support Algorithm T.E.= 0.1	0.158	3	0.13672
Linear Support Algorithm T.E.= 0.01	0.478	5	0.02663
Linear Support Algorithm T.E.= 0.005	0.535	5	0.00928
Linear Support Algorithm T.E.= 0.001	1.068	7	0.00140

Table 3.6: Results of the data set D3.4

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	5.171	16	0
Relaxed Region Algorithm	2.230	8	0
Linear Support Algorithm T.E.= 0	2.348	15	0
Linear Support Algorithm T.E.= 0.1	0.389	5	0.06367
Linear Support Algorithm T.E.= 0.01	0.692	7	0.01219
Linear Support Algorithm T.E.= 0.005	0.727	7	0.01051
Linear Support Algorithm T.E.= 0.001	1.267	9	0.00093

Table 3.7: Results of the data set D3.5

From these five tables, it can be seen that in the case where only a small number of supports are needed to construct a value function, there is no significant difference in CPU times among these methods. However, when a larger number of supports are needed to form a value function, then the relaxed region algorithm and the linear support algorithm are much more efficient than the Monahan algorithm. The CPU times required for the relaxed region algorithm and the linear support algorithm are most often considerably less than a half of the CPU time required by the Monahan algorithm.

The performance of the approximation method is still very impressive. The CPU times required for the approximation method are much less than those for the other methods to find the exact solution. The actual maximal error is about the same or less than the tolerable error in one iteration. Considering that the difference between one period maximal and minimal reward ranges from 6.1 to 9.7, or the maximal value at

the end of the twentieth iteration ranges from 119 to 175, the actual error is remarkably small.

Now consider the second group of randomly generated data. This group of data includes five data sets with four states, four actions, and four signals. The data are listed in D4.1 to D4.5 in Appendix 1. The number of supports and maximal error at the end of the twentieth iteration, and the CPU times are shown in Table 3.8 to 3.12.

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	58.820	26	0
Relaxed Region Algorithm	11.549	22	0
Linear Support Algorithm T.E.= 0	19.861	25	0
Linear Support Algorithm T.E.= 0.1	0.991	5	0.18222
Linear Support Algorithm T.E.= 0.01	3.609	11	0.00922
Linear Support Algorithm T.E.= 0.005	8.348	13	0.00394
Linear Support Algorithm T.E.= 0.001	11.787	16	0.00156

Table 3.8: Results of the data set D4.1

These five sets of data require more than 20 supports to form their value functions at the end of the twentieth iteration. Since there are more supports in each iteration, the CPU times required to solve these problems are much longer than the data in the previous group.

This group of data can be divided into three subgroups. The first subgroup contains

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	61.479	26	0
Relaxed Region Algorithm	13.365	19	0
Linear Support Algorithm T.E.= 0	23.493	25	0
Linear Support Algorithm T.E.= 0.1	2.293	8	0.16191
Linear Support Algorithm T.E.= 0.01	7.830	13	0.00905
Linear Support Algorithm T.E.= 0.005	10.301	14	0.00308
Linear Support Algorithm T.E.= 0.001	16.890	20	0.00038

Table 3.9: Results of the data set D4.2

data sets D4.1 to D4.3. The results for this subgroup are obtained by all methods. The Monahan algorithm requires more than twice the CPU time to solve these problems as the other two algorithms. The performance of the approximation method is excellent. For example, the approximate solutions with a tolerable error of 0.1 for every iteration only require 1.2% to 3.8% of the CPU times required for the Monahan algorithm to solve these problems. The maximal errors for these approximations are very small when compared with the maximal value at the end of the twentieth iteration or the difference between one period maximal and minimal reward.

The second subgroup contains data set D4.4. Due to numerical problems, no result is generated by the relaxed region algorithm. For this set of data, the Monahan algorithm needs 762 CPU seconds to solve; however, the linear support algorithm requires only 172 CPU seconds to reach solution, which is only about 22.6% of the CPU time

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	79.154	30	0
Relaxed Region Algorithm	28.633	32	0
Linear Support Algorithm T.E.= 0	36.327	30	0
Linear Support Algorithm T.E.= 0.1	0.891	5	0.21916
Linear Support Algorithm T.E.= 0.01	4.301	9	0.02196
Linear Support Algorithm T.E.= 0.005	7.522	13	0.00813
Linear Support Algorithm T.E.= 0.001	21.877	20	0.00049

Table 3.10: Results of the data set D4.3

required by the Monahan algorithm. The performance of the approximation method is even more impressive. With a tolerable error of 0.1 for each iteration, it only takes 2.152 CPU seconds to reach solution and this is less than 0.3% of the CPU time required by the Monahan algorithm or 1.25% of the CPU time required by the linear support algorithm for finding the exact solution. The maximal error is only 0.01826. Consider the maximal value at the end of the twentieth iteration, 154.62, or the difference between one-period maximal and minimal reward, 6.8. This maximal error is so small that it can be ignored in this case.

The third subgroup contains data set D4.5. In this set of data, all three algorithms for finding the exact solution generate more than the pre-set maximal number of supports, 50, at the sixth iteration. Therefore, none of the results are generated. Even though the exact solution is impossible or difficult to obtain, the approximation

	CPU Times	At the end of the 20th Iteration	
		# of supports	Maximal Error
Monahan's Algorithm	762.173	32	0
Relaxed Region Algorithm	Result is not obtained		
Linear Support Algorithm T.E.= 0	172.282	33	0
Linear Support Algorithm T.E.= 0.1	2.152	6	0.01826
Linear Support Algorithm T.E.= 0.01	2.604	7	0.01297
Linear Support Algorithm T.E.= 0.005	9.140	10	0.00546
Linear Support Algorithm T.E.= 0.001	57.226	19	0.00214

Table 3.11: Results of the data set D4.4

method still works very well. It takes only 3.113 CPU seconds to solve this problem if the tolerable error for each iteration is set to 0.1 or 45.688 CPU seconds if the tolerable error is set to a relatively small number, 0.001.

Since the exact solution is unknown, the exact maximal error cannot be computed. As discussed before, the error bound can be calculated as $n \cdot T.E.$. However, from the previous examples, the error bound computed by this method is too big. A narrower error bound is desired. By the triangle inequality,

$$\|v^* - v\| \leq \|v^* - \tilde{v}\| + \|\tilde{v} - v\|$$

where v^* is the exact solution, v is the approximate solution whose error bound is desired, and $\tilde{v}$ is another approximate solution whose error bound is known. Since the approximate solution with a tolerable error of 0.001 at every iteration is the best solution obtained, this solution can be chosen as the reference solution, $\tilde{v}$, where $\|v^* - \tilde{v}\| \leq 0.02$.

Therefore, only $\|\tilde{v} - v\|$ has to be computed in order to know the error bound for the approximate solution v. The error bounds shown in Table 3.12 are computed by this method. These error bounds are still very small. It appears from this example that very good approximate solutions are obtained by the approximation method, particularly since these error bounds are probably considerably overestimated.

		At the end of the 20th Iteration	
	CPU Times	# of supports	Error bound
Monahan's Algorithm	over 50 supports at the 6th iteration		
Relaxed Region Algorithm	over 50 supports at the 6th iteration		
Linear Support Algorithm T.E.= 0	over 50 supports at the 6th iteration		
Linear Support Algorithm T.E.= 0.1	3.113	8	0.11084
Linear Support Algorithm T.E.= 0.01	9.892	14	0.03778
Linear Support Algorithm T.E.= 0.005	32.259	20	0.02998
Linear Support Algorithm T.E.= 0.001	45.688	24	0.02000

Table 3.12: Results of the data set D4.5

Since the generated supports are added one at a time into the support set at every iteration of the linear support algorithm, it is easy to develop an approximation method by limiting the maximal number of supports at each iteration. Table 3.13 shows the results of limiting the maximal number supports at each iteration to be 10, 15, and 20, respectively, for the data set D4.5. When approximate error for each iteration is known, the error bound can be computed as

$$\sum_{k=1}^{n} \beta^{n-k} e_k$$

where e_k is the maximal approximate error in the k-th iteration. By this method, the error bounds are 0.85161, 0.14986, and 0.07117, for maximal number of supports of 10, 15, and 20, respectively. However, since a more accurate result is known, a narrower error bound can be obtained by the triangle inequality discussed above. These narrower error bounds are shown in Table 3.13.

	CPU Times	At the end of the 20th Iteration	
		# of supports	Error bound
Maximal # of supports = 10	4.219	10	0.08210
Maximal # of supports = 15	14.479	15	0.03587
Maximal # of supports = 20	35.283	20	0.02655

Table 3.13: Results of data set D4.5 by using the approximation method which limits the maximal number of supports in every iteration

When it is difficult to choose *à priori* tolerable error for every iteration, it can be very useful to limit the maximal number of supports at every iteration in the approximation method. The selection of the maximal number of supports only depends on the number of actions and number of signals in the problem under consideration. When a reasonable number is chosen as the maximal number of supports, an approximate solution can usually be obtained although the error bound cannot be determined beforehand.

The last group of randomly generated data contains only one set of data, D5.1 in Appendix 1. This is a set of five states, three actions and three signals. The results of

	CPU Times	At the end of the 20th Iteration	
		# of supports	Error bound
Monahan's Algorithm	73.139	41	0
Relaxed Region Algorithm	Result is not obtained		
Linear Support Algorithm T.E.= 0	Result is not obtained		
Linear Support Algorithm T.E.= 0.1	0.725	4	0.03799
Linear Support Algorithm T.E.= 0.01	2.364	6	0.02653
Linear Support Algorithm T.E.= 0.005	2.668	7	0.02343
Linear Support Algorithm T.E.= 0.001	10.622	12	0.02000

Table 3.14: Results of the data set D5.1

this set of data are shown in Table 3.14.

In this set of data, the relaxed region algorithm and the linear support algorithm cannot complete the calculations. This difficulty is caused by too many supports being generated and the procedure of finding all vertices of relaxed regions being not sufficiently stable under this situation. As shown in the previous examples as well as this example, the approximation method can reduce the number of supports generated. As a result, the instability of the procedure for finding all vertices of a relaxed region is resolved. All four levels of approximation can obtain the results within a relatively short time.

By the triangle inequality, the error bounds are 0.03799, 0.02653, 0.02343, and 0.02000 for tolerable errors of 0.1, 0.01, 0.005, and 0.001, respectively. This fact also shows that a large number of generated supports can be represented by a very small

number of supports and only a very small error occurs. For example, the 41 supports generated by the Monahan algorithm can be represented by only 4 supports and the maximal error occurred is less than 0.03799. This example shows that the approximation method can quickly find a stable and accurate approximation.

VII. Conclusion

In this chapter, four algorithms for solving finite horizon POMDP problems are discussed.

Sondik's one–pass algorithm was the first systematic solution procedure for solving finite horizon POMDP problems. Since the unification of several regions is usually required to form a support region and all vertices of these region have to be found, it is clear that Sondik's method needs more CPU time to solve a problem than does the relaxed region algorithm or the linear support algorithm where the number of relaxed regions generated is the same as the number of supports.

The Monahan algorithm is simple to code. When there is only a small number of generated supports at each iteration, the Monahan algorithm can be more efficient than the relaxed region algorithm or the linear support algorithm. However, when the number of generated supports increases, it is clear, as evidenced by the numerical examples shown in Section VI, that the relaxed region algorithm and the linear support algorithm need less CPU time to solve a problem than does the Monahan algorithm.

It is difficult to compare the efficiency of the relaxed region algorithm and the linear support algorithm. However, the linear support algorithm has two advantages over the relaxed region algorithm. The first advantage is that the constraint set which defines a relaxed region is easy to set up for the linear support algorithm. This constraint set also gives more stable results for finding all vertices of a relaxed region. More importantly, the linear support algorithm can be used as an approximation method. Both the relaxed region algorithm and the linear support algorithm can provide the vertices of all support regions. These vertices can be used to compute the maximal and minimal difference of two piecewise linear functions.

The approximation method is the only method capable of solving a problem with a large number of supports. The approximation method reduces the number of generated supports with a small error. Reducing the number of supports not only significantly decreases the CPU time required for an iteration, but also decreases the possibility of numerical error caused by two or more very similar supports. As a result, a stable and relatively accurate solution can be obtained for more complex problems within reasonable CPU time. The approximation method can be performed by either setting the tolerable error for each iteration or limiting the number of supports generated for each iteration.

CHAPTER 4

ALGORITHMS FOR INFINITE HORIZON POMDP

In the previous chapter, the algorithms for finite horizon POMDP problems were discussed. In this chapter, the algorithms for infinite horizon discounted POMDP problems will be presented.

The discount factor β, in this chapter, is assumed to be $0 < \beta < 1$. The assumption about the discount factor is important since it guaranties that H and H_δ are contractions. Moreover, under this assumption Sawaragi and Yoshikawa (1970) have shown that there is a stationary optimal policy for a POMDP. Hence, only stationary policies have to be considered.

Although only stationary policies have to be considered in an infinite horizon discounted POMDP, there are uncountably many stationary policies available because the state space Π is continuous. Therefore, the convergence of the algorithm within finite time is not guaranteed. Moreover, the limit of a piecewise linear function is not necessarily piecewise linear. Papadimitriou and Tsitsiklis (1987) pointed out that infinite horizon POMDP problems are not combinatorial problems and do not appear to be exactly solvable by finite algorithms. However, if only an ϵ–optimal solution is required, these difficulties may be resolved. The theme of the chapter is to find an ϵ–optimal policy.

The main aim of this chapter is to develop a special class of algorithms for infinite horizon discounted POMDP, called iterative discretization procedures (IDP), which can

find an ϵ–optimal solution efficiently. This chapter is organized in the following way. The existing algorithms will be discussed in Section I. Section II introduces some of the basic results used in this chapter. Applying the approximation method discussed in Chapter 3 to the successive approximation method to get an ϵ–optimal solution is the topic of Section III. Section IV discusses the methods to find some useful values for termination criteria. The iterative discretization procedure is developed in Section V. Methods for accelerating convergence for the iterative discretization procedure are discussed in Section VI. The iterative discretization procedure with the approximation policy improvement is presented in Section VII. Section VIII provides the numerical comparisons of algorithms discussed in this chapter.

I. Existing Algorithms for Infinite Horizon POMDP

The most straightforward approach for solving an infinite horizon POMDP problem is the standard successive approximation method. As discussed in Chapter 2, a POMDP problem has the contraction property. Following Theorem 1 of Denardo (1967), it is easy to show that an ϵ–optimal solution can be obtained in a finite number of iterations. However, an iteration in the successive approximation method is similar to solving one stage of a finite horizon POMDP. As discussed in Chapter 3, to solve one stage of a finite horizon POMDP is not an easy task. Moreover, Papadimitriou and Tsitsiklis (1987) have shown that a finite horizon POMDP is a PSPACE–complete problem, so this approach is not efficient.

Sondik (1971,1978) introduced a policy iteration algorithm. This algorithm is based on the assumption of a finitely transient policy property. If a policy is finitely transient, then the state space Π can be partitioned into a finite number of convex regions such that, for any given signal, all states in one region will map onto the same region under this policy. However, it is difficult to verify that a given policy is finitely transient. Hence, a transient policy is used to approximate the given policy. Since partitioning the state space is not easy, it is difficult to perform this algorithm.

Recently, White and Scherer (1986) proposed a reward revision algorithm for an infinite horizon POMDP. Their algorithm is an accelerated successive approximation algorithm. In their algorithm, the problem is approximated by a completely observable, finite state MDP and the reward is revised between two standard policy improvement steps. They presented several examples with two states for which the speed of convergence is reduced by ten times compared with the standard successive approximation algorithm.

Although not necessarily the most efficient method, discretizing the state space is widely used for solving continuous state space MDP. Bertsekas (1975) showed that as the discretization grids become finer and finer, the performance of the resulting suboptimal policies comes arbitrarily close to that of the optimal. Kakalik (1965) used this method to solve a POMDP. He divided the state space Π into equal area grids and each grid was considered as a state. In this way, a finite number of states is obtained. The usual finite state MDP techniques can, then, be used to solve this problem. The resulting solutions for each state are used to represent the whole grid; that is, a piecewise

constant function is used to represent the value function. This method is easy to use and techniques developed for finite MDP can be applied. The disadvantage of this method is that a large number of grids might be required to get a reasonable approximation to an optimal solution. Indeed, this finite state MDP might become more difficult to solve than the original problem.

There are some other methods that have received less attention. Satia and Lave (1973) developed an implicit enumeration algorithm for computing ϵ–optimal solution to an finite horizon POMDP. Brumelle and Sawaki (1978) and Sawaki (1980) developed a modified policy iteration algorithm to solve an infinite horizon POMDP.

There are some special algorithms suitable only for some special cases. Wang (1976, 1977) considered a two action algorithm. Buckman and Miller (1979) reformulated the problem as a regenerative stopping problem. Algorithms of this nature are very efficient although they are not suitable for general infinite horizon POMDP problems.

II. Preliminaries

In this section, some results which will be used in later sections are developed.

Let Ω be a nonempty set in R^N. Let B be the collection of real-valued bounded functions with domain Ω. Define a metric $\|\cdot\|$ on B by $\|u-v\| = \sup_{x\in\Omega} |u(x)-v(x)|$ where $u, v \in B$, and let V be a subset of B which is complete in this metric. Let $u, v \in B$, we say $u = v$ if $u(x) = v(x)$ for all $x \in \Omega$ and $u \leq v$ if $u(x) \leq v(x)$ for all

$x \in \Omega$.

Recall that a mapping $G : V \to V$ is called a contraction if for some β strictly between 0 and 1, $\|Gu - Gv\| \leq \beta\|u - v\|$ $\forall u, v \in V$. Then by the principle of contraction mapping, there exists a unique fixed point, v^*, in V such that $Gv^* = v^*$ (Èl'sgol'c 1964).

The following theorem is an extension of Theorem 12.2.1 in Ortega and Rheinboldt (1970) and can be proved by a simple modification of the their proof.

THEOREM 4.1:

Let $G : V \to V$ be a contraction mapping, and assume $V_0 \subset V$ is a closed set such that $GV_0 \subset V_0$. Let $\{u^k\}$ be any sequence of functions in V_0 and set $\mu_k = \|Gu^k - u^{k+1}\|$, $k = 0, 1, \ldots$. Also let $v^0 \in V_0$ and define $v^{k+1} = Gv^k$ for $k = 0, 1, \ldots$. Let v^* be the unique fixed point of G in V (of course, $v^* \in V_0$). Then, for $k = 0, 1, \ldots$,

$$\|u^{k+1} - v^*\| \leq [1/(1-\beta)] \cdot [\beta \cdot \|u^{k+1} - u^k\| + \mu_k]; \qquad (4-1)$$

$$\|u^{k+1} - v^*\| \leq \|v^{k+1} - v^*\| + \sum_{j=0}^{k} \beta^{k-j} \cdot \mu_j + \beta^{k+1} \cdot \|v^0 - u^0\|, \qquad (4-2)$$

and $\lim_{k\to\infty} u^k = v^*$ if and only if $\lim_{k\to\infty} \mu_k = 0$.

The following definition is due to Van Nunen (1976).

Definition:

A mapping $\tilde{G}$ from V to V is said to be *μ-contracting* ($\mu \geq 0$) with contraction radius β ($0 < \beta < 1$) if for each $u, v \in V$, we have

$$\|\tilde{G}u - \tilde{G}v\| \leq \beta \cdot \|u - v\| + \mu.$$

LEMMA 4.1:

Let $\tilde{G} : V \to V$ be a μ-contraction mapping. If $u \in V$ is such that $V_0 = \{v \in V : \|v - \tilde{G}u\| \leq \gamma\} \subset V$ where $\gamma = [1/(1-\beta)] \cdot [\beta \cdot \|\tilde{G}u - u\| + \mu]$, then $\tilde{G}V_0 \subset V_0$.

Proof:

Let $v \in V_0$, then

$$\begin{aligned} \|\tilde{G}v - \tilde{G}u\| &\leq \beta \cdot \|v - u\| + \mu \\ &\leq \beta \cdot [\|v - \tilde{G}u\| + \|\tilde{G}u - u\|] + \mu \\ &\leq \beta \cdot \gamma + \beta \cdot \|\tilde{G}u - u\| + \mu \\ &= \gamma. \end{aligned}$$

■

Now let us consider using a mapping $\tilde{G}$ to approximate a contraction mapping G. The following theorem gives us a basic result for this approximation.

THEOREM 4.2:

Let $G : V \to V$ be a contraction mapping on V with constant β and $\tilde{G} : V \to V$ be another mapping for which

$$\|\tilde{G}v - Gv\| \leq \mu \quad \forall v \in V.$$

Suppose for some $\tilde{v}^0 \in V$, $V_0 = \{v \in V : \|v - \tilde{G}\tilde{v}^0\| \leq \gamma\}$ where $\gamma = [1/(1-\beta)] \cdot [\beta \cdot \|\tilde{G}\tilde{v}^0 - \tilde{v}^0\| + 2\mu]$. Then the sequence $\{\tilde{v}^k\}$ defined by $\tilde{v}^{k+1} = \tilde{G}\tilde{v}^k$ for $k = 0, 1, \ldots$ remains in V_0 and

$$\begin{aligned} \|\tilde{v}^{k+1} - v^*\| &\leq [\beta/(1-\beta)] \cdot \|\tilde{v}^{k+1} - \tilde{v}^k\| + \mu_k/(1-\beta) \\ &\leq [\beta/(1-\beta)] \cdot \|\tilde{v}^{k+1} - \tilde{v}^k\| + \mu/(1-\beta) \qquad (4-3) \end{aligned}$$

where $\mu_k = \|G\tilde{v}^k - \tilde{G}\tilde{v}^k\|$. *Moreover, if* $\{v^k\}$ *is the sequence defined by* $v^{k+1} = Gv^k$ *for* $k = 0, 1, \ldots$ *with* $v^0 = \tilde{v}^0$, *then*

$$\begin{aligned} \|\tilde{v}^{k+1} - v^*\| &\leq \|v^{k+1} - v^*\| + \sum_{j=0}^{k} \beta^{k-j}\mu_j \\ &\leq \|v^{k+1} - v^*\| + \sum_{j=0}^{k} \beta^j \mu \end{aligned} \tag{4-4}$$

where v^* *is the unique fixed point of* G *in* V_0.

Proof:

The proof is divided into four parts.

(i) $\tilde{G}$ is a 2μ–contraction mapping.

Let $u, v \in V$, then

$$\begin{aligned} \|\tilde{G}u - \tilde{G}v\| &\leq \|\tilde{G}u - Gu\| + \|Gv - \tilde{G}v\| + \|Gu - Gv\| \\ &\leq \mu + \mu + \beta \cdot \|u - v\| \\ &= \beta \cdot \|u - v\| + 2\mu. \end{aligned}$$

Therefore, by Lemma 4.1, $\tilde{G}V_0 \subset V_0$.

(ii) Now we show $GV_0 \subset V_0$.

Let $v \in V_0$, then $Gv \in V$ and

$$\begin{aligned} \|Gv - \tilde{G}\tilde{v}^0\| &\leq \|Gv - G\tilde{v}^0\| + \|G\tilde{v}^0 - \tilde{G}\tilde{v}^0\| \\ &\leq \beta \cdot \|v - \tilde{v}^0\| + \mu \\ &\leq \beta \cdot (\|v - \tilde{G}\tilde{v}^0\| + \|\tilde{G}\tilde{v}^0 - \tilde{v}^0\|) + \mu \\ &\leq \beta \cdot \gamma + \beta \cdot \|\tilde{G}\tilde{v}^0 - \tilde{v}^0\| + \mu \end{aligned}$$

$$\leq \beta \cdot \gamma + \beta \cdot \|\tilde{G}\tilde{v}^0 - \tilde{v}^0\| + 2\mu$$

$$\leq \beta \cdot \gamma + (1-\beta) \cdot \gamma$$

$$= \gamma$$

Therefore, $Gv \in V_0$ for all $v \in V_0$.

(iii) Now we will show that if $v^0 = \tilde{v}^0$, then $Gv^0 \in V_0$.

$\|Gv^0 - \tilde{G}\tilde{v}^0\| = \|G\tilde{v}^0 - \tilde{G}\tilde{v}^0\| \leq \mu \leq \gamma$. Therefore, $Gv^0 \in V_0$.

(iv) Then by Theorem 4.1 and $\mu_j \leq \mu$ for $j = 0, 1, \ldots, k$, the inequalities (4–3) and (4–4) are established. ■

As k approaches infinity, $\|v^{k+1} - v^*\|$ approaches 0, and $\sum_{j=0}^{k} \beta^j \cdot \mu$ approaches $\mu/(1-\beta)$. Then, by Theorem 4.2, the maximal distance between the approximation function $\tilde{v}^k$ and the fixed point v^* will be less than or equal to $\mu/(1-\beta) + \epsilon$ for all k sufficiently large.

When k equals 0, the inequalities $(4-3)$ and $(4-4)$ can be rewritten as

$$\|\tilde{v}^1 - v^*\| \leq [\beta/(1-\beta)] \cdot \|\tilde{v}^1 - \tilde{v}^0\| + [\mu_0/(1-\beta)]$$

$$\leq [\beta/(1-\beta)] \cdot \|\tilde{v}^1 - \tilde{v}^0\| + [\mu/(1-\beta)] \qquad (4-5)$$

and $$\|\tilde{v}^1 - v^*\| \leq \|v^1 - v^*\| + \mu_0$$

$$\leq \|v^1 - v^*\| + \mu.$$

Since $v^0 = \tilde{v}^0$ and by the contraction mapping assumption,

$$\|\tilde{v}^1 - v^*\| \leq [\beta/(1-\beta)] \cdot \|v^1 - \tilde{v}^0\| + \mu_0$$

$$\leq [\beta/(1-\beta)] \cdot \|v^1 - \tilde{v}^0\| + \mu \qquad (4-6)$$

where $\mu_0 = \|G\tilde{v}^0 - \tilde{G}\tilde{v}^0\|$.

In practice, the current approximation to the solution can be viewed as $\tilde{v}^1$ and the previous approximation to the solution can be viewed as $\tilde{v}^0$. In this case, $(4-5)$ and $(4-6)$ are more useful than $(4-3)$ and $(4-4)$ since the error bound for the current approximation to the solution can be computed. This error bound can be computed and used to determine whether or not the ϵ–optimality has been achieved.

Note that $\mu_0 \leq \mu$. Therefore, if μ_0 is readily available, which is the case in later sections, then the first inequalities in (4–5) and (4–6) should be used. Moreover, if $\|v^1 - \tilde{v}^0\| = \|\tilde{v}^1 - \tilde{v}^0\| + \mu$, then (4–5) and (4–6) give the same error bound. However, if $\|v^1 - \tilde{v}^0\| \leq \|\tilde{v}^1 - \tilde{v}^0\| + \mu$, then (4–6) gives a tighter bound than (4–5). Therefore, if $\|v^1 - \tilde{v}^0\|$ can be computed easily and $\|v^1 - \tilde{v}^0\| \leq \|\tilde{v}^1 - \tilde{v}^0\| + \mu$, then (4–6) is recommended.

III. Approximate Value Iteration

In the previous section, some very general results were discussed. In the next few sections, we will apply these results to the setting discussed in Chapter 2; that is, we will focus on the domain Π, the set of bounded real–valued functions V, and the contraction operator H.

In Chapter 2, the operator H was introduced. The computation of Hv for a given v was the major topic of Chapter 3. In the successive approximation method for an infinite

horizon problem, the operator H is repeatedly applied to find the optimal solution, i.e.,

$$\lim_{n\to\infty} H^n v = v^*.$$

As discussed and shown in the numerical examples in Chapter 3, when a large number of supports are needed to form Hv, it is usually time–consuming to compute Hv. The linear support algorithm discussed in the last chapter can be used to find an approximation solution. In contrast to the operator H, we will refer to this approximation operator as $\tilde{H}$ in this chapter.† The approximate value $\tilde{H}v$ might require much less time to obtain. We might expect that it may be easier to repeatedly apply operator $\tilde{H}$ to find an ϵ–optimal solution for an infinite horizon POMDP problem. We will discuss this issue in this section.

Theorem 4.2 gives a theoretical background for the use of an approximate evaluation for each step of policy improvement. Formula (4–4) shows that $\|\tilde{v}^{k+1} - v^*\| \leq \|v^{k+1} - v^*\| + \sum_{j=0}^{k} \beta^{k-j} \cdot \mu$ where $v^i = Hv^{i-1}$ and $\tilde{v}^i = \tilde{H}\tilde{v}^{i-1}$ for $i = 1, 2, \ldots, k+1$ and $\tilde{v}^0 = v^0$. When k approaches infinity, v^{k+1} approaches v^*. Therefore, for any given $\tilde{\epsilon} > 0$, $\|\tilde{v}^{k+1} - v^*\| \leq \frac{\mu}{1-\beta} + \tilde{\epsilon}\ k$ for k large enough. This implies that the maximal distance between v^* and the result from repeatedly applying approximate policy improvement steps will not be more than ϵ if μ is chosen to be less than $(1-\beta)\cdot(\epsilon - \tilde{\epsilon}$. Of course, in practice, μ is chosen to be much less than $(1-\beta)\cdot\epsilon$ in order to ensure faster convergence.

In practice, inequalities similar to (4–5) and (4–6) are usually used to determine the

† Although $\tilde{H}$ is dependent on a selected error ϵ, the dependency was suppressed from the notation for simplification.

error bound for the current solution. Let $\tilde{v}^k$ be the current solution. If $\|\tilde{H}\tilde{v}^k - \tilde{v}^k\| \leq \frac{(1-\beta)\cdot\epsilon-\mu_k}{\beta}$ or $\|H\tilde{v}^k - v^k\| \leq \frac{(1-\beta)\cdot(\epsilon-\mu_k)}{\beta}$ where $\mu_k = \|H\tilde{v}^k - \tilde{H}\tilde{v}^k\|$, then, following from the inequalities (4–5) and (4–6), $\|\tilde{H}\tilde{v}^k - v^*\| \leq \epsilon$. That is, $\tilde{H}\tilde{v}^k$ is an ϵ–optimal value function.

Successive approximations with extrapolation usually give a better bound for the optimal value function and also reduce the number of iterations required to get an ϵ–optimal value function. The following proposition is a generalization of Proposition 4 on page 237, in Bertsekas (1976). The proof of the following proposition is also a direct generalization of the proof in Bertsekas.

PROPOSITION 4.1:

Let $v \in V$. Then for all $\pi \in \Pi$ and $k = 1, 2, \ldots,$

$$\begin{aligned}(H^k v)(\pi) + \frac{\beta \cdot L_k}{1-\beta} &\leq (H^{k+1}v)(\pi) + \frac{\beta \cdot L_{k+1}}{1-\beta} \\ &\leq v^*(\pi) \\ &\leq (H^{k+1}v)(\pi) + \frac{\beta \cdot U_{k+1}}{1-\beta} \\ &\leq (H^k v)(\pi) + \frac{\beta \cdot U_k}{1-\beta}\end{aligned}$$

where
$$L_k = \inf_{\pi \in \Pi}\{(H^k v)(\pi) - (H^{k-1}v)(\pi)\}$$

and
$$U_k = \sup_{\pi \in \Pi}\{(H^k v)(\pi) - (H^{k-1}v)(\pi)\}.$$

Therefore, if $((\beta \cdot (U_k - L_k))/(1-\beta)) < \epsilon$, then $H^k v + ((\beta \cdot L_k)/(1-\beta))$ is an ϵ–optimal value function.

The above proposition and proof are restricted to the operations with an exact evaluation of H in the policy improvement step. The following proposition extends this result to the approximate policy improvement operator, $\tilde{H}$.

PROPOSITION 4.2:

Assume $Hv \geq \tilde{H}v$ *and* $\|Hv - \tilde{H}v\| \leq \mu$ *for all* $v \in V$. *Then, for all* $\pi \in \Pi$ *and* $k = 1, 2, \ldots,$

$$\tilde{H}^k v(\pi) + \frac{\beta \cdot L_k}{1-\beta} \leq v^*(\pi) \leq \tilde{H}^k v(\pi) + \frac{\beta \cdot U_k + \mu_k}{1-\beta}$$

where

$$L_k = \inf_{\pi \in \Pi} \{(\tilde{H}^k v)(\pi) - (\tilde{H}^{k-1} v)(\pi)\}$$

$$U_k = \sup_{\pi \in \Pi} \{(\tilde{H}^k v)(\pi) - (\tilde{H}^{k-1} v)(\pi)\}$$

$$\mu_k = \sup_{\pi \in \Pi} \{(H^k v)(\pi) - (\tilde{H}^k v)(\pi)\}.$$

Proof:

Since $L_k = \inf_{\pi \in \Pi} \{(\tilde{H}^k v)(\pi) - (\tilde{H}^{k-1} v)(\pi)\}$, then

$$\tilde{H}^{k-1} v(\pi) + L_k \leq \tilde{H}^k v(\pi) \quad \forall \pi \in \Pi \tag{4-7}$$

Apply H to both sides, using the monotonicity of H,

$$H\tilde{H}^{k-1} v(\pi) + \beta \cdot L_k \leq H\tilde{H}^k v(\pi),$$

and, by assumption, $\tilde{H}^k v(\pi) \leq H\tilde{H}^{k-1}v(\pi)$, and (4–7),

$$\tilde{H}^{k-1}v(\pi) + L_k + \beta \cdot L_k \leq H\tilde{H}^k v(\pi).$$

This process can be repeated. First apply H and then apply (4–7) to obtain

$$\tilde{H}^{k-1}v(\pi) + L_k + \beta \cdot L_k + \beta^2 \cdot L_k \leq \tilde{H}^k v(\pi) + \beta \cdot L_k + \beta^2 \cdot L_k \leq H^2\tilde{H}^k v(\pi).$$

After m steps this results in the inequality

$$\tilde{H}^k v(\pi) + \sum_{i=1}^{m} \beta^i \cdot L_k \leq H^m \tilde{H}^k v(\pi).$$

Taking the limit as $m \to \infty$ we obtain

$$\tilde{H}^k v(\pi) + \frac{\beta \cdot L_k}{1-\beta} \leq v^*(\pi),$$

which is the first inequality of this proposition.

Now consider the second inequality. The proof is similar to the first one. Since $U_k = \sup_{\pi \in \Pi}\{(\tilde{H}^k v)(\pi) - (\tilde{H}^{k-1}v)(\pi)\}$,

$$\tilde{H}^{k-1}v(\pi) + U_k \geq \tilde{H}^k v. \qquad (4-8)$$

Apply H to both sides, using the monotonicity of H,

$$H\tilde{H}^{k-1}v(\pi) + \beta \cdot U_k \geq H\tilde{H}^k v(\pi).$$

By assumption, $H\tilde{H}^{k-1}v(\pi) \leq \tilde{H}^k v(\pi) + \mu_k$, and (4–8),

$$\tilde{H}^k v + \mu_k + \beta \cdot U_k \geq H\tilde{H}^k.$$

This process can be repeated. First apply H and then apply (4–8) to obtain

$$\tilde{H}^k v + \mu_k + \beta \cdot \mu_k + \beta \cdot U_k + \beta^2 \cdot U_K \geq H^2 \tilde{H}^k.$$

After m steps this results in the inequality

$$\tilde{H}^k v(\pi) + \sum_{i=1}^{m} \beta^i \cdot U_k + \sum_{i=0}^{m-1} \beta^i \cdot \mu_k \geq H^m \tilde{H}^k v(\pi).$$

Taking the limit as $m \to \infty$ we obtain

$$\tilde{H}^k v(\pi) + \frac{\beta \cdot U_k + \mu_k}{1-\beta} \geq v^*(\pi),$$

which is the second inequality. ■

Corollary:

If $\frac{\beta \cdot (U_k - L_k) + \mu_k}{1-\beta} \leq \epsilon$, *then,* $\tilde{H}^k v + \frac{\beta \cdot L_k}{1-\beta}$ *is an* ϵ*-optimal value function.*

Proof:

The difference between $\tilde{H}^k v(\pi) + \frac{\beta \cdot L_k}{1-\beta}$ and $\tilde{H}^k v(\pi) + \frac{\beta \cdot U_k + \mu_k}{1-\beta}$ is less than or equal to $\frac{\beta \cdot (U_k - L_k) + \mu_k}{1-\beta}$ for all $\pi \in \Pi$. Therefore, if $\frac{\beta \cdot (U_k - L_k) + \mu_k}{1-\beta} \leq \epsilon$, then $\tilde{H}^k v + \frac{\beta \cdot L_k}{1-\beta}$ is an ϵ–optimal value function. ■

Unlike Proposition 4.1, the bound need not decrease monotonically in Proposition 4.2; that is, $\tilde{H}^k v(s) + (\beta \cdot L_k)/(1-\beta) \leq \tilde{H}^{k+1} v(s) + (\beta \cdot L_{k+1})/(1-\beta)$ and $\tilde{H}^{k+1} v(s) + (\beta \cdot U_{k+1} + \mu_{k+1})/(1-\beta) \leq \tilde{H}^k v(s) + (\beta \cdot U_k + \mu_k)/(1-\beta)$ might not be true because the operator $\tilde{H}$ is not monotone.

Since $\|\tilde{H}^k v - \tilde{H}^{k-1}\| = \max\{|L_k|, |U_k|\}$, if both L_k and U_k are same sign, the slightly modified bound provided by Proposition 4.2 will always be smaller or equal to

the bound provided by Theorem 4.2. The quantities L_k and U_K for the examples will be shown in Section VIII to be both positive, thus the bound provided by Proposition 4.2 is better than that of Theorem 4.2.

IV. Methods for Calculating L_k, U_k, and μ_k

In order to obtain an ϵ–optimal solution, L_k, U_k, and μ_k have to be calculated. Since these values are also required in later sections for computing the error bound, method for calculating these values will be reviewed and discussed.

For ease of discussion, let us consider a more general setting. Assume u and v are two paces linear continuous convex functions with a polytope domain Π. Assume the supports of u are in the set $A = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ and the supports of v are in the set $\Xi = \{\xi_1, \xi_2, \ldots, \xi_l\}$ where k and l are finite integers. Then, for all $\pi \in \Pi$,

$$u(\pi) = \max\{\pi \cdot \alpha : \alpha \in A\}$$

$$\text{and} \quad v(\pi) = \max\{\pi \cdot \xi : \xi \in \Xi\}.$$

Let

$$L = \inf_{\pi \in \Pi} \{u(\pi) - v(\pi)\}$$

$$U = \sup_{\pi \in \Pi} \{u(\pi) - v(\pi)\}$$

$$\text{and} \quad \mu = \|u - v\|.$$

It is easy to show that $\mu = \max\{|L|, |U|\}$.

Now consider the function $u - v$. Since both u and v are paces linear functions, $u - v$ is a paces linear and continuous function; however, $u - v$ need not be a convex function. For $\alpha_i \in A$ and $\xi_j \in \Xi$, define $R_{i,j}$ as

$$R_{i,j} = \{\pi \in \Pi : (u - v)(\pi) = \pi \cdot \alpha_i - \pi \cdot \xi_j\}.$$

Let the set $R = \{R_{i,j} : \text{ where } \alpha_i \in A \text{ and } \xi_j \in \Xi\}$. As in Chapter 3, let the support regions for u and v be

$$\tilde{R}_i = \{\pi \in \Pi : \pi \cdot \alpha_i \geq \pi \cdot \alpha \quad \text{where } \alpha_i, \alpha \in A\}$$

$$\text{and} \quad \hat{R}_j = \{\pi \in \Pi : \pi \cdot \xi_j \geq \pi \cdot \xi \quad \text{where } \xi_j, \xi \in \Xi\}.$$

Therefore, the region of $R_{i,j}$ is the intersection of the support regions $\tilde{R}_i$ and $\hat{R}_j$. Since both $\tilde{R}_i$ and $\hat{R}_j$ are convex, $R_{i,j}$ is a convex set. Note that $R_{i,j}$ can be an empty set.

Since $u - v$ is a linear function on the polytope $R_{i,j}$, the maximal and the minimal values of $u - v$ in $R_{i,j}$ are on the extreme points of $R_{i,j}$. Moreover, since both u and v have only a finite number of supports, there is a finite number of regions in R. Then L, U, and μ are on the extreme points of some regions in R.

White and Scherer (1986) developed a linear programming method to implement the idea discussed above calculating L, U, and μ. Let $L_{i,j}$ and $U_{i,j}$ be the minimum and maximum values of $u - v$ in $R_{i,j}$, respectively. To compute $L_{i,j}$, the following linear programming problem can be solved

$$\min \quad \pi \cdot \alpha_i - \pi \cdot \xi_j$$

subject to

$$\pi \cdot \alpha_i \geq \pi \cdot \alpha \qquad \text{for all} \quad \alpha \in A$$

$$\pi \cdot \xi_j \geq \pi \cdot \xi \qquad \text{for all} \quad \xi \in \Xi$$

$$\pi \in \Pi$$

The objective function value is the value of $L_{i,j}$. Analogously, $U_{i,j}$ is the objective value of the maximization of the above linear programming. Having calculated $L_{i,j}$ and $U_{i,j}$ for each region in R, let $L = \min\{L_{i,j}\}$ and $U = \max\{U_{i,j}\}$.

Using such a procedure to determine L and U requires solving $2 \cdot k \cdot l$ linear programming problems. This procedure can represent a significant computational effort, particularly if k and l are large numbers. White and Scherer (1986) suggested an approximation method to find L and U. Let $E = \{e_1, e_2, \ldots, e_m\}$ be a preselected set such that if $e_i \in E$ then $e_i \in \Pi$. Consequently,

$$\tilde{L} = \min\{u(e) - v(e) : e \in E\}$$

$$\tilde{U} = \max\{u(e) - v(e) : e \in E\}$$

$$\text{and} \qquad \tilde{\mu} = \max\{|\tilde{L}|, |\tilde{U}|\}.$$

Clearly, $L \leq \tilde{L}$, $U \geq \tilde{U}$, and $\mu \geq \tilde{\mu}$; White and Scherer did not discuss how to choose the set E to obtain a good approximation.

As discussed earlier, L and U will occur at some of the extreme points of some regions in R. If all of the extreme points of the regions in R are contained in the set E, then $\tilde{L} = L$ and $\tilde{U} = U$. However, it is as difficult to find the extreme points of the regions in R as it is to solve all linear programming problems to find L and U.

Let $\tilde{R} = \{\tilde{R}_i : \alpha_i \in A\}$ and $\hat{R} = \{\hat{R}_j : \xi_j \in \Xi\}$. Also let E_R, $E_{\tilde{R}}$, and $E_{\hat{R}}$ be the extreme points of the regions in R, $\tilde{R}$ and $\hat{R}$, respectively. Clearly, $E_{\tilde{R}} \cup E_{\hat{R}} \subseteq E_R$. More importantly, in our application, if u and v are found by the relaxed region algorithm or the linear support algorithm, then $E_{\tilde{R}}$ and $E_{\hat{R}}$ are readily available for use. Since no extra effort is required to find $E_{\tilde{R}}$ and $E_{\hat{R}}$, it is recommended that $E = E_{\tilde{R}} \cup E_{\hat{R}}$ be used to find $\tilde{L}$, $\tilde{U}$ and $\tilde{\mu}$ to approximate L, U, and μ. In particular for the two system states problem, it can be shown that $E_{\tilde{R}} \cup E_{\hat{R}} = E_R$ and the approximation is exact. Once the approximate termination criterion based on $\tilde{L}$, $\tilde{U}$, and $\tilde{\mu}$ is satisfied, the exact values of L, U, and μ can be computed to verify ϵ–optimality.

V. An Iterative Discretization Procedure for POMDP

In Chapter 3 an approximation $\tilde{H}$ was defined which could be used to apply approximate value iteration to compute an ϵ–optimal value function as discussed in Section III. Although the time required for each iteration of approximate value iteration is much less than under regular successive approximation, for each iteration all vertices of the relaxed regions still have to be found. The procedure of finding all vertices is not an easy task. It is desirable to have a method which can approximate Hv for a given v which does not involve finding all vertices and which reduces the number of iterations of applying operator H. In this section, we present a method which accomplishes these purposes.

Let v_n be a piecewise linear and convex value function with $A_n = \{\alpha^1, \ldots, \alpha^k\}$ as

its support set. For any given π and A_n, $Hv_n(\pi)$ and the corresponding support can be calculated by the formula $(2-5)$ and $(2-6)$; that is

$$Hv_n(\pi) = \max_{d \in D} \quad \pi \cdot \{r^d + \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \alpha_{\pi,d,\theta}\} \qquad (4-9)$$

where $\alpha_{\pi,d,\theta} \in \{\alpha \in A_n : \pi \cdot P^d \cdot Q_\theta^d \cdot \alpha \geq \pi \cdot P^d \cdot Q_\theta^d \cdot \hat{\alpha} \quad \text{for all } \hat{\alpha} \in A_n\}$. For the example shown in Chapter 3, let us choose π as $[0.5, 0.5]$ and β to be 1, then $Hv_n(\pi)$ can be obtained as

$$\begin{aligned}
Hv_n([0.5,0.5]) = \max \quad & [0.5,0.5] \cdot \{ \\
& \begin{bmatrix} -4 \\ 5 \end{bmatrix} + \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \cdot \left(\begin{bmatrix} 0.8 & 0 \\ 0 & 0.6 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 9 \end{bmatrix} + \begin{bmatrix} 0.2 & 0 \\ 0 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 9 \end{bmatrix} \right); \\
& \begin{bmatrix} -2 \\ 3 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \end{bmatrix} \cdot \left(\begin{bmatrix} 0.9 & 0 \\ 0 & 0.4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 9 \end{bmatrix} + \begin{bmatrix} 0.1 & 0 \\ 0 & 0.6 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 9 \end{bmatrix} \right); \\
& \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.6 & 0.4 \\ 0.3 & 0.7 \end{bmatrix} \cdot \left(\begin{bmatrix} 0.9 & 0 \\ 0 & 0.2 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 9 \end{bmatrix} + \begin{bmatrix} 0.1 & 0 \\ 0 & 0.8 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 9 \end{bmatrix} \right) \} \\
= \max \quad & [0.5,0.5] \cdot \left\{ \begin{bmatrix} 0.2 \\ 11 \end{bmatrix}; \begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix}; \begin{bmatrix} 4.4 \\ 8.2 \end{bmatrix} \right\} \\
= & [0.5,0.5] \cdot \begin{bmatrix} 4.0 \\ 9.6 \end{bmatrix} \\
= & \ 6.8
\end{aligned}$$

The support of Hv_n at this state is $[4.0, 9.6]^T$, and the second action is the best for this state. Similarly, if the chosen state is $[0, 1]$, the support of Hv_n at this state is $[0.2, 11]^T$. If the chosen state is $[1, 0]$, then the support of Hv_n at this state is $[4.62, 7.91]^T$. Notice that these three supports form the supports of Hv_n on Π. This implies that if some particular states are chosen and formula $(4-9)$ is applied, then it is not necessary to perform the complicated procedures discussed in Chapter 3 in order to calculate the supports and values of Hv_n. However, the major difficulty with this method is how to select the chosen states such that all necessary supports for Hv_n can be found.

Although it is difficult to select a set of states such that all necessary supports can be found by only using these states in the set, a good selection of states can generate a good approximation of Hv_n. A means of selecting states for this purpose will be discussed later in this section. For now, assume that there is a method for choosing a finite number of states such that a good approximation of Hv_n can be generated. Since the computational time spent in finding supports by using a finite number of states is much less than that for an iteration of H as discussed in Chapter 3 provided not too many states are chosen, it might be worthwhile to approximate Hv_n using a finite number of states to generate supports. A method to solve POMDP can be developed using such an approximation for Hv_n.

For a given value function $v_n \leq v^*$ and the corresponding set of supports A_n, first select a set of k states. Then compute the value of Hv and the corresponding linear supports for these states. The convex piecewise linear function generated by these supports is used to approximate Hv_n. Note that this approximated value might be less than v_n for some $\pi \in \Pi$. In order to obtain $Hv_n \geq v_n$, we should also include the supports in A_n when generating the approximation. Although the approximated solution might not be exactly the same as the result from an iteration of policy improvement (i.e., the H operator), the approximated value function might be reasonably good and the CPU time required for computing this approximation will be much less than that for an iteration of the H operation.

Improvements for these chosen states due to this approximation can be obtained

through computing the difference between the current values and the original values of these states. If the maximal improvement among these chosen states is "large," then the same set of states and their approximate values can be used to find a new approximate value function, since a relatively "large" improvement can be expected for the new approximation. This procedure can be repeated. However, as expected, the improvements for these chosen states become smaller and smaller. It is not a good idea to continue the process using the same set of states if the maximal improvement is very small. Instead, an iteration of policy improvement should be performed. Since only a finite number of states are considered between two iterations of policy improvement, we refer to this period as a discrete phase. This is the basic concept for an iterative discretization procedure.

Now let us discuss the iterative discretization procedure more precisely.

Set $\bar{A}_{n,0} = A_n$, and $\bar{v}_{n,0}(\pi) = \max_i\{\pi \cdot \alpha^i : \ \alpha^i \in \bar{A}_{n,0}\}$. Denote the set of discrete states generated at period n as $\bar{\Pi}_n = \{\pi^1, \pi^2, \cdots, \pi^{k(n)}\}$. Also let $\bar{\delta}$ be an arbitrary policy used in the discrete phase. Define $\bar{v}^{\bar{\delta}(\pi^i)}_{n,m+1}(\pi^i)$ as

$$\bar{v}^{\bar{\delta}(\pi^i)}_{n,m+1}(\pi^i) = \pi^i \cdot [r^{\bar{\delta}(\pi^i)} \ + \ \beta \cdot \sum_{\theta \in \Theta} P^{\bar{\delta}(\pi^i)} \cdot Q^{\bar{\delta}(\pi^i)}_{\theta} \cdot \alpha_{f(\pi^i, \bar{\delta}(\pi^i), \theta, \bar{A}_{n,m})}] \qquad (4-10)$$

and $\bar{v}_{n,m+1}(\pi^i)$ as

$$\begin{aligned} \bar{v}_{n,m+1}(\pi^i) &= \max_{\bar{\delta}(\pi^i)}\{\bar{v}^{\bar{\delta}(\pi^i)}_{n,m+1}(\pi^i)\} \\ &= \max_{d \in D}\{\pi^i \cdot [r^d \ + \ \beta \cdot \sum_{\theta \in \Theta} P^d \cdot Q^d_{\theta} \cdot \alpha_{f(\pi^i, d, \theta, \bar{A}_{n,m})}]\} \\ &= \pi^i \cdot \bar{\alpha}^i(n, m+1) \end{aligned} \qquad (4-11)$$

where $\pi^i \in \bar{\Pi}_n$, and $\alpha_{f(\pi^i,d,\theta,\bar{A}_{n,m})} \in \{\alpha \in \bar{A}_{n,m} : \pi^i \cdot P^d \cdot Q_\theta^d \cdot \alpha \geq \pi^i \cdot P^d \cdot Q_\theta^d \cdot \hat{\alpha}$ for all $\hat{\alpha} \in \bar{A}_{n,m}\}$. Setting

$$\bar{A}_{n,m+1} = \bar{A}_{n,m} \cup \{\cup_{i=1}^{k(n)} \bar{\alpha}^i(n, m+1)\},$$

$\bar{v}_{n,m+1}$ can be defined as

$$\bar{v}_{n,m+1}(\pi) = \max_i \{\pi \cdot \alpha^i : \ \alpha^i \in \bar{A}_{n,m+1}\} \ \forall \pi \in \Pi. \qquad (4-12)$$

Note that, since $\bar{A}_{n,m} \subseteq \bar{A}_{n,m+1}$, it follows that we have $\bar{v}_{n,m+1} \geq \bar{v}_{n,m}$. Moreover, if $\alpha^i, \alpha^j \in \bar{A}_{n,m+1}$ where $i \neq j$, and all the elements of α^i are greater than or equal to those of α^j, then α^j can be deleted from $\bar{A}_{n,m+1}$ without changing the results of the whole process.

The iterative discretization procedure can be summarized as follows:

Step 0. Choose v_0' such that $Hv_0' \geq v_0'$. Set $n = 0$.

Step 1. Compute $v_{n+1} = Hv_n'$.

Step 2. Calculate $U_{n+1} = \sup_{\pi \in \Pi}\{v_{n+1}(\pi) - v_n(\pi)\}$ and $L_{n+1} = \inf_{\pi \in \Pi}\{v_{n+1}(\pi) - v_n(\pi)\}$ using the techniques described in Section IV. If $U_{n+1} - L_{n+1} \leq \frac{\beta}{1-\beta}\epsilon$, go to Step 8; otherwise, go to Step 3.

Step 3. Set $n = n+1$, $m = 0$, $\bar{v}_{n,0} = v_n$, $\bar{A}_{n,0} = A_n$, and select $k(n)$ disjoint states from Π, put them into the set $\bar{\Pi}_n$. Also select a small number $\epsilon_1(n)$ as a reference for stopping this discrete phase, and an integer number $I(n)$ as the largest number of iterations to be performed in this discrete phase.

Step 4. Compute $\bar{v}_{n,m+1}(\pi)$ for $\pi \in \bar{\Pi}_n$. Then find $\bar{A}_{n,m+1}$.

Step 5. Proceed to Step 7 if $\max_{\pi \in \bar{\Pi}_n}(\bar{v}_{n,m+1}(\pi) - \bar{v}_{n,m}(\pi)) \leq \epsilon_1(n)$ or $m \geq I(n)$; otherwise, go to Step 6.

Step 6. Set $m = m + 1$, then go to 4.

Step 7. Set $v'_n = \bar{v}_{n,m+1}$, $A'_n = \bar{A}_{n,m+1}$. Go to 1.

Step 8. Set $v_{n+1}(\pi) = v_{n+1}(\pi) + \frac{\beta}{1-\beta} \cdot L_{n+1}$. Then $\|v^* - v_{n+1}\| \leq \epsilon$ and an ϵ–optimal value function has been found by Proposition 1.

Note that the steps 3 to 7 are the procedures in a discrete phase. If steps 4 to 7 are omitted and step 3 is changed to "Set $n = n + 1$, $v'_n = v_n$, and $A'_n = A_n$, go to 1", then this becomes an ordinary successive approximation procedure.

Example:

One iteration of the discrete phase of iterative discretization procedure is illustrated using the problem posed in Sondik (1978) with the following data:

$$P^1 = \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \quad Q^1 = \begin{bmatrix} 0.8 & 0.2 \\ 0.6 & 0.4 \end{bmatrix} \quad r^1 = \begin{bmatrix} -4 \\ 4 \end{bmatrix}$$
$$P^2 = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \end{bmatrix} \quad Q^2 = \begin{bmatrix} 0.9 & 0.1 \\ 0.4 & 0.6 \end{bmatrix} \quad r^2 = \begin{bmatrix} 0 \\ 3 \end{bmatrix}.$$

Assume that β is 0.9 and $\bar{A}_{1,0} = A_1 = \{\alpha^1, \alpha^2\}$ where $\alpha^1 = [-4, 4]^T$ and $\alpha^2 = [0, 3]^T$. For ease of calculation, two states, $\pi^1 = [0, 1]$ and $\pi^2 = [1, 0]$ are selected. Then $\bar{v}_{1,0}(\pi^1) = 4$ and $\bar{v}_{1,0}(\pi^2) = 0$. Let us also choose $\epsilon_1(1) = 1.25$ as the stopping criterion

for the discrete phase. By (4–10) and (4–11),

$$\begin{aligned}\bar{v}_{1,1}([0,1]) &= \max\{[0,1]\cdot\begin{bmatrix}-3.46\\5.35\end{bmatrix};[0,1]\cdot\begin{bmatrix}1.44\\4.80\end{bmatrix}\}\\ &= [0,1]\cdot\begin{bmatrix}-3.46\\5.35\end{bmatrix}\\ &= 5.35\end{aligned}$$

and

$$\begin{aligned}\bar{v}_{1,1}([1,0]) &= \max\{[1,0]\cdot\begin{bmatrix}-3.46\\5.35\end{bmatrix};[1,0]\cdot\begin{bmatrix}1.44\\4.80\end{bmatrix}\}\\ &= [1,0]\cdot\begin{bmatrix}1.44\\4.80\end{bmatrix}\\ &= 1.44.\end{aligned}$$

Therefore, $\bar{\alpha}^1(1,1) = \begin{bmatrix}-3.46\\5.35\end{bmatrix}$ and $\bar{\alpha}^2(1,1) = \begin{bmatrix}1.44\\4.80\end{bmatrix}$. The set $\bar{A}_{1,1}$ is $\bar{A}_{1,0}\cup\{\bar{\alpha}^1(1,1)\}\cup\{\bar{\alpha}^2(1,1)\}$. However, since $\bar{\alpha}^1(1,1) \geq \alpha^1$ and $\bar{\alpha}^2(1,1) \geq \alpha^2$, α^1 and α^2 can be deleted from $\bar{A}_{1,1}$ to reduce unnecessary calculations and the solution will still be the same. Hence, $\bar{A}_{1,1}$ can be set as $\{\begin{bmatrix}-3.46\\5.35\end{bmatrix},\begin{bmatrix}1.44\\4.80\end{bmatrix}\}$. The value of $\bar{v}_{1,1}(\pi)$ can then be calculated by $\bar{v}_{1,1}(\pi) = \max\{\pi\cdot\begin{bmatrix}-3.46\\5.35\end{bmatrix},\pi\cdot\begin{bmatrix}1.44\\4.80\end{bmatrix}\}$.

Since both $\bar{v}_{1,1}(\pi^1) - \bar{v}_{1,0}(\pi^1)$ and $\bar{v}_{1,1}(\pi^2) - \bar{v}_{1,0}(\pi^2)$ are greater than $\epsilon_1(1)$, π^1 and π^2 are used to perform the second iteration. Similarly, by (4–10) and (4–11),

$$\begin{aligned}\bar{v}_{1,2}([0,1]) &= \max\{[0,1]\cdot\begin{bmatrix}-2.10\\6.81\end{bmatrix};[0,1]\cdot\begin{bmatrix}2.74\\6.11\end{bmatrix}\}\\ &= [0,1]\cdot\begin{bmatrix}-2.10\\6.81\end{bmatrix}\\ &= 6.81\end{aligned}$$

and

$$\bar{v}_{1,2}([1,0]) = \max\{[1,0]\cdot\begin{bmatrix}-2.10\\6.80\end{bmatrix};[1,0]\cdot\begin{bmatrix}2.81\\6.11\end{bmatrix}\}$$

$$= [1,0] \cdot \begin{bmatrix} 2.81 \\ 6.11 \end{bmatrix}$$

$$= 2.81.$$

Therefore, $\bar{\alpha}^1(1,2) = \begin{bmatrix} -2.10 \\ 6.81 \end{bmatrix}$ and $\bar{\alpha}^2(1,2) = \begin{bmatrix} 2.81 \\ 6.11 \end{bmatrix}$. Since $\bar{\alpha}^1(1,2) > \bar{\alpha}^1(1,1)$ and $\bar{\alpha}^2(1,2) > \bar{\alpha}^2(1,1)$, $\bar{A}_{1,2}$ can be set as $\bar{\alpha}^1(1,2) \cup \bar{\alpha}^2(1,2)$. Moreover, since $\bar{v}_{1,2}(\pi^1) - \bar{v}_{1,1}(\pi^1)$ and $\bar{v}_{1,2}(\pi^2) - \bar{v}_{1,1}(\pi^2)$ are greater than $\epsilon_1(1)$, the discrete phase should not be terminated. The states π^1 and π^2 are used to do the third iteration. Similarly, by (4–10) and (4–11),

$$\bar{v}_{1,3}([0,1]) = \max\{[0,1] \cdot \begin{bmatrix} -0.88 \\ 8.01 \end{bmatrix}; [0,1] \cdot \begin{bmatrix} 3.98 \\ 7.36 \end{bmatrix}\}$$

$$= [0,1] \cdot \begin{bmatrix} -0.88 \\ 8.01 \end{bmatrix}$$

$$= 8.01$$

and

$$\bar{v}_{1,3}([1,0]) = \max\{[1,0] \cdot \begin{bmatrix} -0.88 \\ 8.01 \end{bmatrix}; [1,0] \cdot \begin{bmatrix} 4.01 \\ 7.31 \end{bmatrix}\}$$

$$= [1,0] \cdot \begin{bmatrix} 4.01 \\ 7.31 \end{bmatrix}$$

$$= 4.01.$$

Therefore, $\bar{\alpha}^1(1,3) = \begin{bmatrix} -0.88 \\ 8.01 \end{bmatrix}$ and $\bar{\alpha}^2(1,3) = \begin{bmatrix} 4.01 \\ 7.31 \end{bmatrix}$. Since $\bar{\alpha}^1(1,3) > \bar{\alpha}^1(1,2)$ and $\bar{\alpha}^2(1,3) > \bar{\alpha}^2(1,2)$, $\bar{A}_{1,3}$ can be set as $\bar{\alpha}^1(1,3) \cup \bar{\alpha}^2(1,3)$. Since both $\bar{v}_{1,3}(\pi^1) - \bar{v}_{1,2}(\pi^1)$ and $\bar{v}_{1,3}(\pi^2) - \bar{v}_{1,2}(\pi^2)$ are equal to 1.20 which is smaller than the preselected stopping criterion $\epsilon_1(1) = 1.25$, this discrete phase is terminated. Set A_1' as $\{\begin{bmatrix} -0.88 \\ 8.01 \end{bmatrix}, \begin{bmatrix} 4.01 \\ 7.31 \end{bmatrix}\}$. Then do one iteration of the H operator to check whether or not an ϵ–optimal solution has been obtained. ∎

Observe that, in this example, $\bar{v}_{1,1} = Hv_1$ and $\bar{v}_{1,2} = H\bar{v}_{1,1} = H^2v_1$. Therefore, only $\bar{v}_{1,3}$ is an approximation of H^3v_1. Compared with the methods discussed in Chapter 3, the computation shown in this example to obtain $\bar{v}_{1,1}$ and $\bar{v}_{1,2}$ is much simpler than the computation of any method discussed in Chapter 3 for obtaining the values of Hv_1 and H^2v_2. This is the major benefit of using the iterative discretization procedure.

Let us now develop some properties of the iterative discretization procedure.

LEMMA 4.2:

If $v_n = Hv'_{n-1} \geq v'_{n-1}$, *then*

$$v_n \leq \bar{v}_{n,m} \leq \bar{v}_{n,m+1} \leq v'_n \leq Hv'_n \leq v^*$$

where $0 \leq m \leq \bar{I}(n)-1$, *and* $\bar{I}(n)$ *is the number of iterations in the n-th discrete phase.*

Proof:

$v_n \leq \bar{v}_{n,m} \leq \bar{v}_{n,m+1} \leq v'_n$ follows immediately from the definitions of $\bar{v}_{n,m}$ and v'_n.

Let π be an arbitrary state in Π and $m' = \min\{m \geq 1 : \bar{v}_{n,m}(\pi) = v'_n(\pi)\}$. Then

$$\begin{aligned}
Hv'_n(\pi) &= \max_{d \in D}\{\pi \cdot r^d + \beta \sum_{\theta \in \Theta} \Pr(\theta|\pi, d) \cdot v'_n(T(\pi|d, \theta))\} \\
&\geq \max_{d \in D}\{\pi \cdot r^d + \beta \sum_{\theta \in \Theta} \Pr(\theta|\pi, d) \cdot \bar{v}_{n,m'-1}(T(\pi|d, \theta))\} \\
&= H\bar{v}_{n,m'-1}(\pi) \\
&\geq \bar{v}_{n,m'}(\pi) \\
&= v'_n(\pi)
\end{aligned}$$

where the first inequality follows from the monotonicity property, and the second inequality follows from the definition of $\bar{v}_{n,m'}(\pi)$ and the convexity of $\bar{v}_{n,m'}$.

We still have to prove $Hv'_n \leq v^*$. Since $Hv'_{n-1} \geq v'_{n-1}$, we have $v'_{n-1} \leq v^*$. By the monotonicity property, $Hv'_{n-1} \leq v^*$. Then by the monotonicity property and induction, $H\bar{v}_{n,m} \leq v^*$. The result follows. ∎

THEOREM 4.3:

If $v'_0 \leq Hv'_0$ and the sequence of $\{v_n\}$ is defined as in the above algorithm, then the sequence $\{v_n\}$ converges to v^* monotonically.

Proof:

The monotonicity follows directly from Lemma 4.2. We only have to prove the convergence.

Let $H^n = H \circ H^{n-1}$. Then by induction, the monotone property and Lemma 4.2

$$H^n v_0 \leq v_n \leq v'_n \leq v^*.$$

Since H is a contraction mapping, $H^n v_0$ converges to v^* when $n \to \infty$. Therefore, the sequence $\{v_n\}$ converges to v^*. ∎

The initial choice of the $v'_0 \leq Hv'_0$ is the key to getting the monotonic convergence for this algorithm. There are several methods which can be used to satisfy this requirement. One possible method involves starting the algorithm by choosing $v'_0(\pi) = \frac{1}{1-\beta}\{\max_{d \in D}[\min_{1 \leq i \leq N} r^d(i)]\}$ $\forall \pi \in \Pi$. This implies that the set A_0 contains only one vector and each element of this vector is $\frac{1}{1-\beta}\{\max_{d \in D}[\min_{1 \leq i \leq N} r^d(i)]\}$.

PROPOSITION 4.3:

If $v_0(\pi) = \frac{1}{1-\beta}\{\max_{d\in D}[\min_{1\le i\le N} r^d(i)]\}\ \forall \pi \in \Pi$, *then* $Hv_0 \ge v_0$.

Proof:

Let $\hat{d} = \arg\max_{d\in D}\{\min_{1\le i\le N}[r^d(i)]\}$ and $r_0 = \frac{1}{1-\beta}\{\max_{d\in D}[\min_{1\le i\le N} r^d(i)]\}$. Then $r^{\hat{d}}(i) \ge (1-\beta)\cdot r_0$ for $1 \le i \le N$. Let π be an arbitrary state in Π, then

$$\begin{aligned} Hv_0(\pi) &= \max_{d\in D}\{\pi \cdot r^d + \beta \sum_{\theta\in\Theta} \cdot \Pr(\theta|\pi,\hat{d}) \cdot v_0(T(\pi|d,\theta))\} \\ &= \max_{d\in D}\{\pi \cdot r^d + \beta \cdot r_0\} \\ &\ge \pi \cdot r^{\hat{d}} + \beta \cdot r_0 \\ &\ge \pi \cdot (1-\beta) r_0 \cdot 1 + \beta r_0 \\ &= r_0 \\ &= v_0(\pi) \end{aligned}$$

■

Note that during the process, the conditions $Hv_n' \ge v_n'$ and $\bar{v}_{n,m+1} \ge \bar{v}_{n,m}$ are always satisfied. If accurate results are desired, the previous results v_n' and $\bar{v}_{n,m}$ can be used as the new initial values.

Before closing this section, let us discuss the selection of states for a discrete phase. The ideal situation is to choose states that can generate supports which cannot be generated by other chosen states. In this case, supports of Hv_n at these chosen states completely determine Hv_n. However, it may be impossible to find such ideal states. The question then becomes how best to use the currently available information. The best estimation of Hv_n will be the current value v_n, especially when v_n is very close to the optimal solution v^*. If the operation H is performed by the relaxed region algorithm or the linear support algorithm discussed in Chapter 3, then all the extreme points of

the support regions are readily available. These extreme points may be a good start for quickly finding the supports. However, when a large number of such states are generated, a large amount of time may be required to perform an iteration in a discrete phase. The simple average of the vertices of each support region, which will be inside this region, may be an ideal alternative since it can reflect the information we now have, and thereby significantly reduce the number of chosen states.

Unlike the methods discussed in the previous chapter, the information generated in the current iteration in a discrete phase is not used in the current iteration. More precisely, the computation to generate $\bar{v}_{n,m}(\pi^i)$ is independent of the computation to generate $\bar{v}_{n,m}(\pi^j)$ for all the selected discrete states π^i and π^j. These computations depend only on the information generated from the previous iteration, i.e., $\bar{A}_{n,m-1}$. The advantages of this independence of information can be exploited by developing programs for parallel processing computers, thereby reducing the computational time required. The standard successive approximation algorithm and policy iteration algorithm developed by Sondik (1978) are not suitable for parallel processing. The feasibility of using parallel processing computers is an advantage peculiar to the iterative discretization procedure.

VI. Accelerating the Convergence

As mentioned in the previous section, only a few discrete states are used in an iteration in a discrete phase. Since discrete MDP is well developed, some of the techniques for

discrete MDP might, at least in some sense, be applied to POMDP to accelerate the convergence. In this section, three such methods are discussed. They are the Gauss–Seidel method, the action elimination procedure, and the modified policy iteration algorithm.

1. Gauss–Seidel Method:

Denardo (1982) discussed three methods for accelerating successive approximation for a discrete MDP. The second method, the Gauss–Seidel method, used the latest information in successive approximations. This method may be described as:

$$v_n(i) = \max_{d \in D}\{r^d(i) \ + \ \beta \sum_{i>j} P_{ij}^d \cdot v_n(j) \ + \ \beta \sum_{i \leq j} P_{ij}^d \cdot v_{n-1}(j)\}.$$

That is, all the values available before computing the $v_n(i)$ can be used for computing $v_n(i)$. A similar approach, easily applied to the POMDP setting, is discussed below.

Define $\bar{A}_{n,m}^1 = \bar{A}_{n,m}$ and $\bar{A}_{n,m}^i = \bar{A}_{n,m} \cup (\cup_{j<i} \bar{\alpha}^j(n, m+1))$ for $i > 1$; that is, all of the supports including those just generated in the current iteration are in $\bar{A}_{n,m}^i$. When the value of the state π^i is computed, all supports in $\bar{A}_{n,m}^i$ are used as candidates for $\alpha_{f(\pi^i,d,\theta,\bar{A}_{n,m}^i)}$. Then we have

$$\begin{aligned} \bar{v}_{n,m+1}(\pi^i) &= \max_{d \in D}\{\pi^i \cdot [r^d \ + \ \beta \sum_{\theta \in \Theta} P^d \cdot Q_\theta^d \cdot \alpha_{f(\pi^i,d,\theta,\bar{A}_{n,m}^i)}]\} \\ &= \pi^i \cdot \bar{\alpha}^i(n, m+1) \end{aligned} \qquad (4-13)$$

where $\alpha_{f(\pi^i,d,\theta,\bar{A}_{n,m}^i)} = \{\alpha \in \bar{A}_{n,m}^i : \ \pi^i \cdot P^d \cdot Q_\theta^d \cdot \alpha \geq \pi^i \cdot P^d \cdot Q_\theta^d \cdot \hat{\alpha} \ \forall \hat{\alpha} \in \bar{A}_{n,m}^i\}$.

The algorithm is similar to the one discussed in Section V, with the exception that in step 4, the new $\bar{v}_{n,m+1}(\pi)$ discussed in this section is used.

Example:

Let us illustrate the Gauss–Seidel method for the problem in the previous section. First compute $\bar{v}_{1,1}([0,1])$ using equation (4–13):

$$\begin{aligned}\bar{v}_{1,1}([0,1]) &= \max\{[0,1]\cdot\begin{bmatrix}-3.46\\5.35\end{bmatrix};[0,1]\cdot\begin{bmatrix}1.44\\4.80\end{bmatrix}\}\\ &= [0,1]\cdot\begin{bmatrix}-3.46\\5.35\end{bmatrix}\\ &= 5.35.\end{aligned}$$

The result of $\bar{v}_{1,1}([0,1])$ is the same as the example shown in the previous section. Now the generated support $\begin{bmatrix}-3.46\\5.35\end{bmatrix}$ is put into the support set $\bar{A}^1(1,0)$. Therefore, the support set $\bar{A}^1(1,0)$ contains three supports: $\begin{bmatrix}-4\\4\end{bmatrix}$, $\begin{bmatrix}0\\3\end{bmatrix}$, and $\begin{bmatrix}-3.46\\5.35\end{bmatrix}$. Since, $\begin{bmatrix}-3.46\\5.35\end{bmatrix} > \begin{bmatrix}-4\\4\end{bmatrix}$, the support $\begin{bmatrix}-4\\4\end{bmatrix}$ can be deleted from $\bar{A}^1(1,0)$. Now by using $\bar{A}^1(1,0)$ and equation (4–13), $\bar{v}_{1,1}([1,0])$ can be computed as

$$\begin{aligned}\bar{v}_{1,1}([1,0]) &= \max\{[1,0]\cdot\begin{bmatrix}-3.46\\5.35\end{bmatrix};[1,0]\cdot\begin{bmatrix}1.83\\5.26\end{bmatrix}\}\\ &= [1,0]\cdot\begin{bmatrix}1.83\\5.26\end{bmatrix}\\ &= 1.83\end{aligned}$$

Note that the value $\bar{v}_{1,1}([1,0])$ generated here is larger than the one generated in the example of the previous section.

Since both $\bar{v}_{1,1}([0,1]) - \bar{v}_{1,0}([0,1])$ and $\bar{v}_{1,1}([1,0]) - \bar{v}_{1,0}([1,0])$ are greater than ϵ_1, the second iteration is performed. The support set $\bar{A}_{1,1}$ is $\{\begin{bmatrix}-3.46\\5.35\end{bmatrix}, \begin{bmatrix}1.83\\5.26\end{bmatrix}\}$. Then,

by equation (4–13),

$$\begin{aligned}\bar{v}_{1,2}([0,1]) &= \max\{[0,1]\cdot\begin{bmatrix}-1.74\\7.19\end{bmatrix};[0,1]\cdot\begin{bmatrix}3.19\\6.50\end{bmatrix}\}\\ &= [0,1]\cdot\begin{bmatrix}-1.74\\7.19\end{bmatrix}\\ &= 7.19\end{aligned}$$

Include the support $\begin{bmatrix}-1.74\\7.19\end{bmatrix}$ with the support set $\bar{A}_{1,1}$ and then compute $\bar{v}_{1,2}([1,0])$:

$$\begin{aligned}\bar{v}_{1,2}([1,0]) &= \max\{[1,0]\cdot\begin{bmatrix}-1.74\\7.19\end{bmatrix};[1,0]\cdot\begin{bmatrix}3.55\\7.00\end{bmatrix}\}\\ &= [1,0]\cdot\begin{bmatrix}3.55\\7.00\end{bmatrix}\\ &= 3.55.\end{aligned}$$

Since both $\bar{v}_{1,2}([0,1]) - \bar{v}_{1,1}([0,1])$ and $\bar{v}_{1,2}([1,0]) - \bar{v}_{1,1}([1,0])$ are greater than ϵ_1, the procedure should not be terminated. Follow the same procedure to perform the third and fourth iterations. The support set after the fourth iteration is $\bar{A}_{1,4} = \{\begin{bmatrix}1.16\\10.09\end{bmatrix},\begin{bmatrix}6.35\\9.78\end{bmatrix}\}$. Now perform the fifth iteration:

$$\begin{aligned}\bar{v}_{1,5}([0,1]) &= \max\{[0,1]\cdot\begin{bmatrix}2.33\\11.26\end{bmatrix};[0,1]\cdot\begin{bmatrix}7.26\\10.57\end{bmatrix}\}\\ &= [0,1]\cdot\begin{bmatrix}2.33\\11.26\end{bmatrix}\\ &= 11.26\end{aligned}$$

Add $\begin{bmatrix}2.33\\11.26\end{bmatrix}$ into $\bar{A}_{1,4}$ and then compute $\bar{v}_{1,5}([1,0])$:

$$\begin{aligned}\bar{v}_{1,5}([1,0]) &= \max\{[1,0]\cdot\begin{bmatrix}2.33\\11.26\end{bmatrix};[1,0]\cdot\begin{bmatrix}7.48\\10.90\end{bmatrix}\}\\ &= [1,0]\cdot\begin{bmatrix}7.48\\10.90\end{bmatrix}\\ &= 7.48\end{aligned}$$

Now both $\bar{v}_{1,5}([0,1]) - \bar{v}_{1,4}([0,1])$ and $\bar{v}_{1,5}([1,0]) - \bar{v}_{1,4}([1,0])$ are less than ϵ_1, the procedure should be terminated. ■

Comparing the results of this example and the one in the previous section, we observe that the values of the selected states grow much faster in the Gauss–Seidel method, implying that this method may reduce the computational time.

THEOREM 4.4

If $Hv_0' \geq v_0'$ and $\bar{v}_{n,m}$ is computed from the Gauss–Seidel method, then

$$v_n \leq \bar{v}_{n,1} \leq \bar{v}_{n,m} \leq v_n' \leq Hv_n'.$$

The sequence $\{v_n\}$ converges to v^* monotonically.

Proof:

The proof is similar to the proof of Lemma 4.2 and Theorem 4.3 and it follows by induction and the monotone property. ■

Although we cannot prove Gauss–Seidel method will accelerate the convergence rate, numerical examples have shown that this method can reduce computation time.

2. Action Elimination:

To obtain results from (4–11), all available actions have to be used in the computations. If a large number of actions are available, this procedure may take a long

time to compute. However, if one or more actions can be identified as suboptimal for a chosen state, these actions need not be considered for this particular state. This idea is difficult to apply to a continuous state space since there are uncountably many states; however, the action elimination procedure can be applied in the iterative discretization procedure since only finitely many states are considered.

Let $U_n = \sup_{\pi\in\Pi}(Hv'_{n-1}(\pi) - v'_{n-1}(\pi))$ and $L_n = \inf_{\pi\in\Pi}(Hv'_{n-1}(\pi) - v'_{n-1}(\pi))$. As shown in Theorem 1, $Hv_{n-1} \geq v_{n-1}$. Therefore, U_n and L_n are both nonnegative real numbers. The following proposition is similar to the one used by Puterman and Shin (1982).

PROPOSITION 4.4:

Let $\bar{v}_{n,1}$ be the value calculated from (4-11) and (4-12). Then

$$\begin{aligned} v_n(\pi) + \frac{1}{1-\beta}\cdot L_n \cdot \mathbf{1} &\leq \bar{v}_{n,1}(\pi) + \frac{\beta}{1-\beta}\cdot L_n \cdot \mathbf{1} \\ &\leq v^*(\pi) \\ &\leq v_n(\pi) + \frac{1}{1-\beta}\cdot U_n \cdot \mathbf{1} \end{aligned}$$

THEOREM 4.5

Suppose in the n-th iteration that

$$\pi \cdot r^{\hat{a}} + \beta \sum_{\theta\in\Theta} \Pr(\theta|\pi,\hat{a}) v_n(T(\pi|\hat{a},\theta)) + \frac{\beta}{1-\beta}\cdot U_n \cdot \mathbf{1} \leq \bar{v}_{n,1}(\pi) + \frac{\beta}{1-\beta}\cdot L_n \cdot \mathbf{1}$$

where $\bar{v}_{n,1}$ is calculated from (4-11) and (4-12), then the action $\hat{a}$ is non optimal in state π.

Proof:

The proof directly follows from the Proposition 4.4 and MacQueen's lemma (1967).

■

The action elimination procedure discussed above is only useful in the standard procedure presented in Section III. To apply the action elimination procedure directly to the Gauss–Seidel method, a result similar to Theorem 4.5 is needed. However, this task is difficult to achieve since computing the improvement on $\bar{v}_{n,1}$ is not easy. To overcome this difficulty, the procedure may be appropriately modified.

Formulas (4–11) and (4–12) can be used to compute $\bar{v}_{n,1}$. Following Theorem 4.5, the suboptimal actions for $\pi \in \bar{\Pi}_n$ can be identified. The Gauss–Seidel method is then applied to compute $\bar{v}_{n,m+1}$, taking into account only those actions which are not suboptimal.

As discussed in Section IV, to compute U_n and L_n is not an easy task. Therefore, the action elimination method is useful only when there are a large number of actions. When the problem involves only very few actions, the action elimination method may not be able to reduce the computational time because the exact computation has to be performed.

3. Modified Policy Iteration:

The modified policy iteration algorithm for discrete MDP has been discussed in Puterman and Shin(1978). The same idea can be directly applied to this setting.

The procedure for computing Hv_n can be viewed as the policy improvement procedure in the modified policy iteration algorithm. The discrete phase can then be viewed as the value iteration in the modified policy iteration. That is, when we derive the set of discrete states in $\bar{\Pi}_n$, we also choose an action corresponding to each state in $\bar{\Pi}_n$. Then $\bar{v}_{n,m+1}^{\bar{\delta}(\pi)}(\pi)$ instead of $\bar{v}_{n,m+1}(\pi)$ is computed each iteration in this discrete phase. Since only one action is chosen for each selected state, it can save computational effort when a large number of actions are available. For each selected state, the action chosen can be the one used in Hv. Therefore, no extra effort is needed for selecting the actions.

Example:

Consider the example discussed in Section V and under the the section on the Gauss–Seidel method. If action 1 is taken for the state $[0,1]$, and action 2 for the state $[1,0]$, all the results are the same in both examples; however, when $\bar{v}_{n,m}([0,1])$ is computed only the first action has to be calculated, and when $\bar{v}_{n,m}([1,0])$ is computed only the second action has to be calculated. No maximization has to be performed. Since only one action is considered for each state, the computational time can be drastically reduced. ∎

THEOREM 4.6

Let $\bar{v}_{n,m}$ be computed from the modified policy iteration algorithm, then

$$v_n \leq \bar{v}_{n,1} \leq \bar{v}_{n,m} \leq v'_n \leq Hv'_n$$

and the sequence v_n converges to v^* monotonically.

Proof:

The proof follows from induction and the monotone property. ■

We remark that in modified policy iteration algorithm, only one action in each state $\pi \in \bar{\Pi}_n$ is considered. There is a trade–off between finding a potentially better solution and reducing computational time. When the policy for those states in $\bar{\Pi}_n$ is close to the policy chosen by the standard iterative discretization procedure discussed in the previous section, the value function derived from the modified policy iteration might be close to or equal to the one obtained from the standard method which performs optimization at every iteration. In this case, the modified policy iteration algorithm can reduce the time in finding values for different actions. On the other hand, if the actions used for states in $\bar{\Pi}_n$ are suboptimal, then the result obtained from a modified iteration might be far away from the solution obtained from the standard method, and might need more iterations for convergence.

The Gauss–Seidel method is unsuitable for parallel processing, whereas the action elimination method and modified policy iteration are suitable, making it advantageous to use the latter two methods.

VII. The Iterative Discretization Procedure with Approximate Policy Improvement

The iterative discretization procedure described in Sections V and VI used an exact policy improvement step (i.e., the operator H). As discussed in Chapter 3, the evaluation of H might be difficult when there are a large number of supports forming the value function. A natural question arises: can the approximation of the operator H, $\tilde{H}$ in Section III, be used in the iterative discretization procedure? In this section, this problem and a solution will be presented.

In Sections V and VI, the convergence of the iterative discretization procedures was demonstrated using the monotonicity property of the operator H. Unfortunately, the monotonicity property does not hold for the operator $\tilde{H}$. In fact, when $\tilde{H}$ is substituted for H in the algorithm, v_n will not necessarily converge to v^*. However, we will show that v_n is ϵ–optimal when n is sufficiently large. We will also determine error bounds of v_n which provide a termination criterion.

Now let us discuss the convergence of v_n to an ϵ–optimal neighbourhood. For simplicity, introduce the following notation which will be used in next theorem and proof. Let $v_0' \in V$ and $v_0' \leq v^*$. Define an operator $\bar{H}$ with the property that $\bar{H}^{m+1}v_k \geq \bar{H}^m v_k \geq v_k$ for all $m \geq 1$. We remark that the operations in the discrete phase discussed in the two previous section satisfy the assumption of $\bar{H}$. Define $\tilde{H}v_k' = v_{k+1}$ and $\bar{H}^{m+1}v_k = \bar{H} \circ \bar{H}^m v_k$. Moreover, let $n(k)$ be the number of operations of $\bar{H}$ in the k–th discrete phase and $\bar{H}^{n(k)}v_k = v_k'$. Note that, unlike the previous two sections, the only

limitation for choosing the initial value function is that it should not be larger than v^*, i.e. $v_0' \leq v^*$. Since $v_0' \leq v^*$, it is also true that $v_k \leq v^*$ and $v_k' \leq v^*$.

THEOREM 4.7:

If v_0' is the chosen starting value function such that $v_0' \leq v^*$, then an ϵ–optimal value function can be obtained by applying the iterative discretization procedure with approximation policy improvement.

Proof:

Let v^* be the optimal value function.

$$\begin{aligned}
\|\tilde{H}v_k' - v^*\| &\leq \|\tilde{H}v_k' - Hv_k'\| + \|Hv_k' - v^*\| \\
&\leq \mu + \beta \cdot \|v_k' - v^*\| \\
&= \mu + \beta \cdot \|\bar{H}^{n(k)}v_k - v^*\| \\
&\leq \mu + \beta \cdot \|v_k - v^*\| \\
&= \mu + \beta \cdot \|\tilde{H}v_{k-1}' - v^*\|.
\end{aligned}$$

where the third inequality arises since $v^* \geq \bar{H}^{n(k)}v_k \geq v_k$. Therefore, by recursion,

$$\|\tilde{H}v_k' - v^*\| \leq \sum_{i=0}^{k} \beta^i \cdot \mu + \beta^k \cdot \|v_0' - v^*\|.$$

When k approaches infinity, then the right hand side approaches $\mu/(1-\beta)$. If μ is chosen to be less than $(1-\beta)\cdot\epsilon$, then an ϵ–optimal solution is obtained for sufficiently large k. ■

Theorem 4.7 shows that the iterative discretization procedure with approximate policy improvement obtains an ϵ–optimal value function. However, this theorem does

not provide any information as to when an ϵ–optimal solution is obtained. An easy way to determine a stopping criterion is to use the values of v'_k and $\tilde{H}v'_k$. We can view v'_k as the current result and $\tilde{H}v'_k$ as the updated value. These two values can be used to decide whether or not an ϵ–optimal value function has been obtained. In this case, the result shown in Section III can be used. Therefore, if $\|\tilde{H}v'_k - v'_k\| \leq \frac{(1-\beta)\cdot\epsilon-\mu_k}{\beta}$ where $\mu_k = \|Hv'_k - \tilde{v}'_k\|$, then $\|\tilde{H}v'_k - v^*\| \leq \epsilon$. Similarly, Proposition 4.2 can also be used; that is, if $\frac{\beta\cdot(U_k-L_k)+\mu_k}{(1-\beta)} \leq \epsilon$, then $\tilde{H}v'_k + \frac{\beta\cdot L_k}{1-\beta}$ is an ϵ–optimal value function where $L_k = \inf_{\pi\in\Pi}\{(\tilde{H}v'_k)(\pi) - v'_k(\pi)\}$, $U_k = \sup_{\pi\in\Pi}\{(\tilde{H}v'_k)(\pi) - v'_k(\pi)\}$, and $\mu_k = \sup_{\pi\in\Pi}\{(Hv'_k)(\pi) - (\tilde{H}v'_k)(\pi)\}$.

Similarly, we can apply approximate policy improvement to the accelerated convergence methods discussed in Section VI. As might be expected, when a large number of supports are needed to form a value function, these methods might converge faster than the methods discussed in Section VI. The method discussed in this section together with the accelerated convergence techniques discussed in Section VI are especially well suited to complicated problems which require large numbers of supports to form a value function. Numerical comparisons for these methods will be provided in the next section.

VIII. Numerical Examples

In this section, several sets of test data are used to compare the efficiency of the algorithms discussed in this chapter. The basis of comparison is CPU time. All algorithms were implemented as Fortran 77 programs which were run on the Amdahl 5860

with FPU at the University of British Columbia.

The algorithm for computing Hv or its approximate value, $\tilde{H}v$, is the linear support algorithm which was developed in Chapter 3. Unless otherwise specified, $\frac{\beta \cdot (U_k - L_k) + \mu_k}{1-\beta}$ is used as the termination criterion. The vertices of relaxed regions are used to compute the approximate termination criterion as discussed in Section IV. If the approximate termination criterion is satisfied, the linear programming method is used to compute the exact termination criterion. The IMSL routines are used as the linear programming solver.

The test data can be divided into two groups. The first group contains the well known testing data discussed in Sondik (1978). The second group contains several sets of randomly generated data for the problems with three, four, and five system states. Unless otherwise specified, the discount factor β is 0.9.

1. Sondik's Testing Problem:

This test problem is the only testing data shown in the literature (Platzman 1981, White 1980, White and Scherer 1986). We will study this set of data first.

Since this is a two system states problem, the values of U_k, and L_k can be found exactly by the vertices of the relaxed regions. Linear programming is not required to find these values for this problem. The CPU times, the number of iterations, and error bounds for algorithms using the exact policy improvement discussed in this chapter are recorded and shown in Table 4.1.

Method	CPU Time (Seconds)	Number of Iterations	Number of Supports	Error Bound
Action Elimination	0.035	4	3	0.000275
Gauss-Seidel Method	0.040	4	3	0.000137
Modified Policy Iteration	0.035	4	3	0.000275
Standard IDP	0.041	4	3	0.000283
Successive Approximation	0.027	7	3	0.000730

ϵ for Stopping Criterion: 0.01
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.001
Tolerable Error for Policy Improvement: 0.0

Table 4.1: Results of Sondik's Example (with Exact Policy Improvement)

From Table 4.1, all of these algorithms converge really quickly. Among these algorithms, the standard successive approximation method requires the least CPU time although the error bound is slightly higher than other methods. This is because it only takes seven iterations to get a convergence result for the standard successive approximation. Although other algorithms only take four iterations to converge, they perform several iterations in a discrete phase. As a result, for this simple problem, the standard successive approximation method performs better than other methods.

The result for this problem by using the algorithms with tolerable error for the policy improvement is shown in Table 4.2. This result is similar to the result in Table 4.1 because this testing problem has a finitely transient optimal policy.

If $\|Hv - v\|$ is used to calculate the error bound, the result is shown in Table 4.3. This result is similar to the results shown in two previous tables except for the

Method	CPU Time (Seconds)	Number of Iterations	Number of Supports	Error Bound
Action Elimination	0.035	4	3	0.000275
Gauss-Seidel Method	0.040	4	3	0.000137
Modified Policy Iteration	0.035	4	3	0.000275
Standard IDP	0.041	4	3	0.000283
Successive Approximation	0.024	7	3	0.000944

ϵ for Stopping Criterion: 0.01
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.001
Tolerable Error for Policy Improvement: 0.0005

Table 4.2: Results of Sondik's Example (with Approximate Policy Improvement)

Method	CPU Time (Seconds)	Number of Iterations	Number of Supports	Error Bound
Action Elimination	0.036	4	3	0.005768
Gauss-Seidel Method	0.041	4	3	0.004944
Modified Policy Iteration	0.036	4	3	0.005356
Standard IDP	0.052	5	3	0.006729
Successive Approximation	0.307	71	3	0.009476

ϵ for Stopping Criterion: 0.01
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.001
Tolerable Error for Policy Improvement: 0.0

Table 4.3: Results of Sondik's Example (Use $\|Hv - v\|$ to Compute Error Bound)

standard successive approximation method. The standard successive approximation method requires seventy–one iterations to get a convergence result. It also requires 6 to 8 times more CPU time to complete the computation. This is due to the fact that

there are many iterations in a discrete phase of any IDP method and each iteration in a discrete phase does a similar job as an iteration in the standard successive approximation method. Therefore, unlike the standard successive approximation, the IDP methods are not sensitive to the methods of computing error bound.

Since this problem has a finitely transient optimal policy and only three linear supports are needed to form the optimal value function, the algorithms with tolerable error for the policy improvement obtain the exactly same results as presented in Table 4.3 if $\|Hv - v\|$ is used to compute the error bound.

2. Randomly Generated Data:

This group consists five sets of data range from three to six system states. These data are listed in the Appendix 2. We did not use the data in Appendix 1 because the calculations either converged too fast or did not converge within a reasonable time limit.

The first set of randomly generated data is a three system states, three actions, and three signal problem. If the algorithms with the exact policy improvement are used, there are too many linear supports generated (more than fifty supports), which is more than the design of the code can handle. However, if the algorithms with approximate policy improvement are used, the problem of space can be resolved. This is one of the advantages of using the algorithms with approximate policy improvement. The CPU time, number of iterations, number of supports and error bound for each method are

shown in Table 4.4. In order to compare the methods in more detail, we also show the CPU time less the time spent in using linear programming to calculate U_n and L_n.

Method	CPU Time		Number of Iterations	Number of Supports	Error Bound
	Total	No LP			
Action Elimination	4.045	3.676	8	13	0.059359
Gauss-Seidel Method	4.093	3.718	10	13	0.058961
Modified Policy Iteration	3.922	3.561	10	13	0.058961
Standard IDP	3.303	2.940	8	13	0.062985
Successive Approximation	11.555	11.271	18	12	0.083387

ϵ for Stopping Criterion: 0.1
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.01
Tolerable Error for Policy Improvement: 0.005

Table 4.4: Results of Data Set 1 (with Approximate Policy Improvement)

From this example, we can see that all IDP methods require much less CPU time and have better error bounds than the standard successive approximation method. The IDP methods just need about one third of the CPU time of the standard successive approximation method. Among the IDP methods, standard IDP requires slightly less CPU time but has slightly larger error bound. Although both the action elimination method and standard IDP require eight iterations to get the solution and the action elimination method has fewer iterations in the discrete phases (not shown in Table 4.4), the action elimination method requires more time than standard IDP. This might be due to the fact that not many actions are eliminated and special effort is required to check which actions can be eliminated.

In order to find the effect of the action elimination method, let us consider the second data set. This is a three system states, six actions, and three signals problem. The results of algorithms with exact policy improvement are shown in Table 4.5.

Method	CPU Time		Number of Iterations	Number of Supports	Error Bound
	Total	No LP			
Action Elimination	0.669	0.484	5	11	0.030075
Gauss-Seidel Method	1.090	0.905	5	11	0.020606
Modified Policy Iteration	0.816	0.631	5	11	0.042023
Standard IDP	0.647	0.537	4	9	0.028022
Successive Approximation	3.356	3.132	8	11	0.034083

ϵ for Stopping Criterion: 0.1
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.01

Table 4.5: Results of Data Set 2 (with Exact Policy Improvement)

From Table 4.5, we can see that the action elimination method requires less CPU time than the Gauss–Seidel method and the modified policy iteration although all three methods require five iterations to obtain the solutions. This result is as we expected since there are relatively large number of actions available and the action elimination method can omit the suboptimal actions. The modified policy iteration method only uses one action for each selected state in a discrete phase; however, in this case, the suboptimal actions might be chosen for some selected states. As a result, the modified policy iteration method requires more iterations in one discrete phase (not shown in Table 4.5). The standard IDP method requires slightly less CPU time to obtain a

solution than the action elimination method because it only takes four iterations to obtain its solution although it has more iterations in the discrete phases. As expected, the standard successive approximation takes the longest time to obtain its solution.

The effect on the number of actions can be detected more easily by the algorithms with approximate policy improvement since there are fewer supports in the solutions. The time spent in computing policy improvement will be less if there are fewer supports in the support set. The results of using data set 2 are shown in Table 4.6.

Method	CPU Time		Number of Iterations	Number of Supports	Error Bound
	Total	No LP			
Action Elimination	0.466	0.391	5	7	0.045400
Gauss-Seidel Method	0.692	0.618	4	7	0.080924
Modified Policy Iteration	0.448	0.374	5	7	0.050268
Standard IDP	0.591	0.519	4	7	0.017349
Successive Approximation	0.971	0.903	8	7	0.057928

ϵ for Stopping Criterion: 0.1
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.01
Tolerable Error for Policy Improvement: 0.005

Table 4.6: Results of Data Set 2 (with Approximate Policy Improvement)

From Table 4.6, we can see that although requiring more iterations than the Gauss–Seidel Method and the standard IDP method, the modified policy iteration method and the action elimination method require less CPU time to obtain their solutions. This result is to be expected since the time required for the action elimination method or

the modified policy method does not relate directly to the number of available actions. The modified policy iteration method needs slightly less CPU time than the action elimination method because only one action is chosen for each state and no extra effort is required for the selection of actions.

The next set of randomly generated data, data set 3, is a four system states, four actions, and four signals problem. Table 4.7 contains the results of algorithms with exact policy improvement steps.

Method	CPU Time		Number of Iterations	Number of Supports	Error Bound
	Total	No LP			
Action Elimination	16.537	16.044	4	14	0.022665
Gauss-Seidel Method	16.647	16.206	4	13	0.022324
Modified Policy Iteration	16.501	16.060	4	13	0.022324
Standard IDP	16.683	16.201	4	14	0.006259
Successive Approximation	> 100 Seconds				

ϵ for Stopping Criterion: 0.1
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.01

Table 4.7: Results of Data Set 3 (with Exact Policy Improvement)

From Table 4.7, we find that the CPU time required for all IDP methods are about the same. However, the standard successive approximation method is unable to yield a solution within 100 seconds.

In contrast to the results shown in Table 4.7, the standard successive approxima-

tion method with approximate policy improvement steps can obtain its solution within twelve seconds. (See Table 4.8.) Among the IDP methods shown in Table 4.8, the CPU time required varies significantly. The action elimination method can obtain its solution in two seconds. However, the Gauss–Seidel method needs 6.492 seconds to obtain its solution. This is because the action elimination method eliminates most of the suboptimal actions in every iteration and only a few actions are considered.

Method	CPU Time		Number of Iterations	Number of Supports	Error Bound
	Total	No LP			
Action Elimination	1.926	1.756	4	9	0.064620
Gauss-Seidel Method	6.492	6.153	6	9	0.057871
Modified Policy Iteration	6.323	5.988	6	9	0.057871
Standard IDP	3.425	3.256	4	9	0.051241
Successive Approximation	11.547	11.222	8	9	0.091206

ϵ for Stopping Criterion: 0.1
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.01
Tolerable Error for Policy Improvement: 0.005

Table 4.8: Results of Data Set 3 (with Approximate Policy Improvement)

For the last two data sets, data set 4 and data set 5, the algorithms with the exact policy improvement steps generate more supports than the fifty supports that the design of the codes can accommodate. Therefore, no results are generated. Moreover, if an error bound of 0.1 is used in the previous three data sets, then the algorithms with approximate policy improvement will also generate more than fifty supports. Therefore, in these two data sets, only the algorithms with approximate policy improvement are

considered, and the error bound is set to be 0.5.

Now let us consider the fourth data set which is a five system states, five actions, and five signals problem. The results can be obtained in relatively short time. From Table 4.9, we find that all IDP methods require much less CPU times than the standard successive approximation method. Among the IDP methods, the modified policy iteration method requires less CPU time than other methods in this data set.

Method	CPU Time		Number of Iterations	Number of Supports	Error Bound
	Total	No LP			
Action Elimination	2.296	2.142	5	8	0.396919
Gauss-Seidel Method	1.991	1.807	4	9	0.452781
Modified Policy Iteration	1.882	1.732	4	8	0.362355
Standard IDP	2.322	2.171	4	8	0.373904
Successive Approximation	9.187	8.850	10	8	0.469870

ϵ for Stopping Criterion: 0.5
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.05
Tolerable Error for Policy Improvement: 0.025

Table 4.9: Results of Data Set 4 (with Approximate Policy Improvement)

The last set of randomly generated data is a six system states, four actions, and six signals problem. The results for the algorithms with approximate policy improvement steps are shown in Table 4.10. From this table, we find that the CPU time requirements for all of the algorithms are about the same. The modified policy iteration requires the least CPU time among all methods. As we discussed before, there are several iterations

of computations in each discrete phase of IDP methods. In this example, the standard successive approximation method, which does not involve any discrete phase, only takes one or two more iterations of policy improvement than the IDP methods. However, the standard successive approximation method requires more CPU time than any of the IDP methods, which implies that each iteration of policy improvement might require more CPU time to perform than the computation of several iterations in a discrete phase.

Method	CPU Time		Number of Iterations	Number of Supports	Error Bound
	Total	No LP			
Action Elimination	26.919	26.205	4	13	0.347050
Gauss-Seidel Method	27.250	26.532	4	13	0.346822
Modified Policy Iteration	24.497	23.792	5	13	0.319670
Standard IDP	26.892	26.189	5	13	0.279474
Successive Approximation	27.910	27.387	6	12	0.330867

ϵ for Stopping Criterion: 0.5
$\epsilon_1(n)$ for Stopping Discrete Phase: 0.05
Tolerable Error for Policy Improvement: 0.025

Table 4.10: Results of Data Set 5 (with Approximate Policy Improvement)

The above examples clearly demonstrate that there are benefits to be reaped in using the algorithms with the approximate policy improvement steps. For the same accuracy, the algorithms with the approximate policy improvement steps generate fewer linear supports than their counterparts with the exact policy improvement. As a result, less computer memory and CPU time are needed for the algorithms with approximate

policy improvement. The algorithms with approximate policy improvement steps might also produce a more stable solution since there are fewer chances that two linear supports are similar. From the above examples, we have also found that the IDP methods require less CPU time to obtain the solution than the standard successive approximation method. These examples also suggest that the fewer iterations of policy improvement required is one of the major reasons that IDP methods are more efficient. Of course, we cannot conclude that IDP methods are more efficient than the standard successive approximation just from these limited examples. However, these limited examples have shown that a large portion of CPU time might be saved by using IDP methods.

What is the best method among IDP algorithms? This is difficult to answer from these limited examples. We have found from these examples that no single method always performs better than other methods. This is because all performance results are data dependent. Only a rule of thumb might be formulated: when there are a large number of available actions, the action elimination method and the modified policy iteration method are recommended since these two methods only consider a subset of available actions.

CHAPTER 5

PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES WITH CONTINUOUS SIGNAL DISTRIBUTIONS

In the last few chapters, partially observable Markov processes with finite discrete signal space were studied. The assumption of finite discrete signal space is restrictive since in many contexts it is more natural to model the signal space as continuous instead of finite. For example, in a machine replacement problem, the signal might be the temperature of this machine. This temperature could be any value within a certain range. As another example, consider blood pressure, which is a good indicator of certain diseases and which is frequently modeled with a continuous distribution. Moreover, if there are a large number of discrete signals, it is sometimes easier to model the problem as a continuous signal space. Therefore, it is desirable to extend the algorithms discussed in the last two chapters to a more general signal space.

There has been some research on POMDP with a general signal space. Whittle (1982) discussed some of the applications of POMDP to statistical inference and sequential design. Nir (1986) discussed control problems with two system states. However, these researchers relied on the particular structure of the problems studied. In this chapter, a more general approach to the problem will be discussed.

The plan of this chapter is as follows: assumptions, notation, and formulation of POMDP with a continuous signal space are presented in Section I; a method to convert a POMDP problem with uniformly distributed signal processes to a POMDP with

finite signals is presented in Section II; and applications of the algorithms developed in Chapter 3 and 4 to solve the POMDP problems with continuous signals (under certain assumptions hold) are discussed in Section III.

I. Assumptions, Notation, and Formulation

The basic assumptions of this chapter are the same as in Chapter 2. The only difference is the nature of the signal processes. In this chapter, we assume that the signal distributions have density functions, whereas in Chapter 2 we assume that they are discrete. The parameters of these probability density functions depend on the system state as well as the action taken in the previous decision epoch. More precisely, for each system state $i \in S$, each action $d \in D$, and each time $t = 0, 1, 2, \ldots$, there is a probability density function $f^d_{i,t}(\cdot)$ on the set of signals. For the infinite horizon problem, we assume that the probability density function is time invariant and the dependency on t is suppressed from the notation.

Let $\pi \in \Pi$ be defined as in Chapter 2. Given that the current distribution on the state space is π and action d is used, the conditional density function on the set of signal is $\tilde{f}(\cdot|\pi, d)$. Then this conditional density function can then be computed as

$$\tilde{f}(\theta|\pi, d) = \sum_{k=1}^{N} \sum_{j=1}^{N} \pi_j \cdot P^d_{j,k} \cdot f^d_k(\theta).$$

Or, in matrix form,

$$\tilde{f}(\theta|\pi, d) = \pi \cdot P^d \cdot \tilde{Q}^d_\theta \cdot \mathbf{1}, \qquad (5-1)$$

where $\tilde{Q}_\theta^d$ is a diagonal matrix with $f_i^d(\theta)$ as its diagonal elements and $\mathbf{1}$ is an N–dimensional column vector with all elements being 1. Analogous to the definition of $T(\pi, d, \theta)$, define $\tilde{T}(\pi, d, \theta)$ as the probability distribution of the system state at the next time epoch, given that the probability distribution of the current system state is π, the action applied is d, and the signal obtained in the next time epoch is θ; i.e.,

$$\tilde{T}_i(\pi, d, \theta) = \Pr(X_{t+1} = i | \pi, Y_t = d, Z_{t+1} = \theta),$$

and

$$\tilde{T}(\pi, d, \theta) = [\tilde{T}_i(\pi, d, \theta)].$$

Then, by Bayes' rule,

$$\begin{aligned} \tilde{T}_i(\pi, d, \theta) &= \Pr(X_{t+1} = i | \pi, Y_t = d, Z_{t+1} = \theta) \\ &= \frac{\sum_{j=1}^N \pi_j \cdot p_{j,i}^d \cdot f_i^d(\theta)}{\sum_{k=1}^N \sum_{j=1}^N \pi_j \cdot p_{j,k}^d \cdot f_k^d(\theta)}. \end{aligned} \tag{5-2}$$

Or, in matrix form,

$$\tilde{T}(\pi, d, \theta) = \frac{\pi \cdot P^d \cdot \tilde{Q}_\theta^d}{\tilde{f}(\theta | \pi, d)} = \frac{\pi \cdot P^d \cdot \tilde{Q}_\theta^d}{\pi \cdot P^d \cdot \tilde{Q}_\theta^d \cdot \mathbf{1}}. \tag{5-3}$$

If $\tilde{f}(\theta | \pi, d) = 0$, then $\tilde{T}(\pi, d, \theta)$ can be arbitrary and will not affect the following analysis.

As in Chapter 2, let v_t and v_{t+1} be bounded convex continuous value functions at time t and $t+1$, respectively. Then, for $\pi \in \Pi$,

$$\begin{aligned} v_t(\pi) &= \max_{d \in D} E\{r^d(X_t) + \beta \cdot v_{t+1}(\tilde{T}(\pi, d, \theta)) | \pi, d\} \\ &= \max_{d \in D} \{\sum_{i=1}^N \pi_i \cdot r^d(i) + \beta \cdot \int_{\theta \in \Theta} \tilde{f}(\theta | \pi, d) \cdot v_{t+1}(\tilde{T}(\pi, d, \theta)) d\theta\}. \end{aligned} \tag{5-4}$$

If v_{t+1} is piecewise linear function on the domain Π, then the formula (5–3) can be rewritten as

$$v_t(\pi) = \max_{d \in D}\{\sum_{i=1}^{N} \pi_i \cdot r^d(i) + \beta \cdot \int_{\theta \in \Theta} \tilde{f}(\theta|\pi, d) \cdot v_{t+1}(\tilde{T}(\pi, d, \theta))d\theta\}$$

$$= \max_{d \in D}\{\sum_{i=1}^{N} \pi_i \cdot r^d(i) + \beta \cdot \int_{\theta \in \Theta} \pi \cdot P^d \cdot \tilde{Q}^d_\theta \cdot \alpha^{g(\pi,d,\theta)} d\theta\} \qquad (5-5)$$

where $\alpha^{g(\pi,d,\theta)}$ is support of v_{t+1} and $\tilde{T}(\pi, d, \theta) \cdot \alpha^{g(\pi,d,\theta)} \geq \tilde{T}(\pi, d, \theta) \cdot \alpha^k$ for all α^k which are supports of v_{t+1}. Notice that the formula (5–5) is similar to the corresponding formula for the discrete signal case. The only difference is that the summation is replaced by an integral.

White and Harrington (1980) showed that if v_{t+1} is a convex function, so is v_t. However, unlike the finite signal case, v_t need not be piecewise linear even though v_{t+1} is piecewise linear. This property makes continuous signal problems more difficult than discrete signal problems.

II. Uniformly Distributed Signal Processes

A uniform distribution is commonly used to model a process without much available information. Nir (1986) studied a two–state POMDP with uniformly distributed costs as its signals. We discuss this distribution separately from others because this problem can be reformulated as a discrete signal space problem. The algorithms developed for the discrete signal space can then be applied to solve this type of problem and are more efficient than the method which will discussed in later sections.

Let Θ be the signal space. Also, at decision epoch t, let $\Theta^d_{t,i}$ be the signal space for the process given that the system state is i and that the decision taken at previous decision epoch is d. The probability density function for the signal in this signal space is uniformly distributed.

A trivial case occurs if the state can be deduced for sure from the observed signal; i.e., $\Theta^d_{t,i} \cap \Theta^d_{t,j} = \emptyset$ for all pairs of states i and j. This can clearly be formulated as a completely observable MDP. For example, suppose in a two state control problem that a machine in the good state has an operating cost in the range of 100 to 250 dollars per day; however, if the machine is in the bad state, the operating cost is in the range of 300 to 450 dollars per day. So if the current operating cost is 350 dollars, it is obvious that the system is in the bad state.

Of course, the above technique fails if the supports of the signal distributions overlap. However, if the signal distributions are uniform, then the problem can be reformulated as a POMDP with finite signal space. Let $\tilde{\Theta}^d_{t,i} = \Theta \backslash \Theta^d_{t,i}$ and $\bar{\Theta}^d_{t,i} = \{\Theta^d_{t,i}, \tilde{\Theta}^d_{t,i}\}$. Then $\bar{\Theta}^d_{t,i}$ is a partition of the signal space Θ. Let $\Theta_t = \{\hat{\Theta}_{t,1}, \hat{\Theta}_{t,2}, \ldots, \hat{\Theta}_{t,k}\}$ be the product partition of $\bar{\Theta}^d_{t,i}$ for all system states i and actions d. Since there are only a finite number, N, of system states, there are only a finite number of elements in Θ_t. The key to converting a uniformly distributed signal problem to a discrete signal problem is that the only information provided by the signal is the cell of the partition in which it occurs.

Each element in Θ_t can be viewed as a signal in a finite signal space problem.

For example, consider a machine with an operating cost that depends on its condition. When the machine is in excellent condition, the operating cost is uniformly distributed between \$100 and \$250 per day. If the machine is in fair condition, the operating cost is uniformly distributed between \$200 and \$350 per day. And if the machine is in bad condition, the operating cost is uniformly distributed between \$300 and \$450 per day. Thus the cost can be divided into five regions – that is, \$100 to \$200, \$200 to \$250, \$250 to \$300, \$300 to \$350, and \$350 to \$450. Each region can be viewed as a distinct signal. Since the observations are uniformly distributed in their signal space, the probabilities of each of the newly defined signals can be calculated by the integral of the density function on each of the regions in Θ_t. For example, there is 66.67% chance that the operating cost is between \$100 and \$200, and 33.33% of chance the operating cost is between \$200 and \$250 if the machine is in excellent condition. Similarly, the probability for each cost range can be computed for the fair and the bad machine conditions. Therefore, this problem is converted to a discrete signal space problem.

The key reason that a POMDP with uniformly distributed signal processes can be modeled as a POMDP with a finite number of signals is because $\tilde{T}(\pi, d, \cdot)$ is constant over each element of Θ_t. Let us show this result in the following lemmas.

LEMMA 5.1:

For every system state $i \in S$, $f_i^d(\cdot)$ is constant on every element of Θ_t.

Proof:

Let θ_1 and θ_2 be any two arbitrary signals in any $\hat{\Theta}_{t,j} \in \Theta_t$. By the method discussed above for generating the elements in Θ_t, either $\hat{\Theta}_{t,j} \cap \Theta_{t,i}^d = \emptyset$ or $\hat{\Theta}_{t,j} \subseteq \Theta_{t,i}^d$

for all system states i. If $\hat{\Theta}_{t,j} \cap \Theta^d_{t,i} = \emptyset$, then $f^d_i(\theta_1) = f^d_i(\theta_2) = 0$. If $\hat{\Theta}_{t,j} \subseteq \Theta^d_{t,i}$, then, by the uniform assumption, $f^d_i(\theta_1) = f^d_i(\theta_2)$. ■

Although this lemma is trivial, it is the primary reason why POMDP with uniformly distributed signal processes can be reformulated as POMDP with a finite number of signals. Moreover, in the proof of this lemma, we only require that $\hat{\Theta}_{t,j}$ is a subset of $\Theta^d_{t,i}$ or $\hat{\Theta}_{t,j} \cap \Theta^d_{t,i} = \emptyset$ for all system states i and that the signal processes are uniformly distributed on $\hat{\Theta}_{t,j}$. Hence, we can extend this result to more general cases where the signal processes are step functions. We will discuss this issue later in this section.

Since $f^d_i(\cdot)$ is constant on any element of Θ_t, then $\tilde{Q}^D_\theta$ is the same for all θ in any element of Θ_t. Moreover, $\tilde{f}(\theta|\pi, d) = \pi \cdot P^d \cdot Q^d_\theta \cdot \mathbf{1}$, therefore, $\tilde{f}(\cdot|\pi, d)$ is constant on every element in Θ_t. Now we can show that $\tilde{T}(\pi, d, \cdot)$ is constant on each element of Θ_t.

LEMMA 5.2:

$\tilde{T}(\pi, d, \cdot)$ is constant on each element of Θ_t.

Proof:

Let $\hat{\Theta}_{t,j}$ be an arbitrary element of Θ_t and θ_1, θ_2 be any two arbitrary signals in $\hat{\Theta}_{t,j}$. Since $\tilde{Q}^d_{\theta_1} = \tilde{Q}^d_{\theta_2}$ and $\tilde{f}(\theta_1|\pi, d) = \tilde{f}(\theta_2|\pi, d)$, by (5–3), $\tilde{T}(\pi, d, \theta_1) = \tilde{T}(\pi, d, \theta_2)$. The result follows. ■

Now let us show that if the signal processes are uniformly distributed, then Equation (5–4) can be rewritten as Equation (2–4). Let us consider the integration part of

Equation (5–4) first. Let θ_i be an arbitrary signal in $\hat{\Theta}_{t,i}$. By Lemma 5.2,

$$\begin{aligned}
&\int_{\theta\in\hat{\Theta}_{t,i}} \tilde{f}(\theta|\pi,d)\cdot v_{t+1}(\tilde{T}(\pi,d,\theta))d\theta \\
=&v_{t+1}(\tilde{T}(\pi,d,\theta_i))\cdot\int_{\theta\in\hat{\Theta}_{t,i}} \tilde{f}(\theta|\pi,d)\ d\theta \\
=&v_{t+1}(\tilde{T}(\pi,d,\theta_i))\cdot \Pr(\theta\in\hat{\Theta}_{t,i}|\pi,d) \\
=&\Pr(\theta\in\hat{\Theta}_{t,i}|\pi,d)\cdot v_{t+1}(\tilde{T}(\pi,d,\theta_i)).
\end{aligned}$$

By Lemma 5.2, each element in Θ_t can be viewed as a signal. Define $T(\pi,d,\hat{\Theta}_{t,i}) = \tilde{T}(\pi,d,\theta_i)$ for $\theta_i\in\hat{\Theta}_{t,i}$. Since there are only a finite number of elements in Θ_t, Equation (5–4) can be rewritten as

$$\begin{aligned}
v_t(\pi) &= \max_{d\in D}\{\sum_{i=1}^{N}\pi_i\cdot r^d(i)+\beta\cdot\int_{\theta\in\Theta_t}\tilde{f}(\theta|\pi,d)\cdot v_{t+1}(\tilde{T}(\pi,d,\theta))d\theta\} \\
&= \max_{d\in D}\{\sum_{i=1}^{N}\pi_i\cdot r^d(i)+\beta\cdot\sum_{\hat{\Theta}_{t,i}\in\Theta_t}\int_{\theta\in\hat{\Theta}_{t,i}}\tilde{f}(\theta|\pi,d)\cdot v_{t+1}(\tilde{T}(\pi,d,\theta))d\theta\} \\
&= \max_{d\in D}\{\sum_{i=1}^{N}\pi_i\cdot r^d(i)+\beta\cdot\sum_{\hat{\Theta}_{t,i}\in\Theta_t}\Pr(\hat{\Theta}_{t,i}|\pi,d)\cdot v_{t+1}(T(\pi,d,\hat{\Theta}_{t,i}))\},
\end{aligned}$$

where $\Pr(\hat{\Theta}_{t,i}|\pi,d) = \int_{\theta\in\hat{\Theta}_{t,i}}\tilde{f}(\theta|\pi,d)d\theta$. The last equality is the same as Equation (2–4). Therefore, Equation (5–4) can be rewritten as (2–4); that is, POMDP with uniformly distributed signal processes can be reformulated as POMDP with a finite number of signals.

As discussed above, the key reason that POMDP with uniformly distributed signal processes can be formulated as POMDP with finite number of signals is that the density function $f_i^d(\cdot)$ is constant on every element of Θ_t for every system state $i\in S$ and action $d\in D$. Therefore, if other POMDP problems have this property, by following the same

procedures we can show that they can be formulated as finite signal POMDP problems. One example is POMDP problems where signal processes are step functions.

Let us consider a POMDP problem which signal processes are step function. Assume that $\bar{\Theta}^d_{t,i}$ is a partition of signal space Θ_t, and the density function is constant on $\bar{\Theta}^d_{t,i}$ for every system state $i \in S$ and action $d \in D$. A product partition of $\bar{\Theta}^d_{t,i}$ for all system state i and action d can be performed and denoted as $\Theta_t = \{\hat{\Theta}_{t,1}, \hat{\Theta}_{t,2}, \ldots, \hat{\Theta}_{t,k}\}$. Therefore, the density function $f^d_i(\cdot)$ is constant in each element in Θ_t for each system state i and action d. By following the same procedures as discussed above, it can be shown that $\tilde{T}(\pi, d, \cdot)$ is also constant on each element in Θ_t. Therefore, POMDP whose signal processes are step functions can also be formulated as POMDP with finite signals.

In practice, there might not be many applications whose signal processes are step functions. However, step functions can naturally be used to approximate any probability distribution. After the signal processes have been approximated by step functions, the problem can then be solved by the methods discussed in Chapters 3 and 4. Using this approach, the general problem with continuous signal space can be solved.

Now let us discuss how to use step functions to approximate the distributions of the signal processes. For simplification, we assume that the signal space is time–invariant, and the time subscript, t, is suppressed. For each action $d \in D$, and system state $i \in S$, the signal space Θ is divided into a finite number of connected subsets, $\Theta^d_i = \{\Theta^d_{i,k}\}$, such that $\cup_k \Theta_{i,k} = \Theta$ and $\Theta^d_{i,k} \cap \Theta^d_{i,l} = \emptyset$ for all pair of k and l. In order to have a good approximation, we usually require the density function to be continuous in each of the

subsets.

The product partition of Θ_i^d for all system states $i \in S$ and actions $d \in D$ can be performed and denoted as $\hat{\Theta} = \{\hat{\Theta}_1, \hat{\Theta}_2, \ldots, \hat{\Theta}_l\}$. If the conditional density function $\tilde{f}(\theta|\pi, d)$ is defined as in the first section, then

$$\Pr(\theta \in \hat{\Theta}_k|\pi, d) = \int_{\theta \in \Theta_k} \tilde{f}(\theta|\pi, d)d\theta \qquad \forall \pi \in \Pi, \ d \in D, \quad \text{and } \Theta_k \in \hat{\Theta}.$$

Since $f_i^d(\theta)$ is continuous in each subset in $\hat{\Theta}$ for all i and d, $\tilde{f}(\theta|\pi, d)$ is also continuous in each set in $\hat{\Theta}$.

Since each set in $\hat{\Theta}$ can be viewed as a signal and the conditional density functions for these signals are defined, we now have a finite signal POMDP problem. The accuracy of the approximation depends on how the signal space is partitioned. Let us now develop an error bound for this approximation.

Let π be an arbitrary state in Π, v a given value function, Hv the exact value function, and $\bar{H}v$ the value function obtained from the approximation. If $d \in D$ is the action used for Hv at π and $Hv(\pi) \geq \bar{H}v(\pi)$, then

$$\begin{aligned}
Hv(\pi) - \bar{H}v(\pi) &\leq \pi \cdot r^d + \beta \cdot \int_{\theta \in \Theta} \tilde{f}(\theta|\pi, d) \cdot v(\tilde{T}(\pi, d, \theta))d\theta \\
&\quad - (\pi \cdot r^d + \beta \cdot \sum_{\hat{\Theta}_k \in \hat{\Theta}} \Pr(\hat{\Theta}_k|\pi, d) \cdot v(T(\pi, d, \hat{\Theta}_k))) \\
&= \beta \cdot \int_{\theta \in \Theta} \tilde{f}(\theta|\pi, d) \cdot v(\tilde{T}(\pi, d, \theta))d\theta - \beta \cdot \sum_{\hat{\Theta}_k \in \hat{\Theta}} \Pr(\hat{\Theta}_k|\pi, d) \cdot v(T(\pi, d, \hat{\Theta}_k)) \\
&= \beta \cdot \sum_{\hat{\Theta}_k \in \hat{\Theta}} \int_{\theta \in \hat{\Theta}_k} \tilde{f}(\theta|\pi, d) \cdot v(\tilde{T}(\pi, d, \theta))d\theta \\
&\quad - \beta \cdot \sum_{\hat{\Theta}_k \in \hat{\Theta}} \Pr(\hat{\Theta}_k|\pi, d) \cdot v(T(\pi, d, \hat{\Theta}_k))
\end{aligned}$$

$$\leq \beta \cdot \sum_{\hat{\Theta}_k \in \hat{\Theta}} \left(\int_{\theta \in \hat{\Theta}_k} \tilde{f}(\theta|\pi, d) \cdot M_k d\theta - \Pr(\hat{\Theta}_k|\pi, d) \cdot L_k \right)$$
$$= \beta \cdot \sum_{\hat{\Theta}_k \in \hat{\Theta}} \Pr(\hat{\Theta}_k|\pi, d) \cdot (M_k - L_k)$$
$$\leq \beta \cdot (\tilde{M}_k - \tilde{L}_k)$$

where $M_k = \max_{\theta \in \hat{\Theta}_k} v(\tilde{T}(\pi, d, \theta))$, $L_k = \min_{\theta \in \hat{\Theta}_k} v(\tilde{T}(\pi, d, \theta))$, $\tilde{M}_k = \max_{\hat{\pi} \in \Pi} v(\hat{\pi})$, and $\tilde{L}_k = \min_{\hat{\pi} \in \Pi} v(\hat{\pi})$. A similar approach can be used if $Hv(\pi)$ is less than $\tilde{H}v(\pi)$. Note that $M_k - L_k$ can be made arbitrarily small by dividing the signal space into very small regions, and in this case, the value function obtained will be very close to the exact solution.

III. Methods for Solving POMDP with Continuous Signal Space

In the previous section, we discussed how to solve POMDP problems with continuous signal spaces by using step functions to approximate the signal processes. However, in order to obtain a value function which is close to the optimal value function, we might need to construct a step function with a large number of steps; that is, we might need to solve a finite signal POMDP problem with a large number of signals. To solve a problem with a large number of signals is not easy. In this section, we will introduce a method which can solve this problem without using step functions to approximate its signal processes if certain assumptions hold.

As discussed in the first section, even if v is a piecewise linear function, Hv need not be piecewise linear. Moreover, integral signs have replaced summation signs in the

formulas (5–4) and (5–5). Therefore, unlike POMDP with a finite number of signals, the construction of linear supports for any given state π and action d is not trivial or a by–product of the procedure for finding the value of $Hv(\pi)$. The algorithms discussed in Chapters 3 and 4 are based on the linear supports for the given states. As a result, they cannot be directly applied to problems with continuous signal space. However, if certain assumptions which will be discussed later are satisfied, we can find linear supports for given states and actions. Then methods similar to those discussed in the previous two chapters can be applied to problems with a continuous signal space.

We now focus on how to calculate the value and the support of $Hv(\pi)$ for arbitrary $\pi \in \Pi$ if v is a piecewise linear function (i.e., the evaluation of (5–5)). Let A be the set of all linear supports of v. If $\tilde{\alpha}$ is a support in A and d is an action in the action space, define $\Theta_{\pi,d,\tilde{\alpha}}$ as

$$\Theta_{\pi,d,\tilde{\alpha}} = \{\theta \in \Theta : \ \pi \cdot P^d \cdot \tilde{Q}^d_\theta \tilde{\alpha} \geq \pi \cdot P^d \cdot \tilde{Q}^d_\theta \alpha \qquad \forall \alpha \in A\}. \tag{5-6}$$

Apply (5–6), then (5–5) can be rewritten as

$$\begin{aligned} Hv(\pi) &= \max_{d \in D}\{\sum_{i=1}^{N} \pi_i \cdot r^d(i) + \beta \cdot \int_{\theta \in \Theta} \pi \cdot P^d \cdot \tilde{Q}^d_\theta \cdot \alpha^{g(\pi,d,\theta)} d\theta\} \\ &= \max_{d \in D}\{\sum_{i=1}^{N} \pi_i \cdot r^d(i) + \beta \cdot \sum_{\tilde{\alpha} \in A} \int_{\theta \in \Theta_{\pi,d,\tilde{\alpha}}} \pi \cdot P^d \cdot \tilde{Q}^d_\theta \cdot \tilde{\alpha} d\theta\} \\ &= \max_{d \in D}\{\sum_{i=1}^{N} \pi_i \cdot r^d(i) + \beta \cdot \sum_{\tilde{\alpha} \in A} \pi \cdot P^d \cdot \int_{\theta \in \Theta_{\pi,d,\tilde{\alpha}}} \tilde{Q}^d_\theta \cdot \tilde{\alpha} d\theta\}. \end{aligned} \tag{5-7}$$

In order to simplify the calculation of $\int_{\theta \in \Theta_{\pi,d,\tilde{\alpha}}} \pi \cdot P^d \cdot \tilde{Q}^d_\theta \cdot \tilde{\alpha} d\theta$, let us assume that for any given $\pi \in \Pi$, $d \in D$, and $\tilde{\alpha} \in A$, there are only a finite number of connected sets in $\Theta_{\pi,d,\tilde{\alpha}}$. Moreover, we also assume that the boundary of $\Theta_{\pi,d,\tilde{\alpha}}$ has measure zero.

Since $\tilde{Q}_\theta^d$ is a diagonal matrix with $f_i^d(\theta)$ as its i–th diagonal element and $\tilde{\alpha}$ is an N–dimensional vector with $\tilde{\alpha}_i$ as the i–th element, we define

$$\zeta_{\pi,\tilde{\alpha}}^d = \begin{pmatrix} \int_{\theta\in\Theta_{\pi,d,\tilde{\alpha}}} f_1^d(\theta)\cdot\tilde{\alpha}_1 d\theta \\ \int_{\theta\in\Theta_{\pi,d,\tilde{\alpha}}} f_2^d(\theta)\cdot\tilde{\alpha}_2 d\theta \\ \vdots \\ \int_{\theta\in\Theta_{\pi,d,\tilde{\alpha}}} f_N^d(\theta)\cdot\tilde{\alpha}_N d\theta \end{pmatrix}. \qquad (5-8)$$

Therefore, (5–7) can be rewritten as

$$\begin{aligned} Hv(\pi) &= \max_{d\in D}\{\sum_{i=1}^N \pi_i\cdot r^d(i) + \beta\cdot\sum_{\Theta_{\pi,d,\tilde{\alpha}}} \pi\cdot P^d\cdot\int_{\theta\in\Theta_{\pi,d,\tilde{\alpha}}} \tilde{Q}_\theta^d\cdot\tilde{\alpha}d\theta\} \\ &= \max_{d\in D}\{\sum_{i=1}^N \pi_i\cdot r^d(i) + \beta\cdot\sum_{\Theta_{\pi,d,\tilde{\alpha}}} \pi\cdot P^d\cdot\zeta_{\pi,\tilde{\alpha}}^d\} \\ &= \max_{d\in D}\{\pi\cdot[r^d + \beta\cdot\sum_{\Theta_{\pi,d,\tilde{\alpha}}} P^d\cdot\zeta_{\pi,\tilde{\alpha}}^d]\}. \qquad (5-9) \end{aligned}$$

We remark that $r^d+\beta\cdot\sum_{\Theta_{\pi,d,\tilde{\alpha}}} P^d\cdot\zeta_{\pi,\tilde{\alpha}}^d$ is a N–dimensional column vector and a linear support of Hv at π.

Although the assumptions about $\Theta_{\pi,d,\tilde{\alpha}}$ are strong, these assumptions are true for many commonly used distributions. More importantly, π, P^d, and $\tilde{\alpha}$ are known before $\Theta_{\pi,d,\tilde{\alpha}}$ is computed. Therefore, in many situations, it is possible to verify whether the given signal processes will have a finite number of connected sets in $\Theta_{\pi,d,\tilde{\alpha}}$ before the actual calculation is started. For example, if the signal processes are exponential distributions and there are only two system states, we can show that the assumptions hold. Now let us use the exponential distribution as a example.

Example

Let us illustrate the method for computing linear supports for POMDP with the

following example:

$$P^1 = \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \qquad r^1 = \begin{bmatrix} -4 \\ 4 \end{bmatrix}$$
$$P^2 = \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 0 \\ 3 \end{bmatrix},$$

The signal processes have the following density functions: for $\theta > 0$,

$$f_1^1(\theta) = e^{-\theta},$$

$$f_2^1(\theta) = 10 \cdot e^{-10 \cdot \theta},$$

$$f_1^2(\theta) = 3 \cdot e^{-3 \cdot \theta},$$

$$f_2^2(\theta) = 2 \cdot e^{-2 \cdot \theta},$$

and $f_1^1(\theta) = f_2^1(\theta) = f_1^2(\theta) = f_2^2(\theta) = 0$ for $\theta \leq 0$. Let us also assume that β is 0.9, $\pi = [0,1]$, and $A = \{\alpha^1, \alpha^2\}$ where $\alpha^1 = [-4,4]^T$ and $\alpha^2 = [0,3]^T$.

We first compute $\Theta_{\pi,1,\alpha^1}$; i.e., we have to find $\theta \in \Theta$ such that

$$[0\ 1] \cdot \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} e^{-\theta} & 0 \\ 0 & 10 \cdot e^{-10\theta} \end{bmatrix} \cdot \begin{bmatrix} -4 \\ 4 \end{bmatrix} \geq [0\ 1] \cdot \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} e^{-\theta} & 0 \\ 0 & 10 \cdot e^{-10\theta} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$

After solving this inequality, we have $\Theta_{\pi,1,\alpha^1} = \{0 < \theta \leq 0.1\}$. Hence $\Theta_{\pi,1,\alpha^2}$ is $0.1 < \theta \leq \infty$.

Similarly, we can find $\Theta_{\pi,2,\alpha^1}$. However, $\Theta_{\pi,2,\alpha^1} = \{\theta < -1.77\}$ which is outside the domain of density functions, so $\Theta_{\pi,2,\alpha^1} = \emptyset$. and $\Theta_{\pi,2,\alpha^2} = \{\theta > 0\}$. Now apply $\Theta_{\pi,1,\alpha^1}$, $\Theta_{\pi,1,\alpha^2}$, $\Theta_{\pi,2,\alpha^1}$, and $\Theta_{\pi,2,\alpha^2}$ in (5–9), to obtain

$$Hv([0\ \ 1]) = \max\{$$
$$[0\ \ 1] \cdot \left\{ \begin{bmatrix} -4 \\ 4 \end{bmatrix} + 0.9 \cdot \begin{bmatrix} 0.8 & 0.2 \\ 0.5 & 0.5 \end{bmatrix} \cdot \left(\begin{bmatrix} \int_0^{0.10} -4 \cdot e^{-\theta}\, d\theta \\ \int_0^{0.10} 4 \cdot 10 \cdot e^{-10\theta}\, d\theta \end{bmatrix} \right.$$

$$+\begin{bmatrix} 0 \\ \int_{0.10}^{\infty} 3 \cdot 10 \cdot e^{-10\theta}\, d\theta \end{bmatrix}\Bigg)\Bigg\};$$

$$[0 \quad 1] \cdot \left\{ \begin{bmatrix} 0 \\ 3 \end{bmatrix} + 0.9 \cdot \begin{bmatrix} 0.5 & 0.5 \\ 0.4 & 0.6 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \int_{0}^{\infty} 3 \cdot 2 \cdot e^{-2\theta}\, d\theta \end{bmatrix} \right\}\}$$

$$= \max\{[0 \quad 1] \cdot \begin{bmatrix} -3.62 \\ 5.46 \end{bmatrix};\quad [0 \quad 1] \cdot \begin{bmatrix} 1.35 \\ 4.62 \end{bmatrix}\}$$

$$= 5.46$$

Therefore, Hv at $[0,1]$ is 5.46 and the linear support corresponding to this state is $[-3.62, 5.46]^T$. ■

Now let us look at the algorithms discussed in Chapter 3. Signals are intrinsic to the partition method and Sondik's one–pass algorithm. In Monahan's method, signals are only used to generate candidate linear supports. However, the signals do not formally appear in the relaxed region algorithm and linear support algorithm. These two algorithms only assume that, for any given state, it is possible to find a linear support corresponding to this state. The assumption of a finite number of signals guarantees an easy method to compute linear supports. If there is a method to generate the linear support for any given state, then the assumption of a finite number of signals is not required for the relaxed region algorithm or linear support algorithm.

Provided that a support of a value function can be constructed at any given state, either the relaxed region algorithm or the linear support algorithm can be used to solve a POMDP which has continuous signals. However, in practice, only the linear support algorithm can be used because, for POMDP with a continuous signal space, Hv need not be a piecewise linear function even v is. If the relaxed region algorithm is used, the termination criterion might never be satisfied. As was discussed in Chapter 3, the

linear support algorithm can be used to approximate Hv, and can be used to find an approximate solution $\tilde{H}v$ which is a piecewise linear function.

Similarly, the algorithms discussed in Chapter 4 only use signals in the computation of the linear supports. However, since it might not be possible to calculate the exact value of Hv for POMDP with continuous signals as discussed above, only those algorithms which permit approximations of Hv can be applied; that is, the methods presented in Sections III and VII in Chapter 4.

The assumptions made in this section are only to guarantee that a linear support can be found for any given state. If there is any other method which can produce the linear support for any given state in Π, then the linear support algorithm and the methods shown in Sections III and VII in Chapter 4 can still be applied. We can expect that many POMDP problems having a continuous signal space either satisfy the assumptions in this section or a method exists for computing the linear support for any given state and therefore can be solved using linear support methods. Problems for which linear supports cannot be calculated for all states in Π can still be solved by using step functions to approximate the signal processes as discussed in the previous section, although this method might not be efficient.

BIBLIOGRAPHY

Albright, S. 1979. Structural Results for Partially Observable Markov Decision Processes. *Operations Research* **27**, 1041–1053.

Åstrom, K. 1965. Optimal Control of Markov Processed with Incomplete State Information. *Journal of Mathematical Analysis and Applications* **10**, 174–205.

Åstrom, K. 1969. Optimal Control of Markov Processes with Incomplete State information, II. The Convexity of the Loss Function. *Journal of Mathematical Analysis and Applications* **26**, 403–406.

Bellman, R. 1958. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey.

Bertsekas, D. P. 1975. Convergence of Discretization Procedures in Dynamic Programming. *IEEE Transactions Automatic Control* **AC-21**, 415–419.

Bertsekas, D. P. 1976. *Dynamic Programming and Stochastic Control.* Academic Press, New York.

Brumelle, S., and K. Sawaki. 1978. Generalized Policy Improvement for Simple Dynamic Programs with an Application to Partially Observable Markov Decision Processes. Working Paper 546, Faculty of Commerce, University of British Columbia, Vancouver.

Buckman, A. G. and B. L. Miller. 1979. Optimal Investigation as a Regenerative Stopping Problem. Working Paper 289, Western Management Science Institute, University of California at Los Angles, California.

Denardo, E. V. 1967. Contraction Mapping in the Theory Underlying Dynamic Programming. *SIAM Reviews* **9**, 165–177.

Denardo, E. V. 1982. *Dynamic Programming Models and Applications.* Prentice-Hall,Inc. Englewood Cliffs, New Jersey.

Drake, A. 1962. *Observation of a Markov Process Through a Noisy Channel.* Unpublished Sc.D. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Eagle, J. N. 1984. The Optimal Search for a Moving Target When the Search Path Is Constrained. *Operations Research* **32**, 1107–1115.

Eckles, J. 1968. Optimum Maintenance with Incomplete Information. *Operations Re-*

search **16**, 1058–1067.

Èl'sgol'c, L. É. 1964 *Qualitative Methods in Mathematical Analysis.* American Mathematical Society. Providence, Rhode Island

Howard, R. A. 1960. *Dynamic Programming and Markov Processes.* The M.I.T. Press, Cambridge, Massachusetts.

Hsu, K. and S. Marcus. 1980. Decentralized Control of Finite State Markov Processes. *Proceedings 19th IEEE Conference on Decision and Control*, 143–148.

Hughes, J. S. 1977. Optimal Internal Audit Timing. *Accounting Review* **LII**, 56–58.

Hughes, J. S. 1977. A Note on Quality Control under Markovian Deterioration. *Operations Research* **28**, 421–423.

Kakalik, J.S. 1965. Optimum Policies for Partially Observable Markov System. Technical Report TR–18, Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Kaplan, R. 1969. Optimal Investigation Strategies with imperfect Information. *Journal of Accounting Research* **7**, 32–43.

Lane, D. E. 1986a. *Dynamics Models of Decision Making by Fishermen.* Unpublished Ph.D. dissertation, University of British Columbia, Vancouver, British Columbia, Canada.

Lane, D. E. 1986b. A Partially Observable Model of Decision Making by Fishermen. Working Paper 86–46. University of Ottawa. Ontario.

Lovejoy, W. S. 1983. *Policy Bounds for Markov Decision Processes with Applications to Fisheries Management.* Unpublished Ph.D. dissertation, University of Delaware.

Lovejoy, W. S. 1987. Some Monotonicity Results of Partially Observed Markov Decision Processes. *Operations Research* **35**, 736–743.

Mattheiss, T. H. 1973. An Algorithm for Determining Irrelevant Constraints and All Vertices in Systems of Linear Inequalities. *Operations Research* **21**, 247–260.

Mattheiss, T. H., and D. S. Rubin. 1980. A Survey and Comparison of Methods for Finding All Vertices of Convex Polyhedral Sets. *Mathematics of Operations Research* **5**, 167–185.

MacQueen, J. 1967. A Test for Suboptimal Actions in Markovian Decision Problems.

Operations Research **15**, 559–561.

Monahan, G. E. 1982. A Survey of Partially Observable Markov Decision Processes: Theory, Models, and Algorithms. *Management Science* **28**, 1–16.

Nir, Abraham. 1986. *Optimal Control of a Partially Observable Markov Chain.* Unpublished Ph.D. dissertation, Northwestern University, Evanston, Illinois.

van Nunen, J.A.E.E. 1976. *Contracting Markov Decision Processes* Amsterdam, Mathematisch Centrum (Mathematical Centre Tracts, no. 71).

Ohnishi, M., Kawai, H., and H. Mine. 1986. An Optimal Inspection and Replacement Policy under Incomplete State Information. *European Journal of Operational Research.* **27**, 117–128.

Ohnishi, M., Mine, H., and H. Kawai. 1986. An Optimal Inspection and Replacement Policy under Incomplete State Information: Average Cost Criterion. in *Stochastic Models in Reliability Theory*, 187-197. Springer-Verlag, Berlin.

Ortega, J. M., and W. C. Rheinboldt. 1970. *Iterative Solution of Nonlinear Equations in Several Variables* Academic Press. New York.

Papadimitriou C.H. and J. N. Tsitsiklis. 1987. The Complexity of Markov Decision Processes. *Mathematics of Operations Research* **12**, 441–450

Platzman, L. 1980. Optimal Infinite-Horizon Undiscounted Control of finite Probabilistic Systems. *SIAM Journal of Control and Optimization* **18**, 362–380

Platzman, L. 1981. A feasible Computational Approach to Infinite-Horizon Partially-Observed Markov Decision Problems. mimeograph, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia.

Puterman, M. L., and M. C. Shin. 1978. Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science* **24**, 1127–1137.

Puterman, M. L., and M. C. Shin. 1982. Action Elimination Procedures for Modified Policy Iteration Algorithms. *Operations Research* **30**, 301–318.

Rhenius, D. 1974. Incomplete Information in Markovian Decision Models. *Annals of Statistics* **2**, 1327–1334.

Ross, S. 1971. Quality Control Under Markovian Deterioration. *Management Science* **17** 587–596.

Satia, J. and R. Lave. 1973. Markovian Decision Processes with Probabilistic Observation of States. *Management Science* **20** 1–13.

Sawaki, K. 1980. Piecewise Linear Markov Decision Processes with an Application into Partially Observable Models. in *Recent Developments in Markov Decision Processes* (R. Hartley *et al.*, Eds.) Academic Press, New York.

Sawaki, K. 1983. Transformation of Partially Observable Markov Decision Processes into Piecewise Linear Ones. *Journal of Mathematical Analysis and Applications* **91**, 112–118.

Sawaragi, Y. and T. Yoshikawa. 1970. Discrete-Time Markovian Decision Processes with Imcomplete State Observation. *Annals of Mathematical Statistics* **41**, 78–86.

Smallwood, R. D., and E. J. Sondik. 1973. The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Operations Research* **21**, 1071–1088.

Sondik, E. J. 1971. *The Optimal Control of Partially Observable Markov Processes.* Unpublished Ph.D. dissertation, Stanford University, Stanford, California.

Sondik, E. J. 1978. The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Cost. *Operations Research* **26**, 282–304.

Striebel, C. 1965. Sufficient Statistics in the Control of Stochastic Systems. *Journal of Mathematical Analysis and Applications* **12**, 576–592.

Wang, R. 1976. Computing Optimal Quality Control Policies–Two Actions. *Journal of Applied Probability* **13**, 826–832.

Wang, R. 1977. Optimal Replacement Policy Under Unobservable States. *Journal of Applied Probability* **14**, 340–348.

White, C. 1976. Optimal Diagnostic Questionnaires Which Allow Less Than Truthful Responses. *Information and Control* **32**, 61–74.

White, C. 1977. A Markov Quality Control Process Subject to Partial Observation. *Management science* **23**, 843–852.

White, C. 1978. Optimal Inspection and Repair of a Production Process Subject to Deterioration. *Journal of Operational Research Society* **29**, 235–243.

White, C. 1979a. Bounds on Optimal Cost for a Replacement Problem with Partial Observation. *Naval Research Logistics Quarterly* **26**, 415-422.

White, C. 1979b. Optimal Control-Limit Strategies for a Partially Observed Replacement Problem. *International Journal of Systems Science* **10**, 321–331.

White, C. 1980. Monotone Control Laws for Noisy, Countable-State Markov Chains. *European Journal of Operational Research* **5**, 124–132.

White, C. C., and D. P. Harrington. 1980. Application of Jensen's Inequality to Adaptive Suboptimal Design. *Journal of Optimization Theory and Application* **32**, 89–99

White, C. C., and W. T. Scherer. 1986. Accelerated Successive Approximation Algorithms for Partially Observed Markov Decision Processes. Working Paper, University of Virginia, Charlottesville, Virginia.

Whitt, W. 1978. Approximations of Dynamic Programs, I. *Mathematics of Operations Research* **3**, 231–243.

Whittle, P. 1982. *Optimization Over Time.* Volume II. John Wiley & Sons Ltd. Chichester.

APPENDIX 1
RANDOM GENERATED TESTING DATA FOR FINITE HORIZON ALGORITHMS

DATA SET D3.1

Number of States = 3

Number of Actions = 3

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} 0.573 & 0.346 & 0.081 \\ 0.416 & 0.441 & 0.143 \\ 0.103 & 0.390 & 0.507 \end{bmatrix} \quad Q^1 = \begin{bmatrix} 0.646 & 0.199 & 0.155 \\ 0.106 & 0.643 & 0.251 \\ 0.120 & 0.220 & 0.660 \end{bmatrix} \quad r^1 = \begin{bmatrix} 2.500 \\ 7.400 \\ 1.300 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.357 & 0.345 & 0.298 \\ 0.370 & 0.460 & 0.170 \\ 0.169 & 0.192 & 0.639 \end{bmatrix} \quad Q^2 = \begin{bmatrix} 0.704 & 0.116 & 0.179 \\ 0.0 & 0.840 & 0.160 \\ 0.188 & 0.135 & 0.677 \end{bmatrix} \quad r^2 = \begin{bmatrix} 5.800 \\ 3.600 \\ 6.600 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.305 & 0.239 & 0.456 \\ 0.388 & 0.356 & 0.256 \\ 0.139 & 0.271 & 0.590 \end{bmatrix} \quad Q^3 = \begin{bmatrix} 0.702 & 0.211 & 0.087 \\ 0.304 & 0.581 & 0.115 \\ 0.145 & 0.238 & 0.617 \end{bmatrix} \quad r^3 = \begin{bmatrix} 7.200 \\ 7.000 \\ 4.900 \end{bmatrix}$$

DATA SET D3.2

Number of States = 3

Number of Actions = 3

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} 0.256 & 0.693 & 0.051 \\ 0.002 & 0.273 & 0.725 \\ 0.578 & 0.063 & 0.359 \end{bmatrix} \qquad Q^1 = \begin{bmatrix} 0.712 & 0.066 & 0.222 \\ 0.089 & 0.790 & 0.121 \\ 0.014 & 0.126 & 0.860 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 7.100 \\ 9.300 \\ 5.300 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.350 & 0.239 & 0.411 \\ 0.244 & 0.237 & 0.519 \\ 0.227 & 0.323 & 0.450 \end{bmatrix} \qquad Q^2 = \begin{bmatrix} 0.763 & 0.041 & 0.196 \\ 0.081 & 0.566 & 0.353 \\ 0.092 & 0.031 & 0.877 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 8.300 \\ 8.600 \\ 7.800 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.482 & 0.045 & 0.473 \\ 0.108 & 0.485 & 0.407 \\ 0.445 & 0.158 & 0.397 \end{bmatrix} \qquad Q^3 = \begin{bmatrix} 0.504 & 0.061 & 0.435 \\ 0.080 & 0.705 & 0.215 \\ 0.194 & 0.015 & 0.791 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 0.900 \\ 3.900 \\ 8.800 \end{bmatrix}$$

DATA SET D3.3

Number of States = 3

Number of Actions = 3

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} 0.349 & 0.180 & 0.471 \\ 0.860 & 0.093 & 0.047 \\ 0.070 & 0.472 & 0.458 \end{bmatrix} \quad Q^1 = \begin{bmatrix} 0.677 & 0.193 & 0.130 \\ 0.166 & 0.572 & 0.262 \\ 0.143 & 0.247 & 0.611 \end{bmatrix} \quad r^1 = \begin{bmatrix} 9.900 \\ 0.200 \\ 2.400 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.295 & 0.144 & 0.561 \\ 0.272 & 0.104 & 0.624 \\ 0.443 & 0.314 & 0.243 \end{bmatrix} \quad Q^2 = \begin{bmatrix} 0.660 & 0.165 & 0.175 \\ 0.130 & 0.792 & 0.078 \\ 0.205 & 0.083 & 0.712 \end{bmatrix} \quad r^2 = \begin{bmatrix} 5.900 \\ 7.400 \\ 5.500 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.284 & 0.672 & 0.044 \\ 0.635 & 0.300 & 0.065 \\ 0.042 & 0.545 & 0.413 \end{bmatrix} \quad Q^3 = \begin{bmatrix} 0.745 & 0.143 & 0.112 \\ 0.133 & 0.712 & 0.155 \\ 0.115 & 0.260 & 0.625 \end{bmatrix} \quad r^3 = \begin{bmatrix} 3.700 \\ 4.900 \\ 9.800 \end{bmatrix}$$

DATA SET D3.4

Number of States = 3

Number of Actions = 3

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} 0.336 & 0.436 & 0.228 \\ 0.304 & 0.272 & 0.424 \\ 0.167 & 0.578 & 0.255 \end{bmatrix} \quad Q^1 = \begin{bmatrix} 0.574 & 0.227 & 0.199 \\ 0.135 & 0.745 & 0.120 \\ 0.062 & 0.294 & 0.644 \end{bmatrix} \quad r^1 = \begin{bmatrix} 3.700 \\ 3.000 \\ 6.200 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.339 & 0.525 & 0.136 \\ 0.435 & 0.525 & 0.040 \\ 0.374 & 0.243 & 0.383 \end{bmatrix} \quad Q^2 = \begin{bmatrix} 0.640 & 0.180 & 0.180 \\ 0.252 & 0.584 & 0.164 \\ 0.236 & 0.024 & 0.740 \end{bmatrix} \quad r^2 = \begin{bmatrix} 5.700 \\ 0.100 \\ 1.700 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.143 & 0.173 & 0.684 \\ 0.020 & 0.826 & 0.154 \\ 0.601 & 0.079 & 0.320 \end{bmatrix} \quad Q^3 = \begin{bmatrix} 0.916 & 0.054 & 0.030 \\ 0.101 & 0.843 & 0.056 \\ 0.095 & 0.215 & 0.690 \end{bmatrix} \quad r^3 = \begin{bmatrix} 7.400 \\ 0.400 \\ 7.000 \end{bmatrix}$$

DATA SET D3.5

Number of States = 3

Number of Actions = 3

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} 0.445 & 0.222 & 0.333 \\ 0.500 & 0.173 & 0.327 \\ 0.204 & 0.553 & 0.243 \end{bmatrix} \qquad Q^1 = \begin{bmatrix} 0.686 & 0.182 & 0.132 \\ 0.138 & 0.786 & 0.076 \\ 0.279 & 0.083 & 0.638 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 5.200 \\ 4.600 \\ 4.100 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.234 & 0.064 & 0.702 \\ 0.549 & 0.218 & 0.233 \\ 0.061 & 0.466 & 0.473 \end{bmatrix} \qquad Q^2 = \begin{bmatrix} 0.698 & 0.131 & 0.171 \\ 0.283 & 0.624 & 0.093 \\ 0.005 & 0.202 & 0.793 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 0.800 \\ 6.800 \\ 6.900 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.535 & 0.313 & 0.152 \\ 0.114 & 0.870 & 0.016 \\ 0.325 & 0.360 & 0.315 \end{bmatrix} \qquad Q^3 = \begin{bmatrix} 0.567 & 0.234 & 0.199 \\ 0.243 & 0.641 & 0.116 \\ 0.186 & 0.044 & 0.770 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 9.000 \\ 9.300 \\ 0.800 \end{bmatrix}$$

DATA SET D4.1

Number of States = 4

Number of Actions = 4

Number of Signals = 4

Action 1:

$$P^1 = \begin{bmatrix} 0.355 & 0.321 & 0.100 & 0.224 \\ 0.433 & 0.181 & 0.342 & 0.044 \\ 0.254 & 0.155 & 0.065 & 0.526 \\ 0.066 & 0.420 & 0.172 & 0.342 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} 0.519 & 0.192 & 0.154 & 0.135 \\ 0.161 & 0.551 & 0.093 & 0.195 \\ 0.112 & 0.158 & 0.662 & 0.068 \\ 0.157 & 0.126 & 0.055 & 0.662 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 0.2 \\ 8.2 \\ 6.8 \\ 8.1 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.094 & 0.311 & 0.262 & 0.333 \\ 0.241 & 0.173 & 0.233 & 0.353 \\ 0.351 & 0.259 & 0.273 & 0.117 \\ 0.064 & 0.174 & 0.454 & 0.308 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} 0.571 & 0.013 & 0.162 & 0.254 \\ 0.158 & 0.547 & 0.105 & 0.190 \\ 0.089 & 0.180 & 0.629 & 0.102 \\ 0.109 & 0.198 & 0.149 & 0.544 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 0.8 \\ 1.0 \\ 5.4 \\ 3.9 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.455 & 0.182 & 0.135 & 0.228 \\ 0.370 & 0.280 & 0.005 & 0.345 \\ 0.220 & 0.276 & 0.270 & 0.234 \\ 0.403 & 0.499 & 0.071 & 0.027 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} 0.650 & 0.121 & 0.126 & 0.103 \\ 0.175 & 0.538 & 0.134 & 0.153 \\ 0.064 & 0.086 & 0.676 & 0.174 \\ 0.204 & 0.232 & 0.010 & 0.554 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 2.2 \\ 6.7 \\ 2.6 \\ 1.7 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} 0.255 & 0.255 & 0.230 & 0.260 \\ 0.294 & 0.108 & 0.417 & 0.181 \\ 0.025 & 0.143 & 0.352 & 0.480 \\ 0.267 & 0.298 & 0.194 & 0.241 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} 0.672 & 0.103 & 0.134 & 0.091 \\ 0.004 & 0.694 & 0.109 & 0.193 \\ 0.159 & 0.166 & 0.621 & 0.054 \\ 0.070 & 0.172 & 0.123 & 0.635 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 7.9 \\ 1.5 \\ 7.4 \\ 1.7 \end{bmatrix}$$

DATA SET D4.2

Number of States = 4

Number of Actions = 4

Number of Signals = 4

Action 1:

$$P^1 = \begin{bmatrix} 0.252 & 0.210 & 0.240 & 0.298 \\ 0.291 & 0.123 & 0.149 & 0.437 \\ 0.171 & 0.075 & 0.475 & 0.279 \\ 0.294 & 0.010 & 0.396 & 0.300 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} 0.700 & 0.088 & 0.131 & 0.081 \\ 0.075 & 0.699 & 0.149 & 0.077 \\ 0.222 & 0.016 & 0.710 & 0.052 \\ 0.088 & 0.252 & 0.149 & 0.511 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 1.1 \\ 9.2 \\ 4.2 \\ 9.9 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.375 & 0.192 & 0.150 & 0.283 \\ 0.462 & 0.121 & 0.077 & 0.340 \\ 0.120 & 0.640 & 0.092 & 0.148 \\ 0.222 & 0.283 & 0.129 & 0.366 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} 0.644 & 0.156 & 0.163 & 0.037 \\ 0.040 & 0.750 & 0.088 & 0.122 \\ 0.050 & 0.204 & 0.655 & 0.091 \\ 0.020 & 0.038 & 0.154 & 0.788 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 0.8 \\ 8.1 \\ 9.5 \\ 7.1 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.255 & 0.140 & 0.309 & 0.296 \\ 0.180 & 0.395 & 0.032 & 0.393 \\ 0.286 & 0.229 & 0.273 & 0.212 \\ 0.157 & 0.012 & 0.517 & 0.314 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} 0.713 & 0.002 & 0.139 & 0.146 \\ 0.116 & 0.667 & 0.020 & 0.197 \\ 0.076 & 0.159 & 0.678 & 0.087 \\ 0.176 & 0.083 & 0.096 & 0.645 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 5.5 \\ 7.3 \\ 9.3 \\ 1.3 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} 0.316 & 0.188 & 0.378 & 0.118 \\ 0.333 & 0.320 & 0.228 & 0.119 \\ 0.280 & 0.346 & 0.069 & 0.305 \\ 0.149 & 0.185 & 0.392 & 0.274 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} 0.728 & 0.080 & 0.072 & 0.120 \\ 0.102 & 0.612 & 0.208 & 0.078 \\ 0.198 & 0.123 & 0.540 & 0.139 \\ 0.037 & 0.170 & 0.260 & 0.533 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 9.7 \\ 8.0 \\ 8.4 \\ 4.0 \end{bmatrix}$$

DATA SET D4.3

Number of States = 4

Number of Actions = 4

Number of Signals = 4

Action 1:

$$P^1 = \begin{bmatrix} 0.016 & 0.086 & 0.634 & 0.264 \\ 0.184 & 0.237 & 0.280 & 0.299 \\ 0.126 & 0.396 & 0.298 & 0.180 \\ 0.344 & 0.342 & 0.110 & 0.204 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} 0.677 & 0.239 & 0.066 & 0.018 \\ 0.030 & 0.634 & 0.185 & 0.151 \\ 0.172 & 0.002 & 0.671 & 0.155 \\ 0.152 & 0.079 & 0.173 & 0.596 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 5.7 \\ 8.8 \\ 8.3 \\ 3.7 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.092 & 0.313 & 0.151 & 0.444 \\ 0.133 & 0.161 & 0.457 & 0.249 \\ 0.162 & 0.355 & 0.165 & 0.318 \\ 0.342 & 0.178 & 0.304 & 0.176 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} 0.559 & 0.138 & 0.108 & 0.195 \\ 0.071 & 0.638 & 0.136 & 0.155 \\ 0.008 & 0.138 & 0.687 & 0.167 \\ 0.210 & 0.081 & 0.062 & 0.647 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 4.3 \\ 3.2 \\ 0.7 \\ 5.3 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.244 & 0.335 & 0.255 & 0.166 \\ 0.319 & 0.137 & 0.204 & 0.340 \\ 0.216 & 0.278 & 0.140 & 0.366 \\ 0.131 & 0.490 & 0.340 & 0.039 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} 0.503 & 0.085 & 0.177 & 0.235 \\ 0.123 & 0.685 & 0.136 & 0.056 \\ 0.243 & 0.042 & 0.626 & 0.089 \\ 0.135 & 0.129 & 0.116 & 0.620 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 5.8 \\ 8.5 \\ 7.4 \\ 8.3 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} 0.174 & 0.230 & 0.372 & 0.224 \\ 0.297 & 0.283 & 0.164 & 0.256 \\ 0.140 & 0.260 & 0.316 & 0.284 \\ 0.222 & 0.341 & 0.055 & 0.382 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} 0.637 & 0.052 & 0.227 & 0.084 \\ 0.121 & 0.675 & 0.059 & 0.145 \\ 0.185 & 0.032 & 0.675 & 0.108 \\ 0.137 & 0.142 & 0.120 & 0.601 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 7.0 \\ 0.8 \\ 1.0 \\ 5.5 \end{bmatrix}$$

DATA SET D4.4

Number of States = 4

Number of Actions = 4

Number of Signals = 4

Action 1:

$$P^1 = \begin{bmatrix} 0.093 & 0.337 & 0.226 & 0.344 \\ 0.269 & 0.306 & 0.178 & 0.247 \\ 0.255 & 0.352 & 0.264 & 0.129 \\ 0.096 & 0.171 & 0.354 & 0.379 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} 0.634 & 0.154 & 0.124 & 0.088 \\ 0.156 & 0.612 & 0.115 & 0.117 \\ 0.073 & 0.167 & 0.670 & 0.090 \\ 0.124 & 0.106 & 0.156 & 0.614 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 4.2 \\ 4.5 \\ 3.8 \\ 2.7 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.280 & 0.162 & 0.029 & 0.529 \\ 0.205 & 0.251 & 0.322 & 0.222 \\ 0.284 & 0.295 & 0.153 & 0.268 \\ 0.418 & 0.379 & 0.181 & 0.022 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} 0.675 & 0.058 & 0.153 & 0.114 \\ 0.302 & 0.549 & 0.112 & 0.037 \\ 0.202 & 0.157 & 0.567 & 0.074 \\ 0.231 & 0.065 & 0.110 & 0.594 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 7.1 \\ 8.9 \\ 5.4 \\ 6.9 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.408 & 0.0 & 0.355 & 0.237 \\ 0.307 & 0.245 & 0.213 & 0.235 \\ 0.072 & 0.346 & 0.419 & 0.163 \\ 0.244 & 0.303 & 0.208 & 0.245 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} 0.563 & 0.119 & 0.236 & 0.082 \\ 0.167 & 0.584 & 0.211 & 0.038 \\ 0.155 & 0.112 & 0.601 & 0.132 \\ 0.091 & 0.146 & 0.091 & 0.672 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 6.2 \\ 2.1 \\ 4.6 \\ 8.9 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} 0.485 & 0.374 & 0.086 & 0.055 \\ 0.280 & 0.030 & 0.649 & 0.041 \\ 0.047 & 0.147 & 0.313 & 0.493 \\ 0.234 & 0.420 & 0.046 & 0.300 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} 0.531 & 0.320 & 0.002 & 0.147 \\ 0.097 & 0.603 & 0.253 & 0.047 \\ 0.089 & 0.136 & 0.590 & 0.185 \\ 0.080 & 0.190 & 0.166 & 0.564 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 6.5 \\ 4.8 \\ 8.4 \\ 8.4 \end{bmatrix}$$

DATA SET D4.5

Number of States = 4

Number of Actions = 4

Number of Signals = 4

Action 1:

$$P^1 = \begin{bmatrix} 0.191 & 0.159 & 0.325 & 0.325 \\ 0.230 & 0.210 & 0.317 & 0.243 \\ 0.105 & 0.370 & 0.101 & 0.424 \\ 0.502 & 0.231 & 0.088 & 0.179 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} 0.536 & 0.140 & 0.173 & 0.151 \\ 0.193 & 0.541 & 0.155 & 0.111 \\ 0.148 & 0.140 & 0.608 & 0.104 \\ 0.146 & 0.023 & 0.147 & 0.684 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 9.4 \\ 2.9 \\ 7.1 \\ 7.2 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.161 & 0.148 & 0.318 & 0.373 \\ 0.280 & 0.334 & 0.336 & 0.050 \\ 0.024 & 0.422 & 0.422 & 0.132 \\ 0.069 & 0.151 & 0.303 & 0.477 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} 0.681 & 0.011 & 0.134 & 0.174 \\ 0.180 & 0.680 & 0.090 & 0.050 \\ 0.017 & 0.262 & 0.504 & 0.217 \\ 0.200 & 0.030 & 0.190 & 0.580 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 0.1 \\ 5.1 \\ 0.8 \\ 8.5 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.330 & 0.200 & 0.174 & 0.296 \\ 0.423 & 0.001 & 0.168 & 0.408 \\ 0.412 & 0.113 & 0.019 & 0.456 \\ 0.243 & 0.034 & 0.309 & 0.414 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} 0.510 & 0.169 & 0.255 & 0.066 \\ 0.122 & 0.624 & 0.125 & 0.129 \\ 0.207 & 0.046 & 0.732 & 0.015 \\ 0.034 & 0.176 & 0.236 & 0.554 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 6.1 \\ 7.1 \\ 5.1 \\ 8.9 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} 0.194 & 0.305 & 0.491 & 0.010 \\ 0.208 & 0.212 & 0.029 & 0.551 \\ 0.193 & 0.415 & 0.181 & 0.211 \\ 0.436 & 0.492 & 0.023 & 0.049 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} 0.576 & 0.287 & 0.041 & 0.096 \\ 0.120 & 0.590 & 0.143 & 0.147 \\ 0.070 & 0.180 & 0.510 & 0.240 \\ 0.040 & 0.254 & 0.200 & 0.506 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 0.3 \\ 7.2 \\ 10.0 \\ 9.6 \end{bmatrix}$$

DATA SET D5.1

Number of States = 5

Number of Actions = 3

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} 0.064 & 0.343 & 0.303 & 0.106 & 0.184 \\ 0.340 & 0.087 & 0.214 & 0.069 & 0.290 \\ 0.189 & 0.200 & 0.402 & 0.097 & 0.112 \\ 0.322 & 0.003 & 0.053 & 0.322 & 0.300 \\ 0.137 & 0.292 & 0.108 & 0.100 & 0.363 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} 0.092 & 0.403 & 0.505 \\ 0.310 & 0.099 & 0.591 \\ 0.431 & 0.289 & 0.280 \\ 0.287 & 0.304 & 0.409 \\ 0.382 & 0.213 & 0.405 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 10.0 \\ 6.0 \\ 4.9 \\ 3.2 \\ 7.4 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} 0.425 & 0.117 & 0.261 & 0.077 & 0.120 \\ 0.108 & 0.298 & 0.092 & 0.320 & 0.182 \\ 0.470 & 0.033 & 0.155 & 0.194 & 0.148 \\ 0.173 & 0.184 & 0.284 & 0.215 & 0.144 \\ 0.154 & 0.136 & 0.236 & 0.128 & 0.346 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} 0.081 & 0.495 & 0.424 \\ 0.736 & 0.024 & 0.240 \\ 0.521 & 0.467 & 0.012 \\ 0.360 & 0.199 & 0.441 \\ 0.040 & 0.447 & 0.513 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 4.5 \\ 0.2 \\ 2.5 \\ 6.5 \\ 5.5 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} 0.005 & 0.226 & 0.292 & 0.264 & 0.213 \\ 0.239 & 0.126 & 0.266 & 0.220 & 0.149 \\ 0.228 & 0.016 & 0.259 & 0.040 & 0.457 \\ 0.092 & 0.019 & 0.518 & 0.034 & 0.337 \\ 0.324 & 0.272 & 0.024 & 0.047 & 0.333 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} 0.226 & 0.403 & 0.371 \\ 0.416 & 0.144 & 0.440 \\ 0.462 & 0.319 & 0.219 \\ 0.260 & 0.103 & 0.637 \\ 0.508 & 0.078 & 0.414 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 2.3 \\ 7.9 \\ 6.2 \\ 6.5 \\ 3.3 \end{bmatrix}$$

APPENDIX 2
RANDOM GENERATED TESTING DATA
FOR INFINITE HORIZON ALGORITHMS

DATA SET 1

Number of States = 3

Number of Actions = 3

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} .483 & .268 & .249 \\ .000 & .000 & 1.000 \\ .000 & .698 & .302 \end{bmatrix} \quad Q^1 = \begin{bmatrix} .557 & .220 & .223 \\ .031 & .665 & .304 \\ .223 & .111 & .667 \end{bmatrix} \quad r^1 = \begin{bmatrix} 5.10 \\ 5.20 \\ 9.50 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} .665 & .335 & .000 \\ .407 & .223 & .369 \\ .695 & .000 & .305 \end{bmatrix} \quad Q^2 = \begin{bmatrix} .624 & .337 & .039 \\ .215 & .706 & .080 \\ .149 & .088 & .762 \end{bmatrix} \quad r^2 = \begin{bmatrix} 5.80 \\ 4.60 \\ 1.40 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} .363 & .361 & .275 \\ .430 & .000 & .570 \\ 1.000 & .000 & .000 \end{bmatrix} \quad Q^3 = \begin{bmatrix} .643 & .171 & .186 \\ .121 & .727 & .152 \\ .256 & .102 & .643 \end{bmatrix} \quad r^3 = \begin{bmatrix} 7.90 \\ 6.80 \\ 7.30 \end{bmatrix}$$

DATA SET 2

Number of States = 3

Number of Actions = 6

Number of Signals = 3

Action 1:

$$P^1 = \begin{bmatrix} .000 & .388 & .612 \\ .580 & .420 & .000 \\ .379 & .180 & .441 \end{bmatrix} \quad Q^1 = \begin{bmatrix} .213 & .451 & .336 \\ .383 & .617 & .000 \\ .359 & .641 & .000 \end{bmatrix} \quad r^1 = \begin{bmatrix} 1.90 \\ 6.20 \\ 9.60 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} .000 & 1.000 & .000 \\ .479 & .000 & .521 \\ .529 & .000 & .471 \end{bmatrix} \quad Q^2 = \begin{bmatrix} 1.000 & .000 & .000 \\ .691 & .000 & .309 \\ .000 & .527 & .473 \end{bmatrix} \quad r^2 = \begin{bmatrix} 2.70 \\ 8.70 \\ 5.60 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} .324 & .291 & .385 \\ .506 & .000 & .494 \\ .000 & 1.000 & .000 \end{bmatrix} \quad Q^3 = \begin{bmatrix} .000 & .643 & .357 \\ .321 & .000 & .679 \\ .486 & .000 & .514 \end{bmatrix} \quad r^3 = \begin{bmatrix} 1.80 \\ 6.20 \\ 8.10 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} .716 & .002 & .282 \\ .171 & .612 & .216 \\ .124 & .214 & .661 \end{bmatrix} \quad Q^4 = \begin{bmatrix} .844 & .043 & .112 \\ .339 & .534 & .127 \\ .273 & .026 & .701 \end{bmatrix} \quad r^4 = \begin{bmatrix} 7.20 \\ 4.10 \\ .40 \end{bmatrix}$$

Action 5:

$$P^5 = \begin{bmatrix} .822 & .058 & .120 \\ .175 & .726 & .099 \\ .215 & .065 & .721 \end{bmatrix} \quad Q^5 = \begin{bmatrix} .742 & .196 & .062 \\ .103 & .641 & .256 \\ .147 & .208 & .645 \end{bmatrix} \quad r^5 = \begin{bmatrix} 2.00 \\ 2.40 \\ 9.80 \end{bmatrix}$$

Action 6:

$$P^6 = \begin{bmatrix} .666 & .033 & .301 \\ .114 & .652 & .233 \\ .217 & .116 & .667 \end{bmatrix} \quad Q^6 = \begin{bmatrix} .681 & .205 & .114 \\ .101 & .734 & .166 \\ .102 & .132 & .766 \end{bmatrix} \quad r^6 = \begin{bmatrix} 6.20 \\ 9.60 \\ 4.30 \end{bmatrix}$$

DATA SET 3

Number of States = 4

Number of Actions = 4

Number of Signals = 4

Action 1:

$$P^1 = \begin{bmatrix} .379 & .271 & .000 & .350 \\ .000 & .341 & .407 & .252 \\ .000 & .434 & .000 & .566 \\ .000 & .414 & .000 & .586 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} .589 & .282 & .018 & .110 \\ .057 & .721 & .144 & .077 \\ .123 & .160 & .561 & .156 \\ .219 & .011 & .115 & .654 \end{bmatrix} \qquad r^1 = \begin{bmatrix} .20 \\ 5.40 \\ 2.50 \\ 8.00 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} .000 & .631 & .000 & .369 \\ .632 & .000 & .000 & .368 \\ .000 & .000 & .523 & .477 \\ .000 & .521 & .000 & .479 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} .542 & .200 & .126 & .133 \\ .180 & .658 & .157 & .005 \\ .181 & .103 & .550 & .166 \\ .094 & .110 & .146 & .649 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 5.80 \\ 8.40 \\ 7.50 \\ 1.70 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} .319 & .339 & .342 & .000 \\ .190 & .332 & .188 & .290 \\ .517 & .000 & .000 & .483 \\ .000 & .378 & .000 & .622 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} .522 & .021 & .169 & .287 \\ .131 & .593 & .113 & .163 \\ .017 & .233 & .612 & .138 \\ .145 & .036 & .149 & .670 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 9.30 \\ 5.60 \\ 9.70 \\ 8.80 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} .000 & 1.000 & .000 & .000 \\ .416 & .000 & .584 & .000 \\ .000 & .199 & .292 & .509 \\ .402 & .000 & .272 & .326 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} .624 & .162 & .155 & .059 \\ .141 & .654 & .140 & .065 \\ .129 & .232 & .600 & .038 \\ .027 & .270 & .124 & .579 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 4.20 \\ 9.40 \\ 4.40 \\ 2.60 \end{bmatrix}$$

DATA SET 4

Number of States = 5

Number of Actions = 5

Number of Signals = 5

Action 1:

$$P^1 = \begin{bmatrix} .000 & .234 & .000 & .292 & .474 \\ .266 & .397 & .000 & .000 & .337 \\ .000 & .000 & .344 & .339 & .318 \\ .000 & .362 & .638 & .000 & .000 \\ .274 & .248 & .000 & .303 & .175 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} .705 & .011 & .041 & .030 & .214 \\ .058 & .655 & .164 & .052 & .071 \\ .056 & .032 & .603 & .148 & .161 \\ .025 & .177 & .157 & .512 & .129 \\ .150 & .112 & .082 & .100 & .555 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 2.40 \\ 5.80 \\ 4.20 \\ 2.70 \\ 2.60 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} .000 & .233 & .315 & .000 & .452 \\ .347 & .360 & .000 & .000 & .293 \\ .192 & .192 & .246 & .105 & .265 \\ .000 & .000 & .279 & .721 & .000 \\ .314 & .000 & .304 & .226 & .157 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} .531 & .210 & .014 & .099 & .146 \\ .175 & .623 & .072 & .040 & .090 \\ .197 & .064 & .542 & .190 & .006 \\ .006 & .082 & .226 & .634 & .052 \\ .111 & .139 & .060 & .064 & .626 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 3.70 \\ 5.10 \\ 7.60 \\ 5.60 \\ 9.00 \\ cr \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} .275 & .257 & .284 & .184 & .000 \\ .000 & .381 & .000 & .358 & .261 \\ .294 & .000 & .000 & .335 & .371 \\ .343 & .264 & .000 & .265 & .128 \\ .595 & .000 & .405 & .000 & .000 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} .525 & .020 & .202 & .074 & .179 \\ .036 & .636 & .149 & .105 & .073 \\ .038 & .079 & .600 & .072 & .210 \\ .009 & .022 & .219 & .522 & .229 \\ .035 & .024 & .059 & .200 & .683 \end{bmatrix} \qquad r^3 = \begin{bmatrix} 5.50 \\ 1.60 \\ 1.00 \\ 3.70 \\ 1.70 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} .167 & .301 & .250 & .282 & .000 \\ .343 & .000 & .207 & .450 & .000 \\ .288 & .000 & .170 & .228 & .314 \\ .416 & .251 & .000 & .333 & .000 \\ .264 & .433 & .000 & .304 & .000 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} .591 & .107 & .044 & .132 & .126 \\ .036 & .735 & .035 & .149 & .045 \\ .002 & .131 & .627 & .129 & .111 \\ .122 & .018 & .181 & .617 & .062 \\ .143 & .079 & .177 & .035 & .566 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 3.90 \\ 5.90 \\ 3.70 \\ 3.90 \\ 4.30 \end{bmatrix}$$

Action 5:

$$P^5 = \begin{bmatrix} .177 & .000 & .390 & .289 & .144 \\ .274 & .286 & .141 & .183 & .116 \\ .284 & .145 & .261 & .310 & .000 \\ .386 & .403 & .000 & .211 & .000 \\ .239 & .000 & .253 & .000 & .508 \end{bmatrix}$$

$$Q^5 = \begin{bmatrix} .571 & .224 & .120 & .078 & .008 \\ .090 & .625 & .122 & .099 & .064 \\ .096 & .185 & .556 & .154 & .010 \\ .024 & .042 & .011 & .726 & .198 \\ .085 & .119 & .000 & .168 & .627 \end{bmatrix} \qquad r^5 = \begin{bmatrix} 3.20 \\ 1.80 \\ 4.30 \\ 1.50 \\ 9.10 \end{bmatrix}$$

DATA SET 5

Number of States = 6

Number of Actions = 4

Number of Signals = 6

Action 1:

$$P^1 = \begin{bmatrix} .106 & .167 & .207 & .196 & .223 & .101 \\ .000 & .357 & .000 & .303 & .341 & .000 \\ .000 & .000 & .196 & .207 & .396 & .201 \\ .000 & .254 & .164 & .153 & .187 & .242 \\ .125 & .234 & .000 & .239 & .138 & .265 \\ .200 & .229 & .159 & .000 & .197 & .216 \end{bmatrix}$$

$$Q^1 = \begin{bmatrix} .549 & .051 & .091 & .148 & .062 & .098 \\ .029 & .620 & .112 & .097 & .062 & .080 \\ .003 & .007 & .559 & .203 & .044 & .183 \\ .086 & .007 & .136 & .660 & .001 & .110 \\ .086 & .090 & .029 & .126 & .561 & .108 \\ .084 & .099 & .019 & .042 & .144 & .612 \end{bmatrix} \qquad r^1 = \begin{bmatrix} 1.30 \\ 4.40 \\ 1.70 \\ 3.50 \\ 6.90 \\ 9.00 \end{bmatrix}$$

Action 2:

$$P^2 = \begin{bmatrix} .275 & .278 & .000 & .166 & .000 & .281 \\ .000 & .462 & .000 & .000 & .538 & .000 \\ .000 & .000 & .231 & .000 & .228 & .541 \\ .173 & .298 & .177 & .182 & .169 & .000 \\ .114 & .180 & .116 & .119 & .304 & .167 \\ .000 & .236 & .126 & .213 & .270 & .155 \end{bmatrix}$$

$$Q^2 = \begin{bmatrix} .501 & .143 & .148 & .011 & .049 & .147 \\ .028 & .508 & .134 & .123 & .097 & .110 \\ .106 & .133 & .621 & .079 & .023 & .038 \\ .092 & .017 & .150 & .527 & .066 & .148 \\ .110 & .061 & .052 & .111 & .607 & .059 \\ .097 & .089 & .104 & .093 & .105 & .513 \end{bmatrix} \qquad r^2 = \begin{bmatrix} 3.50 \\ 6.90 \\ 1.50 \\ 4.40 \\ 4.00 \\ 1.90 \end{bmatrix}$$

Action 3:

$$P^3 = \begin{bmatrix} .209 & .304 & .204 & .000 & .282 & .000 \\ .299 & .000 & .197 & .000 & .000 & .505 \\ .239 & .242 & .000 & .275 & .243 & .000 \\ .000 & .000 & .325 & .000 & .343 & .332 \\ .213 & .000 & .390 & .000 & .397 & .000 \\ .151 & .127 & .212 & .228 & .000 & .282 \end{bmatrix}$$

$$Q^3 = \begin{bmatrix} .633 & .111 & .069 & .017 & .136 & .035 \\ .030 & .618 & .148 & .154 & .018 & .032 \\ .080 & .074 & .628 & .114 & .035 & .069 \\ .043 & .008 & .060 & .531 & .207 & .151 \\ .112 & .080 & .097 & .101 & .572 & .038 \\ .102 & .115 & .031 & .027 & .123 & .603 \end{bmatrix} \qquad r^3 = \begin{bmatrix} .30 \\ 9.40 \\ 5.80 \\ 2.60 \\ 2.70 \\ 1.70 \end{bmatrix}$$

Action 4:

$$P^4 = \begin{bmatrix} .179 & .345 & .000 & .000 & .269 & .207 \\ .404 & .196 & .401 & .000 & .000 & .000 \\ .186 & .221 & .210 & .000 & .383 & .000 \\ .186 & .201 & .218 & .169 & .000 & .228 \\ .179 & .188 & .182 & .000 & .207 & .244 \\ .268 & .000 & .000 & .402 & .000 & .330 \end{bmatrix}$$

$$Q^4 = \begin{bmatrix} .538 & .035 & .096 & .023 & .175 & .133 \\ .097 & .599 & .109 & .092 & .090 & .013 \\ .038 & .141 & .599 & .016 & .103 & .103 \\ .074 & .075 & .120 & .547 & .134 & .050 \\ .123 & .048 & .166 & .097 & .516 & .049 \\ .103 & .101 & .128 & .038 & .064 & .565 \end{bmatrix} \qquad r^4 = \begin{bmatrix} 6.40 \\ 7.10 \\ 7.60 \\ 6.70 \\ 6.20 \\ 1.10 \end{bmatrix}$$