Policy Filtering for Planning in Partially Observable Stochastic Domains†

Nevin Lianwen Zhang

Technical Report HKUST-CS95-05
February 21, 1995

†Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong

†UAI’95 submission.
Abstract

Partially observable Markov decision processes (POMDP) can be used as a model for planning in stochastic domains. This paper considers the problem of computing optimal policies for finite horizon POMDPs.

In deciding on an action to take, an agent is not only concerned with how the action would affect the current time point, but also its impacts on the rest of the planning horizon. In a POMDP, the future effects of an action are not separable from the effects of the agent’s future behavior. Consequently, one needs to consider the agent’s future behavior in order to properly evaluate the future impacts of an action. One reason that makes POMDPs difficult to solve is that the agent can behave in an exponential number (in the length of the remaining of the planning horizon) of ways. This paper represents the agent’s future behavior in terms of sub-policy-trees and gives a method that reduces the number of possible sub-policy-trees by collapsing similar sub-policy-trees and by pruning inferior sub-policy-trees.
1 Introduction

There is a growing interest in using Markov decision processes (MDP) as a model for planning in stochastic domains (Dean and Wellman 1991, Provan and Clarke 1993, Dean et al 1993, Cassandra et al 1994). In this model, there is a state variable that represents the state of a world. The world evolves stochastically over time. At each time point, the agent obtains some observations about the world and takes an action. The agent receives a reward (or penalty) at each time point depending whether the planning goal is achieved and on the costs of actions. A policy specifies, for each time point, the appropriate action to take in response to each possible contingency. An optimal policy is one that maximizes the expected total reward, i.e. one that leads the agent to achieve the goal with minimum cost.

The world can be either fully observable or only partially observable by the agent. The fully observable case has been studied extensively in the dynamic programming literature (e.g. Bertsekas 1987, White 1993). Dean et al (1993) have proposed a search algorithm to deal with the applications where the world can be in a large number of states.

This paper is concerned with the partially observable case. This case is considerably more difficult than the fully observable case for two related reasons. First, when the agent knows exactly which state the world is currently in, information from the past — past observations and actions — is irrelevant to the current decision. This is the so-called Markov property. On the other hand, when the agent does not have full observation of the world, past information can be used to better estimate the true current state of the world, and hence should be taken into account. The problem is that the number of possible states of
past information increases exponentially with time. This is usually referred to as the “curse of dimensionality”. Zhang and Boerlage (1995) have proposed a method for reducing the number of possible information states by pruning inconsistent information states and by collapsing similar consistent information states.

Second, in deciding on an action to take the agent is not only concerned with how the action would affect the current time point, but also its impacts on the rest of the planning horizon. In a fully observable MDP, the effects of an action is fully observed at the next time point. Hence one is able to separate the effects of an action performed at one time point from the effects of actions performed at other time points. In a POMDP, on the other hand, the effects of an action is not fully observed at the next time point, and one is unable to separate the effects of the current action from the effects of the agent’s future behavior. Hence one needs to consider the agent’s future behavior in order to properly evaluate the future impacts of the current action. The problem is that the agent can behave in an exponential number (in the length of the remaining of the planning horizon) of ways. This paper represents the agent’s future behavior in terms of sub-policy-trees and describes a method that reduces the number of possible sub-policy-trees by collapsing similar sub-policy-trees and by pruning inferior sub-policy-trees. In a sense, this paper is a dual to Zhang and Boerlage (1995).

The organization of the rest of the paper is as follows. Section 2 reviews POMDPs and introduces the concepts of policy trees sub-policy-trees. Section 3 shows how to inductively prune pointwise inferior sub-policy-trees and collapse similar sub-policy-trees. A bound on the sacrifice of optimality is given in Section 4. Section 5 defines inferior sub-
Figure 1: A finite horizon POMDP.

policy-tree classes and shows how they can be pruned. Section 6 provides a concise way for understanding previous works in terms of sub-policy-trees and compares them with the method proposed here in this paper. Conclusions are provided in Section 7.

2  POMDPs and planning in stochastic domains

This section reviews the concept of POMDP. As an motivating example, we consider the path planning problem for an agent who travels over a finite grid. Figure 1 shows a POMDP model for this problem. It consists of three types of variables: random, decision, and value variables, which are respectively drawn as circles, rectangles, and diamonds. The random variable $s_t$ represents the location of the agent at time $t$, and is called the state variable. The random variable $o_t$ stands for the observed location of the agent. The decision variable $d_t$ represents the action the agent takes at time $t$, which could be one of stay, go-east, go-south, go-west, and go-north. Here go-east means to move one step eastward. The value variable $r_t$ encodes the planning goal and criteria for good plans.
The observed location $o_t$ depends on the true location $s_t$, as indicated by the arrow $o_t \rightarrow s_t$. Due to noise in observation, this dependency is probabilistic in nature. The observation $o_t$ also probabilistically depends on the action $d_{t-1}$ of the previous time point, because the observation could be noisier when the agent is moving than when the agent stays still. The dependency of $o_t$ upon $d_{t-1}$ and $s_t$ is numerically characterized by a conditional probability $P(o_t|d_{t-1}, s_t)$.

We assume time begins at $1^1$. When $t=1$, $P(o_1|d_{0}, s_1)$ is to be understood as $P(o_1|s_1)$.

The location $s_{t+1}$ of the agent at the next time point depends on its current location $s_t$ and the current action $d_t$, as indicated by the arcs $s_t \rightarrow s_{t+1}$ and $d_t \rightarrow s_{t+1}$. This dependency is again probabilistic because an action might not have the intended effects due to a couple of reasons. First, the agent might not be able to carry actions accurately. When executing go-north, for instance, there might be some chance of overshooting and sliding sideways. Second, a link in the grid might sometimes be broken. In such a case, the action to move from one end of the link to the other will fail. The dependency of $s_{t+1}$ upon $s_t$ and $d_t$ is numerically characterized by a conditional probability $P(s_{t+1}|s_t, d_t)$.

The agent's initial location start is encoded by letting $P(s_1)$ be 1 when $s_1 = \text{start}$ and be 0 otherwise.

The value (or reward) variable $r_t$ depends on the agent’s current location $s_t$, the current action $d_t$, and the agent’s location $s_{t+1}$ at the next time point. The value variable

---

$^1$Normally time is assumed to start at 0. We choose 1 over 0 for notational concerns.
Figure 2: Another formulation of finite horizon POMDP.

is characterized by a value (or reward) function $r_t(s_t, d_t, s_{t+1})$. The goal to reach location goal and the preference for short plans can be encoded, for example, by setting $r_t(s_t, d_t, s_{t+1})$ as follows:

$$
\begin{align*}
    r_t(s_t, d_t, s_{t+1}) &= \begin{cases} 
        -\text{cost}(d_t) & \text{if } s_t \neq \text{goal} \text{ and } s_{t+1} \neq \text{goal} \\
        \text{reward} - \text{cost}(d_t) & \text{if } s_t \neq \text{goal} \text{ and } s_{t+1} = \text{goal} \\
        -\text{reward} - \text{cost}(d_t) & \text{if } s_t = \text{goal} \text{ and } s_{t+1} \neq \text{goal} \\
        -\text{cost}(d_t) & \text{if } s_t = \text{goal} \text{ and } s_{t+1} = \text{goal},
    \end{cases}
\end{align*}
$$

(1)

where reward is the reward for achieving the goal. Usually, reward should be much larger than the costs of actions.

One can “average out” $s_{t+1}$ from the value function $r_t(s_t, d_t, s_{t+1})$ to get $r_t(s_t, d_t) = \sum_{s_{t+1}} r_t(s_t, d_t, s_{t+1}) P(s_{t+1}|s_t, d_t)$, and reformulate the POMDP in Figure 1 in the way as shown in Figure 2. This is the formulation we will work with in the rest of the paper.

At each time point $t$, the agent needs to decides on an action $d_t$ to take. This decision is to be made based on the collection of information available at time $t$, i.e. the values of
variables $o_1$, $d_1$, $o_{t-1}$, $d_{t-1}$, and $o_t$, as indicated by the dotted arcs in Figures 1 and 2. Those arcs are called informational arcs and are dotted only for the sake of readability.

The number of time steps $N$ considered in a POMDP model is called the horizon of the POMDP. POMDPs are classified into finite horizon POMDPs and infinite horizon POMDPs. This paper considers only finite horizon POMDPs. The problem of choosing an appropriate horizon is out of the scope of this paper. See Zhang and Boerlage (1995) for a preliminary discussion.

### 2.1 Policy trees

From a POMDP, we want to obtain an action selection policy that is optimal in some sense. This section formalizes the concept of optimal policy.

A policy can either be given as a list of decision rules or as a policy tree. This paper uses the latter. Figure 3 illustrates the concept of policy tree. The tree reads as follows. At time 1, if 0 is observed take action number 3; if 1 is observed take action number 1; and so on. Given $o_1=0$, the action for time 2 is action number 0 if $o_2=0$; action number 1 if $o_2=1$; and so on. Given $o_1=1$, the action for time 2 is action number 2 if $o_2=0$, action number 3 if $o_2=1$, and so on so forth.

To define policy trees algebraically, we need the concept of frame. The set of all possible values for a variable $x$ is called the frame of $x$, which will be denoted by $\Omega_x$. Thus, $\Omega_s$ is the set of possible locations the agent can be at; $\Omega_o$ is the set of possible observations; and $\Omega_d$ is the set of possible actions. For a set $X$ of variables, $\Omega_X$ is defined
by $\Omega_X = \prod_{x \in X} \Omega_x$. This paper assumes all variables have finite frames.

An $o_N$-rooted sub-policy-tree, denoted by $\delta^N$, is a mapping $\delta^N : \Omega_{o_N} \to \Omega_{d_N}$. We use $\Omega_{\delta^N}$ to denote the set of all possible $o_N$-rooted sub-policy-trees.

For any $1 \leq t < N$, an $o_t$-rooted sub-policy-tree, denoted by $\delta^t$, is recursively defined to be a mapping: $\delta^t : \Omega_{o_t} \to \Omega_{d_t} \times \Omega_{\delta^{t+1}}$, where $\Omega_{\delta^{t+1}}$ is the set of all possible $o_{t+1}$-rooted sub-policy-trees. For each $o_t$, the sub-policy-tree $\delta^t$ specifies two things: first an action to take for time $t$, which will be denoted by $\delta^t_a(o_t)$; and then an $o_{t+1}$-rooted sub-policy-tree, which will be denoted by $\delta^t_p(o_t)$. For this reason, $\delta^t(o_t)$ will sometimes be written as a vector $(\delta^t_a(o_t), \delta^t_p(o_t))$.

An $o_1$-rooted sub-policy-tree is called a policy tree.

To see an example, let $\delta^1$ be the policy tree shown in Figure 3. Then $\delta^1_a(o_1=0)=3$ and

Figure 3: A policy tree.
\( \delta^1_p(o_1=0) \) is the sub-tree rooted at the upper most \( o_2 \)-node; \( \delta^1_p(o_1=1) \) is the sub-tree rooted at the middle level \( o_2 \)-node; and so on. Let \( \delta^2 \) be the sub-policy-tree rooted at the upper most \( o_2 \)-node in Figure 3. Then \( \delta^2_p(o_2=0)=0 \) and \( \delta^2_p(o_2=0) \) is the sub-tree rooted at the upper most \( o_2 \)-node; \( \delta^2_d(o_2=1)=1 \) and \( \delta^2_p(o_2=1) \) is the sub-tree rooted at the second upper most \( o_2 \)-node; and so on.

This paper allows \( \delta'(o_i) \) to take a special value \texttt{fail}. To the agent, \( \delta'(o_i) = \texttt{fail} \) means giving up attempts to reach goal when \( o_i \) is observed at time \( t \).

A few notes are in order. First to relate back to the abstract and introduction, let us remark that an \( o_i \)-rooted sub-policy-tree \( \delta^i \) can be viewed as a representation of one possible way the agent can behave from time \( t \) to \( N \). Second, the fact that \( \delta^i \) covers the entire period from time \( t \) to \( N \) rather than a single time point explains why \( t \) is in the superscript rather than in the subscript. Finally, the concept of sub-policy-tree defined here is closely related to the concept of \( t \)-step policy tree of Littman (1994).

### 2.2 Value functions and the optimality criterion

For any \( t \), let \( V_t(s_t, d_t, \delta^{t+1}) \) be the expected total rewards the agent receives from time \( t \) to \( N \) if at time \( t \) it is at location \( s_t \) and takes the action \( d_t \), and if it behaves according to the sub-policy-tree \( \delta^{t+1} \) from time \( t+1 \) to \( N \). We call the pair \((d_t, \delta^t)\) a \( d_t \)-rooted sub-policy-tree and call \( V_t(s_t, d_t, \delta^{t+1}) \) the value function of the \( d_t \)-rooted sub-policy-tree. It is not difficult
to see that the value function satisfies the following recursion:

$$V_t(s_t, d_t, \delta^{t+1}) = r_t(s_t, d_t) + \sum_{o_{t+1}, s_{t+1}} V_{t+1}(s_{t+1}, \delta^{t+1}_{d}(o_{t+1}), \delta^{t+1}_{p}(o_{t+1})) P(s_{t+1}, o_{t+1} | s_t, d_t),$$

(2)

where the second term is to be understood as 0 when $t=N$, and when $t=N-1$, the term $V_{t+1}(s_{t+1}, \delta^{t+1}_{d}(o_{t+1}), \delta^{t+1}_{p}(o_{t+1}))$ is to be understood as $V_N(s_N, \delta^{N}_{d}(o_N))$. Also

$$P(s_{t+1}, o_{t+1} | s_t, d_t) = d_{t,f} P(s_{t+1} | s_t, d_t) P(o_{t+1} | s_{t+1}, d_t)^2.$$

In equation (2), we did not consider the case when the agent simply gives up by choosing the sub-policy-tree \texttt{fail}. To take this case into account, we will work with, instead of the value function $V_t(s_t, d_t, \delta^{t+1})$, the function $V_t(s_t, \delta | o_t)$ which stands for the expected total reward the agent receives from time $t$ to $N$ if at time $t$ it is observed at location $o_t$ and is actually at location $s_t$, and if it behaves according to the sub-policy-tree $\delta(t)$ from time $t$ to $N$. It is evident that $V_t(s_t, \delta | o_t)$ depends only on the value of $\delta(t)$ at the given observation $o_t$ while is independent of the values of $\delta(t)$ for other possible observations. Hence, we write it as $V_t(s_t, \delta(t) | o_t)$, or simply as $V_t(s_t, \delta(t))$.

Given an observation $o_t$, $\delta(t) | o_t$ is a sub-tree of, or more precisely a branch of, the $o_t$-rooted sub-policy-tree $\delta(t)$. We will call it an $o_t$-fixed sub-policy-tree. Note that in an $o_t$-rooted sub-policy-tree the value of $o_t$ is not given, while in an $o_t$-fixed sub-policy-tree the value of $o_t$ is given.

---

Note that in the literature, $V_t$ usually denotes the expected total rewards the agent receives during the last $t$ time steps of the planning horizon, i.e. from time $N-t$ to $N$. Thus the recursion goes from $V_{t+1}$ to $V_t$. In this paper, $V_t$ denotes the expected total rewards the agent receives from time $t$ to $N$. Thus the recursion is in the opposite direction, from $V_{t+1}$ to $V_t$. 

9
As a function of \( s_t \), \( V_t(s_t, \delta^t(o_t)) \) will be referred to as the value function of the \( o_t \)-fixed sub-policy-tree \( \delta^t(o_t) \). It is given by

\[
V_t(s_t, \delta^t(o_t)) = \begin{cases} 
0 & \text{if } \delta^t(o_t) = \text{fail} \\
V_t(s_t, \delta^t(o_t), \delta^t(o_t)) & \text{otherwise}
\end{cases}
\]

(3)

where \( V_t(s_t, \delta^t(o_t), \delta^t(o_t)) \) is given by equation (2). The value function is zero when \( \delta^t(o_t) = \text{fail} \), because if the agent gives up attempts to reach the goal, then it does not receive any rewards and it does not incur any costs either.

The expected total reward \( V(\delta^1) \) the agent receives during the entire planning horizon if it behaves according to the policy tree \( \delta^1 \) is given by

\[
V(\delta^1) = \sum_{o_1, s_1} V_t(s_1, \delta^1(o_1)) P(s_1) P(o_1 | s_1).
\]

(4)

We will call \( V(\delta^1) \) the expected value of the policy tree \( \delta^1 \). A policy tree \( \hat{\delta}^1 \) is optimal if

\[
V(\hat{\delta}^1) = \max_{\delta^1} V(\delta^1).
\]

(5)

The quantity \( \max_{\delta^1} V(\delta^1) \) is called the optimal expected value of the POMDP under discussion. To solve a POMDP is to find an optimal policy tree and the optimal expected value.
2.3 An efficiency concern

Except for the special sub-policy-tree fail, the optimal policy tree defined in the previous subsection is the same as in the literature. The definition is wasteful in $V_t(s_t, \delta'(o_t))$. Because the agent’s actual location $s_t$ cannot be too far away from its observed location $o_t$. More specifically, $s_t$ cannot be outside of the set $\Omega_{s_t|o_t} = \{s_t | \sum_{d_{t-1}} P(o_t|s_t, d_{t-1}) > 0\}$. Thus there is no need to define $V_t(s_t, \delta'(o_t))$ for the values of $s_t$ that are not in $\Omega_{s_t|o_t}$. The waste can be substantial since the $\Omega_{s_t|o_t}$ might be only a small portion of $\Omega_{s_t}$.

The waste can be avoided by doing two things. First, recursively define the value function $V_t(s_t, \delta'(o_t))$ only for $s_t \in \Omega_{s_t|o_t}$ through the following equation:

$$V_t(s_t, \delta'(o_t)) = r_t(s_t, \delta''(o_t)) + \sum_{o_{t+1}} \sum_{s_{t+1}} V_{t+1}(s_{t+1}, \delta'(o_{t+1})) P(s_{t+1}, o_{t+1}|s_t, \delta'(o_t)),$$

where the second term is to be understood as 0 when $t=N$. The expression $\delta'(o_{t+1})$ makes sense because $\delta'(o_t)$ is an $o_{t+1}$-rooted sub-policy-tree. Second, rewrite equation (4) as:

$$V(\delta) = \sum_{o_t} \sum_{s_t \in \Omega_{s_t|o_t}} V_t(s_t, \delta'(o_t)) P(s_t) P(o_t|s_t).$$

The concept of $o_t$-fixed sub-policy-tree allows us to avoid wastes by focusing on the locations in $\Omega_{s_t|o_t}$. This is an more important reason for using $o_t$-fixed sub-policy-trees, instead of $d_t$-rooted sub-policy-trees, than the need to account for the special sub-policy-tree
3 Policy filtering

According to Cassandra (1994), if all the decision variable $d_i$ have the same number, say $|\Omega_d|$, of possible values and the observation variable $o_t$ have the same number, say $|\Omega_o|$, of possible values, then the number of possible $o_t$-rooted sub-policy-trees is $|\Omega_d|^{\frac{|\Omega_o|^{N-t+1}-1}{|\Omega_o|-1}}$.

This makes POMDPs difficult to solve. This paper seeks to reduce the number of possible sub-policy-trees. We prune inferior sub-policy-trees and collapse sub-policy-trees that have approximately the same value functions.

3.1 Pointwise inferior sub-policy-trees

Suppose the agent is observed to be at location $o_t$ at time $t$. An $o_t$-fixed sub-policy-tree $\alpha$ is pointwise inferior to another $o_t$-fixed sub-policy-tree $\beta$ if and only if for all $s_t \in \Omega_{s_t|o_t}$,

$$V_i(s_t, \delta^t(o_t) = \alpha) \leq V_i(s_t, \delta^t(o_t) = \beta).$$

When it is the case, we also say that $\beta$ pointwise dominates $\alpha$.

To see an example, consider the case when $o_t$ is so far away from goal that there is no chance for the agent to reach goal in the remaining $N-t+1$ steps of the planning horizon, then $\text{fail}$ dominates all the other possible $o_t$-fixed sub-policy-trees. As another example, suppose $o_t$ is to the North of goal. Then an $o_t$-fixed sub-policy-tree that does not contain
any instance of the action go-south might be dominated by fail.

Pointwise inferior sub-policy-trees can be pruned without affecting the optimal expected value of the POMDP under discussion. This is because that in any policy tree $\delta^1$ that contains $\delta^t(o_t)=\alpha$ as a subtree, if we replace this subtree with $\delta^t(o_t)=\beta$, then the expected value of the resulting policy tree is no smaller than that of $\delta^1$ itself. The idea of pruning pointwise inferior sub-policy-tree is due to Eagle (Cassandra 1994). See Section 6 for more details.

3.2 Sub-policy-trees with similar value functions

Suppose $\alpha$ and $\beta$ are two $o_t$-fixed sub-policy-trees. It might be the case that $\alpha$ and $\beta$ has roughly the same value functions, i.e. $V_i(s_t, \delta^t(o_t)=\alpha)$ and $V_i(s_t, \delta^t(o_t)=\beta)$ are roughly the same for all possible $s_t \in \Omega_{s_t|o_t}$. To see an example, suppose $\alpha$ dictates that the action for time $t$ be go-south and the action for time $t+1$ be go-east regardless of the observation $o_{t+1}$; and suppose $\beta$ dictates that the action for time $t$ be go-east and the action for time $t+1$ be go-south regardless of the observation $o_{t+1}$. Then it is likely that the value functions for $\alpha$ and $\beta$ are roughly the same.

To see a second example, recall that both $\alpha$ and $\beta$ are vectors consisting of two components. Let us write them as $(\alpha_1, \alpha_2)$ and $(\beta_1, \beta_2)$, where $\alpha_1$ and $\beta_1$ are values of $d_t$, and $\alpha_2$ and $\beta_2$ are mappings from $\Omega_{o_{t+1}} \rightarrow \Omega_{(d_{t+1}, \delta_{t+1})}$. The observation $o_{t+1}$ cannot be too far away from $o_t$. Thus if $\alpha_1=\beta_1$, and $\alpha_2(o_{t+1})$ and $\beta_2(o_{t+1})$ are the same except when $o_{t+1}$ are sufficiently far away from $o_t$, then $V_i(s_t, \delta^t(o_t)=\alpha)$ and $V_i(s_t, \delta^t(o_t)=\beta)$ should be the same.
for all possible $s_t \in \Omega_{s_t \mid o_t}$.

Suppose $\alpha$ and $\beta$ have roughly the same value functions. Then in any policy tree $\delta^1$ that contains $\delta^t(o_t) = \alpha$ as a subtree, if we replace the subtree with $\delta^t(o_t) = \beta$, the expected value of the resulting policy tree is roughly that same as that of $\delta^1$. Thus $\alpha$ and $\beta$ can be clustered and be treated as one.

### 3.3 The idea of policy filtering

This subsection shows one way to realize the idea of pruning pointwise inferior sub-policy-trees and the idea of collapsing similar sub-policy-trees.

Given $o_t$, suppose the set of all the possible $o_t$-fixed sub-policy-trees, i.e. the set $\Omega_{\{\delta_t, \delta^{t+1}\}}$, that are not pointwise inferior has somehow been partitioned into disjoint classes such that the sub-policy-trees in each class have roughly the same value functions. Let $L(o_t)$ denote the list of all those classes. We ignore the differences among the sub-policy-trees in the same class and treat them as one entity.

More specifically, we replace the sub-policy-tree variable $\delta^t$ with a sub-policy-tree-class variable $\phi^t$ such that for each $o_t$, $\phi^t(o_t)$ takes the classes in $L(o_t)$ as possible values; and we transform the problem of finding an optimal $\delta^1$ into the problem of finding an optimal $\phi^1$. Suppose an optimal $\phi^1$ has been found. For any $o_t$, $\phi^1(o_t)$ consists of a set $o_t$-fixed sub-policy-trees. Randomly choose one such sub-policy-tree (the probability distribution to be specified later), and take it as a solution policy tree for the POMDP under discussion.

Formally, $\phi^t$ is a mapping from $\Omega_{o_t} \rightarrow \bigcup_{o_t \in \Omega_{o_t}} L(o_t)$ with the restriction that for each $o_t$,
\( \phi^i(o_t) \) can only be one of the classes in \( L(o_t) \). For any \( d_t \neq o_t \), \( \phi^i(o_t) \) cannot be a class in \( L(d_t) \) unless that class is also in \( L(o_t) \). The number of possible values for \( \phi^i \) is \( \prod_{o_t \in \Omega_o} |L(o_t)| \), while the number of possible values for \( \delta^i \) is \( \prod_{o_t \in \Omega_{\delta^i}} |\Omega_{\{d_t, \delta^{i+1}\}}| \). Since \( |L(o_t)| \) is usually smaller than \( |\Omega_{\{d_t, \delta^{i+1}\}}| \), \( \phi^i \) usually has less possible values than \( \delta^i \). Consequently, replacing \( \delta^i \) with \( \phi^i \) makes it easier to solve the POMDP under discussion.

The process of going from \( \delta^i \) to \( \phi^i \) is called policy filtering because pointwise inferior sub-policy-trees and the differences among the sub-policy-trees in the same class are filtered out.

### 3.4 Inductive policy filtering process

The purpose of policy filtering is to avoid dealing with the large number of possible values for the sub-policy-tree variable \( \delta^i \). It thus defeats this purpose to carry out policy filtering by first explicitly computing the value function \( V_t(s_t, \delta^i(o_t)) \). This section presents an inductive approach.

As the induction hypothesis, assume that policy filtering has been carried out for \( \delta^{i+1} \). In other words, we assume that (1) for each \( o_{t+1} \), all the pointwise inferior \( o_{t+1} \)-fixed sub-policy-trees have been pruned and the set of \( o_{t+1} \)-fixed sub-policy-trees that are not pointwise inferior have been partitioned into classes; (2) a sub-policy-tree-class variable \( \phi^{i+1} \) has been introduced such that \( \phi^{i+1}(o_{t+1}) \) take those sub-policy-tree classes as possible values; and (3) that the value function \( V_{t+1}(s_{t+1}, \phi^{i+1}(o_{t+1})) \) have been obtained.

When \( t=N \), \( \phi^{i+1} \) is to be understood as a dummy sub-policy-tree-class variable whose
only possible value is \{\text{fail}\} and $V_{t+1}(s_{t+1}, \phi^{t+1}(a_{t+1}))$ is to be understood as the constant 0. This serves as the base of induction.

Now consider filtering $\delta'$. For each $o_t$, $\Omega_{\{d_t, \delta^{t+1}\} \cup \{\text{fail}\}}$ is the set of all possible $a_t$-fixed sub-policy-trees. Since $\delta^{t+1}$ has been filtered, a natural first step in filtering $\delta'$ is to inherit the filtering of $\delta^{t+1}$ by replacing the $\delta^{t+1}$ in the set $\Omega_{\{d_t, \delta^{t+1}\} \cup \{\text{fail}\}}$ with $\phi^{t+1}$, resulting in $\Omega_{\{d_t, \phi^{t+1}\} \cup \{\text{fail}\}}$. Let $\phi^{t,0}$ be the sub-policy-tree-class variable such that for each $o_t$ the set of all possible values for $\phi^{t,0}(o_t)$ is $\Omega_{\{d_t, \phi^{t+1}\} \cup \{\text{fail}\}}$.

Further filtering needs to be carried out for $\phi^{t,0}$. For this purpose, we need the value function $V_t(s_t, \phi^{t,0}(o_t))$. When $\phi^{t,0}(o_t)=\text{fail}$, $V_t(s_t, \phi^{t,0}(o_t)) = 0$ by definition. When $\phi^{t,0}(o_t) \neq \text{fail}$, $\phi^{t,0}(o_t)$ consists of two components $\phi_d^{t,0}(o_t)$ and $\phi_p^{t,0}(o_t)$, where $\phi_d^{t,0}(o_t)$ is a value of $d_t$ and $\phi_p^{t,0}(o_t)$ is an $o_{t+1}$-rooted sub-policy-tree-class, i.e. a mapping from $\Omega_{o_{t+1}} \rightarrow \Omega_{\phi^{t+1}}$. By the induction hypothesis, we have already obtained the value function $V_{t+1}(s_{t+1}, \phi_p^{t}(a_t)(o_{t+1}))$. Consequently, the value function $V_t(s_t, \phi^{t,0}(o_t))$ is

$$V_t(s_t, \phi^{t,0}(o_t)) = r_t(s_t, \phi_d^{t,0}(o_t)) + \sum_{s_{t+1}, d_{t+1}} V_{t+1}(s_{t+1}, \phi_p^{t,0}(o_t)(o_{t+1}))P(s_{t+1}, o_{t+1}|s_t, \phi_d^{t,0}(o_t)). \quad (9)$$

We also need an exact definition for two numbers to be “roughly the same”. This paper considers two numbers to be “roughly the same” if their difference is no larger than a certain threshold $\sigma$. The constant $\sigma$ will be referred as a clustering threshold.

Now here is a procedure for filtering $\phi^{t,0}$.
Procedure POLICY-FILTERING

- **INPUT:**
  
  1. $V_i(s_t, \phi^{t,0}(o_t))$.
  
  2. $\sigma$ — A clustering threshold.

- **OUTPUT:** $L(o_i)$ — A list of classes that constitute a partition of $\Omega_{\{\hat{d}_t, \phi^{t+1}\}} \cup \{\text{fail}\}$.

  1. Create a class with $\text{fail}$ as the only element and labelled it with $\text{fail}$.

     Initialize $L(o_i)$ to be the list whose only member is the class just created.

  2. **For** $\alpha$ running over $\Omega_{\{\hat{d}_t, \phi^{t+1}\}}$,

     (a) **Check if** $L(o_i)$ contains a class labelled $\alpha'$ such that $\alpha'$ pointwise dominates $\alpha$.

     (b) **If yes,** do nothing.

     (c) **If no,**

     - Traverse $L(o_i)$ and mark all the classes whose label is pointwise dominated by $\alpha$, and

     - **Check if** $L(o_i)$ contains a class labelled $\alpha'$ such that

       $$\max_{s_t \in \Omega_{\hat{d}_t \alpha}} |V_i(s_t, \phi^{t,0}(o_t) = \alpha) - V_i(s_t, \phi^{t,0}(o_t) = \alpha')| \leq \sigma/2.$$ 

       - **If yes,** add $\alpha$ to that class.

       - **If no,** create a new class, label it by $\alpha$, add it to $L(o_i)$. 


3. Delete from $L(o_t)$ all the marked classes.

4. Return $L(o_t)$.

Define $\phi^t$ to be the sub-policy-tree-class variable such that for each $o_t$ the set of possible values for $\phi^t(o_t)$ is $L(o_t)$. Thus, a possible value for $\phi^t(o_t)$ consists of a class of values for $\phi^{t,0}(o_t)$. This is an important point to remember in order to understand the rest of this section and the next section.

Define choosing the sub-policy-tree class $\phi^t(o_t)$ to mean randomly choosing a member $\phi^{t,0}(o_t)$ from the class $\phi^t(o_t)$ such that each member in the class has equal probability of being chosen. Therefore, the value function of $\phi^t(o_t)$ is as follows: for all $s_t \in \Omega_{s_t|o_t}$

$$V_t(s_t, \phi^t(o_t)) = \frac{1}{|\phi^t(o_t)|} \sum_{\phi^{t,0}(o_t) \in \phi^t(o_t)} V_t(s_t, \phi^{t,0}(o_t)),$$

where $|\phi^t(o_t)|$ stands for the number of members in the sub-policy-tree class $\phi^t(o_t)$.

That concludes the description of our inductive policy filtering process.

At the end of induction, we get a sub-policy-tree-class variable $\phi^1$ and a value function $V_t(s_1, \phi^1(o_1))$. The expected value of $\phi^1$, denoted by $V(\phi^1)$, is given by

$$V(\phi^1) = \sum_{o_1} \sum_{s_1 \in \Omega_{s_1|o_1}} V_t(s_1, \phi^1(o_1)) P(s_1) P(o_1|s_1).$$

18
A value $\hat{\phi}^1$ of $\phi^1$ is an $\sigma$-optimal policy tree class if

$$V(\hat{\phi}^1) = \max_{\phi^1} V(\phi^1).$$

The quantity $\max_{\phi^1} V(\phi^1)$ is called the $\sigma$-optimal expected value of the POMDP under discussion.

Since $\phi^1$ might have much less possible values than $\delta^1$, it can be much easier to find a $\sigma$-optimal policy tree class than an optimal policy tree. The next section provides a bound on the difference between the $\sigma$-optimal expected value and the optimal value. A first time reader might want to skip the section.

4 A bound on the sacrifice of optimality

Recall that for any $o_t$ a value for $\phi^t(o_t)$ consists of a class of values for $\phi^{t,0}(o_t)$. Recursively define two functions $V_i(s_t, \phi^t(o_t))$ and $V_i(s_t, \phi^t(o_t))$ as follows: When $\text{fail} \in \phi^t(o_t)$,

$$V_i(s_t, \phi^t(o_t)) = V_i(s_t, \phi^t(o_t))_{\text{def}} 0; \text{ otherwise}$$

$$V_i(s_t, \phi^t(o_t)) = \max_{\phi^t_p(o_t) \in \phi^t(o_t)} \{r_i(s_t, \phi^{t,0}_p(o_t)) + \sum_{s_{t+1}, o_{t+1}} V_{i+1}(s_{t+1}, \phi^{t,0}_p(o_t)(o_{t+1})) \cdot P(s_{t+1}, o_{t+1} | s_t, \phi^{t,0}_d(o_t)) \}, \quad (13)$$

and

$$V_i(s_t, \phi^t(o_t)) = \min_{\phi^t_p(o_t) \in \phi^t(o_t)} \{r_i(s_t, \phi^{t,0}_p(o_t)) + \sum_{s_{t+1}, o_{t+1}} V_{i+1}(s_{t+1}, \phi^{t,0}_p(o_t)(o_{t+1})) \cdot P(s_{t+1}, o_{t+1} | s_t, \phi^{t,0}_d(o_t)) \}, \quad (14)$$

19
where $\nabla_{t+1}(s_{t+1}, \phi_{t+p}^{t,0}(o_t)(o_{t+1}))$ and $\nabla_{t+1}(s_{t+1}, \phi_{t+p}^{t,0}(o_t)(o_{t+1}))$ are to be understood as the constant 0 when $t=N$.

To better understand the above two recursive definitions, we suggest the reader to compare them with equations (9, 10).

Define $\nabla(\phi^1)$ and $\nabla(\phi^1)$ respectively from $\nabla(s_1, \phi^0(o_1))$ and $\nabla(s_1, \phi^1(o_1))$ in the same way as $V(\phi^1)$ was defined from $V_1(s_1, \phi^1(o_1))$.

**Theorem 1** The difference between the $\sigma$-optimal expected value $\max_{\phi^1} V(\phi^1)$ and the optimal expected value $\max_{\phi^1} V(\delta^1)$ is bounded by

$$|\max_{\phi^1} V(\delta^1) - \max_{\phi^1} V(\phi^1)| \leq \max_{\phi^1} \nabla(\phi^1) - \max_{\phi^1} \nabla(\phi^1).$$

(15)

The proof can be found in the appendix.

5 Pruning inferior sub-policy-tree classes

After policy filtering, the number of sub-policy-tree classes might still be too large to deal with. This section introduces the concept of inferior sub-policy-tree classes and shows how they can be pruned. The technique of pruning inferior sub-policy-tree classes is more sophisticated and more expensive than policy filtering.
5.1 Inferior sub-policy-tree classes

A value for $\phi^t(o_t)$ is a set of values for $\phi^{t,0}(o_t)$, which in turn consists of two components $\phi^{t,0}_d(o_t)$ and $\phi^{t,0}_p(o_t)$; for any $o_{t+1}$, a value for $\phi^{t,0}_d(o_t)(o_{t+1})$ is again a set; and so on so forth, forming a hierarchy. From now on, we collapse the hierarchy and regard $\phi^t(o_t)$ simply as consisting of a set of $o_t$-fixed sub-policy-trees.

An $o_t$-fixed sub-policy-tree class $\alpha \in L(o_t)$ is inferior if for any probability distribution $P(s_t)$ over $\Omega_{o_t|o_t}$, there exists another $\alpha' \in L(o_t)$ such that

$$\sum_{s_t \in \Omega_{o_t|o_t}} P(s_t)V(s_t, \phi^t(o_t)=\alpha) \leq \sum_{s_t \in \Omega_{o_t|o_t}} P(s_t)V(s_t, \phi^t(o_t)=\alpha').$$  \tag{16}

**Theorem 2** In a POMDP, inferior $o_t$-fixed sub-policy-tree classes can be pruned from $L(o_t)$ without affecting the $\sigma$-optimal expected value.

**Proof:** Suppose the decision process has been executed to time $t-1$ and observation $o_t$ has been obtained at time $t$. Then the agent’s belief about $s_t$ is the probability distribution $P(s_t|o_t, h_t)$, where $h_t = \{o_1, d_1, \ldots, o_{t-1}, d_{t-1}, o_t\}$. $P(s_t|o_t, h_t)$ is zero outside $\Omega_{s_t|o_t}$. So, it can be regarded as a probability distribution over the set. If the agent behaves according to an $o_t$-fixed sub-policy-tree class $\alpha \in L(o_t)$, then the optimal expected total reward it receives in the remaining of the planning horizon is

$$\sum_{s_t \in \Omega_{o_t|o_t}} P(s_t|o_t, h_t)V(s_t, \phi^t(o_t)=\alpha).$$
If $\alpha$ is inferior, there exists another $o_t$-fixed sub-policy-tree class $\alpha' \in L(o_t)$ such that

$$\sum_{s_t \in \Omega_{s_t} | o_t} P(s_t | o_t, h_t) V_t(s_t, \phi^t(o_t) = \alpha) \leq \sum_{s_t \in \Omega_{s_t} | o_t} P(s_t | o_t, h_t) V_t(s_t, \phi^t(o_t) = \alpha').$$

In other words, using $\alpha'$ instead of $\alpha$ could possibly improve the expected total reward; and in the worst case the expected total reward stays the same. Therefore, pruning $\alpha$ from $L(o_t)$ will not affect the $\sigma$-optimal expected value of the POMDP under discussion. □

5.2 Direct pruning

There are two ways by which inferior $o_t$-fixed sub-policy-tree classes can be pruned: the direct approach and the indirect approach. The basic idea behind the direct approach is to traverse $L(o_t)$ and prune the inferior $o_t$-fixed sub-policy-trees one by one (Monahan 1982).

Enumerate the members of the set $\Omega_{s_t} | o_t$ as 0, 1, ..., $m$. Denote $P(s_t = i)$ by $\pi_i$. Then an $o_t$-fixed sub-policy-tree class $\alpha \in L(o_t)$ is not inferior if and only if there exist $\pi_i$’s that satisfy all the following constraints:

- $\sum_{i=0}^{m} \pi_i [V_i(s_t = i, \phi^t(o_t) = \alpha') - V_i(s_t = i, \phi^t(o_t) = \alpha)] \leq 0, \forall \alpha' \in L(o_t),$

- $\pi_i \geq 0, \forall i,$

- $\sum_{i=0}^{m} \pi_i = 1.$

One way to solve this constraint satisfaction problem is to set up a linear programming problem with those constraints and with the objective function being, for instance, the
constant function 1. This linear programming problem has a solution if and only if the constraint satisfaction problem has one.

Procedure DIRECT-PRUNING

- INPUT: $L(o_t)$ — A list of $o_t$-fixed sub-policy-tree classes.
- OUTPUT: The list of non-inferior sub-policy-tree classes in $L(o_t)$.

1. For $\alpha$ running over $L(o_t)$
   - Solve the aforementioned constraint satisfaction problem.
   - If solutions exist, do nothing,
   - Else remove $\alpha$ from $L(o_t)$.

2. Return $L(o_t)$.

One can incorporate DIRECT-PRUNING into POLICY-FILTERING simply by adding the line “DIRECT-PRUNING($L(o_t)$)” between items 3 and 4.

5.3 Indirect pruning

Given a distribution $P(s_t)$ over $\Omega_{s_t \mid o_t}$, an $o_t$-fixed sub-policy-tree class $\phi^t(o_t) \in L(o_t)$ is called the optimal $o_t$-fixed sub-policy-tree class for $P(s_t)$ if maximizes $\sum_{s_t} P(s_t)V_t(s_t, \phi^t(o_t))$. If there exit more than one such $\phi^t(o_t)$, choose an arbitrary one as the optimal $o_t$-fixed sub-policy-tree class for $P(s_t)$.
The basic idea behind the indirect approach to the pruning of inferior sub-policy-tree classes is as follows: partition the space of all possible probability distributions \( P(s_t) \) over \( \Omega_{s_t|a_t} \) into regions such that each region has the same optimal \( a_t \)-fixed sub-policy-tree class, and prune all the sub-policy-tree classes that are not the optimal sub-policy-tree class for any region. This idea originated from (Sondik 1971 and Cheng 1988) and its realization is too complex to be covered in this paper. For an excellent survey of different realizations, we refer the reader to Cassandra (1994).

6 Related works

Most previous methods for solving POMDPs are built on the idea of pruning inferior sub-policy-trees. This section gives a high level description of those methods and provides a brief comparison between our method and the earlier methods.

To begin with, let us note that previous works do not allow \texttt{fail} to be a possible sub-policy-tree. Consequently, the concepts of optimal policy tree and optimal expected value are slightly different from the one used this paper.

All the earlier methods carry out pruning inductively, in a way very much like what we did in this paper. However, they differ in the types of sub-policy-trees to prune, the inferiority criteria, and the pruning methods.

Sondik (1971), Monahan (1982), and Cheng (1988) prune \( d_t \)-rooted sub-policy-trees. The inferiority criterion they use is as follows. A \( d_t \)-rooted sub-policy-tree \( (d_t = a, \delta^{t+1} = \beta) \) is 

\textit{inferior} if for any probability distribution \( P(s_t) \) there exists another \( d_t \)-rooted sub-policy-tree
such that

\[ \sum_{s_t} P(s_t) V_i(s_t, d_i = \alpha', \delta^{i+1} = \beta') \leq \sum_{s_t} P(s_t) V_i(s_t, d_i = \alpha, \delta^{i+1} = \beta). \] (17)

Those three methods differ in their ways of pruning. Monahan (1982) uses direct pruning, while the other two use indirect pruning. Both being indirect, the pruning methods of Sondik (1971) and that of Cheng (1988) are very different. See Cassandra (1994) for a detailed comparison.

Cassandra (1994) and Littman (1994) prune \( \omega \)-rooted sub-policy-trees. They use what we call the \( d_i \)-inferiority criterion. Given \( d_i \), an \( \omega_{i+1} \)-rooted sub-policy-tree \( \beta \) is \( d_i \)-inferior if for any probability distribution \( P(s_t) \) there exists another \( \omega_{i+1} \)-rooted sub-policy-tree \( \beta' \) such that

\[ \sum_{s_t} P(s_t) V_i(s_t, d_i, \delta^{i+1} = \beta) \leq \sum_{s_t} P(s_t) V_i(s_t, d_i, \delta^{i+1} = \beta'). \] (18)

Both being indirect, the pruning methods of Cassandra (1994) and that Littman (1994) are different. See Littman (1994) for a comparison.

Our method is different from previous method in two ways. First, we push the POMDP through a filtering process and prune the resulting sub-policy-tree classes, rather than pruning the sub-policy-trees themselves. This is advantageous because the pruning process usually involves solving linear programming problems and is much more time consuming than the filtering process. By dealing with sub-policy-tree classes instead of sub-policy-tree
themselves, the number and the complexity of the linear programming problems are reduced.

Second, we prune $a_t$-fixed sub-policy-tree classes instead of $d_t$-rooted sub-policy-trees or $a_t$-rooted sub-policy-trees. This allows us to consider only all the probability distributions $P(s_t)$ over the set $\Omega_{s_t|o_t}$. All the previous methods need to consider all possible distributions $P(s_t)$ over the set $\Omega_{s_t}$. This can mean huge savings in applications such as path planning where the set $\Omega_{s_t|o_t}$ is much smaller than the set $\Omega_{s_t}$.

To end this section, we would like to point out the difference between our method of pruning pointwise inferior sub-policy-trees and that of Eagle (Cassandra 1994). We prune $a_t$-fixed sub-policy-trees while Eagle prunes $d_t$-rooted sub-policy-trees. A $d_t$-rooted sub-policy-tree $(d_t=\alpha, \delta^{t+1}=\beta)$ is pointwise inferior if there exists another $d_t$-rooted sub-policy-tree $(d_t=\alpha', \delta^{t+1}=\beta')$ such that for all $s_t$

$$\sum_{s_t} V_i(s_t, d_t=\alpha, \delta^{t+1}=\beta) \leq \sum_{s_t} V_i(s_t, d_t=\alpha', \delta^{t+1}=\beta'). \quad (19)$$

In this, we also say that $(d_t=\alpha', \delta^{t+1}=\beta')$ pointwise dominates $(d_t=\alpha, \delta^{t+1}=\beta)$.

It is more likely for an $a_t$-fixed sub-policy-tree to pointwise dominate another $a_t$-fixed sub-policy-tree than for a $d_t$-rooted sub-policy-tree to pointwise dominate another $d_t$-rooted sub-policy-tree. For example, there is no reason why a $d_t$-rooted sub-policy-tree that does not contain go-south should be dominated by fail, while it is very likely that fail might pointwise dominate an $a_t$-fixed sub-policy-tree that does not contain go-south if $o_t$ is to the north of goal.
7 Summary

POMDPs are difficult to solve. One reason is that to appropriately evaluate the effects of an action, one needs to consider the agent’s future behaviors and the agent can behave in an exponential number (in the length of the remaining of the planning horizon) of ways. This paper represents the agent’s future behavior as sub-policy-trees, and proposes to reduce the number of possible sub-policy-trees through an inductive pruning-collapsing-pruning process.

At each step of the induction, pointwise inferior sub-policy-trees are first pruned. This idea is originally due to Eagle (Cassandra 1994). The introduction of fail as a possible sub-policy-tree and the use of $a_t$-fixed sub-policy-trees instead of $d_t$-rooted sub-policy-trees enhance the effectiveness of the idea.

Secondly, $a_t$-fixed sub-policy-trees with roughly the same value functions are collapsed. The idea of collapsing similar sub-policy-trees was first mentioned in Cassandra (1994) and is first realized in this paper.

Thirdly, the resulting $a_t$-fixed sub-policy-tree classes that are inferior are pruned by using either direct pruning or indirect pruning. The idea of pruning inferior sub-policy-trees is the cornerstone of most of the previous methods for solving POMDPs. The use of $a_t$-fixed sub-policy-trees (or tree classes) instead of either $a_t$-rooted or $d_t$-rooted sub-policy-trees greatly improve the computational efficiency. The previous algorithms need to consider all the probability distributions over $\Omega_s$, while we need only to consider all the probability distributions over $\Omega_{s[t]}$, which is usually a much smaller set than $\Omega_s$.

Other than the above technical contributions, the introduction of the concepts of
$\alpha$-rooted, $\alpha$-fixed, and $d$-rooted sub-policy-trees provides a framework of viewing previous works.

References


Appendix: Proofs

Proof of Theorem 1: It suffices to prove the following inequalities:

\[
\max_{\phi^1} V(\phi^1) \leq \max_{\phi^1} V(\phi^1) \leq \max_{\phi^1} \nabla(\phi^1), \tag{20}
\]

\[
\max_{\phi^1} \nabla(\phi^1) \leq \max_{\phi^1} V(\delta^1) \leq \max_{\phi^1} \nabla(\phi^1). \tag{21}
\]

Comparing equation (13) with equations (9, 10), we get that for any \( t \)
\[ \nabla_t(s_t, \phi'(o_t)|o_t) \geq V_t(s_t, \phi'(o_t)|o_t). \]

Consequently, \( V(\phi^1) \leq \nabla(\phi^1) \), and hence \( \max_{\phi'} V(\phi^1) \leq \max_{\phi'} \nabla(\phi^1) \). Similarly, we can show \( \max_{\phi'} V(\phi^1) \leq \max_{\phi'} \nabla(\phi^1) \). Thus, inequality (20) is proved.

To show inequality (21), recall that the possible values for \( \phi'(o_t) \) constitute a partition of the set of possible values for \( \delta'(o_t) \). Hence for a particular \( \delta'(o_t) \), there uniquely exists a \( \phi'(o_t) \) such that \( \delta'(o_t) \in \phi'(o_t) \). Define

\[ \nabla_t(s_t, \delta'(o_t)) =_{\delta f} \nabla_t(s_t, \phi'(o_t)), \]

where \( \phi'(o_t) \) is such that \( \delta'(o_t) \in \phi'(o_t) \). Define \( V(\delta^1) \) from \( \nabla_t(s_t, \delta'(o_t)) \) in the usual way. It is evident that

\[ \max_{\delta} V(\delta^1) = \max_{\phi} \nabla(\phi^1). \]

By induction, one can show that for any \( t \)

\[ V_t(s_t, \delta'(o_t)|o_t) \leq \nabla_t(s_t, \delta'(o_t)). \]

Hence \( \max_{\delta} V(\delta^1) \leq \max_{\phi} \nabla(\phi^1) \). Consequently, \( \max_{\delta} V(\delta^1) \leq \max_{\phi} \nabla(\phi^1) \).
Similarly, we can show that $\max_{\phi^1} V(\phi^1) \leq \max_{\delta^1} V(\delta^1)$. Therefore, inequality (21) is proved. \(\square\)