A Recursive Point Filter Algorithm for the Incremental Pruning

by

MAHDI NASER-MOGHADASI, B.S.

A Thesis

In
Computer Science
Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfilment of
the Requirements for
the Degree of
MASTER OF SCIENCES

May, 2010

to my

MOTHER and FATHER

with love

Table of Contents

Abstract

Decision making is one of the central problems in artificial intelligence and specifically in robotics.In most cases this problem comes with uncertainty both in data received by the decision maker/agent and in the actions performed in the environment. One effective method to solve this problem is to model the environment and the agent as a Partially Observable Markov Decision Process (POMDP). A POMDP has a wide range of applications such as: Machine Vision , Marketing, Network troubleshooting, Medical diagnosis etc.

In recent years, there has been a significant interest in developing techniques for finding policies for (POMDPs). Most exact algorithms for a general POMDP use a form of dynamic programming in which a piecewise-linear and convex representation of one value function is transformed into another.

We consider a new technique, called *Recursive Point Filter (RPF)* based on Incremental Pruning (IP) POMDP solver to introduce an alternative method to Linear Programming (LP) filter. It identifies vectors with maximum value in each witness region known as dominated vectors, the dominated vectors at each of these points would then be part of the upper surface. RPF takes its origin from computer graphic.

First, it projects higher-dimensional vectors into 2D vectors then in each recursion, it gets two points as the start and the end points of the 2-states belief boundary. Next, it identifies vectors with maximum value in each middle, start and end belief points of the given interval. If the dominated vector of the medium point is the same with each of the dominated vectors of the start or end points; it adds the vector to the pruned list otherwise it is recursively executed with a new divided interval till termination condition is reached.

A termination condition is reached when the length of the interval boundary is less than a pre-assigned parameter δ. In this thesis, we will test RPF against several POMDP solvers such as the popular Incremental Pruning with Linear Programming filter. Our previous work on Incremental Pruning with Scan Line Point Based filter [14] is also used to measure the relative speed and quality of our new method. We show that a high-quality POMDP policy can be found in lesser time in some cases. Furthermore, RPF has a solution for several POMDP problems that LP and SCF could not converge to in 24 hours. Experiments are run on problems from POMDP literature, and an Average Discounted Reward (ADR) is computed by testing the policy in a simulated environment.

List of Tables

List of Figures

Chapter 1

Introduction

One of the most challenging tasks of an intelligent decision maker or agent is planning, or choosing how to act in such of interactions with environment. Such agent/environment interactions can be often be effectively modelled as a Partially Observable Markov Decision Process (POMDPs).Operation research [15, 23] and stochastic control [1] are two domains where this model can be applied for balancing between competing objectives, action costs, uncertainty of action effects and observations that provide incomplete knowledge about the world . Planning, in the context for a POMDP, corresponds to finding an optimal *policy* for the agent to follow.The process of finding a policy is often referred to as *solving* the POMDP. In the general case, finding an exact solution for this type of problem is known to computationally intractable [25], [8]. However, there have been some recent advances in both approximate and exact solution methods.

The value iteration algorithm for a POMDP was introduced by [23] first.

The value function V for the belief-space MDP can be represented as a finite collection of | S | - dimensional vectors known as α vectors. Thus, V is both piecewise - linear and convex [23].Although its initial success for solving hard POMDP problems, there are two distinct reasons for the limited scalability of a POMDP value iteration algorithm.The more widely reason is dimensionality [13]; in a problem with n physical states, POMDP planners must reason about belief states in an (n -1) dimensional continuous space. The other reason is, the number of distinct action - observation histories grows exponentially with the planning horizon. Pruning is one proposed[3] solution to whittle down the set of histories considered.

In some cases, an agent does not need to know the exact solution of a POMDP problem to perform its tasks. Over the years, many techniques have been developed to compute approximate solutions to POMDP problems. The goal of finding approximate solutions is to find a solution in a fast way within the condition that it does not become too far from the exact solution. Point-based algorithms [17],[24],[27] choose a subset of B of the belief points that is reachable from the initial belief state through different methods and compute a value function only over the belief points in B. After the value function has converged, the belief-point set is expanded with all the most distant immediate successors of the previous set.

PBVI and Perseus use two opposing methods for gathering the belief point sets B. In larger or more complex domains, however, it is unlikely that a random walk would visit every location where a reward can be obtained. PBVI attempts to cover the reachable belief space in a uniform density by always selecting immediate successors that are as far as possible from the B. Perseus, on other hand, simply explores the belief space by performing random trajectories. While the points gathered by PBVI generate a good B set, the time it takes to compute these points makes other algorithms more attractive.

However, approximate methods have the drawback that we cannot precisely evaluate them without knowing the exact solutions for the problems that we are solving.Furthermore, there are crucial domains that need exact solution to control accurately. For example when dealing with human's life or controlling an expensive land rover. Our objective in this thesis is to present an alternative solver to evaluate approximate solutions on POMDP problems with small number of states.

Among current methods for finding exact solutions, Incremental Pruning(IP) [3] is the most computationally efficient . As with most exact and many approximate methods, a set of linear action-value functions are stored as vectors representing the policy. In each iteration of running algorithm, the current policy is transformed into a new set of vectors and then they are filtered. The cycle is repeated for some fixed number of iterations, or until the value function converges to a stable set of vectors, The IP filter algorithm relies on solving multiple linear programs (LP) at each iteration.

Chapter 2

POMDP

2.1 Background

Planning problem is defined as: given a complete and correct model of the world dynamics and a reward structure, find an optimal way to behave.In Artificial intelligence, when the environment is deterministic our knowledge about our surrender describes with set of preconditions. In that case, *Planning* can be addressed by adding those additional knowledge preconditions to traditional planning systems [16].However in stochastic domains, we depart from the classical planning model. Rather than taking plans to be sequences of actions, which may only rarely execute as expected, we take them to be mapping from states - which are situations - to actions that specify the agent's behaviour no matter what may happen [4].

Most research in [12, 6, 28] methods develop partial policies and conditional plans for completely observable domains . These methods assume that the agent in each time steps knows where it is. However, in most real world problems, observability comes with uncertainty. POMDP is an approach to face this problem. One important factor of the POMDP is that there is no distinction drawn between actions taken to change the state of the world and actions taken to gain information by agent.

The value iteration algorithm for POMDP was introduced by [23] first. The value function V for the belief-space MDP can be represented as a finite collection of | S | - dimensional vectors known as α vectors. Thus, V is both piecewise - linear and convex [23].Although its initial success for solving hard POMDP problems, there are two distinct reasons for the limited scalability of a POMDP value iteration algorithm.The more widely reason is dimensionality [13]; in a problem with n physical states, POMDP planners must reason about belief states in an (n -1) dimensional continuous space. The other reason is, the number of distinct action - observation histories grows exponentially with the planning horizon. Pruning is one proposed[3] solution to whittle down the set of histories considered.

There are many ways to approach this problem based on checking which belief states can be reached [30] ,[9] , searching for good controllers [18] and using dynamic programming [23] [10] [13]. Most exact algorithms for general POMDPs use a form of dynamic programming in which a piecewise - linear and convex representation of one value function is transformed into another.

In some cases, an agent does not need to know the exact solution to a POMDP problem to perform its tasks. Over the years, many techniques have been developed to compute approximate solutions to POMDP problems. The goal of finding approximate solutions is to find a solution in a fast way with a condition that it does not fall too far from the exact solution.

A possible approximation is to compute an optimal value function over a finite subset B of the belief space .However an optimal solution over B does not guarantee optimality over belief points not in B. It is therefore possible that for some reachable belief states (which are not included in B), the resulting value function is suboptimal. Point-based algorithms [17],[24],[27] choose a subset of B of the belief points that is reachable from the initial belief space state through different methods and compute a value function only over the belief points in B. After a value function has converged, the belief-point set is expanded with all the most distant immediate successors of the previous set. In [27] suggests to explore the world using a random walk from the initial belief state b_0. The points that were observed during a random walk compose the set B of belief points. The Persus algorithm then iterates over these points in a random order. During each iteration backups are executed over points whose value has not yet improved in current iterations. PBVI and Perseus use two opposing methods for gathering the belief point sets B. In larger or more complex domains, however, it is unlikely that a random walk would visit every location where a reward can be obtained. PBVI attempts to cover the reachable belief space in a uniform density by always selecting immediate successors that are as far as possible from the B. Perseus, on the hand, simply explores the belief space by performing random trajectories. While the points gathered by PBVI generate a good B set, the time it takes to compute these points makes other algorithms more attractive.

Smith and Simmons [24], presented the heuristic search value iteration (HSVI) that maintains both an upper and lower bound over the value function. HSVI traverses the belief space following the upper bound heuristic, and selecting successor belief points where the gap between the bounds is the largest, until some stopping criteria has been reached. HSVI differs considerably from other point-based algorithms in collecting belief points after each iteration and also these collected points depend on the computed value function.

Recently,Shani [22], suggested the forward search value function (FSVI) algorithm. FSVI uses ideas from HSVI, such as traversing the belief space following a heuristic and executing backups in a reversed order. The FSVI heuristic for traversing the belief space relies on an optimal Q function for the underlying MDP.Aside from point-based approaches, a second dominant method is the computation of a policy directly without a value function through the use of finite state controller.Another different approach is using compression techniques to create a smaller model and solving the compressed POMDP instead of the larger one [19].One of the newest method is the use of bounded on-line search in the belief space, with heuristic function to decide which action to execute in real time [21].

However, approximate methods have the drawback that we cannot precisely evaluate them without knowing the exact solutions for the problems that we are solving.Furthermore, they are crucial domains that need exact solution to control accurately. For example when dealing with human's life or controlling an expensive land rover. Our objective in this thesis is to present an alternative solver to evaluate approximate solutions on POMDP problems with small number of states.

Basically, numerous exact algorithms to POMDP have their different ways of finding solution vector set [2] [31]. Most of them start by constructing a finite representation of piece-wise linear convex function over the belief space, then iteratively updating this representation, expanding horizon.In contrast to the approximate technique, exact solvers perform updates for the entire belief space B, and use dynamic programming approach to update their value functions.

Among current methods for finding exact solutions, Incremental Pruning(IP) [3] is the most computationally efficient. As with most exact and many approximate methods, a set of linear action-value functions are stored as vectors representing the policy. In each iteration of running algorithm, the current policy is transformed into a new set of vectors and then they are filtered. The cycle is repeated for some fixed number of iterations, or until the value function converges to a stable set of vectors, The IP filter algorithm relies on solving multiple linear programs (LP) at each iteration. Based on experience, it is known that the quality and speed of the POMDP solver depends on the speed and numerical stability of the LP solver that it uses.It was for this propose that we introduce RPF. RPF selects middle point in a given interval of the belief space recursively, it identifies a vector with the maximum value in each witness region known as a dominated vector.

2.2 Partially Observable Markov Decision Problem

The Partially Observable Markov Decision Problem is a framework that models interactions between an agent and a stochastic partially observable environment [26]. It can be denoted as a tuple $(S, A, R, P, \gamma, Z, 0)$ where

- S represents the set of all possible states.
- A the set of all possible actions.
- $R(s, a)$ is the expected reward after performing action $a \in A$ when the agent is in state $s \in S$.
- $P(s' \mid s, a)$ is the transitional probability to move from one state $s \in S$ to another state $s' \in S$ after performing an action $a \in A$.
- γ is the discount factor which controls the importance of previous rewards in the current computation.
- Z is the set of all possible observations.
- $O(z \mid s)$ is the probability of observing $z \in Z$ in a state $s \in S$.

b can also be defined as a vector of length $\mid S \mid$ specifying a probability distribution over hidden states. The elements of this vector $b(i)$, specify the conditional probability of the agent being in the state s_i, given the initial belief b_0 and the history experienced so far.[11]. This probability distribution represents the agent's *belief that state s is the true state, for all $s \in S$. The* *probability distribution b is commonly referred to as the agent's* belief state. Stated another way, a belief state b is a probability distribution over every state $s \in S$, such that $\sum_{s \in S} b(s) = 1.0$. Based on b, an action $a \in A$ is taken which causes the state to change from the current state s to some state s' according to the state transition probability $P(s' \mid s, a)$. The agent may then receive an observation z. After an action a has been taken and an observation z has been made, the agent updates its current belief b (with the initial belief denoted as b_0) to b'. This new belief is computed using Bayes rule:

$$\begin{aligned} b'(s') &= p(s' \mid z, a, b) \\ &= \frac{1}{p(z \mid a, b)} p(z \mid s') \sum_{s \in S} p(s' \mid a, b, s)\, p(s \mid a, b) \\ &= \frac{1}{p(z \mid a, b)} p(z \mid s') \sum_{s \in S} p(s' \mid a, s)\, b(s) \end{aligned} \tag{2.1}$$

The agent's goal is to select actions that will maximize its long term reward, in other word finding the optimal actions. The agent chooses the optimal actions by using a policy. The policy can be expressed as a mapping from a set of beliefs into actions, denoted by $\pi : \beta \rightarrow A$.

For POMDP problems, a policy has an associated value function, which is defined as the expected future discounted reward the agent will accumulate by following the policy starting from belief b [26]. A value function V is a mapping from the set of beliefs to real numbers denoted by $V : \beta \rightarrow \mathbb{R}$. Its value is retrieved from the following equation $V(b) = max_k \sum_{i=1}^{|S|} v_i^k p_i$ where coefficients of the k^{th} linear function is determined by a $|S|$-dimensional vector $V = [v_0, v_1, \ldots, v_{|S|}]$, and p_i denotes the i^{th} parameter of the belief distribution b [26]. Exact solution methods for POMDPs take advantage of the fact that value functions for belief are piecewise-linear and convex, and thus can be represented using a finite number of hyperplanes in the space of beliefs [23].Computing the next value function estimate is looking one step deeper into the future and it requires taking all possible actions the agent can take and all subsequent observations it may receive. This leads to an exponential growth of vectors with the planning horizon. For a more complete introduction to POMDP please see [29].

The set of linear functions, when plotted, forms a piecewise linear and convex surface over the belief space. Finding the exact optimal policy for a POMDP requires finding this upper surface. At any horizon t, the corresponding value function can be calculated recursively with the Bellman optimality equation:

$$V_t(b,a){=}max_a\left[R\left(b,a\right)+\beta\sum_z V_{t-1}(z\mid a,b)p\left(z\mid a,b\right)\right] \tag{2.2}$$

where

$$Reward(b,a)=\sum_{i=1}^{N}p_iR(s_i,a) \tag{2.3}$$

Value iteration is one standard algorithm used to provide solution to a POMDP problem, it associates values to probability distribution over states.Unfortunately, exact value iteration is intractable for most POMDP problems with more than a few states, because the size of the set of hyperplanes defining the value functions can grow exponentially with each step. As with most existing filtering techniques, RPF prunes vectors that are not part of the solution during every

Algorithm 1 POMDP Solver

```
1: $\Upsilon$ = (0;0,...,0)
2: for $\tau = 1$ to $T$ do
3:    $\Upsilon$' = $\emptyset$
4:    for all (a'; $v_1^k, \ldots, v_n^k) \in \Upsilon$ do
5:       for all control actions $a$ do
6:          for all measurements $z$ do
7:             for $j = 1$ to N do
8:                $v_{a,z,j}^k = \sum_{i=1}^N v_i^k p(z|s_i) p(s_i|a, s_j)$
9:             end for
10:         end for
11:      end for
12:   end for
13:   for all control actions a do do
14:      for all k(1),...,k(M)=(1,...,1) to ($|\Upsilon \ldots, |\Upsilon|$) do
15:         for $i = 1$ to N do
16:            $v_i' = \gamma \left[ r(s_i, a) + \sum_z v_{a,z,i}^{k(z)} \right]$
17:         end for
18:         $(a; v_1', \ldots, v_N')$ to $\Upsilon$'
19:      end for
20:   end for
21:   optional filtering method to prune $\Upsilon$'
22:   $\Upsilon = \Upsilon'$
23: end for
24: return $\Upsilon$
```

Algorithm 2 policyPOMDP($\Upsilon, b = (p_1, \ldots, p_N)$):

```
1: $\hat{u} = \underset{(a;v_1^k,\ldots,v_N^k)\epsilon\Upsilon}{\arg\max} \sum_{i=1}^{N} v_i^k p_i$
2: return $\hat{u}$
```

iteration, until the solution converges. This filter operation appears on line 21 of Algorithm 1. Without filtering, the number of vectors in the set Υ increases exponentially on each iteration by a factor of $|S| \times |A| \times |Z|$. Therefore failing to filter at this stage can exponentially increase the number of vectors to take into consideration to find the policy π. Also, it is worth mentioning that this filtering step is the most important factor affecting the speed of solving POMDPs.

Algorithm 1 calculates the value, as a linear function over the belief space, of taking action a' after taking an action a and performing observation z (line 8), for all states, all actions, and all observations. Every combination of an action and observation will define one linear function [29]. This algorithm computes value functions recursively, so every linear function depends on the previous values. Also, the number of the resulting value functions in line 16 grows exponentially, since every observation is combined with every possible previous function. This exponential growth is what makes POMDP problems intractable in the general case.

Algorithm 2 shows how the value piecewise linear value function is used as the optimal policy. Note that in practice each vector would be "tagged" with the appropriate action, or one would maintain a set of vectors for each action. Besides slowing convergence, Algorithm 1 failure to filter would make selecting actions in Algorithm 2 very slow as well. This has motivated researchers to find new insights to efficiently reduce the number functions on every iteration. The most commonly used filtering technique for finding exact solution, as mentioned earlier, relies on linear programming, which has a disadvantage of being computationally demanding and which may take a long time to converge. This is the reason why we introduce a new filtering technique, which can open up new ways to speed this computation in the future, without using LP packages.

2.3 Pruning

A key source of complexity is the size of the value function representation, which grows exponentially with the number of observations. Fortunately, a large number of vectors in this representation can be pruned away without affecting the values using a linear programming (LP) method. Solving the resulting linear programs is therefore the main computation in the DP update. Given a set of $\mid S \mid$-vectors A and a vector α, *witness region* defines as:

$$R(\alpha, A) = \{x \mid x \succeq 0, x \cdot 1 = x \cdot \alpha \succ x \cdot \alpha', \alpha' \in A \setminus \{\alpha'\}\};$$

This set known as *witness region* includes belief states for which vector α has the largest dot product compared to all the other vectors in A. R(α,A) is *witness region* of vector α because of any belief b can testify that α is needed to represent the picewise-linear convex function given by A $\cup\{\alpha\}$. Having definition of R, Purge function defines as :

$$purge(A) = \{\alpha \mid \alpha \in A, R(\alpha, A) \neq \phi\}; \tag{2.4}$$

It is set of vectors in A that have non-empty witness regions and is precisely the minimum-size set for representing the piecewise-linear convex function given by A.[13]. We can also consider it as *pruning* or *filtering*. With *filtering* those useless vectors in the sense that their witness region is empty are pruned. Since only the vectors that are part of the upper surface are important, discovering which vectors are dominated is difficult. In Incremental Pruning algorithm, linear programs are used to find a witness belief state for which α vector is optima, thus the vector is a part of the upper surface. However , linear programs degrade performance considerably. Consequently, many research efforts have focused on improving the efficiency of vector pruning. The pruning happens at three stages of DP update, namely the projection stage, the cross-sum stage, and the maximization stage.The incremental pruning (IP) algorithm was designed to address this problem by interleaving the cross-sum and the pruning [3].

2.4 Use Vector Pruning in Dynamic Programming

Note that the computation of **V** in Equation 2.2 can be divided into three stages[3]:

$$V^{a,z}(b) = \frac{Reward(b,a)}{\mid Z \mid} + \beta V_{t-1}(z \mid a,b) p\,(z \mid a,b); \tag{2.5}$$

$$V^{a}(b) = \sum_{z \in Z} V^{a,z}(b); \tag{2.6}$$

$$V_t(b,a) = max_{a \in A} V^{a}(b) \tag{2.7}$$

Each of these value functions is piecewise linear and convex, and can be represented by a unique minimum- size set of vectors. We use the symbol of S_a^z, S^a, S$\prime$ $\forall$ a $\in$ **A**, z $\in$ **Z** to refer to these minimum-size sets. The $S_a^z, S^a, S\prime$ sets as defined above can be computed as :

$$S\prime = purge(\bigcup_{a \in A} S_a) \tag{2.8}$$

$$S_a = purge(\bigoplus_{z \in Z} S_a^z) \tag{2.9}$$

$$S_a^z = purge(r(b,a,z) \mid b \in \mathbf{B}) \tag{2.10}$$

where

$$r(b,a,z)(s) = \frac{Reward(s,a)}{\mid Z \mid} + \beta b(s) P(z \mid a, s\prime) P(s\prime \mid a, s) \tag{2.11}$$

Algorithm 3 SIMPLE-FILTER(w, U)

```
1: for each u ∈ $U$
2: if  $w(s) \leq u(s), \forall s \in S$ then
3:     return  $TRUE$
4: else
5:     return  $FALSE$
6: end if
```

Algorithm 4 LP-FILTER(w, U)

```
1: solve the following linear program
2: variables: d,b(s) ∀ s ∈ S
3: maximize d
4: subject to the constraints
5: b.($w$ - $u$) ≥ d, ∀ u ∈ $U$
6: $\sum_{s \in S}$ b(s) = 1
7: if  $d \geq 0$ then
8:     return  $b$
9: else
10:    return  $nil$
11: end if
```

We refer to these three stages as *maximization pruning* ,*cross-sum pruning* and *projection pruning.*

There are two tests for dominated vectors.The simplest method is to remove any vector u that is point-wise dominated by another vector w Algorithm 3.Although this method of detecting dominated vector is fast, it can only remove a small number of dominated vectors. In Algorithm 6, A linear programming method can detect all dominated vectors. Given a set of vectors W, it extracts non-dominated vectors from W and puts them in the set D. Each time a vector w is picked from W; it is tested against D using linear program listed in algorithm 4. The linear program determines whether adding w to D improves the value function represented by D for any belief state b.

Algorithm 5 BEST(b,U)

```
1: $max \leftarrow -\infty$
2: for each $u \in U$
3: if (b.$u \geq max$) or ((b.$u$=$max$) and ( $u \leq_{lex}$ w )) then
4:    $w \leftarrow u$
5:    $max \leftarrow$ b.$u$
6: end if
7: return w
```

Algorithm 6 PRUGE(b,U)

```
1: $D \leftarrow \emptyset$
2: while $W \neq \emptyset$ do
3:    $w \leftarrow$ any element in $W$
4:    if SIMPLE-FILTER($w$,$D$) = $true$ then
5:       $W \leftarrow W$ - $w$
6:    else
7:       b $\leftarrow$ LP-FILTER($w$,$D$)
8:       if $b = nil$ then
9:          $W \leftarrow W$ - $w$
10:      else
11:         $w \leftarrow$ BEST(b, $W$)
12:         $D \leftarrow D \cup w$
13:         $W \leftarrow W$ - $w$
14:      end if
15:   end if
16: end while
17: return $D$
```

If it does, the vector in W that gives the maximal value at belief state b is extracted from W using the Algorithm 5, and is added to D. Otherwise w is a dominated vector and is discarded. The symbol $\leq_{lex}$ in algorithm 5 denotes lexicographic ordering.It is worth to mention that the number of constraints in each linear program for a set of W is the size of the resulting set which in worth case can be large as $|W|$.

In the next chapter, we will present our previous research on using a Scan-

Line based filtering technique to recognize the dominated vectors. Scan-Line filter gets its origin from computer graphics.

Chapter 3

The Scan Line Technique

One of the most challenging tasks of an intelligent decision maker or agent is planning, or choosing how to act in such a way that maximize total expected benefit over a series of interactions with the environment.

Planning, in the context of a POMDP, corresponds to finding an optimal *policy* for the agent to follow. The process of finding a policy is often referred to as *solving* the POMDP. In the general case, finding exact solutions for this type of problem is known to computationally intractable [25, 8]. However, there have been some recent advances in both approximate and exact solution methods.

In some cases, an agent does not need to know the exact solution to a POMDP problem in order to perform its tasks. Often, an approximate solution is adequate. Over the years, many techniques have been developed to compute approximate solutions to POMDP problems. The goal of finding approximate solution is to find a solution very quickly with a condition that the solution is not “too far” from the exact solution, where the definition of what is too far is a problem dependent. It is for this purpose that SCF was introduced; this new technique is based on the scan line method from computer graphic. This approach does not directly imitate the scan line algorithm, which performs a vertical sweep from left to right or right to left. Instead, It starts from a uniform probability distribution, and moves on in a predefined direction.

Many approximate POMDP solvers have been developed over the years. Among these solvers are PERSEUS and PBVI. PERSEUS and PBVI are both point based solvers. They differ by the fact that PERSEUS uses randomly generated belief points instead of an increasing belief set, as in PBVI. However, the two methods use a similar technique, called backup, which can increase the number of belief points in PBVI and readjust the value of some belief points in PERSEUS at each iteration of the POMDP algorithm. Scan-Line approach differs from these two methods by generating belief points once and using them until the solution is found. Also, the belief points that are generated are quite predictable and not entirely random.

Value functions are stored in the form of vectors $v = (v_1, \ldots, v_{|S|})$ which represent hyperplanes over belief space. Beliefs can also be presented in the form of vectors $b = (b_1, \ldots, b_{|S|})$ with a condition that $b_1 + b_2 + \ldots + b_{|S|} = 1$. Given a value function, v, and a belief, b, we can calculate the reward associated with b by the following computation:

$$R = v_1 b_1 + v_2 b_2 + \ldots + x_{|S|} b_{|S|} \tag{3.1}$$

Now, assume that we have a set of value functions $V = (V_1, \ldots, V_n)$. What we need to do is, to generate a belief b and calculate the reward $R_1, \ldots, R_n$ associated with b with respect to V. The value function that generates the maximum reward, from $R_1, \ldots, R_n$, is recorded and is considered part of the solution to the policy for the current problem.

Like all approximations, the quality of the solution depends on how close it is to the true solution. In this technique, the quality of the solution is affected by the way the belief, b, is generated and how b is moved to cover most of the belief space related to the problem.

3.1 Generating the Belief

Instead of generating beliefs that scan the belief space from left to right or right to left, a different approach was taken. It generates a set of beliefs B, with the initial belief b_0, set to have equal probability of being in each existing state, $b_0 = (\frac{1}{|S|}, \ldots, \frac{1}{|S|})$, i.e. b_0 was a uniform distribution over S. Then, a number ϵ is generated with $0 < \epsilon < \frac{1}{|S|}$. To assure that the sum of the probability distribution in b is equal to 1, It moves b in the following way,

$$\begin{aligned} b = \Bigg(& \frac{1}{|S|} + (|S| - x)\epsilon, \frac{1}{|S|} + (|S| - x + 1)\epsilon, \\ & \frac{1}{|S|} + (|S| - x + 2)\epsilon, \ldots, \frac{1}{|S|} - (|S| - x + 2)\epsilon, \\ & \frac{1}{|S|} - (|S| - x + 1)\epsilon, \frac{1}{|S|} - (|S| - x)\epsilon \Bigg). \end{aligned} \tag{3.2}$$

The idea is to make sure that if it adds ϵ to x number of probabilities, then it also subtracts ϵ from another x number of probabilities. The main goal is that each addition of ϵ should be compensated by a subtraction of ϵ so that sum of probabilities equal to 1. The number of belief points that will be in the set B depends on the value of a density parameter α, where $0 < \alpha < 1$.

Algorithm 7 Belief Generator

```
1: for $\epsilon = 0$ to $\frac{1}{|S|}$ step $\alpha$ do
2:   for $k = 1$ to $|S|$ do
3:     for $l = 1$ to $|S|$ do
4:       for $i = 1$ to $|S|$ do
5:         for $j = 1$ to $|S|$ do
6:           if (l - k ) < (j - i) then
7:             continue
8:           end if
9:           for x = k to l do
10:            z[x][index] = z[x][index] - $\epsilon$
11:            reduceSum = reduceSum + $\epsilon$
12:          end for
13:          increasedStep = $\epsilon$
14:          for y = i to j do
15:            z[y][index] = z[y][index] + increasedStep
16:            increasedSum = increasedSum + increasedStep
17:          end for
18:          if increasedSum < reduceSum then
19:            z[j][index] = z[j][index] +(reduceSum - increasedSum)
20:          end if
21:          increasedSum = reduceSum = 0
22:          index = index + 1
23:        end for
24:      end for
25:    end for
26:  end for
27: end for
```

Algorithm 7 shows the algorithm that was used to generate belief points. In this algorithm, the array z is initialized with the initial belief point b_0, as

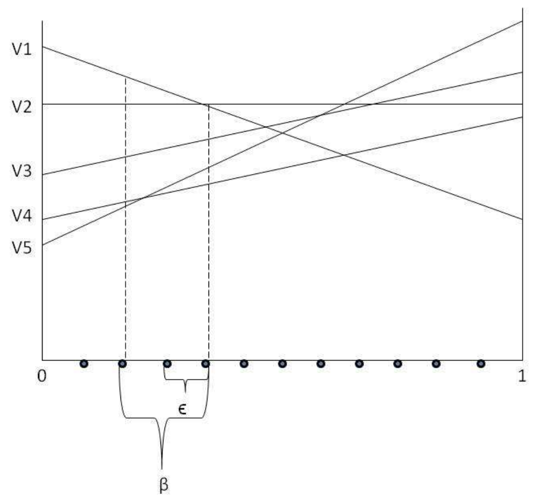

Figure 3.1: 2D Vectors in a 2-states belief space

defined above, in all its indices. Line 1 determines the maximum and minimum boundary of ϵ which are $\frac{1}{|S|}$ and 0 respectively. In line 1, α is added to ϵ on each iteration until it reaches the maximum boundary. Lines 2 through 5 specify the range of indices where values will be increased or reduced. On lines 9 to 12, ϵ is subtracted from the values in index k to l. On lines 14 to 17, ϵ is added to the values in index i to j. If the number of subtractions by ϵ exceeds the number of additions by ϵ, then SCF fixes the difference by adding the value of the number of excess times ϵ to the final index in lines 18 to 20. Belief points are only generated once. The belief set remains constant through all iterations of the POMDP algorithm.

3.2 The Scan Line

As parameter β is used to chose the belief points in B that will participate in the scanning procedure. The value of β indicates how many belief points from the current one will be skipped as the scan is performed.

After the set of belief points B is generated, as shown in figure 3.1 it takes a belief $b \in B$, calculates the expected rewards associated with b, finds the maximum, and records the corresponding vector (i.e. value function) that is associated with the maximum reward. This procedure goes on until it has exhausted all the belief points in B or the number of remaining belief points is less than the value of β or the set of value functions associated with the maximum rewards is equal to the set of value functions.

In case of a small difference between two rewards, it uses α as the deciding factor. Any differences larger than α are considered significant.

The strength of the SCF approach resides in the fact that it was able to solve two problems that LP was not able to finish within the 12 hour limit. Further experiments and tunings need be performed to exploit the strengths of this approach and to increase its speed. In the next chapter we present our approach RPF filter algorithm which is for the Incremental Pruning method. In the RPF filtering technique, hyperplanes are formed from corresponding value function vectors. RPF selects a middle point in a given interval of the 2D belief space recursively, then it identifies the dominated vectors in the both corners of the interval. If the dominated vector in the middle point is same as either the dominated vector of each boundary, it terminates recursion and extends witness region from middle point to that corner. Otherwise, it is called again with the new undiscovered interval.

Chapter 4

Recursive Point Filter (RPF)

4.1 Hyperplane

Value functions, as mentioned earlier, are represented by vectors in $\mathbb{R}^{|S|}$, where each element $s \in S$ of a vector represents the expected long term reward of performing a specific action in a corresponding state s. As the agent does not fully know the state it is currently in, so it would value the effect of taking an action depending on the belief it has for the existing states. This results in a linear function over the belief space for each vector. In other words, the value function simply determines the value of taking an action a given a belief b that determines the probability distribution over S. For example, in a simple problem with two states, if taking action a costs 100 in state s_1, and -50 for s_2, then if the agent believes 50% to be in s_1, then the value of performing a would be valued as 25. Thus, it can be formulated as mapping from the belief space to a real number, which is defined by the expected reward. This mapping can

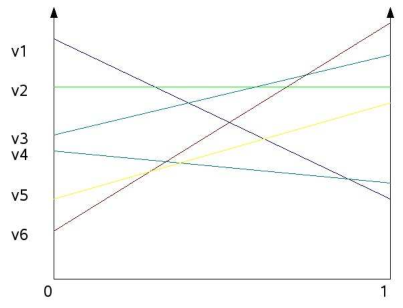

Figure 4.1: 2D Vectors in a 2-states belief space

then be described by functions of hyperplanes, which are generalized by the following equation:

$$H = \{\mathbf{x} \in \mathbb{R}^{|S|} : (\mathbf{v} \cdot \mathbf{x}) = d\} \tag{4.1}$$

with $d \in \mathbb{R}$. In the Equation 4.1, $\mathbf{v} \cdot \mathbf{x}$ denotes the scalar product between vectors $\mathbf{v}$ and $\mathbf{x}$. Each vector then, defines an affine hyperplane (4.1) that passes through these sets of values or points. From here, we refer to lines and planes as hyperplanes, and may use the three terms interchangeably.

In RPF, we begin by pruning completely dominated vectors, which means that we must eliminate any vector V_i that satisfies $V_i < V_j \ \forall b \in \beta$ and $j \neq i$. This property can hold for collinear planes or for planes that are dominated component-wise for all states. We can notice that after eliminating these

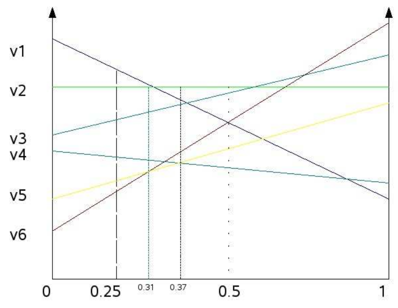

Figure 4.2: Incomplete execution of RPF over a 2-states belief space. (δ=0.05)

vectors, we are only left with intersecting planes.

4.2 RPF

4.2.1 RPF in a 2-states belief space

Since pruning eliminates the set of vectors which are not part of the final solution, it is an important part to make most of the POMDP solvers faster. As shown for instance in Figure 4.1, although there are 6 vectors in a 2-states belief space, only 4 vectors, { V1 ,V2,V3,V6 } will be included as the part of the final solution after pruning.

In this section, we will explain RPF in the 2D space where a belief state is represented by one dimensional coordinate system. We look at the problem of the filtering as identifying vectors that are part of the upper surface. In the diagram of a 2-states problem, belief space is represented by a horizontal line where each point on this line corresponds as the probability of being in the state (Let assume state is s_1), we call this value as $Belief(s_1)$. By knowing the probability of being in the state s_1, the other belief state is calculated by $1 - Belief(s_1)$. Each vector is a linear function of the belief point, and returns the expected reward given a belief state s_1 or s_1. We use this concept in the RPF later.

Algorithm 8 RPF(*InterStart*, *InterEnd*):

```
1: if ($InterEnd$ - $InterStart$) $\prec \delta$ then
2:     return
3: end if
4: $Vsrt$ ←Dominate Vector in $InterStart$
5: $Vend$ ←Dominate Vector in $InterEnd$
6: $Vmid$ ←Dominate Vector in $\frac{(InterStart + InterEnd)}{2}$
7: if $Vsrt = Vend = Vmid$ then
8:     PrunedList.$Append(Vsrt)$
9:     return
10: end if
11: if $Vsrt \neq Vmid$ then
12:     PrunedList.$Append(RPF(InterStart, \frac{(InterStart + InterEnd)}{2}))$
13: else
14:     PrunedList.$Append(Vsrt)$
15: end if
16: if $Vend \neq Vmid$ then
17:     PrunedList.$Append(RPF(\frac{(InterStart + InterEnd)}{2}, InterEnd))$
18: else
19:     PrunedList.$Append(Vend)$
20: end if
21: return PrunedList
```

In each recursion of RPF, it gets two real numbers { *InterStart*, *InterEnd* } as the inputs and returns the dominated vectors list { *PrunedList* }. *InterStart*, *InterEnd* are the starting and ending points in the interval of one dimensional belief space . In the next step, it calculates *MidPnt* as the middle point between *InterStart*, *InterEnd*. It is obvious that initially *InterStart*,*InterEnd* are set to 0 and 1 respectively. If | *InterStart* - *InterEnd* | $\prec \delta$,δ is a positive parameter set before beginning of the recursion; it exits before entering into the main loop of the algorithm as shown in Algorithm 4.2.1. Otherwise, it identifies dominated vectors within given belief intervals :{ *InterStart* to *MidPnt*} and { *MidPnt* to *InterEnd* } . We call the dominated vector with belief value of *MidPnt* as *Vmid* , *InterStart* as *Vsrt* , *InterEnd* as *Vend* respectively (lines 4-6). In the next part of the algorithm, it compares *Vmid* with *Vsrt*; if they are from the same vectors, it adds *Vsrt* into *PrunedList* (line 14), and the witness region is extended by adding the boundary of dominated vectors. In the next recursion it gets new intervals *InterStart*, $\frac{(MidPnt + InterStart)}{2}$ as new arguments for the RPF algorithm (line 12). In this way, it recognizes upper surface vectors from *InterStart* to *MidPnt*. We use the same approach for $\frac{(MidPnt + InterStart)}{2}$, *InterEnd* (lines 16-20), recursively to cover remaining part of the belief space interval.

Figure 4.2 shows an incomplete execution of the RPF for six vectors after four recursive calls. It recognizes upper surface vectors from belief points between 0 to 0.5 (where $\delta \leftarrow 0.05$ is set before the execution) for a 2-states problem. When RPF is terminated, a list of identified dominated vectors will be returned by the *PrunedList*.

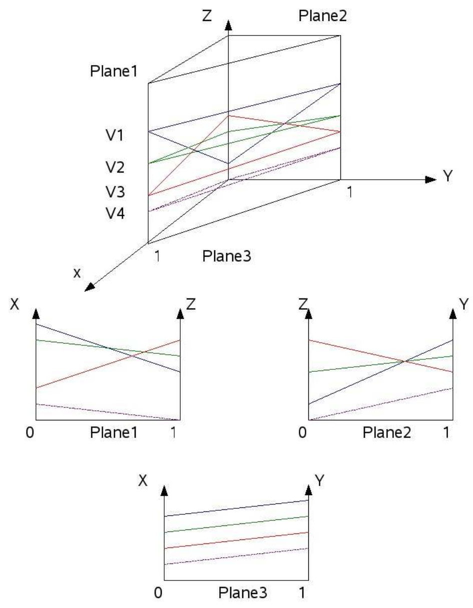

Figure 4.3: Value Function is shown as a plane in a 3-states POMDP problem

4.2.2 RPF in higher dimensions

In the 2D representation, a horizontal axis is the belief space while a vertical axis is showing the values of each vector over the belief space. Since most of the POMDP problems have more than two states; it makes value function to be represented as a set of hyperplanes instead of 2D vectors. In other words, a POMDP problem with | S | number of states makes (| S | -1) - dimensional hyperplanes for representing in the belief space . A POMDP policy is then a set of labelled vectors that are the coefficient of the linear segments that make up the value function. The dominated vector at each corner of the belief space is obviously dominant over some region [20]. As mentioned since RPF receives 2D vectors as the input arguments; the filter algorithm projects every hyperplane into 2D planes and then passes 2D vectors set to RPF algorithm. It sets zero to every component of hyperplane equations except ones that are in the 2D plane equations. Hence, the projections of hyperplanes are 2D vectors, therefore a set of 2D vectors in each plane now can be passed to RPF for filtering. There are $\binom{|S|}{2}$ possible 2D planes where | S | is the number of states. As shown in Figure 4.3, each 3D vector represents value function of one action.As indicated, after each projection, RPF gets a set of 2D vector equations and starts filtering. In filtering process, in each plane, if any of the 2D vectors is part of a 2D upper surface, its corresponding hyperplane index is labelled as dominated vector and its index will add to final pruned vectors list.

Chapter 5

Experimental Results

5.1 Empirical Results

Asymptotic analysis provides useful information about the complexity of an algorithm, but we also need to provide an intuition about how well an algorithm works in practice on a set of problems. Another disadvantage of Asymptotic analysis is that it does not consider constant factors and operations required outside the programs. To address these problems, we have run SCF,LP,RFP on the same machine which had 1.6 GHz AMD Athlon Processor with 2Gb RAM on a set of the benchmark problems from the POMDP literature. These problems are obtained from Cassandra's Online repository [2].

For each problem and method we report the following:

1. size of the final value function (|V|);

2. CPU time until convergence;

3. resulting ADR;

The number of states, actions and observations are represented by S , A and Z.

As with most exact methods, a set of linear action-value functions are stored as vectors representing the policy. In each iteration of running algorithm, the current policy is transformed into a new set of vectors which are then filtered in three stages. As mentioned in chapter 2, each stage produces a unique minimum-size set of vectors. This cycle is repeated until the value function converges to a stable set of vectors. The difference between two successive vector sets make an error bound. The algorithms are considered to have converged when the error bound is less than a threshold of convergence (For example, The ϵ parameter in LP algorithm). Our previous work[14] has shown that by setting ϵ value to 0.02; we converge to a stable set of vectors. However, a final set of vectors after convergence does not guarantee an optimal solution until the performance of the policy is considered under a simulator of the POMDP model. Changing ϵ to a higher value would lead to a non-optimal solution, and on the other hand if it is set to a lower value; it may loop between two sets of vectors because of numerical instability of solvers. Although 0.02 may not be the absolute minimum value for ϵ, but we believe that it is small enough to provide the precision for evaluating policies in the simulator.

TABLE 5.1 shows parameter values of α, β, ϵ and δ for each POMDP problem in the SCF,LP and RPF algorithms. Each of these parameters are initialized to base value depending on the quality of the final POMDP solution and the characteristic of each POMDP model. We have discussed about the weakness of the SCF parameter dependency because of the inappropriate setting of α and β in [14]. As stated above, dynamic programming update is divided into three phases in all POMDP solvers. Therefore, we have the same structure of Incremental Pruning in our solvers that are different in their filtering algorithms.

We have evaluated based on their average CPU time spent to solve each problem. All POMDP solvers were allowed to run until they converged to a solution or they exceed a 24 hours maximum running time. Previous researches on the POMDP solvers [7] , [14], [3], have shown that POMDP exact solvers for the classical test-bed problems either find a solution before 12 hours or they can-not converge. Our hypothesis is, because of the numerical instability they may oscillate between two successive iterations.

TABLE 5.2 summarizes the experiments for all three POMDP solvers.In this table RPF was compared to linear programming filtering (LP) and scan line filtering (SCF) techniques . An x on the table means that the problem did not converge under the maximum time limit set to perform the experiment; therefore we are unable to indicate how many vectors (i.e Value functions) form the final solution. The Vector column on the table indicates how many vectors form the final solution and the Time column shows average CPU time in second over 32 executions . Having higher number of vectors in the final solution and less convergence time are two major positive factors that are considered in our evaluation. We define the term *better* in our evaluation when a POMDP solver can find solution of a problem in lesser time with more final value functions (|V|) than others.

Table 5.1: α, β, ϵ and δ parameters for the SCF, LP and RPF algorithms

	SCF		LP	RPF
Problems	α	β	ϵ	δ
Tiger	0.05	1	0.02	0.02
Shuttle	0.01	5	0.02	0.02
Example	0.001	5	0.02	0.02
Network	0.001	5	0.02	0.02
Hanks	0.001	5	0.02	0.02
Saci	0.001	5	0.02	0.02
4x3	0.001	5	0.02	0.02

Table 5.2:
Experiment I: Descriptions and results presented as the arithmetic mean of 32 run-times.

				Vector			Time(second)		
Problem	$\|S\|$	$\|A\|$	$\|Z\|$	LP	RPF	SCF	LP	RPF	SCF
Tiger	2	3	2	7	9	9	4.68	8.625	9.78
Network	7	4	2	x	83	x	x	203.031	x
Hanks	4	4	2	5	9	x	1.843	3.875	x
Shuttle	8	3	5	22	35	5	44.68	101.031	977.75
Saci	12	6	5	x	43	x	x	600.938	x
4x3	16	4	2	x	436	x	x	72006.625	x
Example	2	2	3	x	4	5	x	1.9062	14.75

From the TABLE 5.2 we can see that RPF found solution on POMDP problems Network, Scai, 4x3 while SCF and LP were not able to converge to a solution before 24 hour limit. In the term of size of the final value function (|V|), RFP had more vectors than both SCF and LP in the Shuttle, and more than LP in Hanks and the Tiger problems. It also shows RPF is faster than SCF in Tiger,Shuttle and the Example but slower than LP approach in the problems that LP solved :Tiger, Hanks and Shuttle. RPF is *better* than the others in POMDP problems Network, Scai and 4x3.

5.2 Simulation

One way to evaluate the quality of policy is to run it under simulator and observe accumulated average of the discounted reward that agent received over several trials. A POMDP policy is evaluated by its expected discounted reward over all possible policy rollouts. Since the exact computation of this expectation is typically intractable, we take a sampling approach, where we simulate interactions of an agent following the policy with the environment. A sequence of interactions, starting from b_0, is called a trial. To calculate ADR, successive polices are tested and rewards are discounted, added and averaged accordingly. Each test starts from an arbitrary belief state with a given policy, Discounted reward is added for each step until the maximum steps limit is reached. The test is repeated for the number of trials. Steps are added among all the trials. The ADR is represented in the form of (mean $\pm$ confidence interval) among all tested policies. In our implementation of ADR, confidential interval is 95% .

$$\frac{\sum_{i=0}^{\#trials} \sum_{j=0}^{\#steps} \gamma^j r_j}{\#trials} \tag{5.1}$$

We computed the expected reward of each such trial, and averaged it over a set of trials, to compute an estimation of the expected discounted reward.

In this experiment, we have computed ADR for a sub-set of POMDP problems where the RPF algorithm and either LPF or SCF techniques have solutions. Since ADR values are noisy for the less number of trials, we have tested different number of trials starting with 500. After several tries, we saw that

Table 5.3: Experiment II: Average Discounted Reward

				Average Discounted Reward		
Problem	$\|S\|$	$\|A\|$	$\|Z\|$	LP	SCF	RPF
Tiger	2	3	2	20.74 ± 0.65	18.78 ± 0.6793	18.12 ± 0.696
Hanks	4	4	2	3.147 ± 0.039	x	3.178 ± 0.039
Shuttle	8	3	5	32.7116 ± 0.1064	33.05 ± 0.104	32.74 ± 0.103
Example	2	2	3	x	51.87 ± 0.16	49.92 ± 0.12

the difference between ADR means with 2000 trials and 2500 are small enough to chose 2000 as the final number of trials in our experiment.We tested such policies in the simulator with 500 steps for each POMDP problem over 2000 trials as shown in TABLE 5.3.

In general, RPF has the near close ADR values to other approaches. This implies RPF policy has a performance similar to SCF and LP for the set of problems we chose. Although LP is the winner in term of the ADR for the Tiger, but it has smaller ADR mean value for the rest of the problems than RPF. One hypothesis is size of |V| in LP policy in TABLE 5.2 for these problems is smaller than RFP; and it also shows that the value of computed ADR mean under a policy is proportional to the size of the final value function (|V|). However, the policies of SCF in the Shuttle and LP in the Tiger problem are two exceptions that with less (|V|) we have observed nearly same or better ADR mean values than with higher (|V|). We believe that it may come from characteristics of the POMDP model, Therefore since these values are nearly close to each other further experiments need to be done to prove our guesses.

Chapter 6

Conclusion

6.1 Discussion

We considered a new filtering technique, called *Recursive Point Filter (RPF)* for Incremental Pruning (IP) POMDP solver to introduce an alternative method for Linear Programming (LP) filter. As suggested in the original work on Incremental Pruning technique, filtering takes place in three stages of an updating process, we have followed the same structure in our implementation to have a fair evaluation with previous approaches. RPF identifies vectors with maximum values in each witness region known as dominated vectors. The dominating vectors at each of these points then become a part of the upper surface. We have shown that a high-quality POMDP policy can be found in the less time in some cases.Furthermore, RPF had solutions for several POMDP problems that LP and SCF were not able to converge in 24 hours. As discussed in the thesis, the quality of POMDP solutions of LP approach depends on the numerical stability of LP solver.Also, LP based filter requires LP libraries, which can be expensive, especially the powerful ones. Because of these reasons, we proposed the idea of filtering vectors as a graphical operator in POMDP solver. In each iteration of the algorithm, vectors that are not part of the upper-surface would be eliminated. SCF and RPF use the same concept but with different algorithmic ways. Although SCF had better performance in some POMDP problems, but it is dependent on the parameters(α,β) for each POMDP problem. It means that SCF parameters have direct effect on the quality of solution and convergence time for each POMDP problem. We discussed about the weakness of SCF with an inappropriate setting of α and β in [14]. RPF was superior on several POMDP problems while SCF was not able to converge to solutions before 24 hours or had lesser number of vectors in the final solutions.

We also included Average Discounted Reward in our evaluation for a subset of POMDP problems where the RPF, LP or SCF techniques have solutions. We tested such policies in the simulator with 500 steps for the POMDP problems over 2000 trials.The promising result is, RPF has a closer ADR mean value than other approaches. This implies RPF policy has a performance similar to SCF and LP for the set of problems we chose.

Although RPF worked better in the small classical POMDP problems, but it has poor performance on bigger sized POMDP problems such as: Mini-hall, Hallway, Mit [2]. Because of this reason and also our thesis consideration on smaller size problems, we believe that for large POMDP problems like those discussed above, approximate techniques would be a better option to choose.

If the complexity of a POMDP problem is close to our set of evaluation, then LP,with the condition that the size is close enough to the Tiger problem, is suggested. If LP is not available, or is expensive and the size of the POMDP problem is not close to the ones that LP was winner in, then RPF is recommended.

6.2 Future Works

Since SCF parameters have important rules either in finding POMDP solution or increasing the size of the final Value function (|V|), we will improve SCF with a dynamic setting parameters approach and compare results with RFP on the sub-set of the POMDP problems where RPF was a winner.

Our initial objective in this research was to present an alternative solver to evaluate approximate solutions on POMDP problems with small number of states. We are going to extend our implementation to use parallel processing over CPU nodes to test RPF on larger POMDP problems like *Hallway* to evaluate solutions of approximate techniques. We also intend to log the number of pruned vectors in each iteration of the algorithms for more consideration on how well each algorithm performs pruning on average after a large number of iterations and when the convergence threshold changes.

Appendix A

Details of some POMDP problems

In this thesis, we have tested RPF in some problems from the POMDP literature. These problems include Tiger,Network,Shuttle etc. The test data is obtained from Cassandra's POMDP file repository page. This appendix gives more detailed descriptions to some of these problems.

A.1 Tiger

This problem appears in Cassandra [4]. There is a tiger behind one of the two doors, left door and right door, and this constitutes to the two states in this problem. The agent has to avoid opening the door of the room where the tiger is. There are three actions that the agent can choose from, opening the left, opening the right door, or listening. If the agent opens the correct door, it receives a reward of 10; otherwise, it receives a penalty of -100. If the agent listens, it can locate the tiger correctly with a probability of 0.85. For example, if the tiger is on the left , the probability that the agent hears the tiger on the left is 0.85, and that it hears on the right is 0.15. The agent receives a penalty of -1 when it listens. Initially, the agent has no knowledge on where the tiger is. Thus, the initial belief state of this problem is an uniform distribution over the two possible states, that the tiger is one the left and that on the right.After opening the door, the agent receives its rewards/penalty, and the whole process starts again.

A.2 Network

This is a 7-states network monitoring problem from Littman.The initial belief state in the simulation is randomly generated. The maximum reward and penalty are 80 and -40 respectively.

A.3 Shuttle

This problem appears in Chrisman [5]. It is a shuttle docking problem, in which the shuttle has to transport supplies between two stations. The shuttle has to go from the most-recently visited station to the least recently visited station,which means, from the docked station to the other station.The shuttle receives a reward of +10 when it attaches to the right station, and a penalty of -3 if it collides with the station.The other action does not lead to any reward or penalty. The initial state is that the shuttle docked in the most-recently visited station, therefore, the initial belief state has probability of 1 to the state representing this situation. The state space of this problem consists of states that represent the relative positions of the shuttle to the most and least recently visited stations.

References

[1] Peter E. Caines. *Linear Stochastic Systems.* John Wiley, New York, New York, April 1988.

[2] A. R Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Process.* PhD thesis, Brown University, Department Of Computer Science, 1998.

[3] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental pruning: A simple, fast, exact algorithm for partially observable Markov decision processes. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence*, 1997.

[4] Anthony R. Cassandra, Leslie P. Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA, 1994.

[5] L Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach.

[6] Thomas Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann Nicholson. Planning under time constraints in stochastic domains. *ARTIFICIAL INTELLIGENCE*, 76:35–74, 1993.

[7] Aisoa Randrianasolo Eddy C.Borera and Larry Pyeatt. Intersection point based pomdp solver. submitted to ICRA2010.

[8] Judy Goldsmith and Martin Mundhenk. Complexity issues in Markov decision processes. In *Proceedings of the IEEE Conference on Computational Complexity*. IEEE, 1998.

[9] E. A. Hansen. Cost-effective sensing during plan execution. In *Proceedings of Twelfth National Conference on Artificial Intelligence.*

[10] Cheng H.T. *Algorithms for Partially Observable Markov Decision Process.* PhD thesis, University of British Columbia,British Columbia, Canada, 1988.

[11] Masoumeh T. Izadi, Doina Precup, and Danielle Azar. Belief selection in point-based planning algorithms for pomdps. In *In AI06*, pages 383–394, 2006.

[12] Jak Kirman, Ann Nicholson, Moises Lejter, and Thomas Dean. Using goals to find plans with high expected utility. In *In Proceedings of the Second European Workshop on Planning*, pages 158–170, 1993.

[13] M.L Littman, A.R Cassandra, and Kaelbling L.P. Efficient dynamic-programming updates in partially observable markov decision process. Technical report, Brown University,Providence ,RI, 1996.

[14] Aisoa Randrianasolo Mahdi Naser-Moghadasi and Larry Pyeatt. Scanline point based pomdp solver. submitted to ICRA2010.

[15] G. E. Monahan. A survey of partially observable Markov decision processes. *Management Science*, 28(1):1–16, 1982.

[16] R.C. Moore. A formal theory of knowledge and action. In J.R. Hobbs and R.C. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex, Norwood, NJ., 1985.

[17] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[18] L.K Platzman. A feasible computational approach to infinite-horizon partially-observed markov decision problems. Technical report, Georgia Institute of Technology, Atlanta,GA, 1981.

[19] P. Poupart and C. Boutilier. Valuedirected compression of pomdps, 2003.

[20] Larry D. Pyeatt and Adele E. Howe. A parallel algorithm for POMDP solution. In *Proceedings of the Fifth European Conference on Planning*, pages 73–83, Durham, UK, September 1999.

[21] Stéphane Ross and Brahim Chaib-Draa. Aems: an anytime online search algorithm for approximate policy refinement in large pomdps. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifi-*

cal intelligence, pages 2592–2598, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[22] Guy Shani, Ronen I. Brafman, and Solomon E. Shimony. Forward search value iteration for pomdps. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 2619–2624, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[23] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21:1071–1088, 1973.

[24] Trey Smith and Reid G. Simmons. Heuristic search value iteration for POMDPs. In *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[25] Matthijs T. J. Spaan. Cooperative active perception using POMDPs. In *AAAI 2008 Workshop on Advancements in POMDP Solvers*, July 2008.

[26] Matthijs T. J. Spaan. Cooperative active perception using POMDPs. In *AAAI 2008 Workshop on Advancements in POMDP Solvers*, July 2008.

[27] Matthijs T. J. Spaan and Nikos Vlassis. Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–20, 2005.

[28] Jonathan Tash and Stuart Russell. Control strategies for a stochastic planner. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1079–1085, 1994.

[29] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, editors. *Probabilistic Robotics*. The MIT Press, June 2006.

[30] R Washington. Incremental markov-model planning. In *Proceedings of TAI-96*.

[31] Nevin L. Zhang and Wenju Liu. A model approximation scheme for planning in partially observable stochastic domains. *J. Artif. Int. Res.*, 7(1):199–230, 1997.