A FAST HEURISTIC ALGORITHM FOR DECISION-THEORETIC PLANNING

BY

POON KIN MAN

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Philosophy
in Computer Science

August 2001, Hong Kong

Copyright © by Poon Kin Man 2001

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize The Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize The Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Poon Kin Man

A FAST HEURISTIC ALGORITHM FOR DECISION-THEORETIC PLANNING

by Poon Kin Man

This is to certify that I have examined the above MPhil thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

Dr. Nevin L. Zhang, Supervisor

Prof. Roland T. Chin, Head of Department

Department of Computer Science
11 August 2001

Acknowledgements

First, I would like to express my gratitude to my supervisor, Dr. Nevin L. Zhang, for his invaluable advice and guidance on my thesis research and on my MPhil study.

Also, I would like to thank Dr. James Kwok and Dr. Helen Shen for being the members of my thesis examination committee.

Thanks should also go to Anthony R. Cassandra. His code as a reference implementation for POMDP algorithms has saved me much time in writing the code for my experiments.

I also want to thank my friends for their care. They have given me an enjoyable life outside study.

Finally, I would like to thank my parents for their love, their continuing support, and their endless tolerance. Also, thank our Lord for His love and for all I have.

Table of Contents

Title Page i

Authorization Page ii

Signature Page iii

Acknowledgements iv

Table of Contents v

List of Figures ix

List of Tables xii

Abstract xiv

1 Introduction **1**
- 1.1 Thesis Overview . . . 1
- 1.2 Thesis Outline . . . 2

2 Background **4**
- 2.1 Markov Decision Processes . . . 4
  - 2.1.1 Model Formulation . . . 4
  - 2.1.2 Optimality Criteria . . . 5
  - 2.1.3 Policies . . . 7
  - 2.1.4 Value Functions . . . 7
  - 2.1.5 Value Iteration . . . 9
  - 2.1.6 Policy Iteration . . . 10
- 2.2 Partially Observable MDPs . . . 10
  - 2.2.1 Model Formulation . . . 12
  - 2.2.2 Belief States . . . 12
  - 2.2.3 Value Iteration . . . 13
  - 2.2.4 Piecewise-Linearity and Convexity . . . 15

2.2.5 Parsimonious Representations . . . 17
2.2.6 Policies . . . 18
2.2.7 An Overview on Exact Algorithms . . . 18

3 A Heuristic Algorithm for Solving POMDP 19
3.1 Introduction . . . 19
3.2 Point-Based Update Approximation . . . 20
3.2.1 Overview . . . 20
3.2.2 Backup Operator . . . 21
3.2.3 Point-Based Update . . . 22
3.2.4 Convergence . . . 23
3.2.5 Initial Vector Set . . . 25
3.2.6 Stopping Criterion . . . 26
3.2.7 Initial Grid . . . 26
3.2.8 The Whole Algorithm . . . 27
3.2.9 Time Complexity . . . 27
3.3 Experimental Results . . . 29
3.3.1 Setup . . . 30
3.3.2 Results . . . 31
3.4 Discussions . . . 32
3.5 Summary . . . 34

4 Grid Expansion Heuristics 35
4.1 Introduction . . . 35
4.2 Expansion Basics . . . 36
4.3 Grid Point Generation Heuristics . . . 37
4.3.1 Extreme Points . . . 37
4.3.2 Initial Belief State . . . 38
4.3.3 Previous/Next-Step Belief States . . . 38
4.3.4 Midpoint . . . 39
4.3.5 Random Belief State . . . 39
4.4 Grid Point Selection Heuristics . . . 39
4.4.1 Value Function Improvement . . . 40
4.4.2 Likelihood of being Encountered in Simulation . . . 42
4.5 Combining the Heuristics . . . 42
4.5.1 k-Largest Improvement Expansion . . . 43

4.5.2 Stochastic Simulation 44
4.5.3 Whole PBUA 45
4.6 Experimental Results 45
4.6.1 Setup 45
4.6.2 Effectiveness of Expansion 47
4.6.3 Performance of Using a Larger Grid 48
4.6.4 Comparison of Different Estimation Methods 50
4.6.5 Comparison of Different Generation Heuristics 54
4.6.6 Combined k-Largest Improvement Expansion 58
4.6.7 Stochastic Simulation 62
4.6.8 Comparison of Different Expansion Heuristics 62
4.7 Discussions and Summary 62

5 Possible Enhancements 70
5.1 Introduction 70
5.2 Lookahead Control 70
5.2.1 Overview 70
5.2.2 Experimental Results 71
5.2.3 Discussions 71
5.3 Sorted Asynchronous Update 74
5.3.1 Overview 74
5.3.2 Experimental Results 75
5.3.3 Discussions 75
5.4 Summary 78

6 Related Works 79
6.1 Introduction 79
6.2 Fixed-Resolution Regular Grid Approach 80
6.3 QMDP Approximation 81
6.4 Incremental Linear Function Approach 82
6.5 Hauskrecht's Grid Approach 86
6.6 Brafman's Grid Approach 87
6.7 Variable-Resolution Regular Grid Approach 88
6.8 Discussions and Summary 89

7 Conclusion **92**
7.1 Future Enhancements . . . 93

A Details of Experiments **95**
A.1 General Experiment Setup . . . 95
A.2 Notations to the Variants of PBUA . . . 96
A.3 Smaller Problems . . . 96
A.3.1 Tiger . . . 97
A.3.2 Manufacturing . . . 97
A.3.3 Network . . . 97
A.3.4 Shuttle . . . 98
A.3.5 4x3 Maze . . . 98
A.4 Larger Problems . . . 99
A.4.1 Zhang's Maze . . . 99
A.4.2 Maze20 . . . 100
A.4.3 Office . . . 101
A.4.4 Tiger-Grid . . . 102
A.4.5 Hallway and Hallway2 . . . 103
A.5 Summary of the Best Policies obtained by PBUA . . . 105

List of Figures

2.1 1-step Optimal Value Function for the Tiger Problem 17

3.1 Solving Time against Number of States 32

3.2 Reaction Time against Number of States 33

4.1 Solving Time with Different Estimation Methods in Zhang's Maze using kLI Expansion 51

4.2 Rewards Received with Different Estimation Methods in Zhang's Maze using kLI Expansion 51

4.3 Rewards Received against Solving Time with Different Estimation Methods in Zhang's Maze using kLI Expansion 52

4.4 Solving Time with Different Estimation Methods in Maze20 using kLI Expansion 52

4.5 Rewards Received with Different Estimation Methods in Maze20 using kLI Expansion 53

4.6 Rewards Received against Solving Time with Different Estimation Methods in Maze20 using kLI Expansion 53

4.7 Solving Time with Different Generation Heuristics in Zhang's Maze using kLI Expansion 55

4.8 Rewards Received with Generation Heuristics in Zhang's Maze using kLI Expansion 55

4.9 Rewards Received against Solving Time with Different Generation Heuristics in Zhang's Maze using kLI Expansion 56

4.10 Solving Time with Different Generation Heuristics in Maze20 using kLI Expansion 56

4.11 Rewards Received with Different Generation Heuristics in Maze20 using kLI Expansion 57

4.12 Rewards Received against Solving Time with Different Generation Heuristics in Maze20 using kLI Expansion 57

4.13 Solving Time using combined kLI Expansion in Zhang's Maze . . 59

4.14 Rewards Received using combined kLI Expansion in Zhang's Maze 59

4.15 Rewards Received against Solving Time using combined kLI Expansion in Zhang's Maze 60
4.16 Solving Time using combined kLI Expansion in Maze20 60
4.17 Rewards Received using combined kLI Expansion in Maze20 . . . 61
4.18 Rewards Received against Solving Time using combined kLI Expansion in Maze20 61
4.19 Solving Time using Stochastic Simulation Expansion in Zhang's Maze 63
4.20 Rewards Received using Stochastic Simulation Expansion in Zhang's Maze 63
4.21 Rewards Received against Solving Time using Stochastic Simulation Expansion in Zhang's Maze 64
4.22 Solving Time using Stochastic Simulation Expansion in Maze20 . 64
4.23 Rewards Received using Stochastic Simulation Expansion in Maze20 65
4.24 Rewards Received against Solving Time using Stochastic Simulation Expansion in Maze20 65
4.25 Solving Time using Different Expansion Heuristics in Zhang's Maze 66
4.26 Rewards Received using Different Expansion Heuristics in Zhang's Maze 66
4.27 Rewards Received against Solving Time using Different Expansion Heuristics in Zhang's Maze 67
4.28 Solving Time using Different Expansion Heuristics in Maze20 . . . 67
4.29 Rewards Received using Different Expansion Heuristics in Maze20 68
4.30 Rewards Received against Solving Time using Different Expansion Heuristics in Maze20 68

5.1 Rewards Received against Solving Time for using PBUA with and without Lookahead Control in Zhang's Maze 72
5.2 Reaction Time for using PBUA with and without Lookahead Control in Zhang's Maze 72
5.3 Rewards Received against Solving Time for using PBUA with and without Lookahead Control in Maze20 73
5.4 Reaction Time for using PBUA with and without Lookahead Control in Maze20 73
5.5 Rewards Received against Solving Time using PBUA with and without SAU in Maze20 77

5.6 Rewards Received against Solving Time using PBUA with and without SAU in Hallway2 77

6.1 Rewards Received against Solving Time using ILFA and PBUA in Zhang's Maze 84

6.2 Rewards Received against Solving Time using ILFA and PBUA in Maze20 84

A.1 Layout of 4x3 Maze 98

A.2 Layout of Zhang's Maze 99

A.3 Layout of Maze20 100

A.4 Layout of Tiger-Grid 102

A.5 Layout of Hallway 103

A.6 Layout of Hallway2 104

List of Tables

2.1 Value Iteration for MDP (Value Iteration) 10
2.2 Policy iteration for MDP (Policy Iteration) 11
2.3 Value Iteration for POMDP (Value Iteration) 15

3.1 Procedure for the Backup Operator (Backup) 22
3.2 Value Iteration in PBUA (PBVI) 28
3.3 Basic Version of PBUA (Basic PBUA) 28
3.4 Experimental Results (Time) of Basic PBUA 31
3.5 Solving Time for two Exact Algorithm Reported by Zhang and Zhang[19] 31
3.6 Experimental Results (Reward) of Basic PBUA 33

4.1 Procedure for Adding a New Grid Point (Add Grid Point) . . 37
4.2 Procedure for k-Largest Improvement Expansion (kLI Expansion) 43
4.3 Procedure for Stochastic Simulation Expansion (SS Expansion) 44
4.4 Pseudo-code for PBUA (PBUA) 45
4.5 Experimental Results for Comparing PBUA With and Without Expansion 48
4.6 Performance of PBUA using Random Expansion with Extreme Points 49

5.1 Procedure for Value Iteration with SAU (PBVI-SAU) 76

6.1 Performance of QMDP Approximation 81
6.2 Comparison between QMDP and PBUA 82
6.3 Properties of the Value Function of ILFA after 30 Update Cycles . 85
6.4 Experimental Results of Hauskrecht's Grid Approach in Maze20 . 87
6.5 Quality of Policies Computed by Brafman's Grid Approach in Tiger-Grid 88

A.1 Properties of some Smaller Problems 97
A.2 Properties of some Larger Problems 99

A.3 Results in Maze20 in the Literature 101
A.4 Results in Office 102
A.5 Results in Tiger-Grid in the Literature 103
A.6 Results in Hallway and Hallway2 in the Literature 104
A.7 Summary of the Best Policies obtained by PBUA 105

A FAST HEURISTIC ALGORITHM FOR DECISION-THEORETIC PLANNING

by Poon Kin Man

Department of Computer Science
The Hong Kong University of Science and Technology

Abstract

Markov decision process is usually used as an underlying model for decision-theoretic planning. It models the world as a number of states and represents the uncertainty in effects of actions by transition probabilities. It also incorporates the idea of utility by specifying the reward or cost of taking an action at a certain state. Partially observable Markov decision process (POMDP) extends this model and allows modeling incomplete information about the current state.

POMDP is computationally more complex due to the uncertainty of the current state. Exact algorithms have been proposed, but they are capable of solving POMDPs with only small number of states. Research efforts have been put into alleviating this situation mainly using two approaches: (1) using heuristic algorithms to find approximate solutions, and (2) factorizing the state space into a more compact representation.

This thesis focuses on the first approach and proposes a fast heuristic algorithm, called point-based update approximation (PBUA). Experimental results show that PBUA allows solving POMDPs in much shorter time than exact algorithms, without compromising the quality of the solution much. With the

addition of an expansion mechanism, it gives policies that are of similar quality as the optimal policies found by exact algorithms in the smaller POMDP problems. In the larger problems, which are out of the capability of the exact algorithms, PBUA computes policies among the best found in the literature, with solving times not worse than most of the other heuristic algorithms.

Chapter 1

Introduction

1.1 Thesis Overview

Planning refers to finding a sequence of actions to achieve a certain goal. In classical AI planning research, the effects of actions are deterministic and the decision making agent has complete knowledge of the world. However, in many real-world problems the effects of actions are uncertain. This leads to the research of *decision-theoretic planning*, in which probability theory is used to represent the uncertainty and utility theory is used to represent the preference between different world states.

Markov decision process (MDP) is usually used as an underlying model in decision-theoretic planning [14, 1]. It models the world as a number of states and represents the uncertainty in effects of actions by transition probabilities. It also incorporates the idea of utility by specifying the reward or cost of taking an action at a certain state. One important assumption is that the probability of the outcomes of an action depends only on the current state but not the path to that state. This simplifies the solution of the model and is one of the reason why this model is usually used.

An extension to MDP is the introduction of partial observability. The extended model is named *partially observable Markov decision* (POMDP), in which the agent does not have complete information about the current state. The agent associates a probability to each of the state and makes decision based on these probabilities.

POMDP is computationally more complex than the MDP, due to the uncertainty of the current state. Exact algorithms has been proposed to solve this problem, but they can be applied to POMDPs with only a small number of states. For example, a 12-state POMDP takes about 27,000 seconds to solve using one of the fastest exact algorithm on an UltraSparc II machine [19]. Therefore, POMDP

is still not widely applied in real-world problems despite of its ability to model the problems well.

To make POMDP practical, the solving time must be dramatically reduced. Two approaches are mainly used. One approach is to use fast heuristic approximations to find a solution that is not exact but can perform well in practice [8]. Another approach is to factorize the state space into a more compact representation [2].

We focus on the first approach and propose a fast heuristic algorithm, called *Point-Based Update Approximation* (PBUA). The main concerns in considering whether a heuristic algorithm is good or not are its running time and the quality of the policy it finds. We will show that PBUA reduces the running time significantly without compromising the quality of policy and allows us to find good solutions for POMDPs that are too large for traditional exact algorithms.

1.2 Thesis Outline

In chapter 2, we briefly review the basics of MDP and POMDP. In particular, we point out the limitations of the algorithms for POMDP.

Chapter 3 exhibits the basic version of a fast heuristic algorithm and shows why it is better than exact algorithms. Experimental results are presented to confirm that this heuristic algorithm is more efficient than exact algorithms, and it can solve larger problems than exact algorithms.

The next chapter extends the basic version of our algorithm by incorporating an expansion mechanism. It also suggests some heuristics that can be used for the expansion mechanism. Experimental results are given to show that the incorporation of the expansion mechanism improves the proposed heuristic algorithm. Different expansion heuristic methods are also compared.

Chapter 5 discusses some possible enhancements to the proposed heuristic algorithm. It also includes some experimental results to see whether these suggested enhancements are useful.

The following chapter surveys some related works. The final chapter concludes the thesis and suggests a few future research directions.

Appendix A is included to show to the details of some of the experiments discussed in this thesis. A description of the experiment setup generally used can also be found there. Some of the best results in the literature is listed for

comparing the performance of the proposed heuristic algorithm along the thesis.

Chapter 2

Background

In this chapter, the MDP model is first reviewed. Our discussion is then extended to the POMDP model. Basic ideas behind the algorithms for those models are also explained along with discussions. This lays the foundation for discussions in the subsequent chapters.

2.1 Markov Decision Processes

In a *Markov decision process* (MDP), the world is represented by a number of states. The agent in this MDP can take a number of actions, after which the world state may change from one to another. Different actions may lead to different state transitions, and these transitions are non-deterministic. This uncertainty is modeled by probability distributions over the possible next states of taking actions from the current state. Reward values are associated to the transitions, to represent different preference over the outcomes.

A main feature of MDP is its *Markov property*, which holds if the transition probabilities from any states to other states depend only on the current state and not on the previous history. This property simplifies the decision process by allowing one to make decision based on only the current state, without the consideration of the previous history. In addition to that MDP nicely models the uncertainty of the effects and preference over the outcomes, the simplification by the Markov property makes MDP an usual underlying model in decision-theoretic planning [14, 2] and reinforcement learning [10, 17].

2.1.1 Model Formulation

A MDP model can be described by four components:

- a finite set $\mathcal{S}$ of world states;

- a finite set $\mathcal{A}$ of possible actions;
- a state transition function $T : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ which maps a state-action pair to a probability distribution over the entire state space; and
- a reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \Re$ which maps a state-action-state tuple to a real number that represents the immediate reward/cost.

The transition probability of resulting in state $s' \in \mathcal{S}$ after taking action $a \in \mathcal{A}$ in the current state $s \in \mathcal{S}$ is denoted as $P(s'|s,a)$ or $T(s,a,s')$. Also, $R(s,a,s')$ denotes the immediate reward r of resulting in state $s' \in \mathcal{S}$ after taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. Note that costs are represented by negative rewards.

When computing the solution, the expected reward $\rho(s,a)$ of taking action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ is usually used. $\rho(s,a)$ can be computed by

$$\rho(s,a) = \sum_{s' \in \mathcal{S}} R(s,a,s')P(s'|s,a). \tag{2.1}$$

As mentioned before, the Markov property holds in MDP models. This means that the reward and state transition do not depend on the history. This property is put formally here:

Definition 2.1 *(Markov property) Markov property holds if*

$$P(s_{t+1}, r_{t+1}|s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \ldots, s_0, a_0, r_0) = P(s_{t+1}, r_{t+1}|s_t, a_t), \tag{2.2}$$

where s_t, a_t, r_t are the state, action taken, and immediate reward received at time t.

We can see that since the next state s_{t+1} and the immediate reward r_{t+1} depends only on the current state s_t and the action to be taken a_t. Therefore when we have to choose the action $a \in \mathcal{A}$, we only have to consider which state s we are in, without considering the history that leads us to s.

2.1.2 Optimality Criteria

To evaluate the performance of an agent, we need a criterion of optimality that the agent tries to achieve. Three criteria of optimality are commonly used.

The first criterion is usually used in a *finite-horizon model*, in which the optimality is measured by the expected cumulative rewards after acting a finite number of steps T,

$$E\left[\sum_{t=0}^{T} r_t\right], \tag{2.3}$$

where r_t is the immediate reward received at time step t. The finite-horizon model is not very practical, because the number of steps T are seldom known exactly beforehand.

The second criterion measures the *expected future discounted reward*, meaning that the optimality is measured by the expected discounted cumulative rewards over the remaining steps. In a finite-horizon problem, the optimality is measured by

$$E\left[\sum_{t=0}^{T} \gamma^t r_t\right], \tag{2.4}$$

where a discount factor $0 \leq \gamma \leq 1$ is used to indicate the trade-off between the nearer rewards and more distant rewards. As mentioned before, T may not be known usually. We can use an *infinite-horizon* to model problems with unknown T. The optimality becomes to be measured as $T \to \infty$,

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{2.5}$$

where $0 \leq \gamma < 1$. The discount factor is essential for the existence of the above limit. It can also be used to model a process that terminates with probability $1 - \gamma$ that at any step t.

The final criterion commonly used measures the *average reward*, which means the optimality is measured by the expected long-run average reward,

$$\lim_{T \to \infty} E\left[\frac{1}{T}\sum_{t=0}^{T} r_t\right] \tag{2.6}$$

In this thesis, the criterion measuring the expected future discounted reward is used to evaluate the optimality of the a policy.

2.1.3 Policies

A policy prescribes what action an agent should take. Thanks to the Markov property, the policy depends on only the current state and not history of states. Therefore, a policy $\pi : \mathcal{S} \to \mathcal{A}$ tells the agent to take an action $a \in \mathcal{A}$ given the current state $s \in \mathcal{S}$.

However, a policy may depend on the number of remaining time step t or not. In a finite-horizon problem, the total number of time steps T is finite. Obviously, a good policy should depend t, since the agent should choose an action that leads to a faster but maybe smaller reward when t is smaller, while it should choose an action that leads to a larger but maybe slower reward when t is larger. Thus, policies used in this model usually depend on t. In an infinite-horizon problem, T is unknown and is assumed to be infinite. The optimal balance between the amount of reward to be received and the amount of time to get the reward is represented by the discounted factor γ. As γ is constant, policies used in this model usually do not depend on t, which is assumed to be infinite and is fixed anyway.

A *stationary* policy π is a policy that does not depend on the number of remaining steps t. The same policy is used over time and is usually used in problems with infinite horizon. In contrast, a *non-stationary* policy, π_t, is a policy that depends on t. It prescribes an action based on the current state, with the consideration that there are t steps remaining. Non-stationary policy is usually used in problems with finite horizon.

2.1.4 Value Functions

The agent usually prefers some world states over the others, since some states likely lead to greater rewards while some do not. A *value function* represents this preference and shows the value of being in a certain state. This value function may help to find a better policy by choosing an action that likely results in a state with a higher value over an action that likely results a lower value . It may also be used to evaluate a policy by comparing the values of the states given by one policy to those given by another policy.

Formally, a value function $V : \mathcal{S} \to \Re$ maps a state $s \in \mathcal{S}$ to a real number value. Given a policy π, the value $V_\pi(s)$ is the expected cumulative rewards of starting from state s and executing π.

Finite Horizon

In a finite-horizon problem, non-stationary policy is usually used. Given a non-stationary policy π_t, the value $V_{\pi,t}(s)$ of state s, which is the expected cumulative rewards received from executing π_t for the remaining t steps and starting from state s, can be computed recursively by

$$V_{\pi,t}(s) = \rho(s, \pi_t(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi_t(s)) V_{\pi,t-1}(s'), \tag{2.7}$$

where $V_{\pi,0}$ is defined to be 0, meaning that $V_{\pi,1}(s) = \rho(s, \pi_1(s))$. This t-step value of being in state s and executing policy π_t is the sum of the expected immediate reward and the discounted expected $(t-1)$-step value of the next state s'.

Infinite Horizon

In an infinite-horizon problem, the number of time steps remaining is infinite. Consider the subscript t in equation 2.7. Since $\lim_{t\to\infty} t - 1 = \lim_{t\to\infty} t$, these subscripts can be dropped. This shows the the value function does not depend on time t, and explains why a stationary policy is usually used in the infinite-horizon problem. Similarly to equation 2.7, the value function $V_\pi(s)$ of a stationary policy π is computed recursively by

$$V_\pi(s) = \rho(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V_\pi(s'). \tag{2.8}$$

Optimal Value Functions

In decision-theoretic planning, we would like to find the best policy in a problem. A policy π is defined to be better than or equal to another policy π' if it has an expected reward greater than or equal to that of π' for all states. In other words, $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in \mathcal{S}$. An optimal policy is the one that is better than or equal to all other policies. The optimal policy and its corresponding value function are denoted as π^* and V^* respectively. The optimal policy should choose the action that gives the best expected value of the subsequent state. Thus, the optimal value of a state is the maximum value that

any action can gives. The optimal value function can then be found by

$$V_t^*(s) = \max_{a \in \mathcal{A}} \left\{ \rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_{t-1}^*(s') \right\} \tag{2.9}$$

in a finite-horizon problem and by

$$V^*(s) = \max_{a \in \mathcal{A}} \left\{ \rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s') \right\} \tag{2.10}$$

in an infinite-horizon problem.

Q-Value Functions

A *Q-value function*, $Q : \mathcal{S} \times \mathcal{A} \to \Re$, maps a state-action pair to a real number. This Q-value function is also called *action-value function*. As the latter name shows, it gives the expected cumulative rewards of taking an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ and executing a policy π afterwards. The Q-value of this state-action pair (s, a) is denoted as $Q_\pi(s, a)$. The Q-function in a finite-horizon problem can be computed by

$$Q_{\pi,t}(s,a) = \rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_{\pi,t-1}(s'). \tag{2.11}$$

The counterparts of it in an infinite-horizon problem and for the optimal case are

$$Q_\pi(s,a) = \rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V_\pi(s') \tag{2.12}$$

and

$$Q^*(s,a) = \rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^*(s'). \tag{2.13}$$

2.1.5 Value Iteration

The optimal value function can be obtained by using dynamic programming and equation 2.10. The algorithm using this approach is shown in table 2.1. This method is called *value iteration*, since it computes the optimal value function iteratively by improving an sub-optimal value function continuously. This procedure stops until the maximum difference among all states in $\mathcal{S}$ between value functions obtained in two consecutive steps is smaller than a threshold ϵ. This

VALUE ITERATION(ϵ)
Input: A stopping threshold ϵ
Output: An approximate to optimal value function V^* with Bellman error less than ϵ

Set $V(s) = 0$ for all $s \in \mathcal{S}$
repeat
  $V'(s) \leftarrow V(s)$ for all $s \in \mathcal{S}$
  foreach $s \in \mathcal{S}$
    $V(s) \leftarrow \max_{a \in \mathcal{A}} \{\rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V(s')\}$
until $\max_s |V'(s) - V(s)| \leq \epsilon$
return V

Table 2.1: Value Iteration for MDP (VALUE ITERATION)

difference is known as *Bellman residual*.

Once the optimal value function is given, the optimal policy π^* can be found by selection the action that gives the best expected value averaging over the next state. In other words,

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} \left\{ \rho(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a)V^*(s') \right\} \tag{2.14}$$

2.1.6 Policy Iteration

The optimal policy and its value function can also be found by improving the policy iteratively instead of the value function. A policy is evaluated by a value function using equation 2.8, and the policy is then improved by selecting the actions which gives the best values according to that value function. The process stops until the policy remains unchanged for two consecutive steps. The complete procedure is shown in table 2.2.

2.2 Partially Observable MDPs

The model we discussed in the last section can also be called *fully observable Markov decision process* (FOMDP), because it assumes that the state information is fully accessible. However, in many cases, there is uncertainty in the state

Policy Iteration()
Input: A stopping threshold ϵ
Output: An optimal policy π^* and its value function V^*

Set $V(s) = 0$ and $\pi(s)$ to an arbitrary $a \in \mathcal{A}$ for all $s \in \mathcal{S}$

1. Evaluate policy π.
   repeat
      $V'(s) \leftarrow V(s)$
      foreach $s \in \mathcal{S}$
         $V(s) \leftarrow \rho(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s))V(s')$
   until $\sup_s |V(s') - V(s)| \leq \epsilon$

2. Improve policy π
   $\pi'(s) \leftarrow \pi(s)$ for all $s \in \mathcal{S}$
   foreach $s \in \mathcal{S}$
      $\pi(s) \leftarrow \arg\max_{a \in \mathcal{S}} \{\rho(s, a) + \sum_{s' \in \mathcal{S}} P(s'|s, a)V(s')\}$

3. Check if optimal
   if $\pi'(s) \neq \pi(s)$ for any $s \in \mathcal{S}$
      goto step 1
   else
      return π and V

Table 2.2: Policy iteration for MDP (Policy Iteration)

information. For example, the agent is not told the exact position in a maze, or it may acquire a wrong perception with a small probability. Incomplete observability is modelled in a *partially observable Markov decision process* (POMDP). In a POMDP, the agent does not know its current state exactly. However, it can estimate its current state by taking observation after taking an action. It maintains an internal state, which is a probability distribution over the possible current states, and updates this internal state after receiving an observation. It then makes decision based on this internal state.

2.2.1 Model Formulation

Recall that as discussed previously, four components — a state space $\mathcal{S}$, an action space $\mathcal{A}$, a state transition function T, and a reward function R — are required to describe an MDP. In addition to these four components, two more components are required for a POMDP:

- a finite set $\mathcal{Z}$ of possible observations; and
- an observation function $O : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{Z})$, which maps a state-action pair to a probability distribution over the observation space.

We denote the probability of getting observation $z \in \mathcal{Z}$ after taking action $a \in \mathcal{A}$ and resulting in state $s' \in \mathcal{S}$ as $P(z|s', a)$. Note that this probability does not depend on the current state s in our model.

2.2.2 Belief States

As in FOMDP, the agent in POMDP does not maintain the whole history when it chooses an action. This history is summarized by a probability distribution over the state space $\mathcal{S}$, where a probability value is used to indicate the degree of belief that the state $s \in \mathcal{S}$ is the current state. As this distribution shows the agent's belief, it is called *belief state*, and is denoted as b. It is represented by a vector of length $|\mathcal{S}|$, with each element $b(s)$, for any $s \in \mathcal{S}$, a probability that s is the current state. Since b is a probability distribution over the state space $\mathcal{S}$, $\sum_{s \in \mathcal{S}} b(s) = 1$ and $0 \leq b(s) \leq 1$ for any $s \in \mathcal{S}$. These two conditions are called the *simplex constraints*. The belief state space the space of all belief states and is denoted as $\mathcal{B}$.

In a POMDP, the agent is not told of the current state. It can estimate the likelihood of the next state using only the observation received, and then update its belief state according to the action taken and the previous belief state. Given a current belief state b, the resulting belief state b_a^z after taking action a and receiving z is updated by

$$b_a^z(s') = \frac{\sum_{s \in \mathcal{S}} P(z, s'|s, a) b(s)}{P(z|b, a)}, \tag{2.15}$$

where $P(z, s'|s, a) = P(z|s', a)P(s'|s, a)$ and $P(z|b, a)$ is a normalizing factor given by $P(z|b, a) = \sum_{s \in \mathcal{S}} \sum_{s' \in \mathcal{S}} P(z, s'|s, a)b(s)$. The update belief state b_a^z is also denoted as $\tau(b, a, z)$ in the literature.

$P(z|b, a)$ can be treated as the probability of observing z after taking action a in belief state b, as its notation suggests. It can be computed more efficiently by

$$P(z|b, a) = \sum_{s' \in \mathcal{S}} P(z|s', a) \sum_{s \in \mathcal{S}} P(s'|s, a) \tag{2.16}$$

For convenience, the expected reward function $\rho(b, a)$ is often expressed in terms of the current belief state b, as the expected reward of taking an action a in belief state b. In fact, the belief state is only an internal state of the agent, but not of the model. Therefore, the transition of the belief states does not give any reward, but only does the transition of the actual states. However, we can average the expected reward $\rho(s, a)$ over the probability of the current state s to define $\rho(b, a)$ as

$$\rho(b, a) = \sum_{s} b(s)\rho(s, a). \tag{2.17}$$

2.2.3 Value Iteration

Suppose V_t^* is the optimal value function when there are t steps left. V_t^* can be computed using dynamic programming similar to equation 2.9. However, since we do know the exact next state, we estimate the value that we have in the next step according to the observation that we receive after the current step. V_t^* is therefore computed by

$$V_t^*(b) = \max_{a \in \mathcal{A}} \left\{ \rho(b, a) + \gamma \sum_{z \in \mathcal{Z}} P(z|b, a) V_{t-1}^*(b_a^z) \right\}, \tag{2.18}$$

where $V_0^* = 0$, b_a^z and $P(z|b, a)$ are given by equations 2.15 and 2.16 respectively.

Similarly in an infinite-horizon problem, V^* is computed by

$$V^*(b) = \max_{a \in \mathcal{A}} \left\{ \rho(b,a) + \gamma \sum_{z \in \mathcal{Z}} P(z|b,a) V^*(b_a^z) \right\}. \tag{2.19}$$

Let Γ be a set of real-valued functions $V : \mathcal{B} \rightarrow \Re$. Define an operator $H : \Gamma \rightarrow \Gamma$ as

$$HV(b) = \max_{a \in \mathcal{A}} \left\{ \rho(b,a) + \gamma \sum_{z \in \mathcal{Z}} P(z|b,a) V(b_a^z) \right\}. \tag{2.20}$$

We can see that this definition makes $V^* = HV^*$. It is well known that H is an *isotone* mapping and a *contraction* under the supremum norm, where the meaning of these two properties is given by the following definitions.

Definition 2.2 *Let U and V belong to the set of functions Γ. An isotone mapping H implies that if $U \leq V$, $HU \leq HV$.*

Definition 2.3 *Assume $||\cdot||$ is a supremum norm. A mapping H is a contraction under the supremum norm, if $||HV - HU|| \leq k||V - U||$ for all U, V and for some $0 \leq k < 1$.*

These two properties are sufficient to show the existence of $\lim_{n \rightarrow \infty} H^n V$, where $H^n V$ means applying an operator H to V for n times. It can be shown that this limit converges to V^*. Therefore, we can use dynamic programming to compute V^* by applying H iteratively to the current value function V. The algorithm for value iteration for POMDP is given in table 2.3. The Bellman error ϵ in the value iteration can be used to estimate the precision, denoted as δ, of the current value function V by the following theorem.

Theorem 2.1 *Let $\epsilon = \sup_{b \in \mathcal{B}} |V(b) - V'(b)|$, where V' is the previous-step value function of V. Then*

$$\sup_{b \in \mathcal{B}} |V(b) - V^*(b)| \leq \frac{\gamma \epsilon}{1 - \gamma} \tag{2.21}$$

and

$$\sup_{b \in \mathcal{B}} |V'(b) - V^*(b)| \leq \frac{\epsilon}{1 - \gamma} \tag{2.22}$$

hold.

Notice that this value iteration algorithm cannot be used in practice, since the belief state space $\mathcal{B}$ is continuous and all belief states $b \in \mathcal{B}$ cannot be enumerated

VALUE ITERATION(ϵ)
Input: A stopping threshold ϵ
Output: An optimal value function V^* with Bellman error less than ϵ

Initialize $V(b)$ for all $b \in \mathcal{B}$
repeat
  $V' \leftarrow V$
  Update $V \leftarrow HV'$ for all $b \in \mathcal{B}$
until $\sup_b |V'(b) - V(b)| \leq \epsilon$
return V

Table 2.3: Value Iteration for POMDP (VALUE ITERATION)

finitely. However, it shows the framework of value iteration and more details on how to handle this continuous belief space are discussed later.

2.2.4 Piecewise-Linearity and Convexity

As we can see the belief state space $\mathcal{B}$ is continuous, the value function in POMDP cannot be represented finitely as that in MDP. Along with the fact that the belief state $b \in \mathcal{B}$ cannot be enumerated finitely, this makes the value iteration algorithm given in table 2.3 impracticable.

Fortunately, Sondik shows that the optimal value function over a finite horizon can be represented by a *piecewise-linear and convex* (PWLC) function[16]. This means the value function of a POMDP can be represented by a finite number of vectors.

To explain why a t-step optimal value function V_t^*, is PWLC we first consider the case of t equal to 1. Assume we know the current state $s \in \mathcal{S}$, the value $V_{a,1}$ of taking action $a \in \mathcal{A}$ in s is equal to the expected reward $\rho(s, a)$. Since the belief state b is a probability distribution over $\mathcal{S}$, we can average $V_{a,1}(s)$ over $\mathcal{S}$ to give the value $V_{a,1}(b)$ for taking action a at the current belief state b. In other words, $V_{a,1}(b) = \sum_{s \in \mathcal{S}} V_{a,1}(s) \cdot b(s)$. We can then define a vector $\alpha_{a,1}$ with length $|\mathcal{S}|$, such that $V_{a,1}(b)$ is given by the inner product of $\alpha_{a,1}$ and b. This means, $V_{a,1}(b) = \alpha_{a,1} \cdot b$, where the elements of $\alpha_{a,1}$ is given by $\alpha_{a,1}(s) = \rho(s, a)$ for all $s \in \mathcal{S}$.

After showing the concept of a vector α, we now discuss how the V_1^* can be constructed. An optimal 1-step policy π_1^* chooses the action that gives the

maximum value at the current belief state, and the optimal value $V_1^*(b)$ equals that maximum value. Put it in another way, $\pi_1^*(b) = \arg\max_{a\in\mathcal{A}} \alpha_{a,1} \cdot b$ and $V_1^*(b) = \max_{a\in\mathcal{A}} \alpha_{a,1} \cdot b$. We can now see that, V_1^* can be represented as a vector set $\mathcal{V}_1^* = \{\alpha_{a,1}|a \in \mathcal{A}\}$, where the value of a belief state is given by the maximum inner product with the vectors. The fact that the 1-step optimal value function V_1^* is represented by a set of linear vectors and the value is given by the upper surface of these vectors explains the piecewise-linearity and convexity of V_1^*.

An example of V_1^* for a 2-state POMDP problem is shown in figure 2.1. This POMDP problem is referred as Tiger problem and is given by Cassandra[5]. In this problem, a tiger is behind either the left or right door. There are 3 actions, to open the left door, to open the right door, and to listen and make observation. More detailed descriptions of this problem can be found in appendix A. Since there are two states we can represent the value function against the belief of only one of the states s, where the belief $b(s')$ of the other state s' is given by $b(s') = 1 - b(s)$. We can see there are three vectors in the figure, representing the value of taking the three different actions. For the action open-left, the corresponding vector shows that the expected reward is 10 when $b(\text{tiger-left}) = 0$ (the tiger is behind the right door), and is -100 when $b(\text{tiger-left}) = 1$ (the tiger is behind the left door). V_1^* is the upper surface of the three vectors.

Now consider a general t-step optimal value function. Equations 2.18 and 2.20 shows that $V_t^* = HV_{t-1}^*$. It can be proved that the operator H preserves the piecewise-linearity and convexity of V_{t-1}^*. Since the optimal value function of the first time step V_1^* is PWLC, the optimal value function of V_t^* of any subsequent time step t is also PWLC. Suppose $\mathcal{V}_t^*$ is the set of vectors representing V_t^*. A vector $\alpha \in \mathcal{V}_t^*$ represents the state values of executing a t-step policy, which prescribes an action based on the belief state over a t-step horizon, in this general case.

In an infinite-horizon problem, the number of time steps T is infinite, we can use the fact that $\lim_{t\to\infty} V_t^* = \lim_{t\to\infty} H^{t-1}V_1^* = V^*$ to find the optimal value function V^*. Although the optimal value function V_t^* in a finite horizon problem for any time step t is piecewise linear, V^* is not piecewise linear in many infinite-horizon problems. However, given a sufficiently large t, we can use the piecewise linear function V_t^* to approximate any non-linear value function as closely as desired.

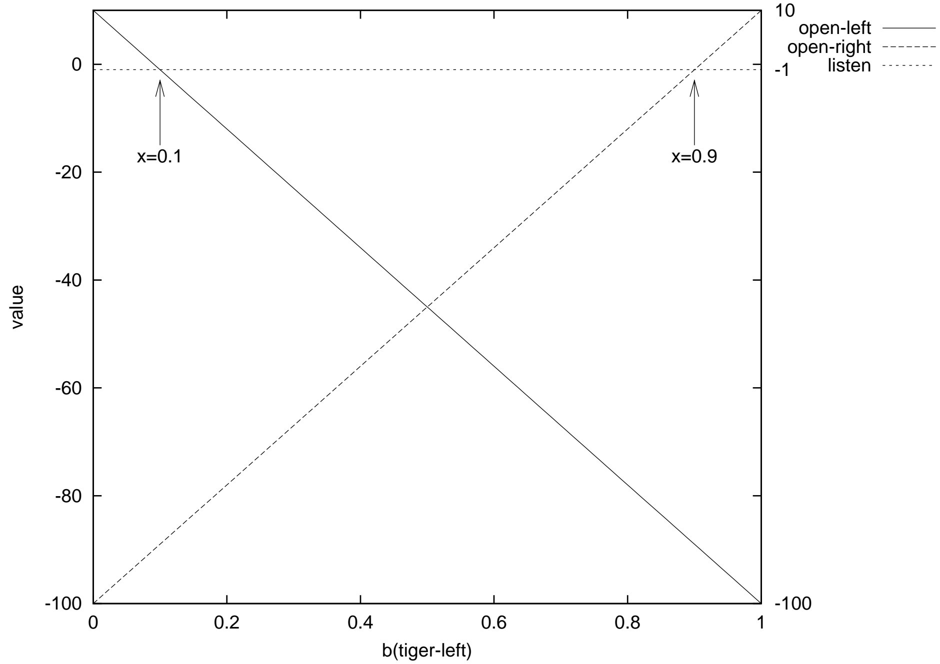

Figure 2.1: 1-step Optimal Value Function for the Tiger Problem

2.2.5 Parsimonious Representations

Assume the V^*_{t-1} is represented by a finite set of linear vectors $\mathcal{V}_{t-1}$. After applying the operator H to V^*_{t-1}, the total number of vectors in $\mathcal{V}_t$ becomes $|A||\mathcal{V}_{t-1}|^{|\mathcal{Z}|}$ in the worst case. However, some vectors generated can be pruned without changing the value function since they are dominated by the other vectors, meaning that they are located under the upper surface graphically. Those vectors that can be removed without affecting the resulting value function is *extraneous*, while the others are *useful*. A set of vectors is *parsimonious* if it does not contain any extraneous vector.

Further we can define the ideas of witness region and witness point. Given a set $\mathcal{V}$ and a vector $\alpha \in \mathcal{V}$, the witness region $R(\alpha, \mathcal{V})$ is defined by

$$R(\alpha, \mathcal{V}) = \{b \in \mathcal{B} | \alpha \cdot b > \tilde{\alpha} \cdot b, \forall \tilde{\alpha} \in \mathcal{V} \setminus \{\alpha\}\}. \quad (2.23)$$

This means the witness region $R(\alpha, \mathcal{V})$ for a vector α is where it appears at the top of the set of vectors $\mathcal{V}$. For example, $[0, 0.1)$ is the witness region for the vector corresponding to action open-left in figure 2.1. This witness region thus testifies that α is useful in $\mathcal{V}$. Any belief state that is inside $R(\alpha, \mathcal{V})$ is called a

witness point for α.

2.2.6 Policies

As the previous discussion shows, $V_t^*(b)$ can be computed by finding the best vector α' in $\mathcal{V}_t^*$ that gives the maximum inner product with belief state b. In other words,

$$V_t^*(b) = \max_{\alpha \in \mathcal{V}_t^*} \alpha \cdot b \tag{2.24}$$

The associated action of α' is used as the action prescribed by the optimal policy for the t-th step.

2.2.7 An Overview on Exact Algorithms

An *exact algorithm* finds the optimal value function in an infinite-horizon problem by approximating it using t-step optimal value function V_t^* over a finite horizon with large t.[1] It computes V_t^* by using the operator H as in table 2.3. However, as shown previously, the number of vectors can increase exponentially in an update step of H. Therefore, using all the vectors to represent the value function may make an algorithm intractable. Fortunately, many of those possible vectors are extraneous and can be pruned without affecting the value function. Thus, the main problem of the exact algorithms is how to update the value function using H only with the useful vectors to improve the time complexity.

Two approaches are usually used to increase computational efficiency. The first one is based on a generate-and-test paradigm [16, 13]. It enumerates all the possible vectors and then test their usefulness. Those extraneous vectors are then pruned. An extension to this is to interleave the generate and test stages, and do the pruning before the vectors are completely constructed [4]. The second approach is based on a search-and-generate paradigm. It searches for the witness points before constructing the vectors. It then uses those witness points to generate the vectors, so that only useful vectors are generated [9].

[1] The exact solution for a POMDP is the one that gives zero Bellman error in value iteration. However, it is usually impossible to have zero Bellman error, as it may take infinite number of iteration steps. Therefore, an exact algorithm usually stops when the Bellman error falls under a certain threshold, and this is why the solution given by an exact algorithm is usually an approximation.

Chapter 3

A Heuristic Algorithm for Solving POMDP

3.1 Introduction

The previous chapter reviews the MDP and POMDP, and gives an overview on the exact algorithms. The exact algorithms use a t-step optimal value function V_t^* to approximate the optimal value function V^*. Theoretically, we can approximate this optimal value function as closely as possible, using the piecewise linear and convex V_t^*. However, it is practicable for POMDP problems with only small state spaces, due to the high computational complexity of the exact algorithms.

The main obstacle for exact algorithms is the exponential increase in number of vectors after each update cycle in value iteration. The number $|\mathcal{V}_t^*|$ of vectors used for representing V_t^* may become $O(|\mathcal{A}|^{|\mathcal{Z}|^{t-1}})$ in the worst case, after applying the operator H for $t-1$ times by starting from V_1^*. This imposes not only a space burden but also a time burden, because after ignoring all other time needed to construct the value function, we still need at least $O(\mathcal{A}^{|\mathcal{Z}|^{t-1}})$ time for enumerating the vectors and store them. Although in practice, $|\mathcal{V}_t^*|$ could be much smaller after pruning, it could still be large enough to make an exact algorithm infeasible for larger problems. For example, using probably the fastest algorithm, it took more than 7 hours to solve a 12-state POMDP problem on an UltraSparc II machine[19]. And whether it is possible to solve a larger problem, such as one with 20 states, is still questionable. This limitation in the size of POMDP thus makes POMDP not widely applied in practice.

To overcome this obstacle, we need some other ways to solve the POMDP problem, instead of exact algorithms. Recent research has been conducted in mainly two directions. The first direction is to use some other representations to approximate V^*, instead of using the complex V_t^*. The algorithms going in this direction are referred in thesis as *heuristic algorithms*, or approximation

algorithms. For example, Lovejoy[12], Brafman[3], Hauskrecht[7, 8], and Zhou and Hansen[20] use grid-value pairs to approximate the optimal value function V^*. These algorithms update and improve the value of the grid points using value iteration. Then the values of the non-grid belief states are interpolated with the grid-value pairs, so that the value functions $\widehat{V}$ approximate to V^* are properly defined for the entire belief state space $\mathcal{B}$.

Another direction is to abstract the state space into a more compact representation. For instance, Boutilier and Poole[2] shows how to use tree-like structure to represent the state space. Then an exact algorithm is applied to solve the POMDP by exploiting this compact tree representation.

This thesis focuses on the first direction, that is, using heuristic algorithms to solve POMDP.

3.2 Point-Based Update Approximation

3.2.1 Overview

As mentioned in the previous section, alternative ways to exact algorithms should be explored to solve POMDP problems with large state space. Based on this idea, this section proposes a fast heuristic algorithm, namely *point-based update approximation* (PBUA).

PBUA is a grid-based heuristic algorithm, meaning that it uses a grid G of belief points b^G in the update steps of value iteration. However, unlike some of the heuristic algorithms listed in the previous section, PBUA uses a vector set $\mathcal{V}$ instead of grid-value pairs to represent an approximate value function $\widehat{V}$. It updates the $\mathcal{V}$ based on the search-and-generate paradigm as of exact algorithms.

PBUA reduces the time complexity of the exact algorithms in two aspects. First, in the search-and-generate paradigm used by exact algorithms, witness points are located, usually by linear programming, before the generate stage. This search can be computationally very expensive. PBUA eliminates the need of doing this search by using a fixed grid G of belief points. It treats these grid points as the witness points and generates vectors for the updated value function using them.

Second, PBUA updates the value function in a different way. This different update method allows only one vector per grid point to be added to the updated

value function. Thus, PBUA overcomes the $O(|\mathcal{A}|^{|\mathcal{Z}|^{t-1}})$ obstacle in exact algorithms by limiting $|\mathcal{V}_t|$ to only the number of grid points $|G|$, which is usually only a small multiple of the size of the state space $|\mathcal{S}|$. Obviously the grid points should have missed many witness regions, due to the relatively small number of $|G|$. However, experimental results show that PBUA is able to use the resulting value function $\widehat{V}$ to obtain good policies for the POMDP problems in practice.

The basic version of PBUA is based on two ideas found in the literature. First, the use of grid-based update for constructing a vector set to approximate an optimal value function can be traced its root back to Lovejoy[12]. The second, and more crucial, idea of using the modified update comes from Zhang and Zhang[18, 19].

3.2.2 Backup Operator

Before describing the whole algorithm, we first need to see how a vector is generated with a given witness point. We define *backup operator* as an operator that generates a new useful vector given a belief state and a vector set.

Recall the operator H defined in equation 2.19. $V = HV'$ can be written in terms of $\mathcal{V}'$ as follows:

$$V(b) = \max_{a\in\mathcal{A}} \left\{ \sum_{s\in\mathcal{S}} \rho(s,a)b(s) + \gamma \sum_{z\in\mathcal{Z}} \max_{\alpha\in\mathcal{V}'} \sum_{s\in\mathcal{S}} \sum_{s'\in\mathcal{S}} P(z,s'|s,a)\alpha(s')b(s) \right\} \tag{3.1}$$

The vector $\beta \in \mathcal{V} = H\mathcal{V}'$ that gives the maximum value with belief state b can be found in three steps:

1. For each a and z, find the vector α_a^z which gives the maximum inner product with next step belief state b_a^z given a current belief state b, action taken a, and observation received z. This can be found by

$$\alpha_a^z = \arg\max_{\alpha\in\mathcal{V}'} \alpha \cdot b_a^z. \tag{3.2}$$

2. For each action a, compute vector β_a by,

$$\beta_a(s) = r(s,a) + \gamma \sum_{z\in\mathcal{Z}} \sum_{s'\in\mathcal{S}} P(z,s'|s,a)\alpha_a^z(s). \tag{3.3}$$

BACKUP(b,$\mathcal{V}$)
Input: b is the belief point where the this backup is perform at; and $\mathcal{V}$ is the set of vectors on which this backup is performed
Output: $\beta = \texttt{backup}(b, \mathcal{V})$

$v_{a,\max} \leftarrow -\infty$ and $\beta \leftarrow \texttt{nil}$
foreach $a \in \mathcal{A}$
  foreach $z \in \mathcal{Z}$
    $\alpha_a^z \leftarrow \arg\max_{\alpha \in \mathcal{V}} \alpha \cdot b$

  foreach $s \in \mathcal{S}$
    $\beta_a(s) \leftarrow r(s,a) + \gamma \sum_{z \in \mathcal{Z}} \sum_{s' \in \mathcal{S}} P(z, s'|s, a) \alpha_a^z(s)$

  if $\beta_a \cdot b > v_{a,\max}$
    $v_{a,\max} \leftarrow \beta_a \cdot b$
    $\beta \leftarrow \beta_a$

return β

Table 3.1: Procedure for the Backup Operator (BACKUP)

3. Find the vector β that gives the maximum inner product with b among β's

$$\beta = \arg\max_{\beta_a} \beta_a \cdot b. \tag{3.4}$$

   Break ties lexicographically if there exists more than one such vector.

Now define $\texttt{backup}(b, \mathcal{V})$ as the vector β. The procedure for finding $\texttt{backup}(b, \mathcal{V})$ is given in table 3.1.

It can be shown that $\beta = \texttt{backup}(b, \mathcal{V})$ is a useful member of $H\mathcal{V}$[19].

3.2.3 Point-Based Update

As discussed before, PBUA improves the time complexity in two aspects. It saves the time spent on searching the witness points and limits the number of vectors to the number of grid points. More technical descriptions on how PBUA achieves these are given here.

Let G be a set of belief points $b^G \in \mathcal{B}$. Suppose there is a set $\mathcal{V}'$ of vectors representing the current approximate value function $\widehat{V}'$, with $|\mathcal{V}'| \leq |G|$. We associate a vector α_i' to each grid point $b_i^G \in G$, such that α_i gives the maximum

inner product with b_i^G. In other words,

$$\alpha'_i = \arg\max_{\alpha' \in \mathcal{V}'} \alpha' \cdot b_i^G \tag{3.5}$$

Note that for two different grid points b_i^G, b_j^G, where $i \neq j$, α'_i may be equal to α'_j. In our implementation, we store only one vector for both b_i^G and b_j^G, and associate this vector with both of them. However, for convenience, we use two different notations α'_i and α'_j to refer to this same vector during the discussion.

PBUA treats the grid points b_i^G as the witness points to perform update. The vectors for the updated vector set $\mathcal{V}$ are generated by performing $\texttt{backup}(b_i^G, \mathcal{V}')$ at all grid points $b_i^G \in G$. As each grid point b_i^G leads to only one backup vector β_i, the number of backup vectors is upper bounded by $|G|$.[1] Therefore, we can use all the backup vectors β_i to construct the updated vector set $\mathcal{V}$, such that we can keep $|\mathcal{V}| \leq |G|$. However, the value function updated in this way does not converge in value iteration[12].

To make it converge, PBUA constructs $\mathcal{V}$ in a different way. Instead of using β_i immediately, it compares the inner products $\beta_i \cdot b_i^G$ and $\alpha'_i \cdot b_i^G$, and uses the vector that gives a larger inner product. In other words,

$$\alpha_i = \begin{cases} \alpha'_i & \text{if } \alpha'_i \cdot b_i^G \geq \beta_i \cdot b_i^G \\ \beta_i & \text{otherwise} \end{cases}$$

where $\alpha_i \in \mathcal{V}$ is the vector associated with b_i^G after update. This update method is referred as *point-based update*.

3.2.4 Convergence

One question about using point-based update is whether this update leads to a converging value iteration. The following proof gives an affirmative answer.

Suppose V_n is value function, represented by a set $\mathcal{V}_n$ of vectors, obtained by PBUA at the n-th update. Define operator T as the update operator used in PBUA such that $V_n = TV_{n-1}$ and $\mathcal{V}_n = T\mathcal{V}_{n-1}$.

Proposition 3.1 *If $V_{n-1} \leq V^*$, $V_n = TV_{n-1} \leq V^*$.*

[1] The numbers are not necessarily equal because two grid points may lead to the same backup vector, so the number of backup vectors may be smaller than $|G|$

Proof: Assume H is the operator defined by equation 2.20. Also, suppose that

$$\mathcal{W}_n = \{\beta_i | \beta_i = \texttt{backup}(b_i^G, \mathcal{V}_{n-1}), \forall b_i^G \in G\}$$

is the set of all backup vectors generated at the n-th update step. Since

$$\beta_i = \texttt{backup}(b_i^G, \mathcal{V}_{n-1}) \in H\mathcal{V}_{n-1},$$

it follows that $\mathcal{W}_n \subset H\mathcal{V}_{n-1}$. The operator T selects either $\alpha_i \in \mathcal{V}_{n-1}$ or β_i to construct $\mathcal{V}_n = T\mathcal{V}_{n-1}$. Therefore,

$$\mathcal{V}_n \subset (\mathcal{V}_{n-1} \cup \mathcal{W}_n) \subset (\mathcal{V}_{n-1} \cup H\mathcal{V}_{n-1}).$$

Because $V_n(b) = \max_{\alpha \in \mathcal{V}_n} \alpha \cdot b$,

$$V_n(b) \leq \max\left\{V_{n-1}(b), HV_{n-1}(b)\right\}, \tag{3.6}$$

for any belief state $b \in \mathcal{B}$.

Recall that H is isotone, which means if $V_{n-1} \leq V^*$, $HV_{n-1} \leq HV^*$. Now assume $V_{n-1} \leq V^*$, this implies $HV_{n-1} \leq HV^* = V^*$. Since both V_{n-1} and HV_{n-1} in equation 3.6 are less than V^*, $V_n \leq V^*$ and the proposition follows. $\square$

Proposition 3.2 *The operator T maintains monotonicity (increasing) of the value function over the grid points. In other words,*

$$V_n(b_i^G) = TV_{n-1}(b_i^G) \geq V_{n-1}(b_i^G)$$

for all $b_i^G \in G$.

Proof: For each grid point b_i^G, the operator T chooses among β_i and $\alpha_i \in \mathcal{V}_{n-1}$ the vector that gives the larger inner product with b_i^G to construct $\mathcal{V}_n = T\mathcal{V}_{n-1}$. Moreover, due to equation 3.5, $V_{n-1}(b_i^G) = \alpha_i \cdot b_i^G$. Therefore,

$$\begin{aligned} V_n(b_i^G) &= \max\{(\alpha_i \cdot b_i^G), (\beta_i \cdot b_i^G)\} \\ &= \max\{V_{n-1}(b_i^G), (\beta_i \cdot b_i^G)\} \\ &\geq V_{n-1}(b_i^G), \end{aligned}$$

and the proposition follows. □

Proposition 3.3 *Given a fixed grid G and an initial value function $V_0 \leq V^*$, the value function updated by operator T in each step of value iteration converges on the grid points. Moreover, the value function remains the same during the value iteration after a sufficiently large number of iteration steps.*

Proof: From propositions 3.1 and 3.2 and a fixed grid G, it is obvious that values at the grid points converge, as follows:

$$\lim_{n \to \infty} V_n(b_i^G) = \lim_{n \to \infty} T^n V_0(b_i^G) = \bar{V}(b_i^G),$$

for all $b_i^G \in G$, where $\bar{V}(b_i^G)$ is a value function with values at grid point converged. Since point-based update selects a backup (new) vector β_i over the original vector α_i only when $\beta_i \cdot b_i^G > \alpha_i \cdot b_i^G$, no β_i are selected after the values at the grid points have converged. Therefore, point-based update does not change the value function in the subsequent steps and the proposition follows. □

Other than the convergence question in value iteration for PBUA, there is still another question of whether the updated value function leads to a better policy after each step. Proposition 3.2 shows that the values of the grid points are improving during the value iteration, but it does not guarantee the same thing for the non-grid points. Moreover, PBUA may not be able to find a backup vector that gives a greater inner product with a grid point. Thus the value iteration may stop too early and the resulting value function may not be good enough. Fortunately, the experimental results clear these concerns. Although we have not measured the quality of the lower bounds imposed by the resulting value functions, we can see that these value functions lead to satisfactory policies. Moreover discussions can be found in a later section when the experimental results are presented.

3.2.5 Initial Vector Set

Proposition 3.3 is based on an assumption that the initial value function V_0 is a lower bound to the optimal value function V^*. We now show how we obtain this V_0.

Recall that the value of a state is the expected future cumulative reward it receives by starting from that state and executing the optimal policy. Therefore, if all the expected rewards in the POMDP are greater than zero, these state values must be non-negative. As a result, we can use a set $\{\mathbf{0}\}$ consisting of only a zero vector as the initial vector set $\mathcal{V}_0$, such that $V_0 \leq V^*$.

When some of the expected rewards in a POMDP are negative, we apply the idea shown in Zhang and Zhang[19]. Let $m = \min_{s,a} \rho(s,a)$ be the minimal expected reward, $c = m/(1-\gamma)$ be the discounted sum of receiving m for infinite steps, and α_c be a vector of which the components are all c. We can use the singleton set $\{\alpha_c\}$ as the initial vector set $\mathcal{V}_0$, such that $V_0 \leq V^*$.

The choice of initial vector set $\mathcal{V}_0$ is summarized by as follows,

$$\mathcal{V}_0 = \begin{cases} \{\mathbf{0}\} & \text{if } \min_{s,a} \rho(s,a) >= 0 \\ \{\alpha_c\} & \text{otherwise} \end{cases}, \tag{3.7}$$

where $c = \min_{s,a} \rho(s,a)/(1-\gamma)$.

3.2.6 Stopping Criterion

We see that the value function remains the same when the values at the grid points converge. This means the value iteration can stop when the values at the grid points converge. Therefore, we use a stopping criterion that measures the maximum difference of the values among the grid points that are given by the value functions in two consecutive steps. In particular, the value iteration stops when this difference falls below a threshold ϵ,

$$\max_{b_i^G \in G} |V_n(b_i^G) - V_{n-1}(b_i^G)| < \epsilon.$$

We call the difference $|V_n(b_i^G) - V_{n-1}(b_i^G)|$ the amount of improvement δ_i at grid point b_i^G, since it shows how much the value function has improved at b_i^G.

3.2.7 Initial Grid

Proposition 3.3 shows that the value iteration produces a converging value function given that the grid is fixed. Therefore, PBUA constructs a grid initially and keeps it unchanged during the value iteration.

Other than this, PBUA has no more restriction concerning the grid. However, there are two main considerations. First, since PBUA performs backup on the grid points, the selection of them may affect the quality of the policy obtained. Second, the time complexity of PBUA depends on the size of the grid points. But at the same time, more grid points may lead to a better value function. This creates a trade-off between the amount of computational time and the quality of policy.

In our implementation, we used an initial grid consisting of only extreme points.[2] This keeps the size of grid relatively small. We also used a randomly constructed initial grid to compare the significance on the choice of grid. The experimental results are shown in a later section.

3.2.8 The Whole Algorithm

Table 3.2 gives the procedure of the value iteration in PBUA. In each update step, it backs up at all the grid points and improves the values by selecting the vector that gives a better inner product at each grid point. It stops until the maximum amount of improvement $\delta_{\max}$ for a step falls below a threshold ϵ.

The amount of improvement may oscillate before it converges to a value smaller than ϵ. Therefore, we check the maximum cumulative amount of improvement Δ for the whole value iteration and repeat a value iteration if $\Delta > \epsilon$. This whole basic version of PBUA is given in table 3.3.

3.2.9 Time Complexity

After presenting the complete algorithm, we now analyze its time complexity, and show how it is better than exact algorithms.

The time complexity of doing a `backup`$(b_i^G, \mathcal{V}_{n-1})$ is

$$O(|\mathcal{A}||\mathcal{S}||\mathcal{Z}||\mathcal{V}_{n-1}| + |\mathcal{A}||\mathcal{S}|^2|\mathcal{Z}|),$$

where the former term corresponds to the time taken to find α_a^z and the latter term to β_a as in table 3.1. This backup operator is the same for some exact algorithms. We can see that the time complexity depends on $|\mathcal{V}_{n-1}|$, can be

[2]The extreme points are those belief states that have one component with value 1 and the others 0. This implies there are only $|\mathcal{S}|$ number of extreme points for a POMDP with state space $\mathcal{S}$.

PBVI(G,$\mathcal{U}$, ϵ)
Input: G is a grid of belief points, $\mathcal{U}$ is a set of vectors before value iteration, ϵ is a stopping threshold
Output: A set of vectors $\mathcal{V}$ after value iteration, and the maximum cumulative amount of improvement $\Delta_{\max}$ for a grid point in the whole update

$\Delta_i \leftarrow 0$ and $\delta_i \leftarrow 0$ for all $b_i^G \in G$
repeat
  $\mathcal{V} \leftarrow \emptyset$
  foreach $b_i^G \in G$
    $\beta_i \leftarrow \texttt{backup}(b_i^G, \mathcal{U})$

    if $\beta_i \cdot b_i^G > \alpha_i \cdot b^G$
      $\delta_i \leftarrow \beta_i \cdot b^G - \alpha_i \cdot b^G$
      $\alpha_i \leftarrow \beta_i$
    else
      $\delta_i \leftarrow 0$

    $\mathcal{V} \leftarrow \mathcal{V} \cup \alpha_i$
    $\Delta_i \leftarrow \Delta_i + \delta_i$
  $\mathcal{U} \leftarrow \mathcal{V}$
until $\max_i \delta_i < \epsilon$
return $\{\mathcal{V}, \max_i \Delta_i\}$

Table 3.2: Value Iteration in PBUA (PBVI)

BASIC PBUA(ϵ)
Input: ϵ is the stopping threshold
Output: A vector set $\mathcal{V}$, which can be used as an estimate of the optimal value function and to obtain a policy

Construct a grid G by some heuristics, such as using random points or extreme points
Initialize $\mathcal{V}$ by equation 3.7
Set α_i to the only vector in $\mathcal{V}$ for all $1 \leq i \leq |G|$

repeat
  $\{\mathcal{V}, \Delta\} \leftarrow$ PBVI(G, $\mathcal{V}$, ϵ)
until $\Delta < \epsilon$
return $\mathcal{V}$

Table 3.3: Basic Version of PBUA (BASIC PBUA)

$O(|\mathcal{A}|^{|\mathcal{Z}|^{n-2}})$ in an exact algorithm, where the n-th step optimal value function V_n^* is used as V_n. Therefore, it shows that how much it can be faster if we limit $|\mathcal{V}_{n-1}| \leq |G|$, where $|G|$ is usually only a few times of $|\mathcal{S}|$.

In each update cycle, the `backup` operator is applied at every belief point in the grid G. Therefore it takes

$$O(|G||\mathcal{A}||\mathcal{S}||\mathcal{Z}||\mathcal{V}_{n-1}| + |G||\mathcal{A}||\mathcal{S}|^2|\mathcal{Z}|)$$

time to complete point-based update in an iteration step. Since $|\mathcal{V}| \leq |G|$, the time complexity of an update step becomes

$$O(|G|^2|\mathcal{A}||\mathcal{S}||\mathcal{Z}| + |G||\mathcal{A}||\mathcal{S}|^2|\mathcal{Z}|),$$

which is polynomial time only compared to super-exponential time of many exact algorithms.

3.3 Experimental Results

In this section, we present the some experimental results to show the effectiveness of PBUA. When judging whether a heuristic method is successful, the following criteria are considered:

1. Computational time – this measures the time used in computing a solution, which includes a policy. As the objective of using heuristic methods is to speed up the computation, this criteria is important.

2. Scalability – this is similar to computational time, but it shows how much the computational time increases with a increase in size of state space. Some algorithms may be fast with small state space, but may not when the state space is large. This depends more on the order of the polynomial terms of $|\mathcal{S}|$, $|\mathcal{A}|$, and $|\mathcal{Z}|$, but less on the constant factor in the asymptotic time. This is also an important criteria since we would like to apply the POMDP model in more complex problems.

3. Quality of policy – this measures the performance of the agent in action, where the policy found is used to prescribe actions for the agent. This performance is usually given by the average rewards received during simulation. Since a heuristic method is not an exact algorithm, it may produce

a sub-optimal policy. We do not want to compromise the quality much for the computation speed since we also want our agent to perform well.

4. Reaction time – this measures the time for an agent to decide on an action using an optimal policy or an optimal value function. Although reaction time is usually very small (much less than a second) on a general purpose computer, it may matter when it is applied in some situations other than our experiments. For example, it may be applied in robotic control, where building on low-end hardware are desired if we want to popularize its use. Moreover, in multi-agent system, a computer may need to make a decision for more than one agent at a time. Therefore, we also need to consider the reaction time, which depends mainly on how a value function is represented and how values are computed from this representation.

From the experiments presented, we look into the different aspects of PBUA to judge its level of success.

3.3.1 Setup

We have conducted experiments on a variety of POMDP problems in the literature. We used two types of initial grids, both of which has a size the same as that of the state space. The first type of grid is generated randomly, while the second type of grid consists of only the extreme points. The stopping threshold ϵ (see table 3.3) is 0.01. In all the simulations run, the maximum number of steps allowed is 100. The results shown were obtained by running 5000 trials.

The reaction time were not measured directly. Instead, we measured the total time taken to run a whole simulation and calculated the average time to complete a simulation step. Thus, this includes the time spent for simulation in addition to the reaction time. However, it can give an idea when comparing the reaction time used by different methods, since the time spent for simulation should be the same. Also, note that the all the time measured are in seconds.

Besides the description shown here, more detailed explanations of the setup of the experiments can be found in appendix A.

Problem		$Time_{solve}$		$Time_{reaction}$	
Name	$\|\mathcal{S}\|$	PBUA$_R$	PBUA$_E$	PBUA$_R$	PBUA$_E$
Tiger	2	0.030	0.030	7.3e-6	7.7e-6
Manufacture	3	0.0200	0.0200	8.8e-6	8.8e-6
Network	7	0.170	0.0901	1.6e-5	1.9e-5
Shuttle	8	0.170	0.150	1.3e-5	1.4e-5
4x3	11	0.410	0.590	1.9e-5	2.5e-5
Zhang's Maze	11	0.270	0.320	1.8e-5	2.2e-5
Maze20	20	0.360	0.670	3.5e-5	4.7e-5
Office	35	8.89	15.7	8.1e-5	1.0e-4
Tiger-Grid	36	53.2	45.5	1.3e-4	1.0e-4
Hallway	60	97.9	51.0	5.0e-4	4.8e-4
Hallway2	92	128	161	1.0e-3	1.0e-3

Table 3.4: Experimental Results (Time) of Basic PBUA

	Tiger	Network	Shuttle
VI	79.14	12478	5199
VI1	0.56	253	30

Table 3.5: Solving Time for two Exact Algorithm Reported by Zhang and Zhang[19]

3.3.2 Results

Table 3.4 shows the solving time and reaction time measured for different problems. PBUA$_R$ refers to PBUA with randomly generated initial grid, and PBUA$_E$ refers to PBUA with initial grid consisting of extreme points. The results show that PBUA can converge, in a sense that improvement is getting small, within a short time when compared with exact algorithms. To get a feeling on how long exact algorithms take to compute a policy, table 3.5 shows the computation time spent by one of the fastest exact algorithms known, which is denoted as VI1. The time of VI1 and PBUA were measured on different machines (UltraSparc II and PC with 400MHz K6-3 CPU respectively), so they cannot be compared directly. However, we can see that some problems solved by exact algorithms in minutes can be solved by PBUA in less than 1 second.

Figures 3.1 and 3.2 plot the solving time and reaction time against the number of states $|\mathcal{S}|$, so that we can see how this time increases with $|\mathcal{S}|$. The reaction time is roughly linear with $|\mathcal{S}|$, but it is hardly to tell for the solving time, possibly because solving time depends on some factors beside of $|\mathcal{S}|$.

Table 3.6 shows the rewards received using the policies found by PBUA in

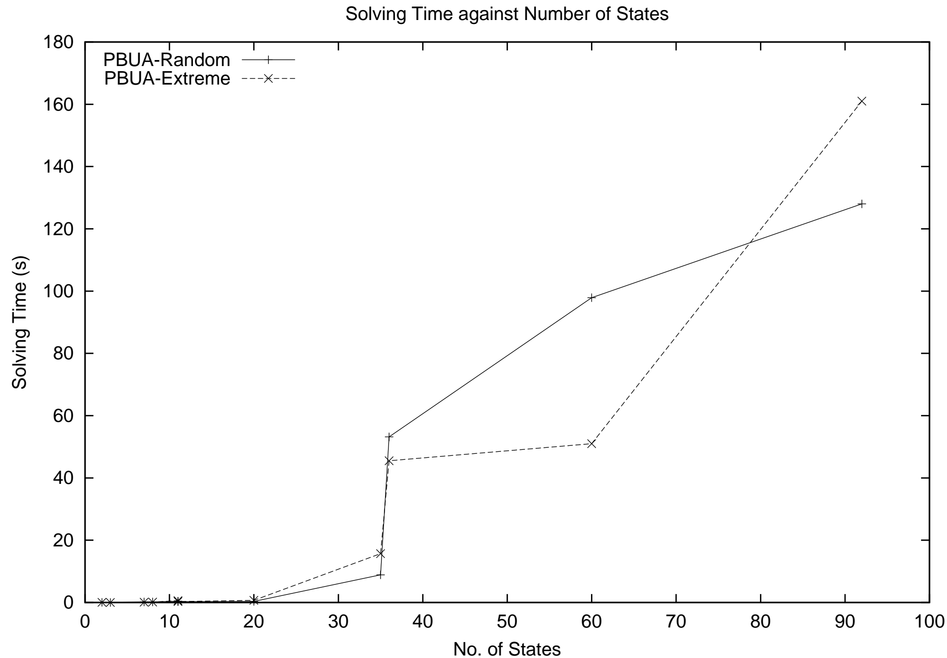

Figure 3.1: Solving Time against Number of States

simulation. It also shows the rewards received when an optimal policy (found by an exact algorithm) and random policy are used. It can be observed that while the quality of policies found by PBUA is quite near to optimal in some problems (Manufacture, Network), it is quite poor also in some problems (Tiger, Zhang's Maze, Office). Moreover, it can be seen that an initial grid constructed with extreme points can produce policy better than that with random points.

3.4 Discussions

From the experimental results, we can see that PBUA can compute policies very quickly compared to exact algorithms. It also allows some larger problems, which have not been attempted by any exact algorithm, to be solved. This fulfills one of the objectives of heuristic algorithms.

However, the policies found are not always very good. This is not desirable because we do not want to compromise computation time for a much poorer policy. The poor policy found can be explained by the fact that the number of vectors used in a value function is insufficient. For example, in the Tiger problem, the optimal policy should be to listen until the agent is very certain on where

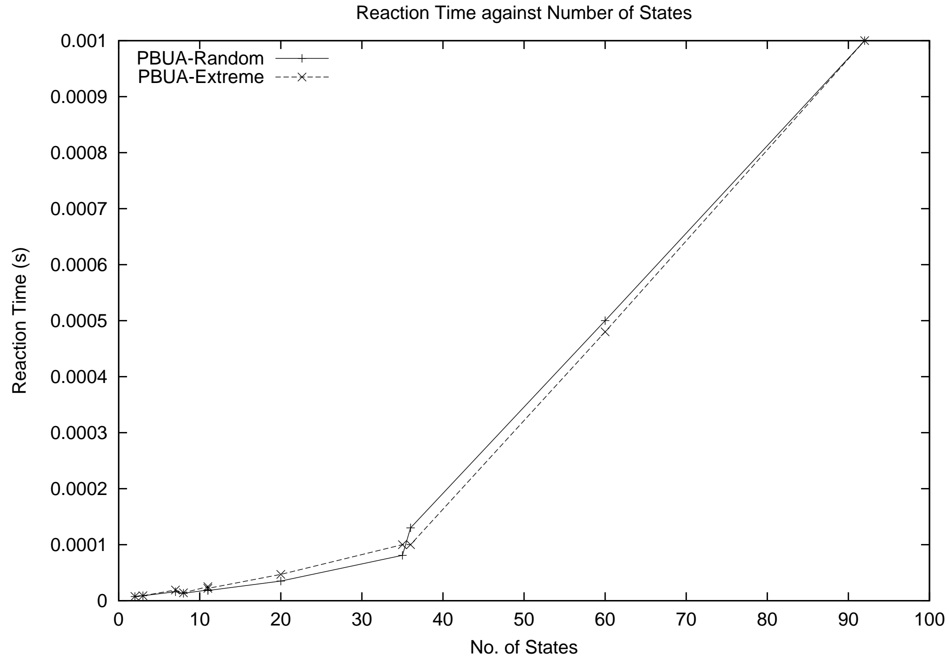

Figure 3.2: Reaction Time against Number of States

Problems	Reward			
	PBUA_R	PBUA_E	Optimal	Random
Tiger	-19.8	-19.8	19.5	-601
Manufacture	7.82	7.81	7.94	-6.97
Network	17.2	262	294	-243
Shuttle	19.6	32.7	32.6	-4.07
4x3	-0.585	1.84	-	-1.03
Zhang's Maze	0.0478	0.289	-	0.0055
Maze20	39.0	43.3	-	33.8
Office	-17.6	-7.44	-	-43.8
Tiger-Grid	1.89	2.11	-	-2.14
Hallway	0.518	0.493	-	0.0437
Hallway2	0.347	0.335	-	0.0252

Table 3.6: Experimental Results (Reward) of Basic PBUA

the tiger is, and then open the door opposite to the location of the tiger (More descriptions on the problem can be found in appendix A). Therefore, an optimal policy should be a choice among 3 different actions. However, since the size of the grid in the basic version of PBUA is 2, at most two vectors can be used in the value function found, which means it is a choice between only at most 2 actions. This shows that this small number of vectors is not enough to represent a good policy. Therefore, we should use a grid with a larger size to compute a good policy.

Moreover, we can see that an initial grid containing extreme points lead to a better policy than that with random points most of the cases. As a result, we should include the extreme points in the grid.

3.5 Summary

In this chapter, we propose a fast heuristic algorithm for POMDP. The core idea of PBUA is its point-based update, which improves the time complexity compared exact algorithms in two ways. First, it backups at the grid points only so it eliminates the need for spending time on search for the witness points. Second, the number of vectors does not increase after each update step, and it is limited to only the number of grid points. This overcomes the obstacle of exponential increase in number of vectors as seen in exact algorithms.

Experiments show that PBUA obtain a solution much faster than exact algorithms. Although the quality of some policies are quite poor, it shows promise to find a quick policy.

Chapter 4

Grid Expansion Heuristics

4.1 Introduction

In last chapter, we use some experiments to show the performance of PBUA. Despite the fact that it runs much faster than exact algorithms, it sometimes produces unsatisfactory policies. This is not acceptable to those smaller problems, which exact algorithms are capable to solve, because we may want to spend more time to solve a problem instead of using a much worse policy. However, for the larger problems, this might be the only way to obtain a better-than-random policy, since they are intractable to exact algorithms. This chapter looks into an extension to the basic version of PBUA, so that it is made suitable to be used in both cases.

As being pointed out at the end of last chapter, the main problem with PBUA is the insufficient number of vectors to represent a good policy. Moreover, the backup points may not be enough for obtaining a good value function. We can increase the size of the initial grid, but this also increases the computational complexity. Also, there is no obvious choice of the grid points to be added, other than the extreme points of the state space. We can add some random points in addition to the extreme points, but this does not guarantee to produce a good policy every time.

The extension to the basic version of PBUA builds on an iterative approach. PBUA first finds a value function quickly using a small grid. Then it expands the grid and finds a hopefully better value function using this larger grid. This expansion continues until either the policy is good enough, or time runs out.

This has two main advantages over merely using a large initial grid without expansion. First, the time spent on finding the solution is probably shorter, because not all the grid points are useful for improving the value function much. Having a larger grid initially may lead to more time spent on those less useful

points. In other words, the value function can be improved faster, though more coarsely, at the beginning with a smaller grid. Then it can be improved finely with a larger grid.

The second advantage is that the value iteration provides more information on the choice of new grid points. Initially, we have only got the parameters of a POMDP and so it is difficult to select the points with this little information. However, after a value iteration, we have got more information. For example, we can see where the value function has improved most, and then we can add more grid points to those areas.

This chapter discusses some heuristic methods for expanding the grid after each value iteration. These heuristic methods are classified into two kinds, generation heuristics and selection heuristics. The first kind finds some new belief points and add them for consideration. This is essential because the entire belief state space is continuous and we have to come up with a finite number of choices of belief points. The second kind selects some belief points from those choices based on some heuristic criteria.

This chapter begins with expansion basics, which show the steps for adding a new belief point to the grid. After that, some generation heuristics and selection heuristics are discussed. We then combine those heuristics to give some practical heuristic methods. Following that are some experimental results for comparing different heuristic methods. Finally, there are some discussions on the experimental results and a summary of this chapter.

4.2 Expansion Basics

PBUA allows a variable grid, so there is no restriction on the belief points that can be added to the grid, except no duplicate points should be added. As each point b_i^G is associated with a vector $\alpha_i \in \mathcal{V}$, we must also associate a vector to the new grid point b_{new}^G. Recall that the value function converges in a sense that the values at the grid points stop increasing, so we would like to start with the highest value that is smaller than the optimal value function. However, we do not know the exact value of the optimal value function, and we only have a lower bounded value function V computed after a value iteration in PBUA. As a result, we start with the value $V(b_{new}^G)$ at the newly added grid point. We can achieve this by associating b_{new}^G with the vector in $\mathcal{V}_n$ that gives the maximum

Add Grid Point(b^G_{new}, G, $\mathcal{V}$)
Input: b^G_{new} is the belief point to be added to the grid G, $\mathcal{V}$ is the set of vectors representing the current value function V
Output: A grid G with b^G_{new} added, and a modified set of vectors $\mathcal{V}$

Add b^G_{new} into G
if $|\mathcal{V}| = 0$
  Initialize $\mathcal{V}$ by equation 3.7
Associate b^G_{new} to the vector, $\alpha_{new} \leftarrow \arg\max_{\alpha \in \mathcal{V}} \alpha \cdot b^G_{new}$
return G, $\mathcal{V}$

Table 4.1: Procedure for Adding a New Grid Point (Add Grid Point)

inner product with it. This has assumed that $\mathcal{V}$ is initialized. When $\mathcal{V}$ is empty, which happens before it is initialized, a vector should be added to it as shown by equation 3.7. A procedure for adding a new grid point is given in table 4.1.

4.3 Grid Point Generation Heuristics

POMDP has a continuous belief state space $\mathcal{B}$ but PBUA uses a discrete grid G. Therefore, we have to find some ways to generate a finite number of new grid points, from which we can choose to add to the original grid. This section suggests some heuristic methods for generate these new grid points. The generated grid points are not guaranteed to be useful to the value iteration. In the next section, we suggests some heuristic methods to decide which points should be added.

4.3.1 Extreme Points

As shown in experiments in last chapter, a grid consisting of extreme points leads to a better policy than that does not. Therefore, one heuristic method for generating the belief points is to generate the extreme points. In particular, an extreme point $b_{ex,s}$ of state s is a belief state that assigns a probability mass of 1 to state s and 0 to the remaining states. Formally,

$$b_{ex,s}(s') = \begin{cases} 1 & \text{if } s' = s \\ 0 & \text{otherwise} \end{cases} . \tag{4.1}$$

These extreme points $b_{ex,s}$ correspond to the belief states that the agent is certain about at which state s the agent is. Since there are $|\mathcal{S}|$ number of states, the total number of extreme points is also $|\mathcal{S}|$.

The fact that inclusion of extreme points improves value function can be explained in this way. When the agent is more certain on its state, it is clearer the expected rewards that it is going to receive. Therefore, the values of these extreme points are more extreme and are more influential to the value iteration. Moreover, including the extreme points might help maintain the overall shape of the value function and might make it similar to that of the optimal value function. Therefore the resulting value function produces a better policy.

4.3.2 Initial Belief State

Another possible choice of belief point is the initial state of a POMDP problem. Some problems specify a particular belief state as the initial belief state. For example, the initial belief state can be a uniform belief state, where equal probability is assigned to each of the state. It can also be a uniform distribution over some of the states, or excluding the goal states. Adding this initial belief state may enhance the policy found by PBUA in that particular problem, because this may give a better estimate to the values of the belief states that the agent might encounter starting from the initial state.

4.3.3 Previous/Next-Step Belief States

We can also generate the previous/next-step belief states using each of the grid points. For a belief state b, the next-step belief states b_{next} are the possible belief states that can be reached from b in one step, and the previous-step belief states b_{prev} are those from where b can be reached in one step. They are also referred as successive and preceding belief states.

Given an action $a \in \mathcal{A}$, an observation $z \in \mathcal{Z}$ and a belief state $b \in \mathcal{B}$, the next-step belief state b_{next} can be found by

$$b^z_{a,next}(s') = \frac{\sum_s P(z, s'|s, a)b(s)}{P(z|b, a)}, \tag{4.2}$$

and the previous-step belief state b_{prev} by

$$b_{a,prev}(s) = \frac{\sum_{s'} P(s'|s,a)b(s')}{\sum_{s,s'} P(s'|s,a)b(s')}. \tag{4.3}$$

Using these two kinds of belief states in grid expansion helps propagate the values at the grid points, but in two directions. Recall that the value of a belief state is the expected future rewards. Therefore, when we use the successive belief states, we are in hope of improving the values of the original grid points, since we may get more accurate values of their future states. When we use the preceding belief states, we are in hope of improving the values of the new grid points, since the original grid points provide more accurate values of the future states of the new grid points.

4.3.4 Midpoint

A midpoint can computed from any pair of points in the grid. This midpoint may be useful because it spreads out the grid , unlike the preceding and successive belief states, which expand on the neighborhood of the grid points. Moreover, this midpoint is the farthest distance from both points of any pair of points. The farther a point from the grid points, the less accurate the estimated value that point may have. Therefore, the midpoint is possibly a point of which value needs largest improvement due to its least accuracy.

For a pair of points b_i and b_j, the midpoint $b_{i,j}$ is given by

$$b_{i,j}(s) = \frac{b_i(s) + b_j(s)}{2} \tag{4.4}$$

4.3.5 Random Belief State

When expanding the grid, we may also take some random belief states into consideration. Using random points can be treated as exploring the belief space for points that cannot be systematically generated as suggested before.

4.4 Grid Point Selection Heuristics

After generating a number of belief points, we have to decide which point should be used to expand the grid. In this section, we discuss two heuristic criteria for

making this decision.

4.4.1 Value Function Improvement

PBUA tries to improve the values at the grid points as much as possible, so that they are as close to the optimal value function as possible. Therefore, one criterion for selecting grid points is the amount of improvements of the values at those points PBUA gets by adding them. Suppose that b_{new} is a candidate belief point, and V and V' are the converged value functions before and after b_{new} is added to the grid. This criterion measures the difference $\Delta = V'(b_{new}) - V(b_{new})$.

This heuristic criterion has several limitations. First, adding a new grid point may not improve the value only at the new grid point, but also the value elsewhere. This criterion does not take this into consideration. Second, the shape of the value function may also be important for obtaining a good policy, especially when the value function is far from the optimal value function. Therefore, increasing the value much at a single point does not necessarily lead to a better policy, since it may distort the shape of the value function and even worsen the policy obtained. Despite these limitations, this criterion may still be useful in guiding the expansion of grid points.

One problem of using this criterion is that it is not trivial to obtain Δ, unless we do a complete value iteration to test for every point, which takes time. We suggest three computationally less expensive ways to estimate Δ. Each of the three methods has its only advantages and disadvantages. It is not clear which one is the best. We present some experimental results later in this chapter to compare them empirically.

Backup

The first way to estimate Δ is to obtain a backup vector at the candidate belief point b_{new}. The estimated amount of improvement $\widehat{\Delta}$ is given by

$$\widehat{\Delta} = \texttt{backup}(b_{new}, \mathcal{V}) \cdot b_{new} - V(b_{new}). \tag{4.5}$$

During a complete value iteration, the backup operator is applied at a grid point for more than one time usually. This estimation uses the first backup to estimate the improvement given by subsequent backups. In equation 4.5, $\widehat{\Delta}$ is

positive only if $\texttt{backup}(b_{new}, \mathcal{V})$ gives a larger inner product than the original vectors in $\mathcal{V}$. In our implementation, we add b_{new} only if $\widehat{\Delta}$ is greater than zero.

This estimation assumes that $\texttt{backup}(b_{new}, \mathcal{V})$ gives a larger inner product at b_{new} than any vector $\alpha \in \mathcal{V}$ can. If this is not the case, that is, if $\widehat{\Delta}$ is negative, b_{new} is not added to the grid.

One limitation of this approach is that $\widehat{\Delta}$ is the improvement of only one step. It is possible that the total improvement caused by the subsequent backups is more than the first backup. Moreover, doing a backup can be relatively slow, since it takes $O(|\mathcal{A}||\mathcal{S}||\mathcal{Z}||\mathcal{V}| + |\mathcal{A}||\mathcal{S}|^2|\mathcal{Z}|)$ time.

Last-Step Value Function Estimation

Another estimation is to use the value function U before the last value iteration. Thus, the estimate is given by

$$\widehat{\Delta} = V(b_{new}) - U(b_{new}). \tag{4.6}$$

This takes only $O(|\mathcal{V}|)$ and is faster than using backup as estimation. However, a large improvement in the last value iteration does not mean a large improvement in the next value iteration. For example, sometimes a large improvement at a belief state means the value at that state is closer to the optimal, and it leaves small room for improvement.

Upper Bound Estimation

The third way is to measure the difference between the values given by the optimal value function and the approximate value function at the candidate belief point. However, if we know the optimal value function, we do not need to compute the approximate value function. Recall that the value function obtained by PBUA is a lower bound to the optimal value function. Therefore, we can use an upper bound to the optimal value function to estimate this room of improvement. Suppose that V_{upper} is a known upper bound value function,

$$\widehat{\Delta} = V_{upper}(b_{new}) - V(b_{new}). \tag{4.7}$$

The room of improvement shows where the value function needs most improvement. However, this is subject to the tightness of the upper bound. A close upper

bound may take much time to compute, while a loose upper bound may mislead PBUA to select the belief states where the upper bound has the largest error. One very fast upper bound is given by QMDP approximation, which is described in chapter 6.

4.4.2 Likelihood of being Encountered in Simulation

The second selection criterion is based on the likelihood that a belief state is encountered in simulation. This is based on an idea that if we get the optimal values of all the belief states that the agent encounters in the simulation, we can derive an optimal policy no matter what the values of the other belief states are. Another idea is that every grid point is associated with a vector, and each vector is associated with an action. Therefore, if we get the optimal actions for these belief states encountered, we can get the optimal policy for these belief states no matter what the entire value function is.

However, there are possibly infinite number of these belief states. Therefore, we can only consider the more likely cases. Moreover, it can be difficult to estimate this likelihood without any extra information in addition to the POMDP parameters. This can be easier if we estimate the likelihood of a successive belief state to a given belief state. For example, with the current approximate value function, we can find the optimal action $a^* \in \mathcal{A}$ for a belief state b. Then, the probability that a successor state $b^z_{a^*}$ occurs after b is equal to the observation probability $P(z|b,a)$, which can be found by equation 2.16. If the initial belief state used in simulation is given by a POMDP problem, we can use it as that given belief state b. If not, we can use the existing grid points as the starting points.

This selection criterion has one limitation. It may be impossible to include all the belief states encountered. Therefore, it may be better to concentrate on improving the value function as a whole rather than concentrate on improving some of the belief state values.

4.5 Combining the Heuristics

After discussing several generation and selection heuristics, we combine them and give two practical heuristic expansion methods in this section.

kLI Expansion(k, G, $\mathcal{V}$)
Input: k is the number of points to be added to grid G, and $\mathcal{V}$ is the vector set corresponding to the current value function V
Output: grid G with at most k points added and the modified $\mathcal{V}$

Initialize a list $\mathcal{L}$, a list of candidate points sorted in descending order by their estimated improvement $\widehat{\Delta}$, to an empty list

while some points can still be generated using any of the generation heuristics
  $b_{new} \leftarrow$ generated point
  Estimate the amount of improvement $\widehat{\Delta}$ using one of the estimation methods
  if $\widehat{\Delta}$ is larger than that of the last element of l and $\widehat{\Delta} > 0$
    Add b_{new} to $\mathcal{L}$ according to the descending order of $\widehat{\Delta}$
    if $|\mathcal{L}| > k$
      Remove the last element of $\mathcal{L}$

foreach $b_l \in \mathcal{L}$
  $\{G, \mathcal{V}\} \leftarrow$ Add Grid Point(b_l, G, $\mathcal{V}$)

return G, $\mathcal{V}$

Table 4.2: Procedure for k-Largest Improvement Expansion (kLI Expansion)

4.5.1 k-Largest Improvement Expansion

The first combined heuristic expansion method is called k-largest improvement (kLI) expansion. kLI expansion selects the k points that give the maximum amount of improvement, estimated by any of the three estimation methods discussed earlier. Moreover, it can use any heuristics to generate new points for consideration.

Table 4.2 shows the procedure for kLI expansion. It first generates some new grid points using any combination of the generation heuristics. Then it uses one of the estimation method to estimate the improvement and chooses the k points that give the largest estimated improvement. This procedure maintains a list $\mathcal{L}$ in descending order of $\widehat{\Delta}$ of the candidate points, and limits the size of $\mathcal{L}$ to less than k. If $\widehat{\Delta}$ of a candidate point is not greater than zero, that point is not added to $\mathcal{L}$, to avoid adding too many useless points to G. After all the points are generated, all the k points in $\mathcal{L}$ are added to G.

```
SS EXPANSION(G, 𝒱, N)
Input: G is the grid used, 𝒱 is the set of vectors, and N is the
limit on the number of trials
Output: an expanded grid G and a modified set of vectors 𝒱

foreach extreme point b_ex
    n ← 0
    b ← b_ex
    while b ∉ G and n < N
        α* ← arg max_{α∈𝒱} α · b
        a* ← associated action of α*
        Select observation z randomly according to the distribution
        of P(z|b, a*)
        b ← b^z_{a*}
        n ← n + 1
    {G, 𝒱} ← ADD GRID POINT(b,G,𝒱)

return G,𝒱
```

Table 4.3: Procedure for Stochastic Simulation Expansion (SS EXPANSION)

A possible modification to this procedure is to use one of the estimation methods alternatively. For example, it may use the backup estimation in an expansion, and then it may last-step value function estimation in the next expansion. This may improve the expansion because more potential witness points are located with different estimation.

4.5.2 Stochastic Simulation

Another combined heuristic expansion method results in an method utilizing stochastic simulation. With stochastic simulation, points that are more likely to be encountered are more probable to be generated and added to the grid G. The stochastic simulation can start with the extreme points or the initial belief state used in simulation. A procedure that uses stochastic simulation starting with extreme points is given in table 4.3.

This procedure adds $|\mathcal{S}|$ number of points to G at each expansion, since there are $|\mathcal{S}|$ extreme points and one point is added for each extreme points. However, in practice, there may not be many possible belief states in a POMDP and most of them have been added to G. Therefore, it may take many trials to generate

```
PBUA(n_e, ε)
Input: n_e is the number of expansions, ε is the stopping threshold
Output: A vector set 𝒱, which can be used as an estimate of the
optimal value function and as a policy

G ← ∅ and 𝒱 ← ∅

for i = 1 to n_e
   {G, 𝒱} ← Expansion(G, 𝒱) using one of the expansion heuris-
   tic methods
   repeat
      {𝒱, Δ} ← PBVI(G, 𝒱, ε)
   until Δ < ε

return 𝒱
```

Table 4.4: Pseudo-code for PBUA (PBUA)

one, which may be unlikely to be encountered in simulation, that is not in G. We put a limit N on the number of trials for finding a point that is not in G for each extreme point, so that it does not spend much time in the while loop to find an unlikely point that may not improve the policy much.

4.5.3 Whole PBUA

The pseudo-code for PBUA is given in table 4.4. After each expansion, the vector set $\mathcal{V}$ is used in value iteration until the cumulative improvement in a complete value iteration is smaller than the threshold ϵ. These expansion and convergence steps are executed for n_e steps. Finally, the value function V represented by $\mathcal{V}$ is returned by PBUA.

4.6 Experimental Results

4.6.1 Setup

In this section, we present some experiments and try to compare the effectiveness of using different expansion heuristics. The experiments included different POMDP problems in literature, as in the previous chapter. The number n_e for the expansion is the number of times that the grid has expanded. We add $|\mathcal{S}|$

number of points in each expansion. In the first expansion, only the $|\mathcal{S}|$ extreme points are added. Therefore PBUA with $N_e = 1$ uses only the extreme points in its grid to compute the policy.

In each simulation of the smaller problems, 10000 trials with maximum of 100 steps in each trial were run. These problems include Tiger, Manufacture, Network, Shuttle, and 4x3. For the other problems, Zhang's Maze, Maze20, Office, Tiger-Grid, Hallway, Hallway2, 5000 trials with maximum of 100 steps in each trial were run in each simulation. This resulted in error of the rewards at 95% significance level less than 4% in most cases.

We denote the variations of PBUA with 4 digits. The first digit refers to the expansion heuristics. This digit corresponds to:

Extreme points only(0) Only extreme points are used and there is no expansion after the first expansion.

Random points(1) In each expansion, $|\mathcal{S}|$ random points are generated and added to the grid, except for the first expansion, in which $|\mathcal{S}|$ extreme points are added.

kLI successive points(2) In each expansion, all the successive belief points are generated for each grid point. $|\mathcal{S}|$ points with the largest estimated amount of improvement are added to the grid.

kLI preceding points(3) In each expansion, all the preceding belief points are generated for each grid point. $|\mathcal{S}|$ points with the largest estimated amount of improvement are added to the grid.

kLI midpoints(4) In each expansion, one midpoint is generated using every pair of grid points. $|\mathcal{S}|$ points with the largest estimated amount of improvement are added to the grid.

kLI stochastic simulation(5) In each expansion, one point is generated using stochastic simulation starting from each grid point. If that generated point is already in the grid, another point is generated from the point just generated. $|\mathcal{S}|$ points with the largest estimated amount of improvement are added to the grid.

kLI combined(6) In each expansion, it combines the generation heuristics used in 2–5. It also generated $|\mathcal{S}|$ number of random points as candidate points.

$|\mathcal{S}|$ points among all the generated points with the largest estimated amount of improvement are selected and added to the grid.

Stochastic simulation from extreme points(7) In each expansion, one point is generated using stochastic simulation from each extreme point. If that generated point is already in the grid, another point is generated from the point previously generated. $|\mathcal{S}|$ points are resulted in each stochastic simulation and are then added to the grid.

The second digit refers to the method used to estimate the amount of improvement for kLI expansion (2–6): (0) for estimation using the backup operator, (1) for estimation using the last-step value function, and (2) for estimation using the QMDP approximation as an upper bound value function. For expansion method 7, a digit 3 refers to a combination of using both last-step value function and QMDP approximation for estimation. It switches the estimation method when the cumulative improvement in the last value iteration drops below 0.1. QMDP approximation is used initially.

The third digit and the last digit is not used in this chapter. They are always 0. As an example, 2100 refers to using kLI expansion with next-step belief points as generation heuristics and using last-step value function for estimation.

The unit of the time reported is second. The time data was collected on a PC with 400MHz AMD K6-3 CPU. In the figures shown, the data points for 1st-5th, 10th, 15th, and 20th expansions are usually included.

4.6.2 Effectiveness of Expansion

The first experiment aims to test whether incorporating the expansion mechanism produces a faster converged value function. We used two versions of PBUA, one without expansion and one with expansion. We used some random points for constructing the grid, in addition to the extreme points. Table 4.5 shows the computational time spent. The Nx in the heading means the final grid size is $N \times |\mathcal{S}|$. For PBUA using expansion, the $|\mathcal{S}|$ random grid points are added after each value iteration. Therefore, N is also the number of expansions in this case. For PBUA not using expansion, the $N \times |\mathcal{S}|$ number of points are added to the initial grid and no more is added after this.

From the results, we can see that using expansion reduced the solving time for Zhang's Maze and Hallway2 significantly. However, in the Maze20 problem,

Problems	Expansion	5x	10x	15x	20x
Zhang's Maze	No	4.00	15.0	30.8	52.9
	Yes	0.90	2.22	3.76	6.17
Maze20	No	6.58	31.0	73.0	140
	Yes	5.13	21.8	106	230
Hallway2	No	2682	11110	26238	45000
	Yes	1369	4519	12489	27898

Table 4.5: Experimental Results for Comparing PBUA With and Without Expansion

the effect is not so obvious and using expansion spent more time for the larger grid. We believe this inconsistency is due to the randomness of the grid points and conclude that expansion generally reduces the solving time.

4.6.3 Performance of Using a Larger Grid

This experiment demonstrates the improved quality of the policies obtained by using a larger grid. In this experiment, we used the random points expansion (1000) for constructing a larger grid. The results are shown in table 4.6. The headings with numbers denote the numbers of expansions that PBUA had taken. The headings "Opt" and "Rand" denote the optimal policy and a random policy.

We can see that the a larger grid improves the quality of the policy found by PBUA. Moreover, we can observe that a close-to-optimal policy can be found significantly faster by PBUA than by exact algorithms for the small problems, in which the optimal policies are known.[1] For the larger problems, we cannot compare the policies obtained by PBUA with the optimal policies, since they cannot be found by exact algorithms. However, the quality of the policies obtained by PBUA are similar to or better than the best policies we have noticed in the literature.[2]

We can observe that the policies obtained stopped improving considerably after the first few expansions. One possible reason is that the policy has reached the optimal already. Another possible reason is that the witness points are harder to be hit when the grid has become larger. Therefore PBUA is not able to improve the policy further. The third reason is that random point expansion is not useful to improve the policies much.

[1] See table 3.5 for an insight to the amount of time needed for exact algorithms.
[2] See appendix A for the results reported in the literature.

Problems		1	5	10	15	20	Opt	Rand
Tiger	T_s	0.02	0.09	0.12	0.12	0.14	-	-
	Reward	-19.8	19.5	19.3	19.9	19.0	19.5	-601
Manufacture	T_s	0.02	0.08	0.09	0.11	0.13	-	-
	Reward	7.82	7.91	7.90	7.91	7.94	7.94	-6.97
Network	T_s	0.09	0.57	1.59	1.73	3.61	-	-
	Reward	263	292	290	292	295	294	-243
Shuttle	T_s	0.16	0.24	0.36	0.59	0.78	-	-
	Reward	32.6	32.6	32.6	32.6	32.7	32.6	-4.07
4x3	T_s	0.65	1.86	2.20	3.35	5.13	-	-
	Reward	1.83	1.87	1.87	1.87	1.87	-	-1.03
Zhang's Maze	T_s	0.30	0.90	2.22	3.76	6.17	-	-
	Reward	0.30	0.52	0.54	0.54	0.53	-	0.00
Maze20	T_s	0.86	5.13	21.8	106	230	-	-
	Reward	43.3	55.4	56.1	57.7	58.4	-	33.8
Office	T_s	15.5	52.1	380	773	1269	-	-
	Reward	-8.45	12.7	21.1	21.2	21.3	-	-43.8
Tiger-Grid	T_s	44.7	257	737	1577	2577	-	-
	Reward	2.15	2.20	2.19	2.24	2.20	-	-2.14
Hallway	T_s	51.0	450	979	2242	4602	-	-
	Reward	0.49	0.52	0.52	0.51	0.51	-	0.04
Hallway2	T_s	161	1369	4519	12489	27898	-	-
	Reward	0.33	0.34	0.34	0.34	0.35	-	0.02

Table 4.6: Performance of PBUA using Random Expansion with Extreme Points

Based on the results, we conclude that a larger grid improves the the quality of the policies given by PBUA. This allows PBUA to find optimal policies for the smaller problems, which exact algorithms is able to solve. This also allows PBUA to find good policies for the larger problems, which is not feasible using exact algorithms.

4.6.4 Comparison of Different Estimation Methods

After showing that expansion is useful to PBUA, we start to compare the different expansion heuristics. First, we present an experiment for comparing the different estimation methods.

We tested the three estimation methods with two generation heuristic methods: successive belief points and midpoint heuristics. We chose these two because they represent two different strategies, one exploring the neighborhood of the grid points and the other spreading the grid. We conducted experiments with the Zhang's Maze problem and Maze20 problem.

From figures 4.1 and 4.4, we can see that the estimation using backup needed most time. This is expected as it takes more time to do a backup. Estimation using last-step value function took more time than that using QMDP approximation, because getting a value from a set of vectors takes more time than from a Q-value function, especially when the vector set becomes larger in size. Note that the time needed for using backup as estimation can take about 4-8 times of that needed for using last-step value function, and about 12-22 times of that needed for using QMDP approximation for a 20-step expansion. This shows that using the backup estimation may become infeasible when the size of the state space and the size of the grid become large.

From figure 4.2, it is hard to tell which estimation method is better than the others. However, it appears that estimation using backup could improve the policy faster during the earlier (2nd–4th) expansions. In figure 4.5, the difference between using different estimation methods is clearer. Expansions using backup as estimation heuristics improve the policies faster and yield the best policies, possibly because the backup gives the best estimate to the amount of improvement. The estimation method with QMDP approximation improved the value function faster than that with last-step value function, but this failed to improve the value function in later expansions. A possible reason is that during the earlier expansions, QMDP approximation tends to choose those points with greater dif-

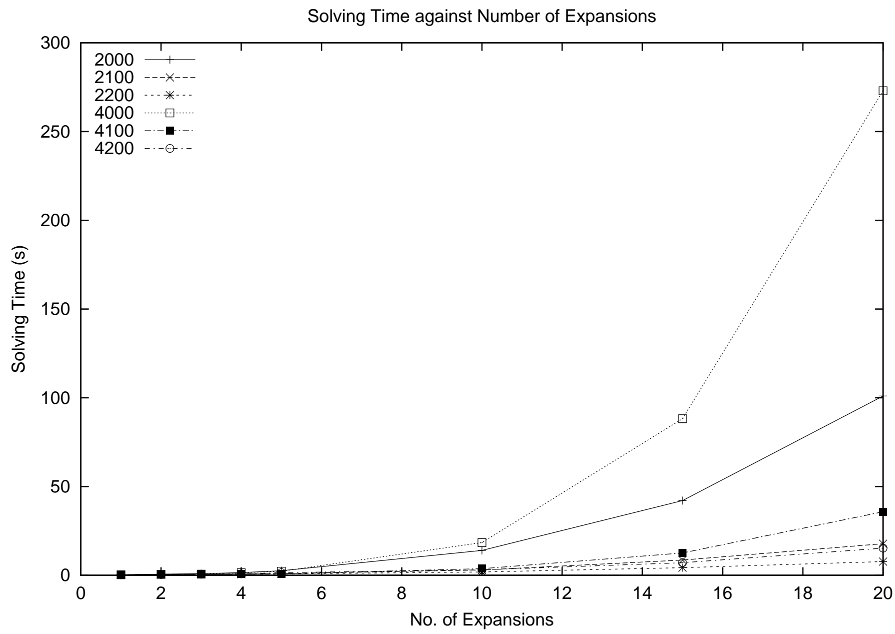

Figure 4.1: Solving Time with Different Estimation Methods in Zhang's Maze using kLI Expansion

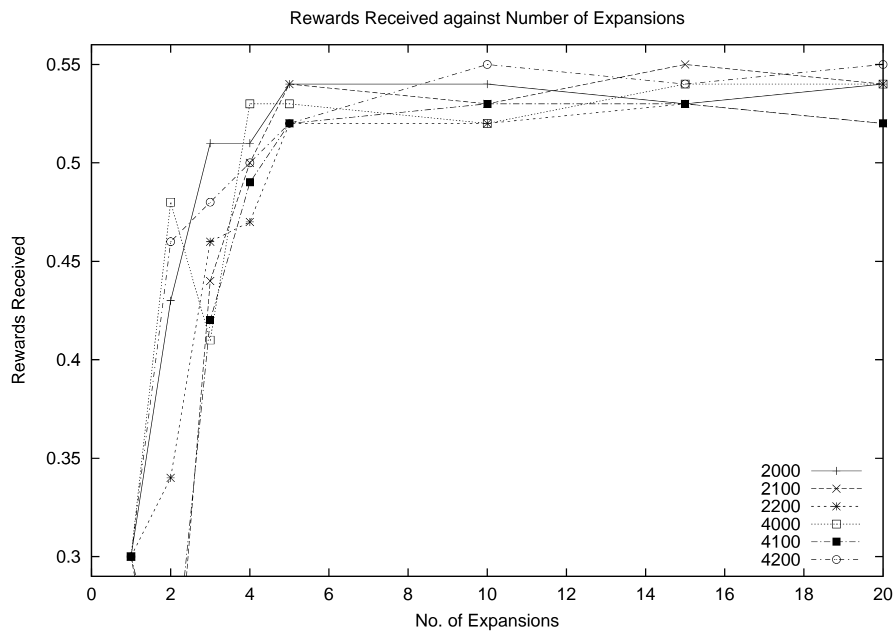

Figure 4.2: Rewards Received with Different Estimation Methods in Zhang's Maze using kLI Expansion

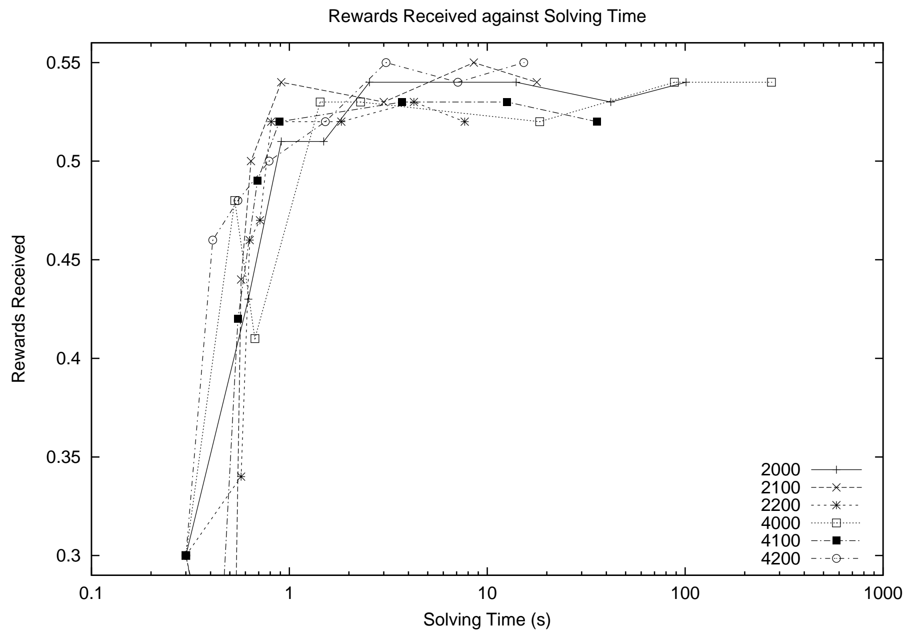

Figure 4.3: Rewards Received against Solving Time with Different Estimation Methods in Zhang's Maze using kLI Expansion

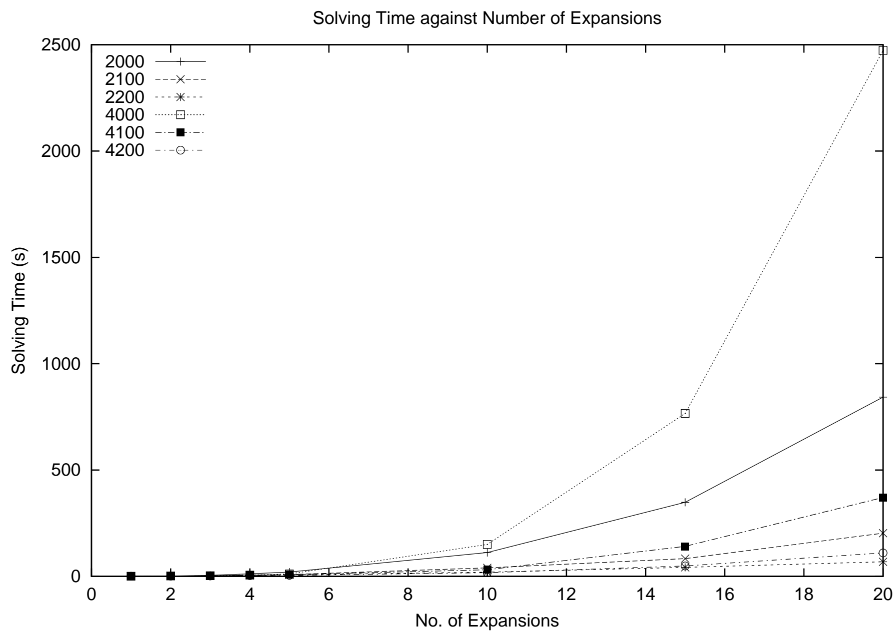

Figure 4.4: Solving Time with Different Estimation Methods in Maze20 using kLI Expansion

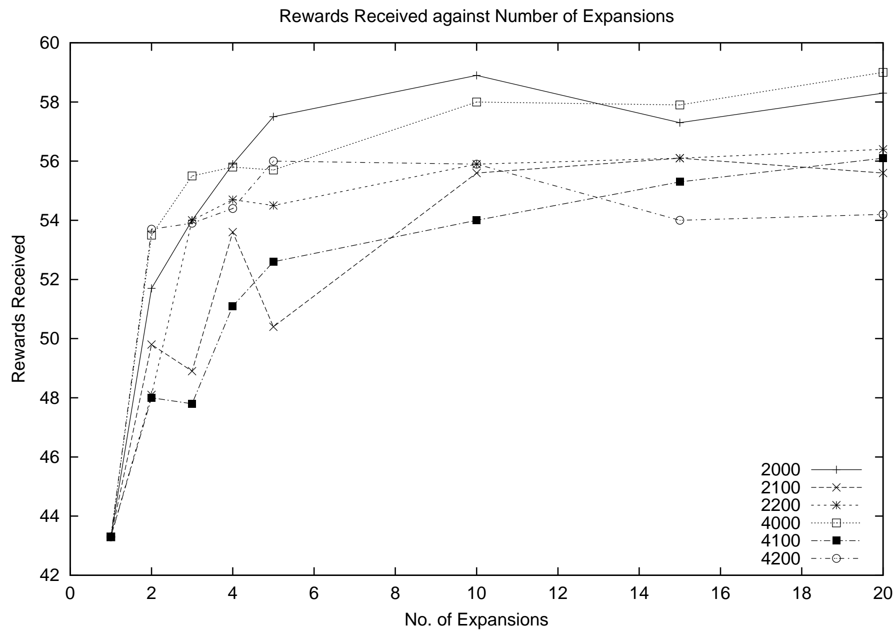

Figure 4.5: Rewards Received with Different Estimation Methods in Maze20 using kLI Expansion

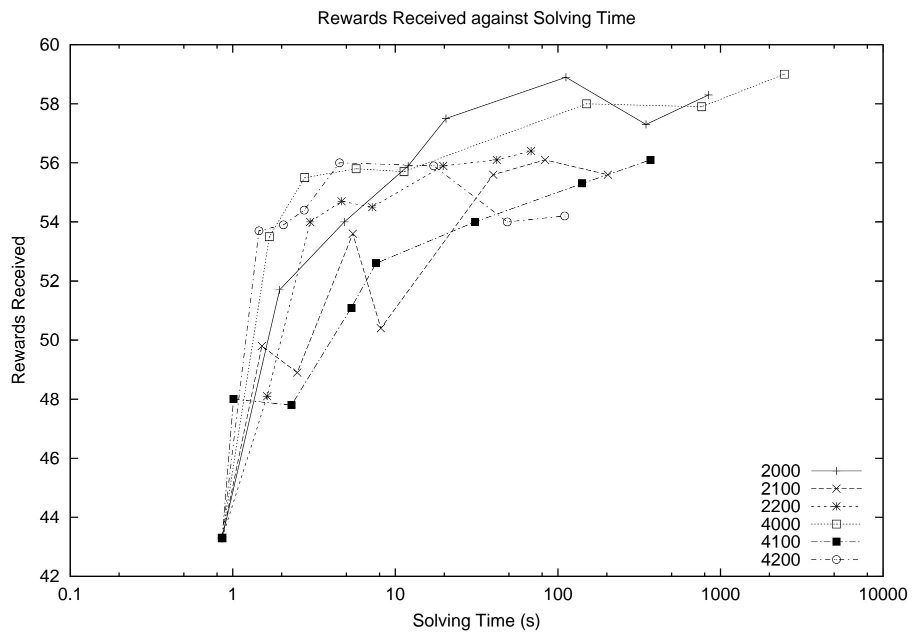

Figure 4.6: Rewards Received against Solving Time with Different Estimation Methods in Maze20 using kLI Expansion

ference to the optimal values, and since the value function computed by PBUA is not close to the optimal, the error of the value function of QMDP approximation is not significant. However, during the later expansions, estimation with QMDP approximation tends to choose the points where the Q-value function has larger errors, so adding those selected points does not improve the policy of PBUA as much as before.

Figures 4.3 and 4.6 shows the quality of the policy against the time spent on PBUA. In the former figure, the difference between using different estimation methods is again not obvious. In the latter figure, it appears that estimation using backup improves the policy faster without regarding to the generation methods. We can also observe that estimation using QMDP approximation improves the policy slightly faster than that using last-step value function.

From the experimental results shown, we believe using backup as estimation method is slightly better than the others, since it allows obtaining a better policy faster. However, it can become slow in larger problems, since the time complexity of the backup operator depends on $|\mathcal{S}|^2$. It appears to be not suitable to the problems with larger state spaces, and so the other two estimation methods still have their values.

4.6.5 Comparison of Different Generation Heuristics

The following experiment compares different generation heuristics. We used the backup estimation method, since it appears to be slightly better than the other two when the state space is not very large. The experiments were carried out with Zhang's Maze and Maze20 problems.

From figures 4.7 and 4.10, we can see that the generation heuristic method using midpoint took the most time to expand, followed by that using next-step belief points. The other two required similar amount of time. This can be explained by the different time requirements of the different generation methods. To generate each point, midpoint considers every pair of grid points, while the others consider only one grid point, so their time complexity are $O(|G|^2)$ and $O(|G|)$, considering that the parameters of POMDP ($|\mathcal{S}|$, $|\mathcal{Z}|$ and $|\mathcal{A}|$) are constant. From each grid point, $|O| \times |A|$ next-step belief points and $|A|$ previous-step belief points are generated. Although in with stochastic simulation, only one point is generated from each grid point, it takes more time to generate a point using stochastic simulation than to generate a previous/next-step belief points. This explains the

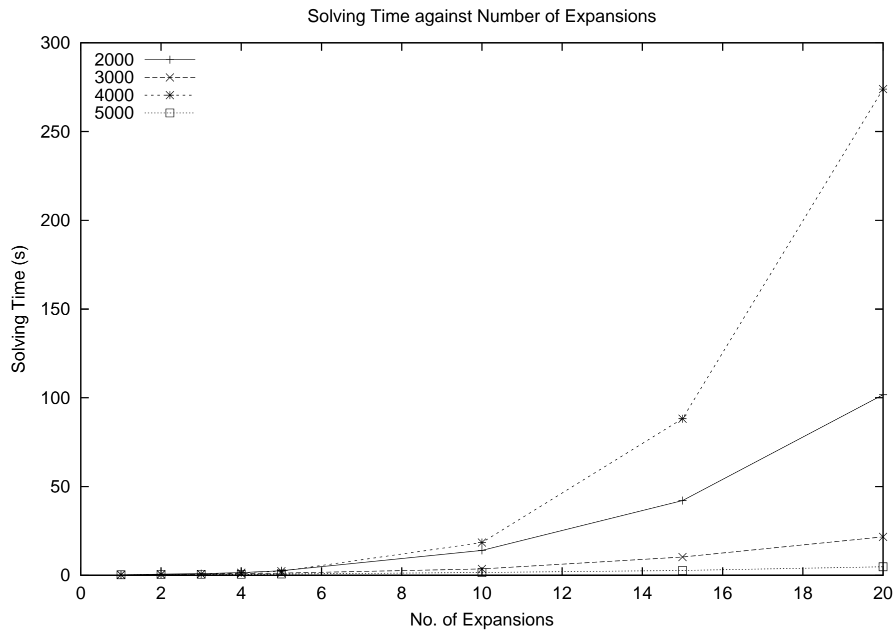

Figure 4.7: Solving Time with Different Generation Heuristics in Zhang's Maze using kLI Expansion

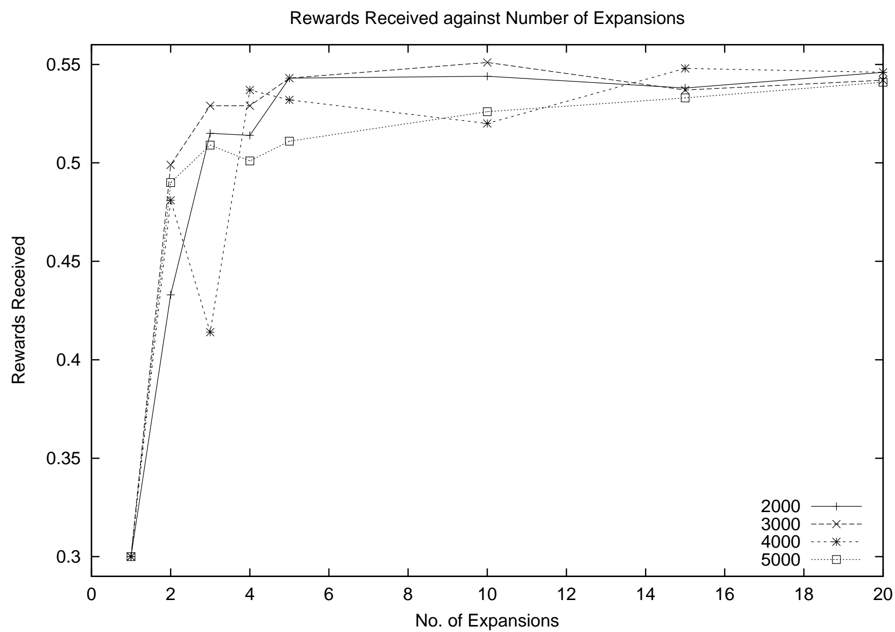

Figure 4.8: Rewards Received with Generation Heuristics in Zhang's Maze using kLI Expansion

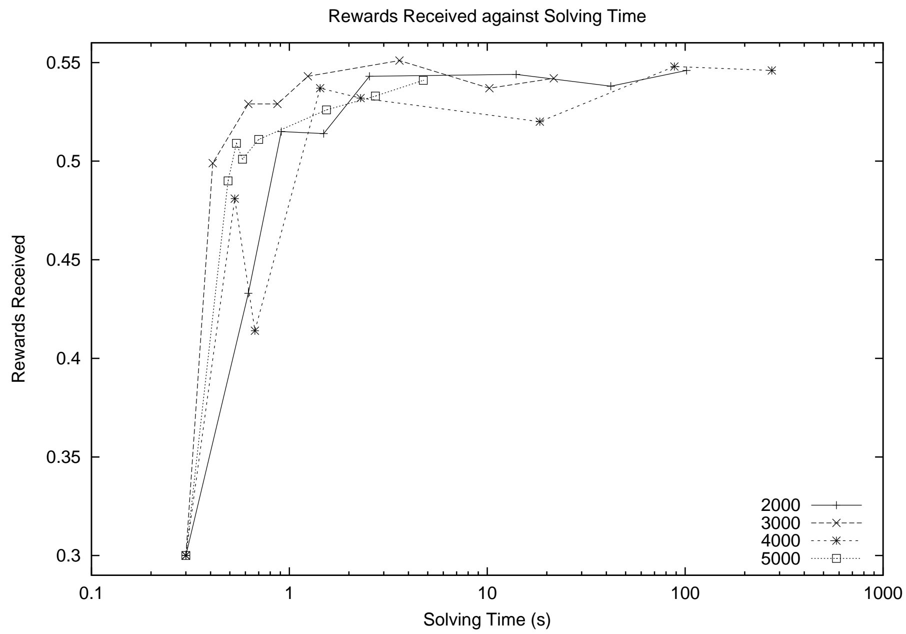

Figure 4.9: Rewards Received against Solving Time with Different Generation Heuristics in Zhang's Maze using kLI Expansion

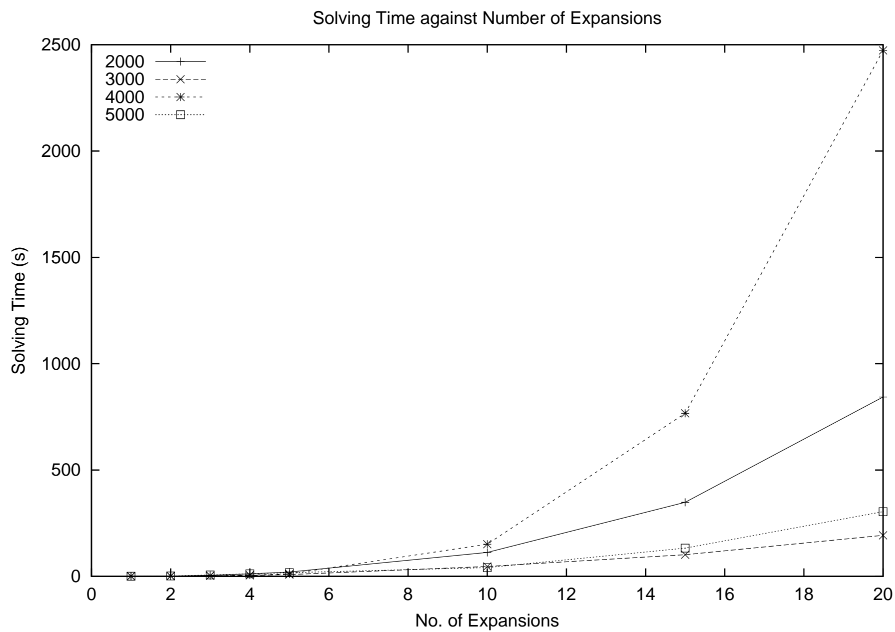

Figure 4.10: Solving Time with Different Generation Heuristics in Maze20 using kLI Expansion

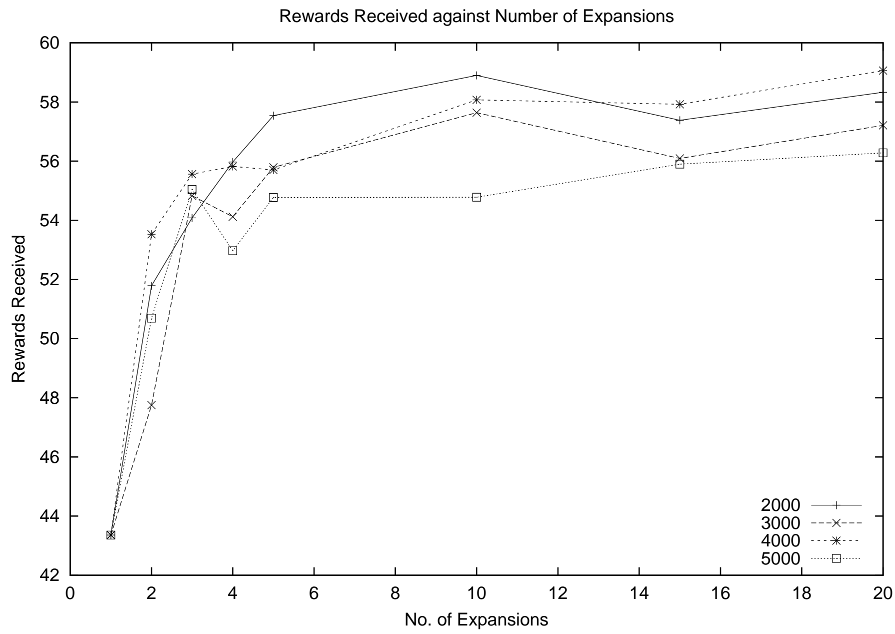

Figure 4.11: Rewards Received with Different Generation Heuristics in Maze20 using kLI Expansion

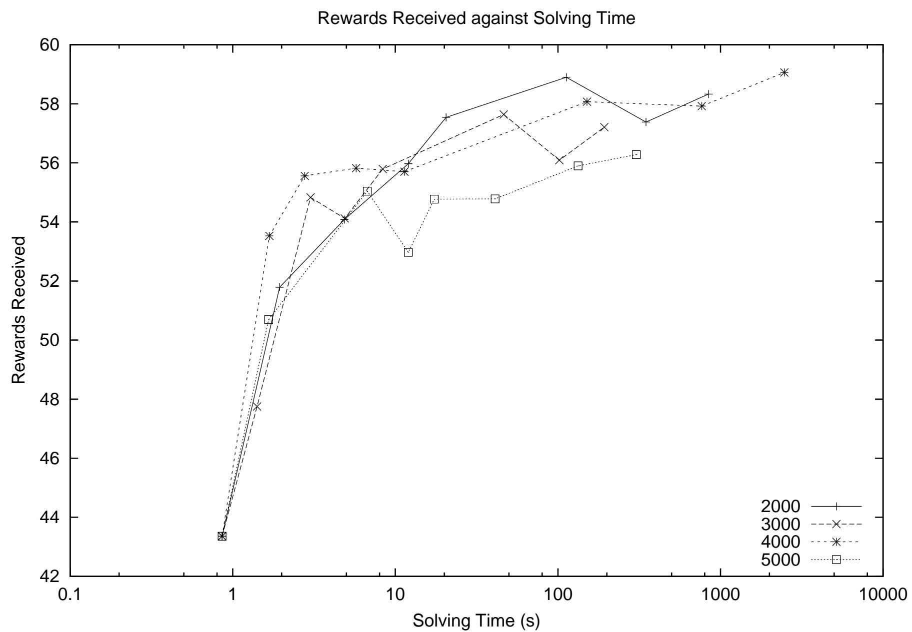

Figure 4.12: Rewards Received against Solving Time with Different Generation Heuristics in Maze20 using kLI Expansion

computation time difference between each generation heuristic methods.

In the Zhang's Maze problem (figures 4.8 and 4.9), the heuristic method using previous-step belief points produced the fastest and best policy, both in terms of number of expansions and solving time. The other three may have either better policy or shorter solving time against number of expansions, therefore the difference between the quality of policy against solving time is not obvious.

In the Maze20 problem, (figures 4.11 and 4.12), it appears that heuristics using midpoints improved the policy faster, while that using next-step belief points produced the best policy. On the other hand, the heuristics using previous-step performed most poorly in this problem.

We can see that there is a clear difference of the time spent on each expansion between different expansion heuristics. However, when comparing the time spent for obtaining the policy of the same quality, the time difference is not significant, especially with different POMDP problems. It is possible that a different generation heuristic method is better in a different problem. and it requires further investigation to confirm this.

4.6.6 Combined k-Largest Improvement Expansion

Since different generation heuristics may perform better in different problems, we come up with an idea that uses all generation heuristics with kLI expansion. We also combine the selection heuristics of using last-step value function and QMDP approximation, so that PBUA switches the use of these estimation methods when the improvement in last update falls below 0.01 (This combined estimation heuristics is denoted by the second digit 3, while the original backup estimation heuristics is denoted by 0).

From figures 4.13 and 4.16, we see that the combined heuristics using backup as estimation, as expected, spent the most computation time. The computation time of the combined generation heuristics with combined estimation method lied between the others with single generation heuristic method with backup estimation. This shows that the faster computation time of estimation using last-step value function and QMDP approximation offsets the larger number of generated points.

However, the quality of the policy found by the combined generation heuristics is disappointing. We can see from figures 4.8, 4.9, 4.11, and 4.12 that it is worse

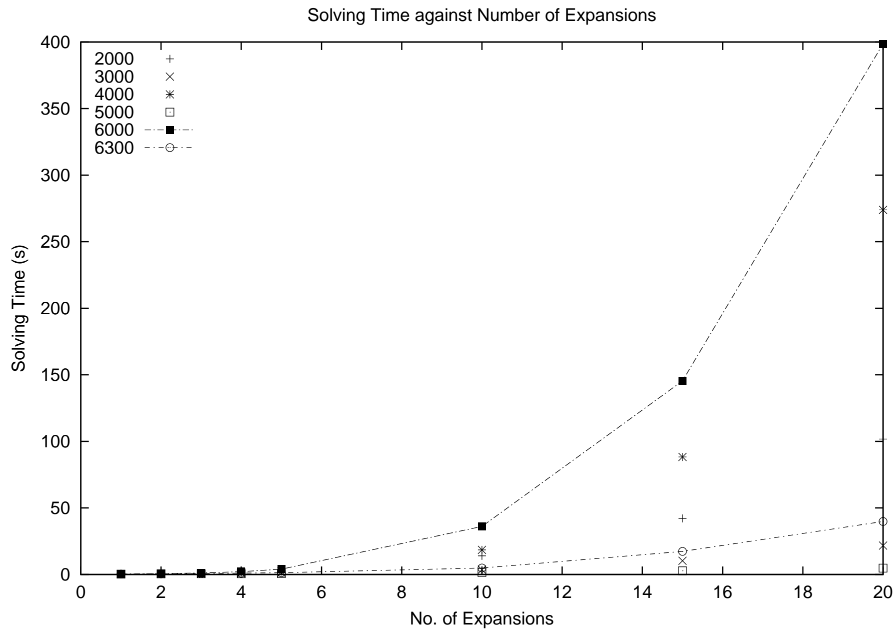

Figure 4.13: Solving Time using combined kLI Expansion in Zhang's Maze

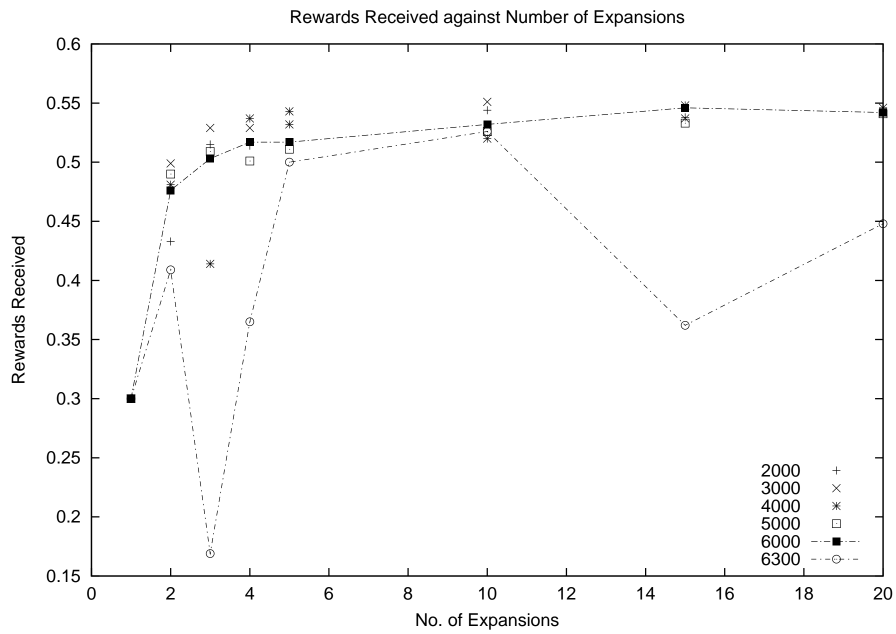

Figure 4.14: Rewards Received using combined kLI Expansion in Zhang's Maze

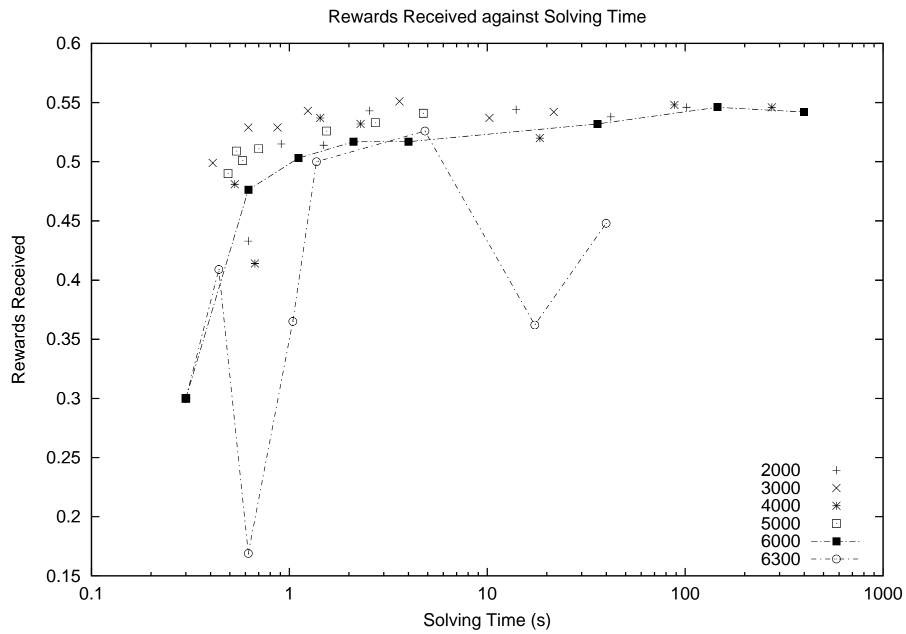

Figure 4.15: Rewards Received against Solving Time using combined kLI Expansion in Zhang's Maze

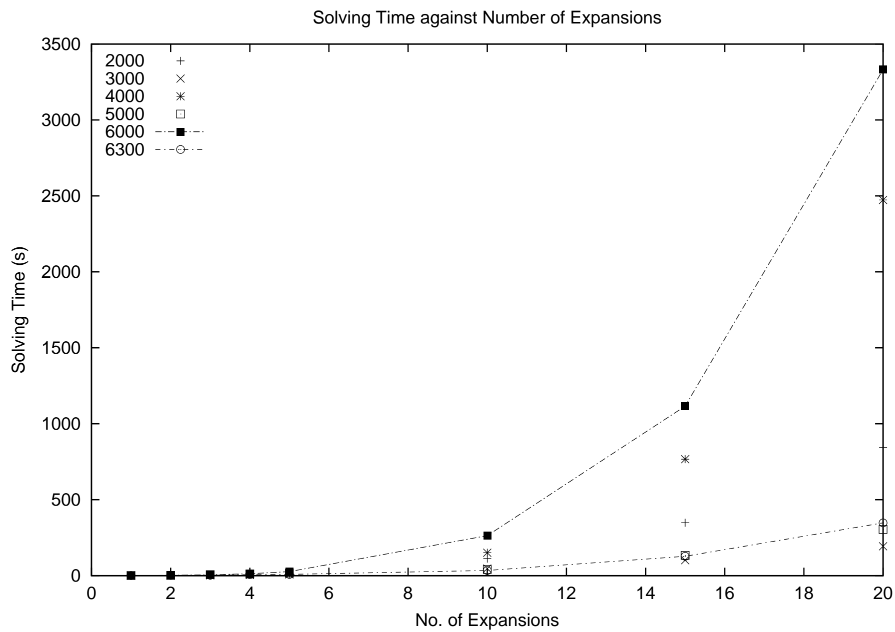

Figure 4.16: Solving Time using combined kLI Expansion in Maze20

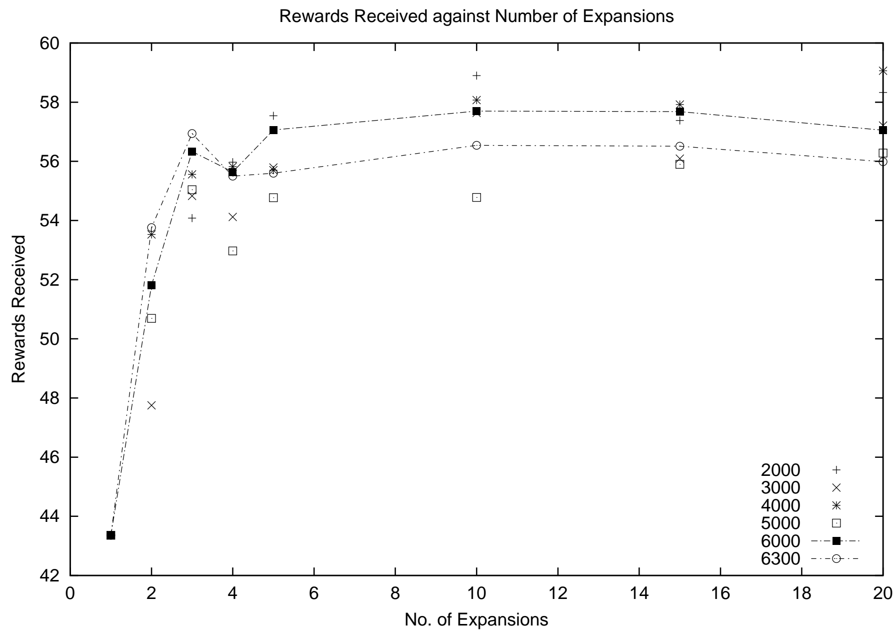

Figure 4.17: Rewards Received using combined kLI Expansion in Maze20

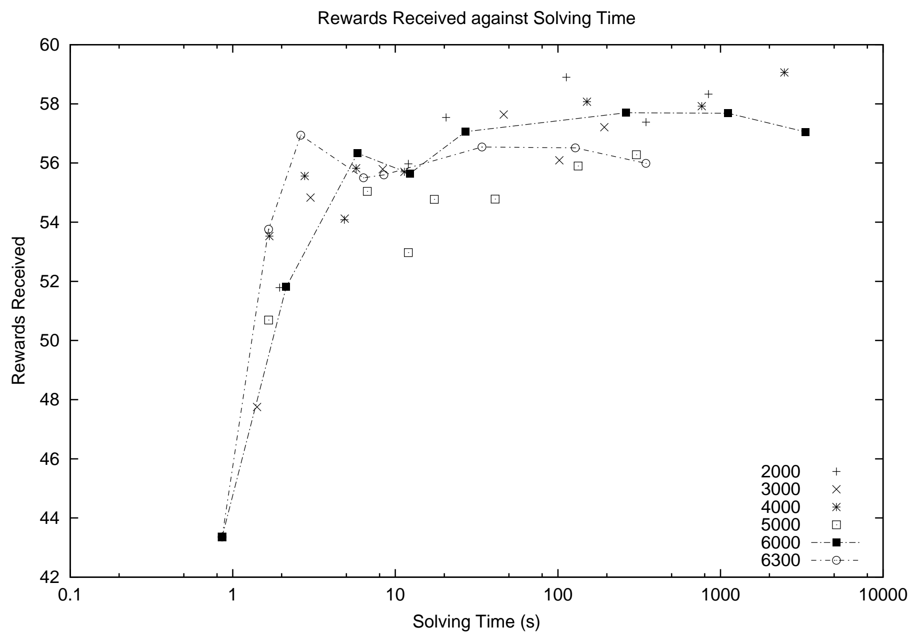

Figure 4.18: Rewards Received against Solving Time using combined kLI Expansion in Maze20

than some of the single generation heuristics.[3] A possible reason is that the using the amount of improvement as selection criterion fails to choose the grid points that improve the policy most. Another possible reason is that the estimation methods fail to estimate the amount of improvement well. Therefore, more less useful points are selected among the larger number of generated points when the generation heuristics are combined. Another observation is that combining the estimation methods using last-step value function and QMDP approximation cannot make it better than estimating with only backup.

4.6.7 Stochastic Simulation

In the next experiment, we compare the stochastic simulation expansion with kLI expansion. Figures 4.19 and 4.22 show that the time used by stochastic simulation expansion is generally less than using kLI expansion. However, from figures 4.20, 4.21, 4.23, and 4.24, we can observe that although not all the generation heuristics with kLI expansion was better than stochastic simulation expansion, at least one generation heuristic method with kLI expansion did. Therefore, we can conclude that stochastic simulation expansion failed to outperform the kLI expansion.

4.6.8 Comparison of Different Expansion Heuristics

The last experiment compares all different expansion heuristics. We also included the random grid expansion (with extreme points) for comparison. Figures 4.19 and 4.22 show that the random grid expansion took the least time in expansion, as we expect. However, from figures 4.20, 4.21, 4.23, and 4.24, we can see that only random grid expansion was slightly less effective than the best of kLI expansion. This shows that our expansion heuristic methods do not guide the expansion very well. Another possibility is that that the policy improvement is not very sensitive to how the grid expands.

4.7 Discussions and Summary

This chapter discusses various heuristic methods for expanding the grid. From the experimental results, we can see that the policy is improved with a larger grid. Also, PBUA is faster with expansion than with a fixed grid without expansion.

[3]This is shown by the existence of data points above the curves.

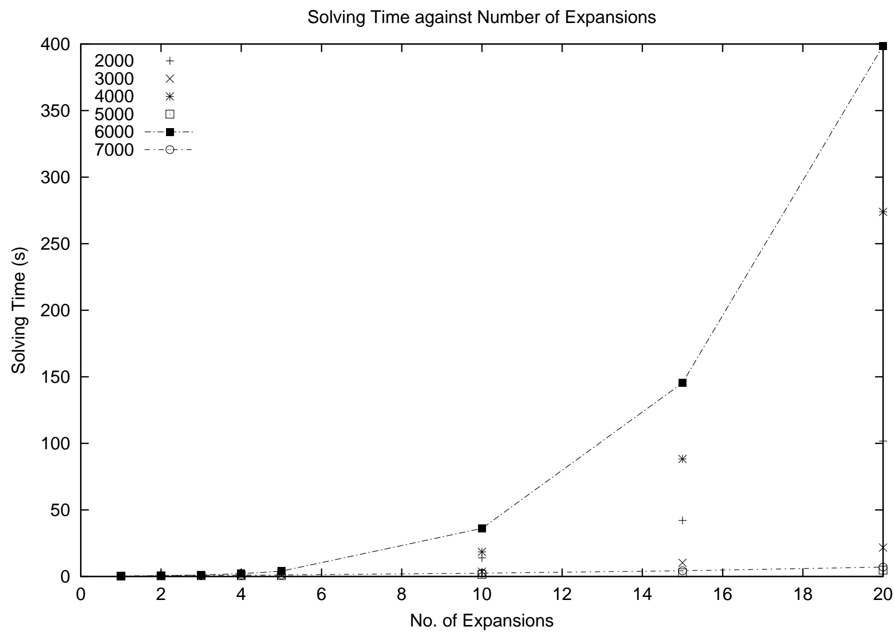

Figure 4.19: Solving Time using Stochastic Simulation Expansion in Zhang's Maze

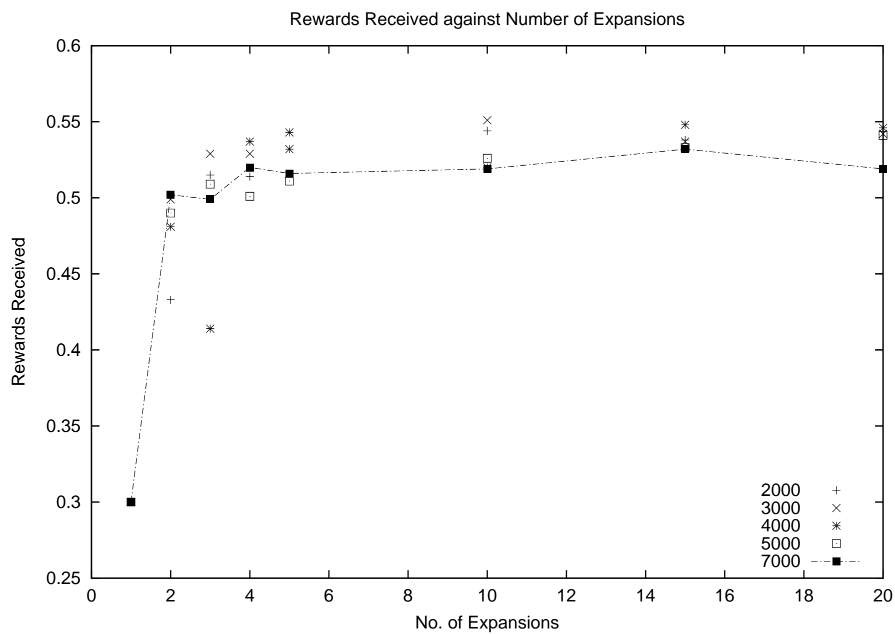

Figure 4.20: Rewards Received using Stochastic Simulation Expansion in Zhang's Maze

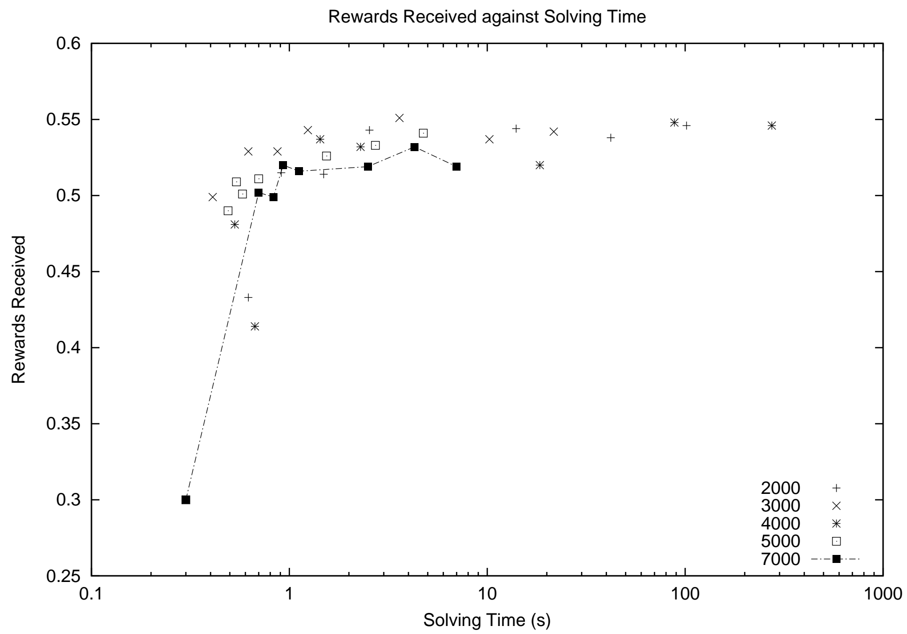

Figure 4.21: Rewards Received against Solving Time using Stochastic Simulation Expansion in Zhang's Maze

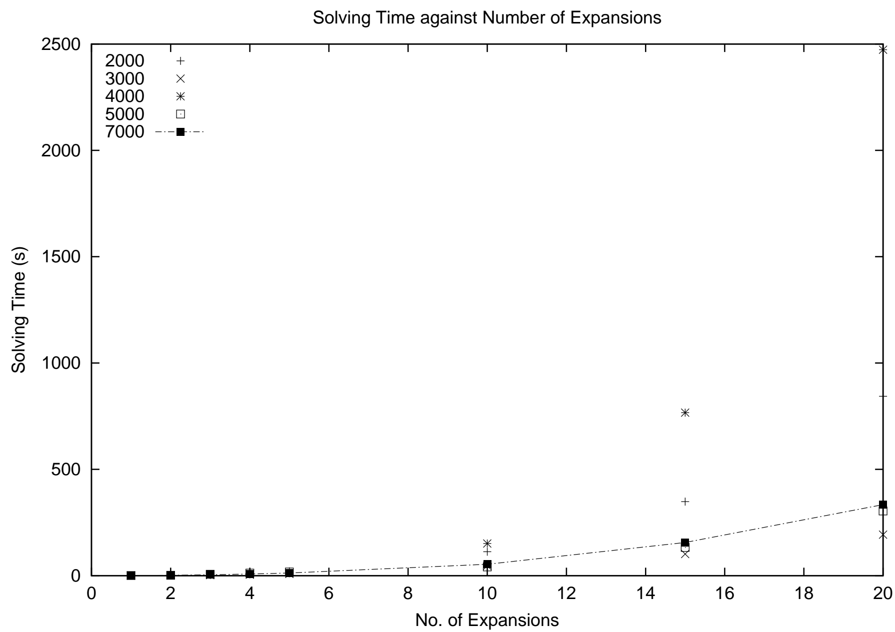

Figure 4.22: Solving Time using Stochastic Simulation Expansion in Maze20

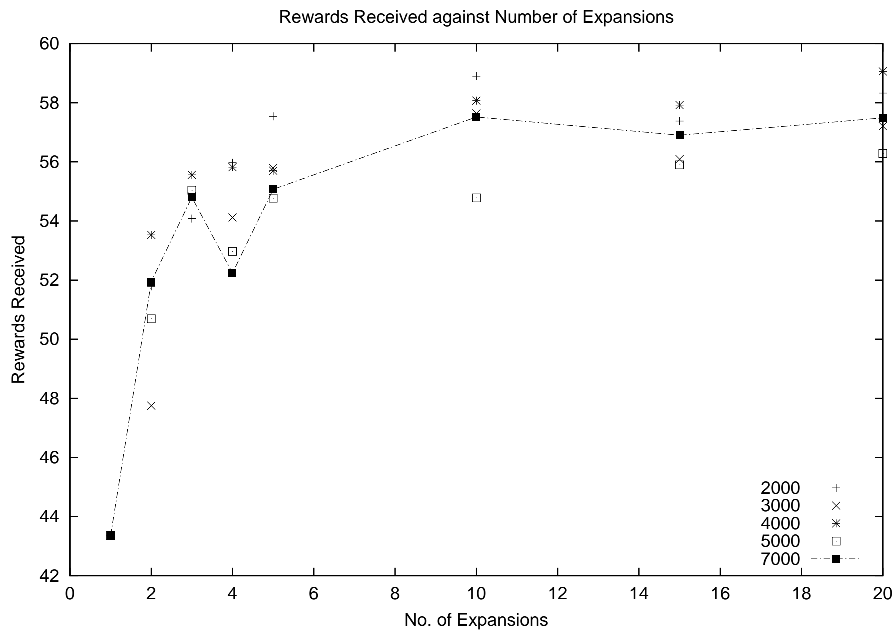

Figure 4.23: Rewards Received using Stochastic Simulation Expansion in Maze20

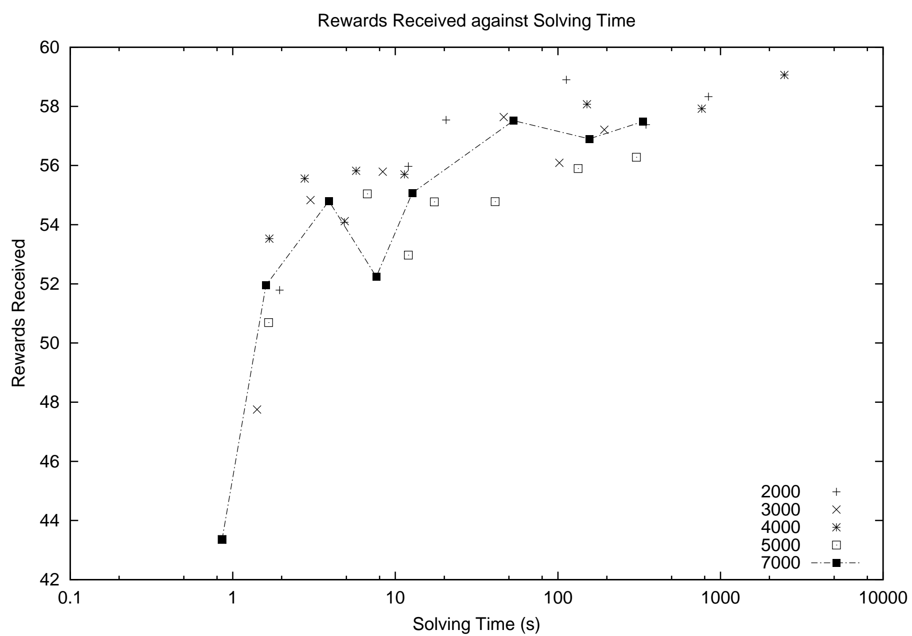

Figure 4.24: Rewards Received against Solving Time using Stochastic Simulation Expansion in Maze20

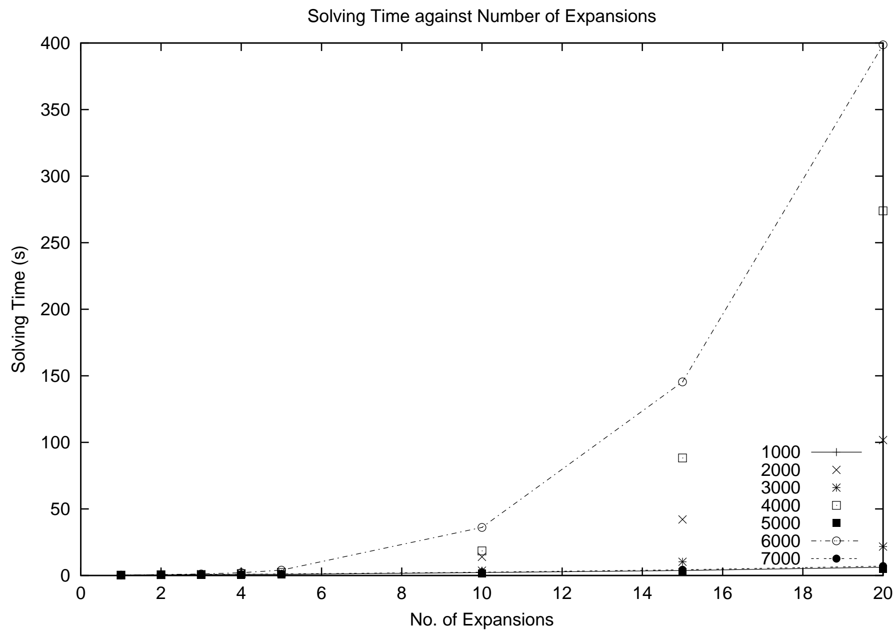

Figure 4.25: Solving Time using Different Expansion Heuristics in Zhang's Maze

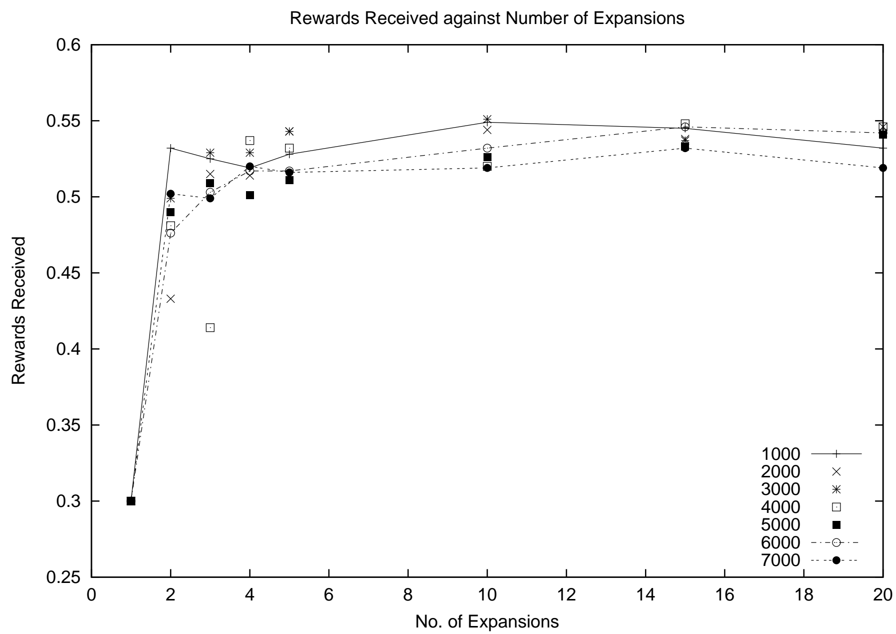

Figure 4.26: Rewards Received using Different Expansion Heuristics in Zhang's Maze

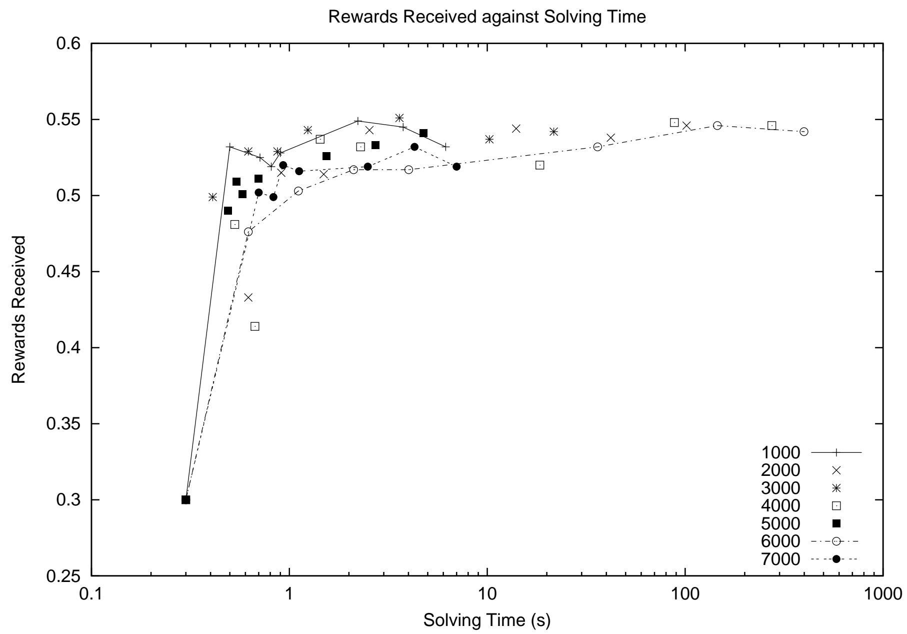

Figure 4.27: Rewards Received against Solving Time using Different Expansion Heuristics in Zhang's Maze

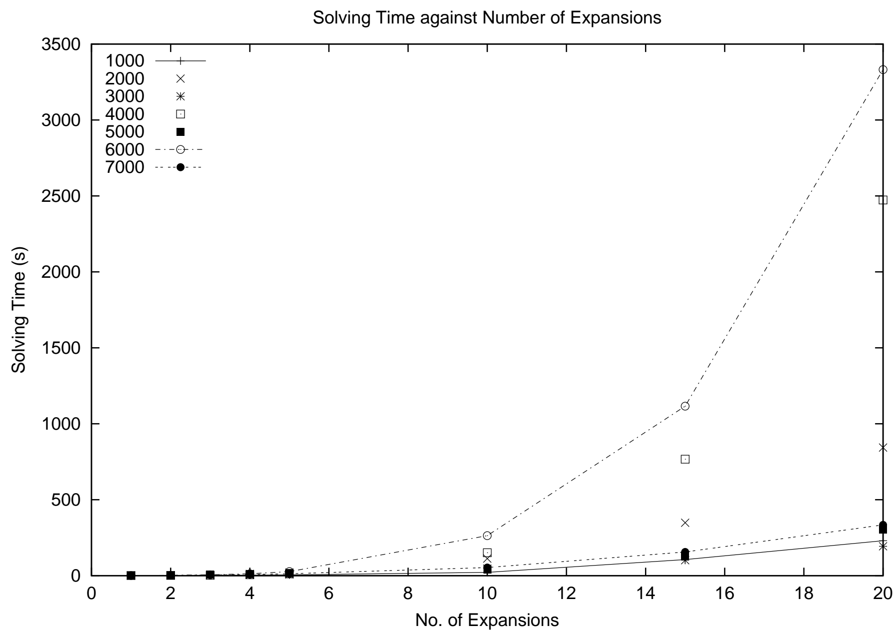

Figure 4.28: Solving Time using Different Expansion Heuristics in Maze20

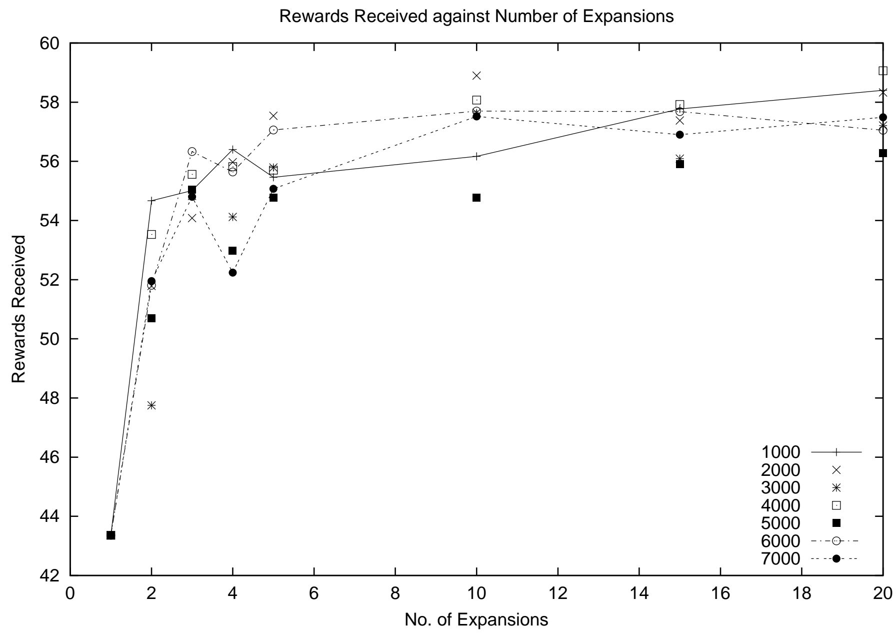

Figure 4.29: Rewards Received using Different Expansion Heuristics in Maze20

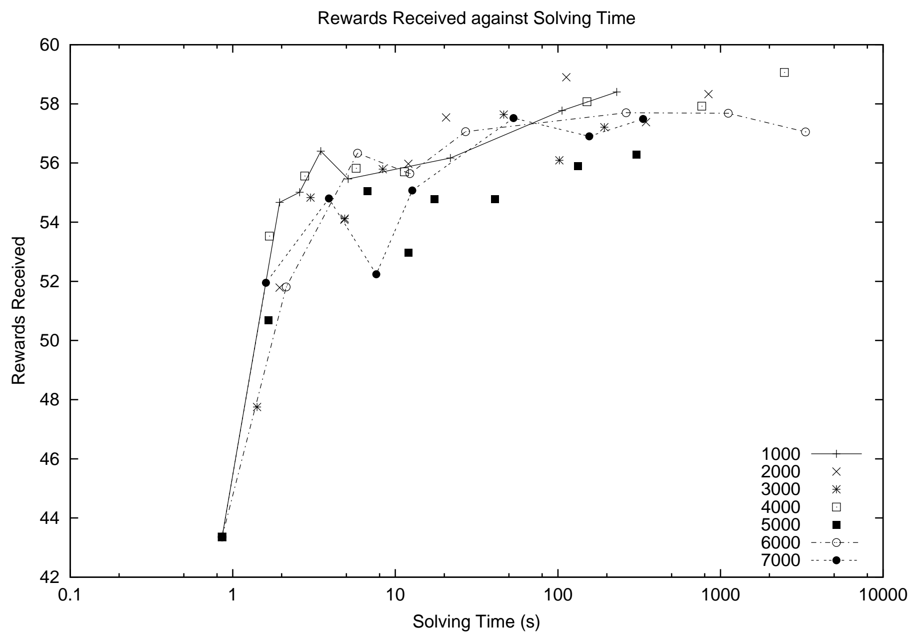

Figure 4.30: Rewards Received against Solving Time using Different Expansion Heuristics in Maze20

Compared with the exact algorithms, PBUA is much faster. Moreover, the quality of policy obtained by PBUA can be similar to that of exact algorithms. This makes PBUA satisfactory for use in different POMDP problems, because in the small problems, PBUA is much faster and does not compromise the quality of policy; and in the larger problems, it is the only choice, as exact algorithms can take an unacceptable computational time.

We present some experiments to compare the different heuristics. However, none of them is significantly better than the others. kLI appears to be the best, but it is only slightly better than a random grid expansion. This shows that the expansion does not benefit much from using a guided expansion rather than a random expansion.

We also compare the different estimation methods for kLI expansion. The backup estimation method is better than the others, as it leads to a faster and better policy, despite of its slower expansion. However, this estimation method can become significantly slower when the state space becomes much larger, since the time complexity of backup depends on $|\mathcal{S}|^2$. Therefore, the other two estimation methods can become more useful when the state space is larger.

Although this chapter does not provide a clear conclusion, it demonstrates the significance of using expansion and provides some ideas on how the grid can be expanded and demonstrates These ideas can be elaborated and may lead to a further improved PBUA with more investigations.

Chapter 5

Possible Enhancements

5.1 Introduction

In the last chapter, we incorporate the expansion mechanism to PBUA, and show that it increases overall performance of PBUA. In this chapter, we examine two other possible enhancements, which people may have thought of. We do some experiments on them to test if they are useful to PBUA.

5.2 Lookahead Control

5.2.1 Overview

Using a lookahead control improves the policy found by some heuristics algorithms [3, 8]. Therefore, this is the first possible enhancement we look into.

A n-step lookahead control means selecting the action that gives the best expected n-step future cumulative reward. In particular, the one-step lookahead policy $\mu : \mathcal{B} \rightarrow \mathcal{A}$ is found by

$$\mu(b) = \arg\max_{a \in \mathcal{A}} \left\{ \rho(b, a) + \gamma \sum_{z \in \mathcal{Z}} P(z|b, a) V(b_a^z) \right\}. \tag{5.1}$$

However, we can see that finding a one-step lookahead control can be a computational burden during runtime, especially when any of $|\mathcal{S}|$, $|\mathcal{A}|$, and $|\mathcal{Z}|$ is large. It may take $O(|\mathcal{A}||\mathcal{S}|^2|\mathcal{Z}|T(V))$ time, where $T(V)$ is the time for computing the value of b using the value function V. In many heuristic algorithms, the time for computing $V(b)$ is the same as the time for computing $\pi(b)$. Therefore, the time for computing a one-step lookahead can be $|\mathcal{A}| \times |\mathcal{S}|^2 \times |\mathcal{Z}|$ times of that for computing a direct policy $\pi(b)$ in the worst case. This shows that using a lookahead control is a trade-off between the quality of policy and the reaction time.

5.2.2 Experimental Results

We have conducted some experiments to test the improvement gained by using lookahead control. The setup of the experiments of PBUA are the same as that in the last chapter. In addition to the first 2 digits introduced in last chapter, the fourth digit refers to the use of lookahead control, where 0 means using direct control and 1 means using 1-step lookahead control. The same value function is used for lookahead control and direct control for the same number of expansions. The data points in the figures correspond to the 1st–5th, 10th, 15th, and 20th expansions.

From figures 5.1 and 5.3, we can observe that lookahead control produced a much better policy than direct control for the first expansion. However, it did not perform significantly better afterwards and can be even worse than the direct control. A possible reason is that PBUA can find a close-to-optimal policy after only a few expansions. Therefore, the lookahead control cannot improve the policy and may become worse due to the randomness in the simulation.

We can also see that the reaction time for using a lookahead control was roughly 10 times of that for using a direct control from figures 5.2 and 5.4. Note that the reaction time is an indirect measurement. It was measured by dividing the total simulation time by the total number of steps taken by the agent. Therefore, it includes the time for simulating the environment transitions. Nevertheless, it gives a good estimate to the reaction time since the simulation time in each step is the same for both direct control and lookahead control. Another observation is that the reaction time increased with the number of expansions. This is expected since the reaction time depends on the size of the vector set, which depends on the number of grid.

5.2.3 Discussions

We see that the lookahead control does not perform better except for the first few expansions. However, it does not further improve the best policy obtained by using only direct control. Moreover, it does not save too much solving time, since the first few expansions usually do not take much time. On the other hand, it increases the reaction time considerably and probably even more when the POMDP problem is larger. Although the reaction time is much less than a second, this increase can becomes significant sometimes, for example, when the

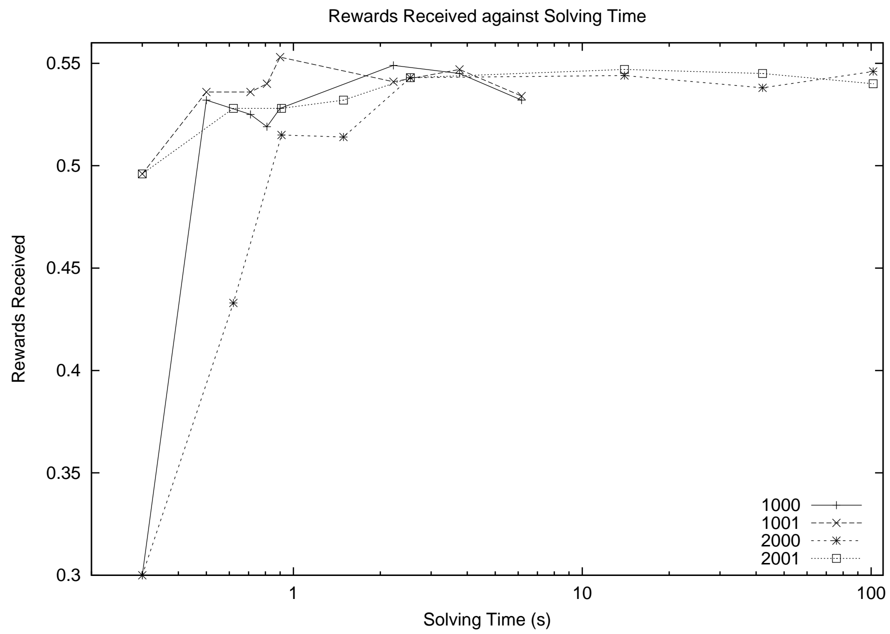

Figure 5.1: Rewards Received against Solving Time for using PBUA with and without Lookahead Control in Zhang's Maze

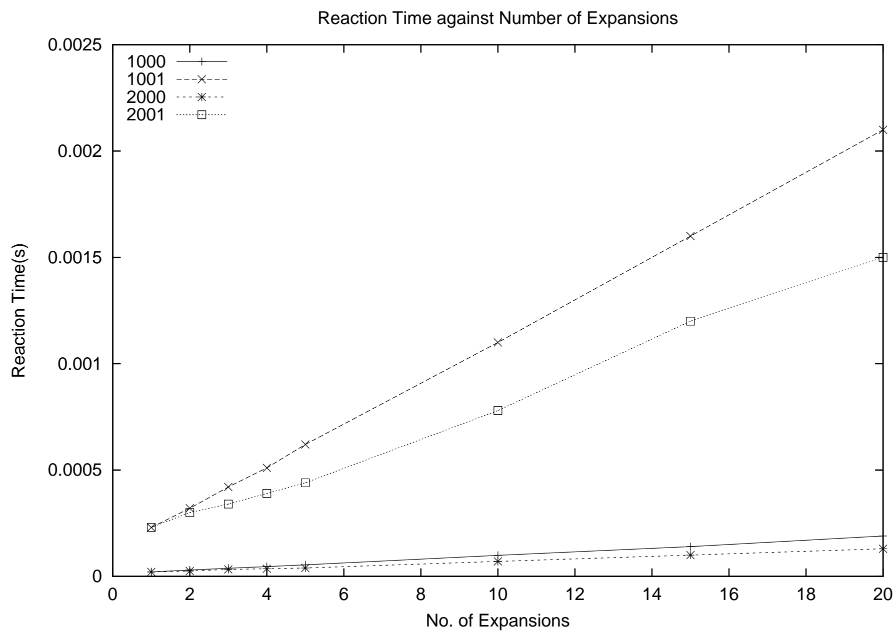

Figure 5.2: Reaction Time for using PBUA with and without Lookahead Control in Zhang's Maze

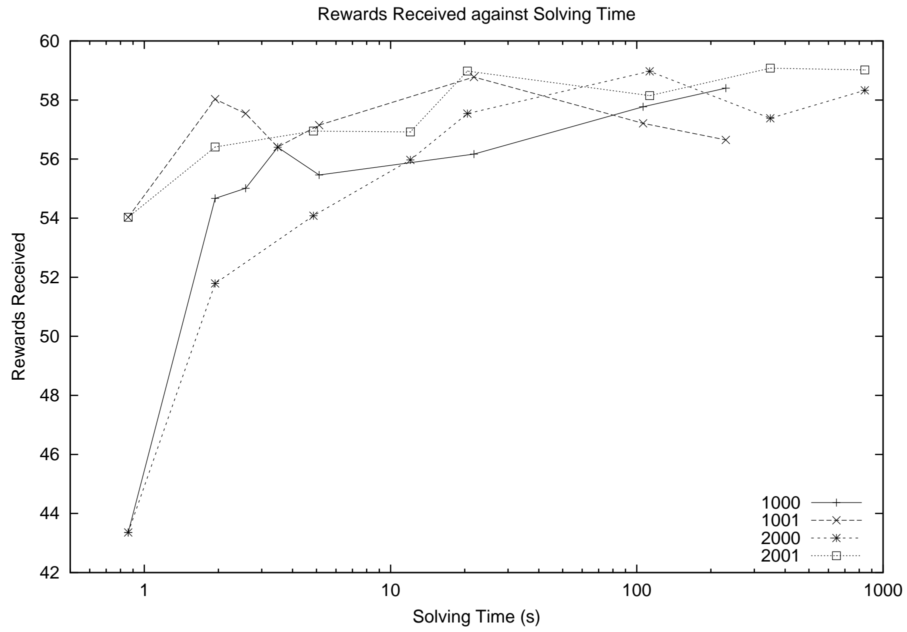

Figure 5.3: Rewards Received against Solving Time for using PBUA with and without Lookahead Control in Maze20

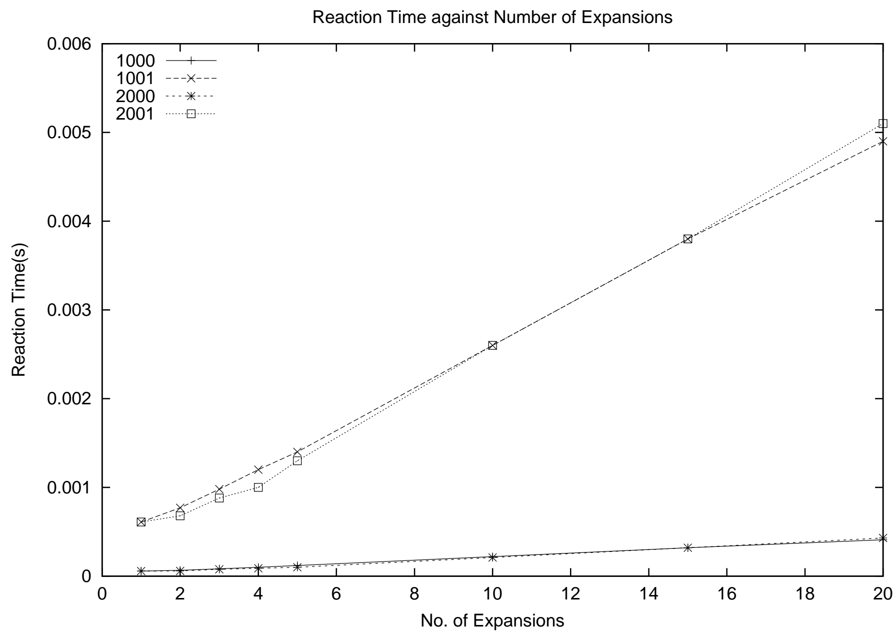

Figure 5.4: Reaction Time for using PBUA with and without Lookahead Control in Maze20

agent has to make many decisions in a short time, when the computer has to make decisions for many agents at a time, or when the control is built on some low-end hardware. Therefore, it is worth to spend more solving time for keeping the reaction time low.

Nevertheless, lookahead can still be useful sometimes, when the solving time should be kept as short as possible. For instance, this is desirable when the agent has to adapt to different (or changing) POMDP problems quickly, and does not have to act in the same POMDP problem for many steps.

Another possible conclusion that we can draw from the observation is that the policy that PBUA obtained is possibly already close to the optimal, because lookahead fails to further improve the policy with only direct control. It may be the reason why further expansions do not improve the policy obtained by both direct control or lookahead control.

5.3 Sorted Asynchronous Update

5.3.1 Overview

The next possible enhancement that we consider is what we call sorted asynchronous update (SAU). Synchronous update refers to updating (modifying) the value function only after the whole update step in value iteration has completed. Before that, it performs backup based on the last-step value function. Therefore, every grid point is getting the same amount of time for update, but some of them may lead to only a small improvement, and much time may be wasted on those grid points. This may be more significant when the grid becomes larger, since the proportion of these less useful points may be larger.

Asynchronous update, on the other hand, modifies the value function immediately after each backup during an update step. SAU belongs to this kind of update. In SAU, the grid points are sorted in the descending order of the improvement to the values of the grid points that they give in the last backup. The grid point that gives the largest improvement is backup next. This makes not every grid point is backed up for the same number of times. Those that give larger improvement are backed up at more.

The procedure for value iteration using SAU is given in table 5.1, with a few modification to the ordinary update, which is given in table 3.2. In this procedure,

we use a heap $\mathcal{H}$ for ordering the grid points for backup. Initially, the amounts of improvement δ_i for all grid points are set to the maximum representable integer and are pushed to $\mathcal{H}$. Therefore, each grid point are backed up once at the beginning. After the first cycle, the points are pushed to $\mathcal{H}$ and is ordered by the δ_i in the previous backup. This procedure stops until m is equal to the size of the grid. This ensures that every point has an improvement δ_i less than ϵ in the last backup.

5.3.2 Experimental Results

We have conducted some experiments to test the effectiveness of using SAU. We tested it in Maze20 problem, and Hallway2 problem, which is larger and has 92 states. We used PBUA with random expansion with two variants of value iteration, one with SAU and one without SAU. The results are shown in figures 5.5 and 5.6. The data points in the figures correspond to the 1st–5th, 10th, 15th, and 20th expansions.

We can observe that in the Maze20 problem, PBUA with SAU was faster than than that without SAU in only the first two and the last two expansions. Moreover, the quality of the policy of the former method was better than the latter one. In the Hallway2 problem, PBUA with SAU is slower than that without SAU, and the quality of the policy obtained by neither of them was significantly better.

5.3.3 Discussions

The results show that SAU does not necessarily speed up PBUA, though it is slightly faster in some cases. It is possibly because not many of the grid points give little improvement and the overhead of using a heap can outweigh the reduced number of backup. Another concern is that asynchronous update may distort the shape of the value function, and may lead to a poorer policy, though our experiments do not show this. Moreover, SAU makes the PBUA slightly more complicated. As a result, even though SAU might speed up PBUA in some cases, we do not think it can enhance PBUA generally.

PBVI-SAU(G, U, ϵ)
Input: G is the grid of belief points, $\mathcal{U}$ is the set of vectors before update, ϵ is the stopping threshold
Output: A set of vectors $\mathcal{V}$ after update, and the maximum amount of cumulative improvement $\Delta_{\max}$ for a grid point in the whole update

$\Delta_i \leftarrow 0$ and $\delta_i \leftarrow \max$ for all $1 \le i \le |G|$
Let $\mathcal{H}$ be a heap which is ordered by the improvement δ_i of the point elements b_i^G
Push b_i^G to $\mathcal{H}$ for all $b_i^G \in G$
Let m be the number of consecutive points that have improvement less than ϵ in the last backup and with initial value 0
$\mathcal{V} \leftarrow \mathcal{U}$

repeat
  $b_i^G \leftarrow$ a grid point, which has the maximum last improvement δ_i, popped from $\mathcal{H}$
  $\beta_i \leftarrow \texttt{backup}(b_i^G, \mathcal{V})$

  if $\beta_i \cdot b_i^G > \alpha_i \cdot b_i^G$
    $\delta_i \leftarrow \beta_i \cdot b_i^G - \alpha_i \cdot b_i^G$
    Replace $\alpha_i \in \mathcal{V}$ by β_i
  else
    $\delta_i \leftarrow 0$

  if $\delta_i > \epsilon$ **then** $m \leftarrow 0$
        else $m \leftarrow m + 1$
  $\Delta(b) \leftarrow \Delta(b) + \delta(b)$
  Push b_i^G to $\mathcal{H}$
until $m >= |G|$

return $\{\mathcal{V}, \max_i \Delta_i\}$

Table 5.1: Procedure for Value Iteration with SAU (PBVI-SAU)

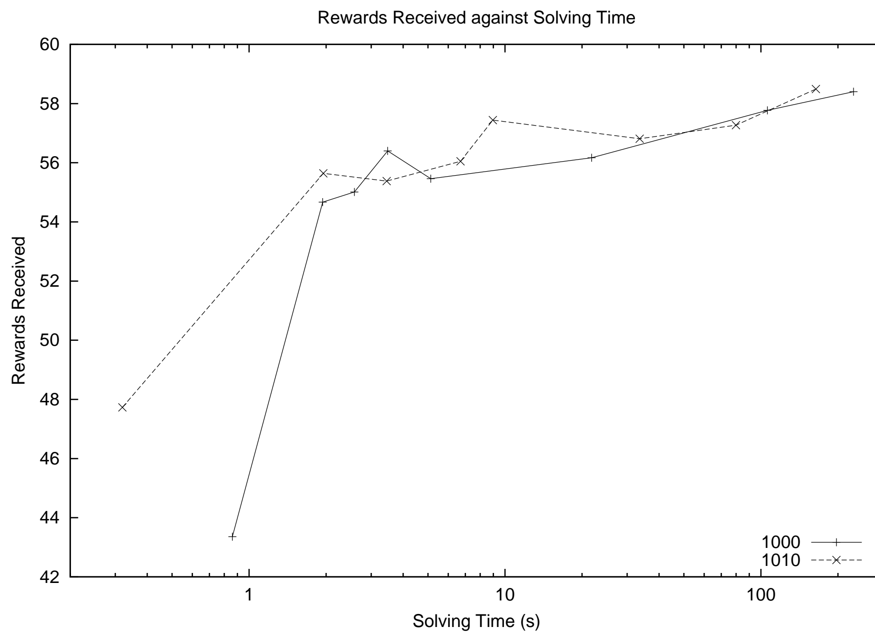

Figure 5.5: Rewards Received against Solving Time using PBUA with and without SAU in Maze20

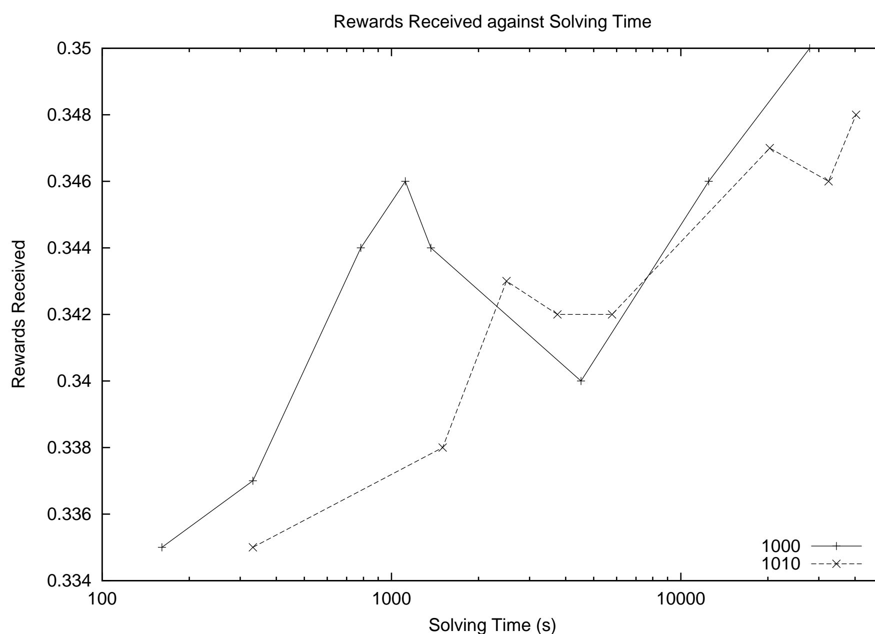

Figure 5.6: Rewards Received against Solving Time using PBUA with and without SAU in Hallway2

5.4 Summary

In this chapter, we discuss two possible enhancements. However, both of them cannot significantly improve the PBUA algorithm. The lookahead control may improve the quality of the policy, but it can increase the reaction time of the agent considerably. Moreover, since PBUA can give a policy of a similar quality to one using lookahead control after some expansions, and these expansions do not take much time, it is worth to spend more solving time rather than allow an increased reaction time. Thus, lookahead control should be used only when a very short solving time is required and a longer reaction time is acceptable.

Another enhancement we discuss is sorted asynchronous update. However, experimental results show that it does not necessarily reduce the solving time and it complicates the PBUA algorithm. As a result, it should not be used generally.

Chapter 6

Related Works

6.1 Introduction

In the previous chapters, we study a heuristic algorithm named PBUA. The idea of using heuristic algorithms is not new, and there are many related works in the literature. In this chapter, we look into these related heuristic algorithms and emphasize the differences between PBUA and them.

All the heuristic algorithms discussed in this chapter uses value iteration to compute an approximation to the optimal value function. Those approximations can be divided into two kinds: upper bound estimates and lower bound estimates. Although both bounds are computed by value iteration, the value functions are represented in different ways and the values of the belief states are computed using different methods.

The upper bound value function is represented by a set of grid-value pairs. A value is associated with each grid point, and this value is updated and improved using value iteration. The value of a belief state other than the grid points is computed by interpolation. On the other hand, the lower bound value function is represented by a set of vectors. Backup is applied to the grid points and the backup vectors, along with the old value function, are used to construct an updated vector set in each step of value iteration.

Both the upper bound method and the lower bound method have to decide how to construct the grid. Moreover, the upper bound method has to consider the interpolation method, while the lower method has to consider the update step in value iteration. In the discussion that follows, we stress on the different strategies the heuristic algorithms uses to solve these problems.

6.2 Fixed-Resolution Regular Grid Approach

The first heuristic algorithm we discuss is fixed-resolution regular grid approach proposed by Lovejoy[12]. An upper bound and a lower bound is computed for the optimal value function using a regular grid with fixed resolution.

The lower bound value function is represented by a set of vectors. In each update step of the value iteration, backup vector are computed for all grid points. The updated vector set is then constructed by putting these backup vectors into a set, without adding any vector from the original vector set. This is how this approach solves the update problem.

The interpolation used in computing the upper bound value function is Freudenthal triangulation, which is elegant and efficient. However, Freudenthal triangulation requires a regular grid. This approach thus uses a regular grid with fixed resolution, due to this constraint. Let M be a positive integer that represents the resolution of the grid. Also, let $I_+^{|\mathcal{S}|}$ denote the set of $|\mathcal{S}|$-vectors that have non-negative integers as their elements. A fixed-resolution regular grid G is defined by

$$G = \left\{ b = \left(\frac{1}{M}\right) m | m \in I_+^{|\mathcal{S}|}, \sum_{s \in \mathcal{S}} m(s) = M \right\}. \tag{6.1}$$

This grid G divides the belief simplex $\mathcal{B}$ into a set of equal-size sub-simplices. The number of the grid points is given by

$$|G| = \frac{(M + |\mathcal{S}| - 1)!}{M!(|\mathcal{S}| - 1)!}. \tag{6.2}$$

The main disadvantage of this heuristic algorithm is the exponential increase in $|G|$ with the resolution M. Therefore, only a grid with small resolution can be used practically, and the error of the bounds cannot be refined arbitrarily. This is different from PBUA, which allows a variable grid. Moreover, this algorithm updates the vector set, which represents the lower bound, in a different way from PBUA. It replaces the old vector immediately with the backup vector, but PBUA chooses the vector, between the old vector α_i and backup vector β_i, that gives a larger inner product with the backup point. Therefore, the lower bound value function computed by PBUA approaches to a fixed value function, but that computed by the fixed-resolution regular grid approach does not.

	Tiger	Manufacturing	Network	Shuttle
QMDP - T_s (s)	< 0.01	< 0.01	0.02	< 0.01
QMDP - Reward	19.0	7.64	284	32.6
Optimal - Reward	19.5	7.94	294	32.6

Table 6.1: Performance of QMDP Approximation

6.3 QMDP Approximation

Another heuristic algorithm using QMDP is proposed by Littman[11]. This algorithm first computes a Q-value function for the underlying MDP, using equation 2.13. The Q-value for a belief state is then computed by

$$Q(b,a) = \sum_{s \in \mathcal{S}} Q(s,a) b(s), \tag{6.3}$$

and the policy is constructed by

$$\pi(b) = \arg\max_{a \in \mathcal{A}} Q(b,a). \tag{6.4}$$

QMDP approximation can be viewed as a grid-based approximation, where the grid consists of only the $|\mathcal{S}|$ extreme points. $|\mathcal{A}|$ number of Q-values are associated with each grid point, and the interpolation is given by equation 6.3. Moreover, QMDP has the same property with those heuristic algorithms with sets of grid-value pairs. The value function given by QMDP is an upper bound for the optimal value function[8].

Since to obtain the Q-value function requires solving a finite-state MDP only, it is considerably fast compared to many algorithms for POMDP. The performance of QMDP approximation is shown in table 6.1. We can see that this algorithm required only very little time compute a policy. Also, the quality of the policy was close to that of the optimal policy.

To compare the performance of PBUA and QMDP approximation, we have done some experiments and tested them in the Zhang's Maze problem and the Maze20 problem. The experimental results are shown in table 6.2. We used both direct control and one-step lookahead control for the QMDP approximation. In the table, PBUA-fast refers to a policy that was computed relatively fast but near to the optimal, while PBUA-best refers to the best policy that was computed in 20 expansions. The expansion heuristic methods used in PBUA were kLI

	Zhang's Maze			Maze20		
	T_s	Rewards	T_r	T_s	Rewards	T_r
Random	-	0.005	1.0e-5	-	33.8	1.9e-5
QMDP-direct	< 0.01	0.002	1.5e-5	0.03	34.1	2.9e-5
QMDP-lookahead	< 0.01	0.015	1.4e-4	0.03	50.1	3.2e-4
PBUA-fast	0.41	0.499	3.0e-5	1.94	51.7	5.9e-5
PBUA-best	3.60	0.551	9.6e-5	112	58.9	2.1e-4

Table 6.2: Comparison between QMDP and PBUA

with preceding belief points and kLI with successive belief points for the Zhang's Maze problem and the Maze20 problem respectively. Both used the backup as estimation. The policies are selected from those obtained after the 2th–10th expansions.

As we can see, the solving time for QMDP approximation is very small. However, the quality computed by QMDP approximation is not good and is near to that of a random policy. It becomes better when a lookahead control is used, but is still worse than that computed by PBUA. Moreover, using a lookahead control makes the reaction time for QMDP approximation larger than that for PBUA. However, when the number of expansions in PBUA increased, the number of grid points and vectors increased, and the reaction time of a PBUA control became near to that for QMDP approximation after 10 expansions.

QMDP approximation allows a very fast heuristic algorithm, but fails to perform well unless a lookahead control, with an increase in reaction time, is used. Moreover, the quality of the policy is still worse than that can be found by PBUA even though a lookahead control is used. The main problem is that it does not allow an expansion of grid points, as in PBUA, to improve the policy found by it. Therefore, QMDP is desirable when a very small solving time is required. Otherwise, we think that PBUA can give a solution by providing both a better policy and smaller reaction time. Also, PBUA allows a more flexible trade-off between solving time and quality of policy.

6.4 Incremental Linear Function Approach

Hauskrecht also proposes a heuristic algorithm, namely incremental linear function approach (ILFA), which constructs a lower bound value function[7, 8]. ILFA is similar to fixed-resolution regular grid and it backs up at the grid points to up-

date the value function. However, it adds all new backup vectors to the original vector set to construct the updated vector set. In other words, the new vector set contains the new backup vectors and the vectors in the old vector set. Let $\mathcal{V}'$ and $\mathcal{V}$ be the vector sets before and after update. Since $\mathcal{V}' \subset \mathcal{V}$, it implies $V \geq V'$. This means each update improves the value function and thus the value function converges. After each update step in the value iteration, the useless vectors are pruned using linear programming.

ILFA uses a fixed grid, which means the grid points remains unchanged. The grid points are selected using different heuristics initially. However, like PBUA, experiments show that ILFA differs little when using different grid construction heuristics.

ILFA and PBUA are similar in the way that both of them back up at the grid points to perform an update step in value iteration. However, the way to update the new value function is different. PBUA maintains only one vector for each grid point, while ILFA adds the new vectors to the original vector. Therefore, $|\mathcal{V}|$ remains unchanged after each update step in PBUA while $|\mathcal{V}|$ increases at most by $|G|$ after each update step in ILFA. Moreover, linear programming is used to prune the resulting vector set after each update step in ILFA, but not in PBUA. The overall result is that ILFA maintains monotonicity over the whole belief space, while PBUA maintains it over only the grid points.

Another difference between ILFA and PBUA is that ILFA uses a fixed grid but PBUA uses expansion to allow a larger grid after the value function converges. Moreover, instead of using a converged value function as in PBUA, ILFA uses the value function after each update to derive the policy. ILFA cannot use a converged value function because although the update method ensures that the value function converges, it may take a long time before so.

Experiments were done to compare the performance between ILFA and PBUA, and the results are shown in figures 6.1 and 6.2. In the experiments, the grid sizes used in ILFA were multiples (1, 3, and 5) of the size of state space. The grid size with multiple 1 consists of extreme points only, while the other two consist of random points in addition to the extreme points. They are denoted by {ILFA<multiple>} in the figures. Note that these grids remained the same during the update steps. The expansion heuristics we used for PBUA was random expansion. In the figures, the data points for ILFA represent the value functions after the 1st, 5th, 10th, 15th, 20th, 25th, and 30th update cycles in the value

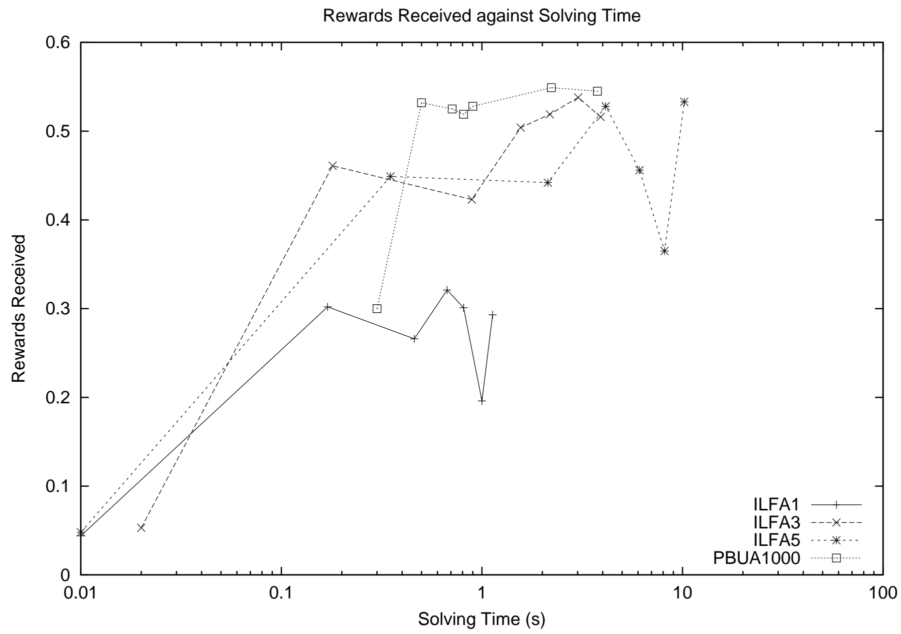

Figure 6.1: Rewards Received against Solving Time using ILFA and PBUA in Zhang's Maze

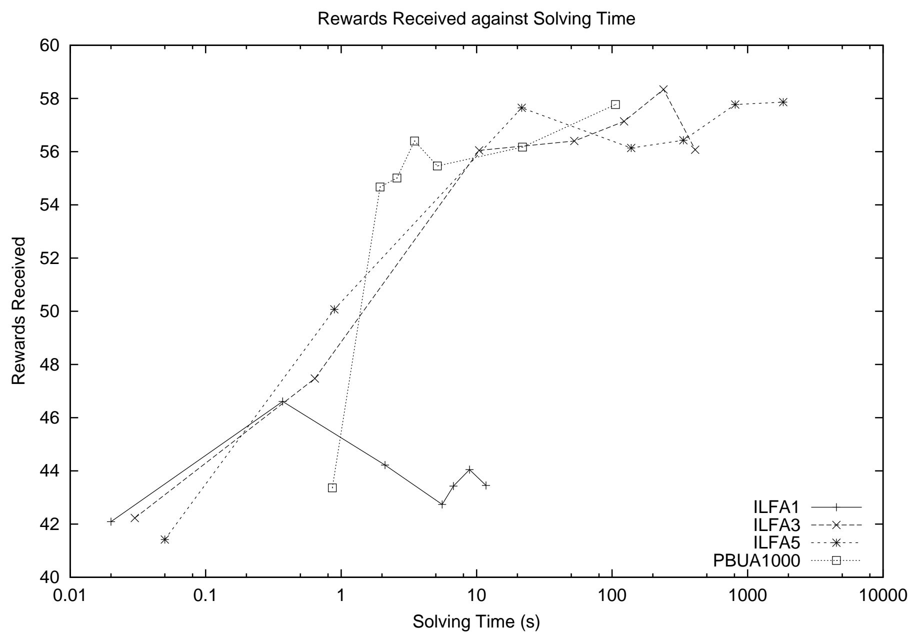

Figure 6.2: Rewards Received against Solving Time using ILFA and PBUA in Maze20

Problem	Method	$\lvert G\rvert$	$\lvert\mathcal{V}\rvert$	$T_s(s)$	$\delta_{\max}$
Zhang's Maze	ILFA1	11	12	1.15	0.26
	ILFA3	33	30	3.91	0.25
	ILFA5	55	52	10.2	0.30
Maze20	ILFA1	20	40	11.7	2.06
	ILFA3	60	346	410	2.30
	ILFA5	100	586	1834	1.83

Table 6.3: Properties of the Value Function of ILFA after 30 Update Cycles

iteration; the data points for PBUA represent the converged value function after the 1st–5th, 10th, and 15th expansions. The initial value function used by both methods were the same, and is given by equation 3.7.

Surprisingly, ILFA gave a fair policy even after only a few update cycles, and the computation time is much faster than that reported in Hauskrecht[8]. In the Zhang's Maze problem, PBUA is better than ILFA after the 2nd expansion. In the Maze20 problem, PBUA was faster to compute a good policy, but ILFA computed the best policy more quickly. Therefore, it is not obvious to say which one is better.

Table 6.3 shows some of the properties of the value function used in ILFA after 30 update cycles in the Zhang's Maze problem and the Maze20 problems. $\delta_{\max}$ is the maximum amount of improvement at the grid points in one update cycle. We can see that it takes much more time for ILFA to converge than for PBUA. For example, for both methods using a grid consisting of extreme points only, it took PBUA 0.3 second to make $\delta_{\max}$ less than 0.01 in Zhang's Maze, while ILFA took 1.15 second to make $\delta_{\max}$ equal to 0.26; it took PBUA 0.86 second to make $\delta_{\max}$ less than 0.01 in Maze20, while ILFA took 11.7 second to make $\delta_{\max}$ equal to 2.06. Another observation is that $\lvert\mathcal{V}\rvert$ might become significantly larger than $\lvert G\rvert$ using ILFA. This large $\lvert\mathcal{V}\rvert$, along with the use of linear programming, accounts for the long convergence time. The update method used in PBUA eliminates these two problems and therefore allows a faster convergence.

Although ILFA has a slow convergence, its policy is still very good. This shows that using vectors to represent the value function can give a good policy even though the value function has not converged. Also, it shows that a closer bound does not guarantee a better policy.

6.5 Hauskrecht's Grid Approach

In addition to ILFA, Hauskrecht proposes another grid approach for computing the upper bound[7, 8]. This grid approach uses a linear point interpolation to find the values of belief states that are not included in the grid. Let $\widehat{V}$ be the interpolated function, and $\widehat{V}(b^G)$ be the value stored for grid point b^G. Since $\widehat{V}$ is an upper bound to the optimal value function, and there are different ways to interpolate the value of a belief state b, this approaches searches for the combination $\bar{\lambda}$ of λ's that gives the minimum value as follows:

$$\widehat{V}(b) = \min_{\bar{\lambda}} \sum_{j=1}^{|G|} \lambda_j \widehat{V}(b_j^G), \tag{6.5}$$

where $0 \leq \lambda_j \leq 1$, $\sum_{j=1}^{|G|} \lambda_j = 1$, and $\sum_{j=1}^{|G|} \lambda_j b_j^G = b$. However, finding this combination of λ's can be relatively slow. Therefore, Hauskrecht limits the space of the $\bar{\lambda}$ to allow a more efficient searching. The interpolation uses only the $|\mathcal{S}|-1$ extreme points and one arbitrary grid point b'. It then finds the combination that gives the minimum value and uses it to compute $\widehat{V}(b)$. The resulting value function is characterized by a sawtooth-shape.

Hauskrecht also uses expansion to improve the grid. The heuristic method used is stochastic simulation. This heuristic method is similar to that used in PBUA and indeed we borrow this idea from Hauskrecht.

This grid approach was tested with the Maze20 problem. The bound quality is reported to be better than that given by QMDP approximation. However, the quality of the policy computed by this grid approach is worse than that by QMDP approximation, when both used lookahead control. The results reported by Hauskrecht are shown in table 6.4. The results for QMDP, PBUA-fast, and PBUA-best are replicated from table 6.1. Note that the times were measured on Sun Ultra 1 Workstation, therefore it should be slower than that measured by us on a PC with K6-3 400MHz CPU. Nevertheless, an indirect comparison shows that the quality of the policy produced by this grid approach is worse than that by PBUA and the solving time could be considerably longer.

Method	$\|G\|$	$T_s(s)$	Reward
Hauskrecht's Grid	40	< 50	40
(with lookahead)	200	~ 500	45
	400	~ 3400	47
QMDP-lookahead	-	(0.03)	50
PBUA-fast	40	(1.94)	52
PBUA-best	200	(112)	59

Table 6.4: Experimental Results of Hauskrecht's Grid Approach in Maze20

6.6 Brafman's Grid Approach

Another grid approach for computing an upper bound is proposed by Brafman[3]. The interpolation used in this approach prefers grid points with less information, since those grid points are usually closer to the belief state b to be interpolated, and closer grid points should give a better interpolated value. This can be done by sorting the grid points into a list in ascending order of the amount of information they encode. One way to estimation this amount of information is by calculating the entropy. After that, the list of grid points are searching from the beginning, until there is a grid point b^G that assigns positive probability only to states that b assigns positive probability. λ_1 is then defined as the maximum constant c that $b - c\ b^G \geq 0$. The process continues by setting $b \leftarrow b - c\ b^G$ to find all the remaining λ's.

To expand the grid, this approach considers whether adding a new belief point to the grid provides useful information. This is determined by how likely a different policy is given and how likely that belief point is reached. The first idea is carried out by comparing for every pair of grid points (b_1^G, b_2^G) the differences $|Q(a_1^*, b_1^G) - Q(a_1^*, b_2^G)|$ and $|Q(a_2^*, b_1^G) - Q(a_2^*, b_2^G)|$ to a threshold, where a_1^* and a_2^* are the optimal actions at b_1^G and b_2^G respectively according to the current value function. If both differences are larger than that threshold, a mixture state $b_{12} = 0.5\ b_1^G + 0.5\ b_2^G$ is considered to be added to the grid. Further, if the probability of obtaining the same observation on both b_1^G and b_2^G is greater than another threshold, b_{12} is then added to the grid.

We borrow the idea of adding the midpoint from Brafman's grid approach and so it is similar to one of the generation heuristic methods in PBUA. However, we consider the amount of improvement given to the value function instead of the how likely a different policy is given by adding that new grid point.

	direct			lookahead			PBUA
$\|G\|$	33	82	174	33	82	174	144
Reward	0.07	0.8	0.94	0.04	2.35	2.27	2.43
STD	0.29	1.35	1.14	0.40	1.21	1.01	0.90
CI (±)	0.05	0.22	0.18	0.06	0.19	0.16	0.14

Table 6.5: Quality of Policies Computed by Brafman's Grid Approach in Tiger-Grid

Brafman did some experiments using this grid approach in the Hallway2 problem and the Tiger-Grid problem. It gave the best result that we have noticed in the literature for Hallway2. However, it did not perform well in the Tiger-Grid problem unless using a lookahead control. The experimental results reported by Brafman are replicated in table 6.5. The results were averaged over 151 simulations, each lasting 500 steps. The "CI" in the table refers to confidence interval at 95% significance level from the mean reward. We can see the confidence interval can be relatively large due to small number of simulations. We have done some experiments using 151 simulations, each lasting 150 steps. The best average reward PBUA got is 2.43, using next-step belief state expansion with backup as estimation method, after spending 461 seconds in 4 expansions, without lookahead control. Since the confidence level is quite large, it is hard to compare this performance with the performance given by Brafman's grid approach using a lookahead control. However, it is obvious that PBUA can achieve a better policy than Brafman's grid approach can, without using a lookahead control.

6.7 Variable-Resolution Regular Grid Approach

An extension to the fixed-resolution regular grid approach is proposed by Zhou et al.[20]. This approach computes an upper bound and a lower bound to the optimal value function, similar to the original version, but it allows regular grid with variable resolution. In the discussion, Zhou focuses on the upper bound value function.

This approach also uses Freudenthal triangulation to perform interpolation, as in the fixed-resolution. Assume the belief point to be interpolated is b. This version searches for the smallest and complete sub-simplex containing b. Smallest means it has the highest resolution and complete means all the vertices of the sub-simplex are contained in the grid. Then Freudenthal triangulation is performed

in this sub-simplex to interpolate the value of b.

To expand the grid, the lower bound value function is used. The differences between the upper bound value function and the lower bound value function are computed for all grid points. The grid point that gives the largest difference is identified. The sub-simplices that contain the successive belief states of that grid point is refined to have a higher resolution. This makes the grid have with different resolutions of sub-simplices.

The grid construction of this grid approach is different from PBUA, since the former has to maintain a regular grid for Freudenthal triangulation while the latter uses a non-regular grid. However, both algorithms uses a similar approach to estimate the error of the grid point. They both uses an upper bound and a lower bound. In PBUA, the QMDP approximation is used since it cannot use a fast interpolation as Freudenthal triangulation to compute an upper bound with a non-regular grid. Indeed, the idea of using an upper bound to estimate the error is adapted from this variable-resolution regular grid, so both methods have a similar idea.

The experiments conducted by Zhou show that this grid approach can compute a policy quickly even for a large grid size. However, only the error bound scores are reported and the quality of policy has not been measured by simulation. It is therefore difficult to compare the performance between PBUA and this grid approach.

6.8 Discussions and Summary

As we can see, the heuristic algorithms discussed in this chapter can be mainly divided into two categories, the upper bound and the lower bound. Upper bound value function is usually represented by a set of grid-value pairs, where values of the belief states not included in the grid are interpolated. This interpolation method is the main problem that an upper bound method has to deal with. Interpolation could be relatively slow, and it constitutes most of the solving time in grid-value heuristic methods. An elegant and efficient solution is to use Freudenthal triangulation, but this requires a fixed-resolution regular grid, which gives an exponential increase in grid size when the resolution increases. This approach is extended to allowing a variable-resolution regular grid. However, the regularity still makes grid size relatively large and many of the grid points may

not be very useful to the interpolation in terms of accuracy of the value.

QMDP approximation solves the underlying MDP and uses the Q-value function to approximate the values of the belief states. The Q-value function can be solved very quickly, and the interpolation to the belief states can be fast. However, the quality of the policies is limited, and it does not allow the expansion to the grid, which consists of only all the extreme points. Hauskrecht and Brafman propose using a variable non-regular grid. However, the interpolation can be slower and it is argued that their interpolation is linear to the grid size, which makes them not scalable to a large grid and a large state space[20]. Moreover, experimental results show that they fail to perform well in some problems, such as the Tiger-Grid problem. Hauskrecht explains that the sawtooth shape of the interpolated value function does not match the shape of the optimal value function correctly[8]. This incompatible shape may affect the quality of the policy, especially that the bound is not close enough to the optimal value function.

On the other hand, a lower bound value function is usually represented by a set of vectors. This eliminates the interpolation problem because the value of any belief state is always given by the maximum inner product of the belief state and a vector included in the vector set. However, the main problem that this approach faces is the exponential increase in the number of vectors after each update step as in the exact algorithms. Lovejoy solves this by backing up at the grid points and put only the backup vector in the updated vector set in each step of value iteration. This limits the size of the vector set to the number of grid points, but the value function may not converge. Hauskrecht uses an incremental methods to allow convergence. The updated vector set consists of all the backup vectors plus the vectors in the original vector set. This limits the increase in the number of vectors to at most the number of the grid points, which is better than the exponential increase in the exact algorithms. However, this relies on linear programming to prune the vectors and the update cycle can take a long time when the number of vectors is large, and the value iteration has to stop well before convergence in practice. This shows how the point-based update of PBUA can be better than these two approaches.

Nevertheless, experimental results show that the incremental linear function approach gives a good policy compared to the grid-value approaches, even though the value function still have not converged. This shows that updating the vectors using backup produces a better policy than updating only the grid values does.

The reason is backing up a vector actually updates the value of all the belief states that give the maximum inner product with that vector, but updating a grid-value pair only updates the value of a particular belief state.

PBUA gives a good policy based on this idea. It also solves that convergence problem and the large increase in the number of vectors at the same time. It selects the vector, between the originally associated vector and the backup vector, that gives a larger inner product to the backup grid point. This maintains the monotonicity at the grid points and at the same time limits the number of vectors to the number of grid points. Although it does not guarantee that the values at non-grid points improve after each update cycle, experiments show that it produces a good policy compared to other heuristic algorithms. The grid can also be expanded to allow a better policy. This provides a flexible trade-off between the quality of policy and the solving time, where a fixed grid does not.

Recent attempts have focused on using the grid-value interpolation to approximate the optimal value function, since it can be faster than using the vector set. However, the policy produced may not be very good due to its incompatible shape to the optimal value function. PBUA shows that using a vector set to represent the value function can be fast and can give a better policy. This shows that future research can put more emphasis on using a lower bound instead of using an upper bound.

Chapter 7

Conclusion

In this thesis, we propose a fast heuristic algorithm, namely point-based update approach (PBUA), for decision-theoretic planning. PBUA uses a set of vectors to represent the value function, and uses a grid-based heuristic update for value iteration to compute this value function. This heuristic update improves the update in exact algorithm on two sides. On one hand, using the grid points for update reduces the time spent on locating the witness points. On the other hand, this heuristic update solves the problem of having an exponential increase in the number of vectors after each update in value iteration. This heuristic update limits the number of vectors to only the number of grids, which is usually only several times of the size of the state space. This dramatic reduction in the number of vectors lowers the time complexity of this heuristic algorithm considerably. This also eliminates the need of pruning extraneous vectors from the vector sets using linear programming, which can be computational expensive. Although this heuristic method does not use a t-step optimal value function to approximate the optimal value function over the infinite horizon, the experiments show that this heuristic algorithm can find policies that are fairly near to the optimal.

Besides the heuristic update, this thesis shows that the grid can be expanded using different heuristic methods. Since the heuristic update maintain monotonicity over the grid points, the value function converges in the value iteration. This allows the grid to be expanded after the value function has converged. With the addition of expansion mechanism, this heuristic algorithm first improves the value function using a smaller grid in a shorter time, and then expands the grid to allow further improvement to the value function. Experimental results show that this heuristic method gives a policy that is of similar quality as that of the optimal policy.

When compared to the exact algorithms, this method can give a close-to-optimal policy in a much shorter time. It can also solve some larger problems that are out of the capacity of any exact algorithms. When compared to other

grid-based heuristic algorithms, this method can give a policy better than or close to the others in many test problems in the literature. Although we have not compared the computation time of this heuristic algorithm directly with the others, we believe that its computational performance is at least comparable to the others.

In addition to showing that the proposed heuristic algorithm is among the best in the literature, we compared the general performance of two different kinds of grid-based heuristic algorithms: one that uses a set of grid-value pairs to interpolate the value function and one that uses a set of vectors to represent the value function. Our observation shows that the latter one gives a much better policy than the former one generally. This is possibly due to the different characteristics of the value functions represented by two different ways. Although using grid-value pairs interpolation allows a faster computation, the poor performance of it in some problems shows its deficiency and suggests that more research effort should be put to using a vector set representation. Moreover, PBUA shows that computation efficiency of update using a vector set representation can be comparable to that using grid-value pairs.

This thesis also has some limitations. Although we discusses some heuristic methods in expanding the grid, the experiments do not show which method is significantly and constantly better than the others. We also do not show that which one performs better under what kind of situations. It needs further investigation to improve this heuristic algorithm. In addition, our analysis shows that although PBUA can compute a policy relatively fast, especially compared to exact algorithms, its time complexity depends heavily on the size of state space. This suggests the possibility of using a compact state space representation to improve the time complexity. However, the heuristic update and the expansion mechanism that PBUA uses may still be useful using a compact state space representation.

7.1 Future Enhancements

As mentioned before, this thesis has some limitations. Here are some of the possible works that can be done in the future to enhance PBUA:

- The expansion heuristic methods are still not mature, based on the fact that they are at most slightly better than a random expansion. The gen-

eration heuristics may need to be improved. However, the main problem of the expansion heuristics is that the selection heuristics cannot guide the expansion well. Therefore, more investigations should be done to improve the selection heuristics.

- A possible extension is to use a non-fixed grid. The grid should remain fixed during a value iteration to ensure convergence of the value function. However, after each value iteration, the grid may change so that the vectors are backed up on more belief states but at the same time the grid can be kept small if necessary.

- Another possible extension is to associate more than one vector to each grid point. For example, $|\mathcal{A}|$ vectors can be used for each grid point, where each vector corresponds to the value of taking a particular action. This possibly improves PBUA based on three intuitions. First, the QMDP approximation is actually an improvement to MDP approximation. This extension is similar to how QMDP approximation extends the MDP approximation. Second, the quality of policy obtained by PBUA with a small grid is limited by the small number of vectors. This number of vectors is increased if more than one vector is used for each grid point, and therefore a small grid can allow a better policy. Third, when obtaining the value of any belief state, especially that not included in the grid, the best vector is chosen among the vectors for all actions. This is similar to performing a lookahead control and shows why this can be possibly better. The $|\mathcal{A}|$ vectors can be obtained as a by-product when doing a backup, so computing them does not need more backup operations. However, since the time for computing a value of a belief state using a vector set representation requires time linear to the size of vector set. Therefore, the overall time for value iteration may still be $|\mathcal{A}|$ of the original time needed. Thus, the improvement in quality of policy and the increase in computation time should be considered together when deciding whether to use this approach.

Appendix A

Details of Experiments

In this thesis, we have tested PBUA in some problems in the literature. These problems include Tiger, Manufacturing, Network, Shuttle, 4x3 Maze, Zhang's Maze, Maze20, Office, Tiger-Grid, Hallway, and Hallway2. The test data is obtained from Cassandra's POMDP file repository page[1], except that of Zhang's Maze and Office. This appendix gives more detailed descriptions to these problems and show the best results that we have noticed in the literature. We divided the problems into smaller problems and larger problems. The smaller problems are usually tested with exact algorithms, since they are usually the only feasible problems to the exact algorithms, with smaller number of states, actions, and observations. On the other hand, larger problems are usually tested with heuristic algorithms, to see how scalable these algorithms are to the larger state space. Before we describe the problems, we show the setup we generally used of the experiments reported in this thesis.

A.1 General Experiment Setup

For the experimental results shown in this thesis, we averaged the results over 10000 simulations for the smaller problems, and over 5000 simulations for the larger problems. Each simulation lasted 100 steps at most, and some were terminated earlier when the agent had entered an absorbing terminal state. This setup gave us an error rate of less than 5% at 95% significance level of most results obtained.

The discount factor we used is 0.95, except for Maze20, in which 0.9 was used. The initial belief state of the simulation were determined problem by problem. Some had an uniform distribution over all the possible states, some had an uniform distribution over all states excluding the goal state, some had an uni-

[1] http://www.cs.brown.edu/research/ai/pomdp/examples/index.html

form distribution over a number of states, and some had an random initial belief state. This initial belief state setup is stated in the descriptions of the problems. The actual initial state of the simulation was decided randomly according to the distribution specified by the initial belief state.

The experiments were implemented with C++. They were run on a PC with a 400MHz AMD K6-3 CPU and 256Mb RAM.

A.2 Notations to the Variants of PBUA

We use a 4-digit notation to denote the variants of PBUA. The first digit refers to the expansion heuristics as follows:

0	extreme point only
1	random points
2	kLI expansion with successive belief points
3	kLI expansion with preceding belief points
4	kLI expansion with midpoints
5	kLI expansion with stochastic simulation
6	kLI expansion combined
7	stochastic simulation

The second digit refers to the estimation method. For expansion heuristic method 2–5, 0 refers to estimation using backup, 1 to estimation using last-step value function, and 2 to estimation using QMDP approximation. For expansion heuristic method 6, 0 refers to estimation using backup and 3 refers to a combined estimation. More information on the first two digits can be found in section 4.6.1.

The third digit corresponds to the use of SAU. 0 refers to value iteration without SAU and 1 refers to value iteration with it. The fourth digit corresponds to the use of lookahead control. Similarly, 0 refers to without using it, and 1 refers to using it. More information on the third and the fourth digit can be found in sections 5.3.2 and 5.2.2 respectively.

A.3 Smaller Problems

The smaller problems refers to Tiger, Manufacturing, Network, Shuttle, and 4x3 Maze. Their properties are shown in table A.1. The average rewards received using the optimal policy and a random policy are also included.

Problem	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	$\|\mathcal{Z}\|$	Optimal	Random
Tiger	2	2	3	19.5	-601
Manufacturing	3	4	2	7.94	-6.97
Network	7	4	2	294	-243
Shuttle	8	3	5	32.6	-4.07
4x3 Maze	11	4	6	-	-1.03

Table A.1: Properties of some Smaller Problems

A.3.1 Tiger

This problem appears in Cassandra et al.[5]. 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 door, 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 receive 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 on the left and that on the right. After opening the door, the agent receives its rewards/penalty, and the whole process starts again.

A.3.2 Manufacturing

This is a 3-state manufacturing machine problem. According to Cassandra's web page, this comes from Ed Sondik's 1971 thesis. The initial belief state of the agent is randomly generated. The maximum reward and penalty in this problem are 1 and -2 respectively.

A.3.3 Network

This is a 7-state 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.

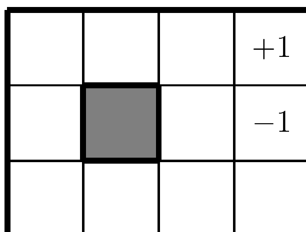

Figure A.1: Layout of 4x3 Maze

A.3.4 Shuttle

This problem appears in Chrisman[6]. 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 a 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. More descriptions on the state space can be found from the data file or from Chrisman's paper.

A.3.5 4x3 Maze

This is a 4x3 Maze problem from Russell and Norvig[15]. The layout of the maze is shown in figure A.1. The agent's goal is to go to the reward state and get as many rewards as possible. The agent can choose to move in one of the four directions. However, the agent may fail to move to its intended direction, with 10% of time that the agent moves to one of the direction perpendicular to its intended action. Moving to the wall leaves the agent at the same location. The agent can perceive correctly if there is a wall on the left and on the right after each action. In addition, the agent knows that whether it is in the reward/penalty state or not. The agent receives a penalty of -0.04 for every action, except that it is moving to the reward or penalty state. The initial belief state is a uniform distribution over all the 11 states.

Problem	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$	$\|\mathcal{Z}\|$	Random
Zhang's Maze	11	6	7	0.00
Maze20	20	6	8	33.8
Office	35	6	23	-43.5
Tiger-Grid	36	5	17	-2.14
Hallway	60	5	21	0.04
Hallway2	92	5	17	0.02

Table A.2: Properties of some Larger Problems

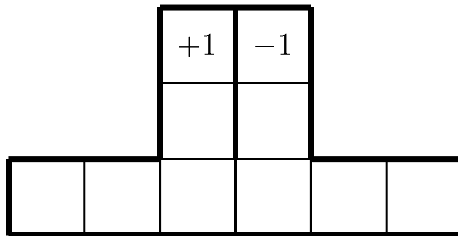

Figure A.2: Layout of Zhang's Maze

A.4 Larger Problems

To show their scalability, heuristic algorithms are usually tested with larger problems. Those we used in thesis include Zhang's Maze, Maze20, Office, Tiger-Grid, Hallway, Hallway2. Their properties are show in table A.2. The average rewards received using a random policy are also included.

As we can see, these problems are larger than the problems previously described. Besides the larger state spaces, the numbers of observations are also larger, and thus they are infeasible to the exact algorithms.

A.4.1 Zhang's Maze

Zhang's Maze appears in Zhang et al.[18]. The layout of the maze is shown in figure A.2. This maze has a terminal state in addition to the 10 location states. It has 4 actions for moving, a look action, and a action for declaring goal.

The move actions have 80% chance of moving to the intended direction, and 10% chance of moving to one of the direction perpendicular of the intended direction. If the agent moves into the walls, it remains at the original position. The agent gets a reward of +1 if it declares goal at the reward position, and a penalty of -1 if at the penalty position. The simulation then goes to the terminal state after this declare action. All actions, except declaring goal at the two special

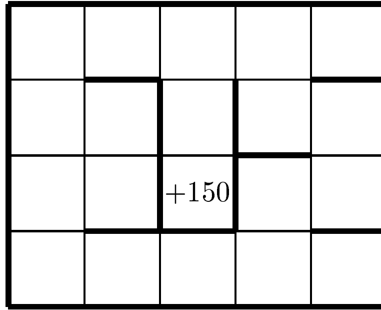

Figure A.3: Layout of Maze20

positions, do not give any reward or penalty.

The look action is the action that gives the agent an observation. The observation made is a sequence consisting of four letters, representing whether there is a wall(w) or nothing(o) in each of the four directions. For example, the reward state gives an observation of woww. However, the observation obtained is noisy. The look action may give a void observation with probability 0.05. It may also give an observation that is supposed to be made in other cells with probability 0.05, if the correct observation and wrong observation differ by no more than two directions. The agent gets the correct observation with the remaining probability, which is around 0.8. This problem is characterized by its symmetry, meaning that there are a few pairs of state that have the same observation. In addition to this, the noisy observation makes this problem hard to solve.

Initially, the agent has a random belief state, except that it has zero belief to the terminal state. The actual initial state is randomly chosen according to the distribution of the initial belief state.

A.4.2 Maze20

This maze problem comes from Hauskrecht[7, 8], and its layout is shown in figure A.3. There are 6 actions: 4 move actions and 2 detect actions. The agent gets a reward of 150 if it takes any action at the goal position. Other than that, the agent gets a small reward for not bumping into the wall — 4 for a move and 2 for a detect action. The initial belief state is randomly generated, and the actual initial state is selected randomly according to that distribution.

The move action of the agent is not perfect. There is 15% each for moving to one of the neighboring direction of the intended direction. A move to the wall keeps the agent at the same position. After the agent takes any action at the goal

Method	Rewards
MDP approximation [a]	~ 37
QMDP approximation - direct [a]	~ 36
QMDP approximation - lookahead [a]	~ 51
Fast informed bound - direct [a]	~ 39
Fast informed bound - lookahead [a]	~ 52
Hauskrecht's grid - lookahead [a]	~ 47
Least-squares fit with linear-Q-function - direct [a]	~ 51
Least-squares fit with linear-Q-function - lookahead [a]	~ 56
Incremental linear function update - direct [a]	~ 58
Incremental linear function update - lookahead [a]	~ 60

[a] See Hauskrecht[8]

Table A.3: Results in Maze20 in the Literature

state, the agent randomly moves to one of three states at the corner according a certain distribution.

The detect actions produce noisy observation. One of them detect the presence of walls at the north-south direction, and the other at the east-west direction. The actions detect correctly with probability 0.75 for a two-wall case, 0.8 for a one-wall case, and 0.89 for a no-wall case.

Some experiments have been conducted on this problems. The best results of some of the attempts are included in table A.3.

A.4.3 Office

This is a 35-state maze problem designed by Zhang and Zhang, modelled after their department[18]. There are 34 location states and a terminal state. The agent has to go to the main office state and take a beep action so that someone will get a mail to the agent. There are 6 actions: 4 move actions, 1 look action and 1 beep action. The agent gets a reward of 50 if it beeps at the right location, and -10 if at other locations. It has a penalty of -2 if it moves into a wall, and a penalty of -1 if it takes a look action.

The observation is more complicated than that in the Zhang's Maze. It is also made up of a string of 4 characters, representing an object observed at the 4 directions. However, there can be a door(d), an empty wall(w), a display board(b), or nothing(o) at each direction. With probability 0.05, a look action produces a void observation, and with 0.05 for each wrong observation, it produces

Method	Rewards
QMDP approximation - direct	2.21
QMDP approximation - lookahead	18.0

Table A.4: Results in Office

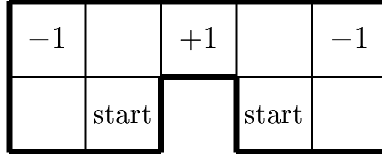

Figure A.4: Layout of Tiger-Grid

an observation that is supposed to be observed at other location if the wrong observation differs from the correct observation by only 1 character. This maze problem is similar to Zhang's Maze in the sense that many pairs (12) of location give the same ideal observation, thus, the agent may feel difficult to locate its position if it does not take many look action.

Initially, the agent has a randomly generated belief state, excluding the goal state. The actual initial state is randomly selected according to that distribution. When the agent beeps at the goal state, it is taken to the terminal state and the simulation ends.

We have conducted some experiments on this maze problem using QMDP approximation. The results is shown in table A.4.

A.4.4 Tiger-Grid

This is a 36-state maze problem appeared on Littman et al.[11]. The layout is shown in figure A.4. Although there are 9 locations in the maze, the agent has 4 orientations at each location, so there are a total of 36 states. The agent starts in either of the starting position, and has to move to the reward position. An interesting feature is that, when the agent starts, it has to make an observation immediately to see if it is on the left or right. Otherwise, after it takes a step up, the observation it makes is the same. As a result, it may go to the penalty position instead of the reward position due to this uncertainty. Littman gives a more detailed description on this problem.

Although Littman have done some experiments on this problem, he measured the median of steps to arrive at the goal and the percentage of arriving at the goal within 251 steps, this makes his result not able to be compared with ours

Method	Rewards
QMDP approximation - direct [a]	0.24
QMDP approximation - lookahead [a]	0.27
Brafman's grid - direct [b]	0.94
Brafman's grid - lookahead [b]	2.35

[a]Obtained from experiments done by us
[b]See Brafman[3]

Table A.5: Results in Tiger-Grid in the Literature

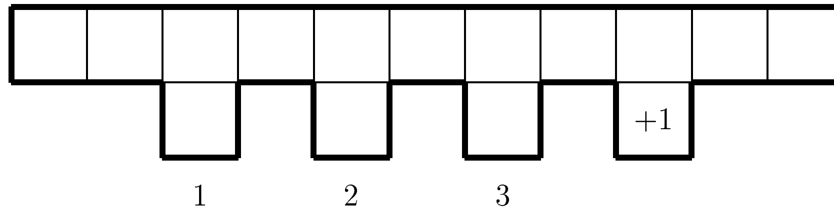

Figure A.5: Layout of Hallway

directly. In our experiment setup, the agent starts over to its initial belief state when it reaches the reward state and therefore we cannot get the same data. Another difference is that the initial belief state of the agent in our experiments were an uniform distribution over heading north at one of the two starting state.

Brafman uses a similar criteria and setup as we used[3]. The experimental results are shown in table A.5.

A.4.5 Hallway and Hallway2

These are the biggest structured problems in the literature, appeared in Littman et al.[11]. Each location has 4 orientations. These combinations of orientation and location constitute to the 60 states and 92 states of these problems. Their layouts are shown in figures A.5 and A.6. Each problem has a goal state that the agent has to reach to receive a reward.

Each problem has 5 actions: stay in place, move forward, turn left, turn right, and turn around. The agent receives reward of 1 only when it moves to the reward state, and nothing otherwise. There are 21 observations in Hallway, and 17 observations in Hallway2. The observations are all combination of walls in each relative direction and the goal observation. In addition, the agent sees 4 different landmarks when it faces south at 4 particular locations in the Hallway problem.

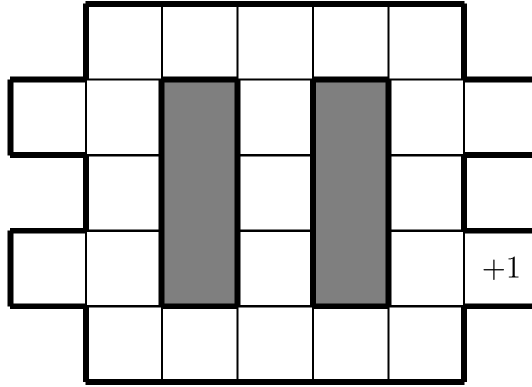

Figure A.6: Layout of Hallway2

	Hallway			Hallway2		
Method	Goal%	Med	Reward	Goal%	Med	Reward
QMDP - no stay [a]	100.0	16	-	57.8	40	-
Human [a]	100.0	15	-	100.0	29	-
Brafman's grid [b]	-	-	-	100	24	-
QMDP - direct [c]	(48.4)	> 100	0.257	(25.4)	> 100	0.087
QMDP - lookahead [c]	(51.0)	14	0.51	(66.7)	37	0.244
Random [c]	(16.7)	> 100	0.043	10.5	> 100	0.025

[a] See Littman[11]
[b] See Brafman[3]
[c] Obtained from experiments done by us

Table A.6: Results in Hallway and Hallway2 in the Literature

The observations and the transitions in these two problems, as in others, are noisy.

The initial belief state is a uniformly distribution over all the states excluding the goal state. After the agent reached the goal state, the simulation terminates.

Some experimental results have been reported in the literature. However, they are in terms of percentage of reaching the goal and the median of number of steps taken to reach the goal, which are different from our average rewards criteria. The results are shown in table A.6. Notice slightly different setups were used in the experiments. Littman and Brafman used 251 simulations, with a maximum of 251 steps each, to record the data. We used 5000 simulations, with a maximum of 100 steps each, to collect the data. Therefore, the goal percentage should not be be compared directly. Moreover, our experimental results should have smaller confidence intervals.

Problem	Method	N_e	T_s	Rewards	Median	Term%
Tiger	1000	15	0.12	19.9	-	-
Manufacturing	4200	10	0.08	7.94	-	-
Network	4100	20	4.35	297	-	-
Shuttle	5100	10	0.20	32.7	-	-
4x3	2100	2	1.21	1.90	-	-
Zhang's Maze	2100	20	4.84	0.56	9	100
Maze20	4000	20	2473	59.1	-	-
Office	2100	20	3304	22.2	13	99.8
Tiger-Grid	3000	20	12116	2.30	-	-
Hallway	1000	5	450	0.53	14	100
Hallway2	1000	20	27898	0.35	27	100

Table A.7: Summary of the Best Policies obtained by PBUA

A.5 Summary of the Best Policies obtained by PBUA

This section summarizes the best policies obtained by PBUA, and the corresponding time taken. Thy are shown in table A.7. The "method" refers to the 4-digit notation to show the variants of PBUA, but only the variants with the first two digits are included in this summary. N_e denotes the number of expansions used to obtain that policy. The solving time is denoted as T_s. The "Rewards" means the averaged discounted rewards. The "Median" refers to the median of steps to the terminal states, and "Term%" is the percentage of reaching the terminal states, which include non-goal state in some problems. Note that the best policies refer to those that has the highest rewards. However, a policy that has the same quality as that of the best policy actually may have a slightly fewer rewards in the experiments, due to the randomness in simulation. If these slightly numerically worse policy is allowed, the time taken to compute them can be much shorter in some cases. Note also that only the PBUA with random point expansion is included for Hallway and Hallway2 in this summary.

Bibliography

[1] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," *Journal of Artificial Intelligence Research*, vol. 11, pp. 1–94, 1999.

[2] C. Boutilier and D. Poole, "Computing optimal policies for partially observable decision processes using compact representations," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, (Portland, Oregon), 1996.

[3] R. I. Brafman, "A heuristic variable grid solution method for POMDPs," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

[4] A. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental pruning: A simple, fast, exact method for partially observable Markov decision processes," in *Proceedings of the Thirteenth Conference on Uncertainty in Artifical Intelligence*, pp. 54–61, 1997.

[5] A. R. Cassandra, L. P. Kaelbling, and M. L. Litttman, "Acting optimally in partially observable stochastic domains," in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, (Seattle, WA), 1994.

[6] L. Chrisman, "Reinforcement learning with perceptual aliasing: The perceptual distinctions approach," in *National Conference on Artificial Intelligence*, pp. 183–188, 1992.

[7] M. Hauskrecht, "Incremental methods for computing bounds in partially observable Markov decision processes," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 734–739, 1997.

[8] M. Hauskrecht, "Value-function approximations for partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, pp. 33–94, 2000.

[9] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partically observable stochastic domains," *Artificial Intelligence*, vol. 101, pp. 99–134, 1998.

[10] L. P. Kaelbling, M. L. Littman, and A. M. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[11] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," in *Proceedings of the Twelfth International Conference on Machine Learning*, (San Francisco, CA), pp. 362–370, Morgan Kaufmann, 1995.

[12] W. S. Lovejoy, "Computationally feasible bounds for partically observed Markov decision processes," *Operations Research*, vol. 39, no. 1, pp. 162–175, 1991.

[13] G. E. Monahan, "A survey of partially observable Markov decision processes: Theory, models, and algorithms," *Management Science*, vol. 28, no. 1, pp. 1–16, 1982.

[14] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* New York: John Wiley & Sons, 1994.

[15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Upper Saddle River, New Jersey: Prentice-Hall, Inc., 1995.

[16] E. J. Sondik, *The Optimal Control of Partially Observable Markov Decision Processes over a Finite Horizon.* PhD thesis, Standford University, 1971.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an Introduction.* Cambridge, Massachusetts: MIT Press, 1998.

[18] N. L. Zhang and W. Zhang, "Value approximation in near-discernible POMDPs: Space-progressive value iteration and a simplication." To appear in Sixth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU) 2001.

[19] N. L. Zhang and W. Zhang, "Speeding up the convergence of value iteration in partially observable Markov decision processes," *Journal of Artificial Intelligence Research*, vol. 14, pp. 29–51, 2001.

[20] R. Zhou and E. A. Hansen, "An improved grid-based approximation algorithm for POMDPs," in *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, (Seattle, Washington), 2001.