Active Sensing for Partially Observable Markov Decision Processes

by

Veronika Koltunova

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2012

AUTHOR'S DECLARATION

Abstract

Context information on a smart phone can be used to tailor applications for specific situations (e.g. provide tailored routing advice based on location, gas prices and traffic). However, typical context-aware smart phone applications use very limited context information such as user identity, location and time. In the future, smart phones will need to decide from a wide range of sensors to gather information from in order to best accommodate user needs and preferences in a given context.

In this thesis, we present a model for active sensor selection within decision-making processes, in which observational features are selected based on longer-term impact on the decisions made by the smart phone. This thesis formulates the problem as a partially observable Markov decision process (POMDP), and proposes a non-myopic solution to the problem using a state of the art approximate planning algorithm Symbolic Perseus. We have tested our method on a 3 small example domains, comparing different policy types, discount factors and cost settings. The experimental results proved that the proposed approach delivers a better policy in the situation of costly sensors, while at the same time provides the advantage of faster policy computation with less memory usage.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Prof. Jesse Hoey for the continuous support of my research, for his patience, motivation, enthusiasm and expertise.

Besides my advisor, I wish to express my sincere thanks to the rest of my thesis committee: Prof. Pascal Poupart and Prof. Robin Cohen for their insightful comments and productive defense discussion.

My sincere gratitude also goes to my colleagues in Computational Health Informatics Laboratory: Marek Grzes, Robby Goetschalckx for assisting in the technical part and thesis proof reading.

I am indebted to my fellow labmate Kirill Erokhin for his extraordinary help and the sleepless nights we were working together on experiment automatization.

I also would like to acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada, particularly in the Engage Grant that provided the necessary financial support for this research.

I am grateful to the administrative and faculty staff of the David R. Cheriton School of Computer Science Department of the University of Waterloo, particularly in persons of Margaret Towell and Paula Roser for helping the departments to run smoothly and for assisting me in many different ways.

Dedication

This work would not have been possible without my friends who helped me get through the hardest months of my life. I am extremely grateful to those people who formed a family for me away from home. They encouraged, supported, understood, and loved me at every moment: Olga Zorin, Roman Blyshchyk, Lev Kisselman. To them and to all of my friends I dedicate this thesis.

Table of Contents

List of Abbreviations

ADD: Algebraic Decision Diagrams

AS: Active Sensing

CPT: Conditional Probability Table

CMDP: Constrained MDP

DP: Dynamic Programming

DT: Decision Tree

FIB: Fast Informed Bound

FSM: Finite State Machine

FSVI: Forward Search Value Iteration

GOSSIP: Goal-oriented sensor selection for intelligent phones

HSVI: Heuristic Search Value Iteration

HMM: hidden Markov Model

IOHMM: Input-Output Hidden Markov Model

La-Casa: Location and Context Aware Safety Assistant

LP: Linear Programming

MDP: Markov Decision Process

MLS: Most Likely State

PBVI: Point-Based Value Iteration

PEMA: Point-based Error Minimization Algorithm

POMDP: Partially Observable Markov Decision Process

PS: Passive Sensing

SARSOP: Successive Approximations of the Reachable Space under Optimal Policies

SP: Symbolic Perseus

SVM: Support Vector Machines

Chapter 1
Introduction

In the heart of AI research there is the field of AI **planning**. It has its main purpose in designing the action strategy for an agent with respect to the environmental feedback. The planning stage is often required for developing an autonomous agent. For example suppose that a robot has to open one of two closed doors when there is a tiger behind one and a reward behind the other. (In the literature the actor who is making the decision is often called an agent.) The agent is interested to estimate the tiger's location in order to avoid opening the door with the tiger. There is a penalty assigned to opening the tiger's door and a reward for the other. To estimate the tiger's location the agent can acquire this information by listening. However, listening has its own cost as well as the challenge that incoming information is noisy. Here, the listening action corresponds to the observation acquisition, or sensing (sensor query action). This problem is a well-known tiger toy problem in the POMDP research literature [Cassandra98, Hoey07]. The planning for the agent is required in order to assist him with a strategy of actions with respect to the current situation and incoming observations from listening.

In this thesis we are focusing on **sequential decision making**, the subarea of planning responsible for developing sequential action plans [Puterman05]. In other words we assist the agent with the action strategy to reach the planning goal by performing a sequence of decisions. Here we focus on the case when time is counted as a discrete sequence of time steps: at every step the decision should be made to balance the trade-off of immediate accomplishments and long-term goals.

One goal of AI is to come up with the techniques applicable to the different areas of the real world. The challenge of this goal is contained in the limitations of the agents' perception abilities. In other words the incoming environment feedback signals may be very complex or noisy. This confusion is called **uncertainty** and affects the knowledge about the state of the world and thus, the decision to make as well as the outcome of the decision.

Planning in a **deterministic** environment is easier due to the fact that the effect of any action on the environment is totally predictable. As soon as we have the uncertainty factor in the acting environment, i.e. we are not 100% sure in the outcome of action, the difficulty of planning drastically increases. Those environments are called **stochastic**.

Partial observability is the special type of stochastisity when the agent is also uncertain about the current state of the environment due to the lack of certainty in the input information from the available sensors. This undetermined stochastic type of the environment is more realistic than fully observable stochastic environments. Hence planning strategies for those environments have potential in terms of applicability.

Generally, partial observability affects with its uncertainty every step of a decision making process. Planning for partially observable environments is facing more complex problems than planning for classic stochastic domains. The problem could be summarized as a cumulative effect of uncertainty through the planning process. In that case, to develop a good strategy, a planning system has to constantly take into consideration the uncertain nature of the environment and incorporate this knowledge into the solution development. That could be done by keeping track of the entire history of actions and feedback in the agent-environment dialogue.

Awareness of the agent about the degree of its uncertainty could make a dramatic difference in the decision making quality. For example, a poor decision could be made in the situation when a robot ignores the low certainty of its state and makes a blind locally best decision before verification of the current state information. In robotics domains those verification actions could be sensor query actions. In a state of a high uncertainty an agent should be able to decide to gather more information from available sources to succeed in a long term goal achievement. Therefore a planning algorithm will have a trade-off problem between the quality of the world knowledge, related acquisition costs and the quality of the outcome solution.

We approach these issues by formulating the problem as a partially observable Markov decision process, or POMDP [Cassandra98]. A POMDP is a probabilistic model that characterizes the world as a set of states, which are not directly observable. A belief state is a distribution of probabilities between all states, a set of possible belief states form a belief space. A set of observations gives information about the states through an observation function (a probability distribution). The observations are the readings from a set of sensors available to the agent. A set of actions are available to the decision maker, and these actions change the state of the world stochastically according to a transition function (another probability distribution). A reward function completes the picture, and assigns a real-valued reward or cost to each state. The POMDP can be used to compute a policy: a function that maps states to actions for the agent to take. This policy gives the action to take in each state that will achieve the best long-term reward.

The POMDP is able to model uncertain effects of actions, deal with uncertainty at the sensing level, fuse different sensor measures, and precisely model long-term trade-offs between system goals and sensor costs. It differs significantly from more reactive based approaches (e.g. neural networks and other supervised classifiers) in that it explicitly encodes domain knowledge instead of a policy of action. In this work we explore a state of the art solution algorithm for discrete complex POMDPs that have flexibility to manage uncertainty, scalability to handle thousands of states and computational efficiency to produce accurate cost-sensitive policies in reasonable time.

In this Chapter we briefly talk about the aspects of decision-theoretic planning and give an overview of the contributions of the research presented in this thesis.

1.1 Planning

The planning framework serves for a constant dialogue between 2 separate components: agent and the environment [Russel10]. However, the tracking of the environment state and decision making happens on the side of the agent. The agent pictures the world as a set of states (could be agent states like location of the robot, or states of any other object or subject). The world could be changed by the agent actions and the result could be observed upon accessible feedback. This kind of interaction repeats to provide the evolution of decision-making process. The planner in this case addresses the problem of controlling this

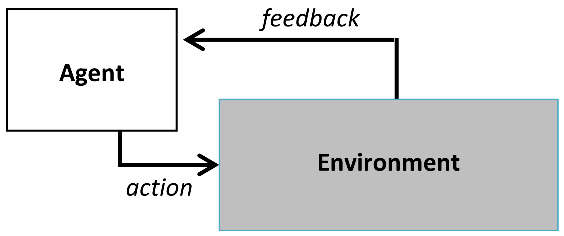


Figure 1.1: The interaction schema between an agent and the environment.

evolution process to make it converge to the given goal. This is achieved by developing a best course of actions with respect to the possible world states.

The schematic representation of the interaction process between an agent and environment is pictured in Figure 1.1. Interaction starts with the initial state and evolves according to the stochastics of actions that influence state transitions and accessible feedback from the world. While the world is represented as the set of states, the agent acts only via the set of actions. However, the complexity of the interaction process is placed into the key components of each POMDP model – actions and observations.

The simplest classic planning framework like STRIPS [Fikes71] has an assumption of being able to directly derive the next state of the world from the current state and taken action. This type of planning framework is called **deterministic**. Here we assume that the next state is uniquely determined by the state-action pair. Evidently a planner has all necessary information to develop a strategy without interacting with the environment as the future action outcome is 100% predictable.

However, deterministic frameworks are too simplified and, thus, are rarely applicable to real world problem domains. To reflect the realistic environment classic planning frameworks were built with introducing an uncertainty about the effects of actions and competing objectives [Horvitz88; Haddawy98]. For example the **decision theoretic approach** [Dean95] uses probability theory to represent uncertainties and utility theory to represent competing objectives. Evolving planning research was greatly influenced by the field of **dynamic programming**. Dynamic programming contains numerous algorithms that approach problems by splitting them into smaller problems [Bellman57].

The core of sequential decision-theoretic planning is a **Markov Decision Process** (MDP) framework [Bellman57]. An MDP model is an extension of the basic generic planning framework, where the agent cannot totally predict the action's effect which cancels the uniqueness quality of the resulting state. In MDP the next state of the world is fully observable. Unlike the nondeterministic case, anytime we are getting the clear exact picture of the world. The objectives or goals are coded by varying the rewards or punishments associated with the states. For example, the desirable states may be assigned with the biggest payoff. As a rule the success of a goal achievement is measured by the selected utility function. The traditional utility function is a function of rewards or payoffs that the agent accumulates over the decision making process. Thus the utility function criterion connects the rewards with the target action plan by guiding the solution process to maximizing utility, thus, optimizing action plans towards the most profitable states.

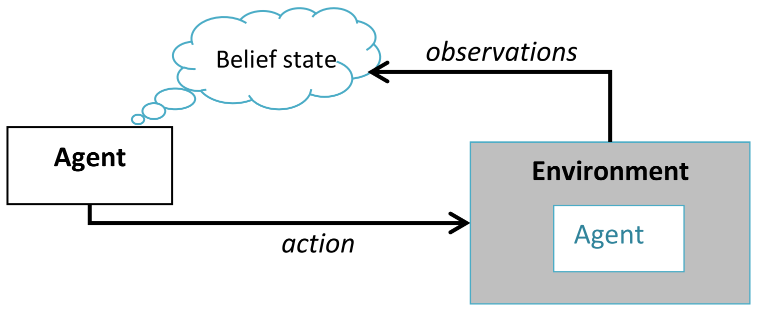


Figure 1.2. One time step of POMDP interaction between an agent and the environment.

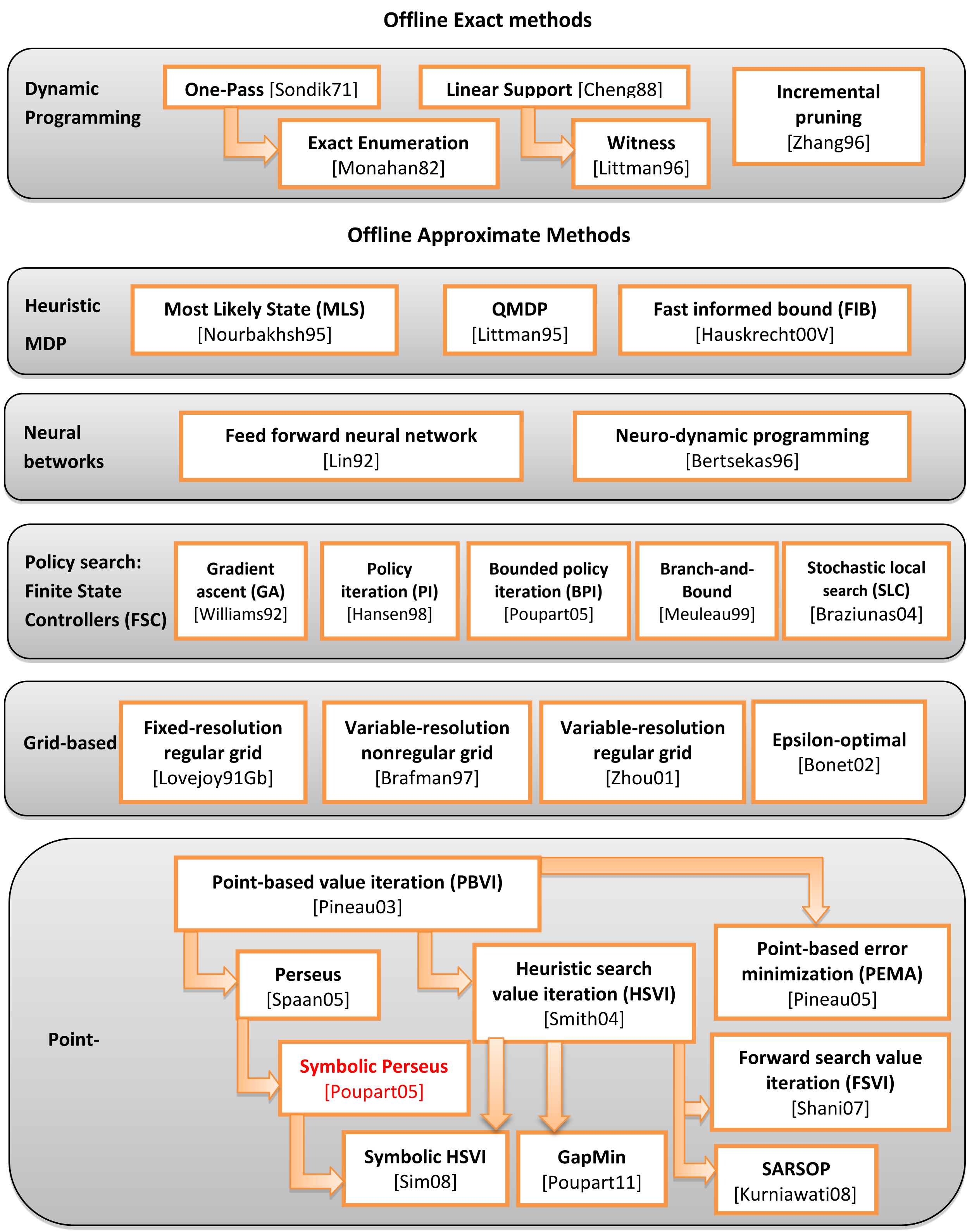
A **Partially Observable Markov Decision Process** (POMDP) [Cassandra98] is an extension of MDPs where the feedback from the world after performing an action cannot be perfectly monitored. That means that the agent has an assumption about the state he is currently in and he verifies the state indirectly via making some observations (see Fig. 1.2). So, he has uncertainty about the current state of the world in addition to the uncertainty of the outcome of the performed action. Hence, POMDP is a rich natural framework to address the issue of a decision making under uncertainty.

The first exact method for solving a POMDP was proposed by Sondik [Sondik71]. Soon there were published several improved methods of exact solving [Monahan82, Littman96, Zhang96, Cassandra97, Zhang01, Feng05]. The direction of research in developing approximate solutions became one of the fast growing in planning for POMDPs. In this thesis, we are focusing on a family of approximate POMDP solvers known as **point-based methods** [Pineau03P; Spaan04].

The key difference of Point-based POMDP planning algorithms is that they approximate POMDP solutions by iterating only on a finite subset of points as the valid representatives of a belief space. Many variations of point-based planning algorithms appeared in the past decade [Pineau03, Smith04, Pineau05; Shani07; Kurniawati08; Spaan04, Pascal11] each is focusing on different area of optimization and application domain.

In this thesis, we generate policies from our model specifications using a combination of the point-based value iteration approach (the Perseus algorithm [Spaan05]) and a structured representation of value functions and beliefs using algebraic decision diagrams (ADDS) [Poupart05]. This “symbolic” ADD approach allows for a compact and efficient representation of value and belief functions, while the point-based approach focuses effort on parts of the belief space that are reachable.

Several excellent algorithmic surveys are still well cited in the POMDP planning research [Lovejoy91; White91; Murphy00]. The 1982 survey paper by Monahan [Monahan82] gives an excellent overview of the base of theory and algorithms for POMDPs. A more technical insight into the algorithmic aspects of existing POMDP solvers can be found elsewhere [Cassandra98; Zhang01; Shani07P; Shani12; Spaan12]. In Figure 1.3 there is a summary of classic offline approaches for solving POMDPs. Precisely, the evolution of point-based methods is illustrated with arrows, pointing to the next algorithm built on the previous.


Offline Exact methods
Dynamic Programming
One-Pass [Sondik71]
Exact Enumeration [Monahan82]
Linear Support [Cheng88]
Witness [Littman96]
Incremental pruning [Zhang96]
Offline Approximate Methods
Heuristic MDP
Most Likely State (MLS) [Nourbakhsh95]
QMDP [Littman95]
Fast informed bound (FIB) [Hauskrecht00V]
Neural betworks
Feed forward neural network [Lin92]
Neuro-dynamic programming [Bertsekas96]
Policy search: Finite State Controllers (FSC)
Gradient ascent (GA) [Williams92]
Policy iteration (PI) [Hansen98]
Bounded policy iteration (BPI) [Poupart05]
Branch-and-Bound [Meuleau99]
Stochastic local search (SLC) [Braziunas04]
Grid-based
Fixed-resolution regular grid [Lovejoy91Gb]
Variable-resolution nonregular grid [Brafman97]
Variable-resolution regular grid [Zhou01]
Epsilon-optimal [Bonet02]
Point-
Point-based value iteration (PBVI) [Pineau03]
Perseus [Spaan05]
Heuristic search value iteration (HSVI) [Smith04]
Point-based error minimization (PEMA) [Pineau05]
Symbolic Perseus [Poupart05]
Forward search value iteration (FSVI) [Shani07]
Symbolic HSVI [Sim08]
GapMin [Poupart11]
SARSOP [Kurniawati08]
Figure 1.3. The classic POMDP solving algorithms.

1.2 Contribution

As smart phones become increasingly outfitted with sensors, and have increasing ability to query the local environment for additional information, the problem of prioritization of information becomes important. This problem involves deciding which sensors to gather information from in order to best accommodate user needs and preferences in a given context. These decisions are important on smart phones, as they impact the amount of processing necessary, and thus the device cost and battery life. Typically, context-aware mobile applications operate in a limited context field such as user location, contacts, and social feeds. Therefore, the output of only the relevant sensors is expected to provide a context focused on a specific area of the application.

For example, suppose a mobile device is being used to assist an elderly person living at home independently. This problem is becoming increasingly relevant due to a rapidly ageing population in many parts of the world. The device may have access to a large set of environmental sensors that are indicative of the activities being carried out. The device's objective is to determine the goals of the user, and to provide assistance in achieving these goals, possibly helping to overcome cognitive limitations of the user: the device is a cognitive orthotic [LoPresti04; Hoey11]. The mobile device can query any of its available sensors, and must do so in a cost-effective way in order to scale down computation on the device, reduce computation time (relative to the temporal bounds placed by the schedule and environmental conditions), and minimize raw cost of the queries It can also query its user for more information about goals, locations, schedule, etc., but such queries may carry significant risk of interruption or other side effects.

A simple example is a weather advisor [GOSSIP11] that reads environmental sensors and gives advice to users based on predicted weather and user preferences. In the weather advisor, the state would correspond to the joint space of the weather states (e.g. rainy, sunny), and the behaviors of the person (e.g. take umbrella, take hat, stay home). The weather states are inferred from the three environmental factors that are measured via sensor readings: pressure, humidity and temperature. In the context of mobile phones which have limited computational power or battery life, the approach presented in this thesis allows a careful cost-sensitive sensor querying based on longer-term impact on the decisions made by the smart phone.

A key observation is that the sensor selection mechanisms are working not only to maximize the amount of information in the system, but also to take into account query costs and long-term effects on goals. The second requirement of non-myopic optimization has been touched in the existing literature (see e.g. [Chai]) but is still challenging [Krause09]. Furthermore, as sensors and the sensed context information are highly correlated, there are situations when context information inferred from different sensors may have overlaps or even conflicts at the reasoning level.

In this thesis, we present an approach to sensor selection for smartphone applications. We formulate the problem as a partially observable Markov decision process (POMDP), and show non-myopic solutions to the problem using approximate planning. The POMDP allows for fusion of information from partially available sensors, and provides a theoretical framework in which the relative utilities of sensing actions can be traded off against those of user goals and preferences. We demonstrated our method on small example domains: on the extended tiger, classic coffee delivery and extended coffee delivery problems formulated as POMDP [Appendix A]. The experiment analysis confirms that the proposed approach provides a resolution for the problem of optimal action strategy in the context-sensitive sequential decision making. We show that compared to the traditional approach, the active sensing approach allows dynamic context-adaptive sensor selection, which results in better policy, while at the same time provides the advantage of faster policy computation with less memory usage for the realistic constraints of costly sensors.

1.3 Outline

The thesis is organized as follows. In Chapter 2 we specify some known POMDP application domains as well as give an overview of existing published research in the area of applying the POMDP framework to the active sensor selection problem. We begin Chapter 3 with the Markov Decision planning and then we outline the theory of Partially Observable Markov Decision Processes. We continue with following the development of Point Based Value iteration POMDP planning methods up to the Symbolic Perseus. In Chapter 4 we give the theoretical, practical and experimental backgrounds for our approach. Experiment details and results can be found in Chapter 5, followed by the conclusions in Chapter 6.

Chapter 2
Background

2.1 Applications of POMDP modeling

POMDP models serve a very general purpose in modeling systems of agent-environment interaction and can be adapted to different goals, information resources or contexts, depending on the purpose of modeling. The convenience of the POMDP framework is that instead of expressing a policy directly it allows one to model the world with stochastic laws and utility functions. POMDP models unify two types of uncertainties: the uncertainty in action results and the uncertainty in observation signals. This provides additional expressivity which makes the POMDP a rich and flexible framework that is more suitable for modeling realistic domains than previously established decision-theoretical planning models.

The classic and still expanding POMDP application area is robotics [Simmons98; Theocharous02; Pineau05; Spaan04]. In robot navigation tasks, the agent is a robot that is moving within some location states. The robot's actions are of two types: the state-transition (e.g., moving to the desired location) and state-observation (verifying of current location). Both types have a nondeterministic effect. Actions are not executed ideally so the next location of robot could not be predicted with 100% certainty. At the same for the sensor information that comes from the observation actions, contains the uncertainty exposed as noise in the readings. POMDPs are successfully integrated in control and navigation of several working robot prototypes [Simmons98, Pineau05]. According to the experimental results, the robotic domain greatly benefits in efficiency and reliability from using POMDP planners in comparison to other frameworks.

Another promising application domain of POMDPs is health informatics [Hauskrecht00; Pineau03, Hoey11; LoPresti04; Hu96]. In this domain POMDP models are used for medical therapy planning [Hauskrecht00, Pineau03, Hu96], health care assistants [Hoey11] and many more. For example in medical therapy planning, the POMDP addresses the general problem of a trade-off between verification diagnostic tests (information-gathering action) and the choice of the corresponding treatment (goal achieving action). Published experiments indicate that generated POMDP solutions appeared to be clinically reasonable and reliable within the test data of the ischemic heart disease patient's database [Hauskrecht00].

A series of papers of Hoey and colleagues [Hoey11; Hoey10; Hoey08; Hoey07A; Hoey07] addresses a problem of assisting people with dementia with their daily home activities. The computerized device, called COACH [Hoey08], is able to autonomously guide an older adult using audio and/or audio-video prompts. It has a POMDP as the basis for a planning system and uses computer vision to monitor the progress of a person with dementia washing their hands to make a decision to prompt when it is necessary.

POMDPs have also been lately successfully applied to the field of the natural language processing. In that area a most cited POMDP application is dialogue management [Roy00; Williams07]. Spoken dialogue systems have used probabilistic reasoning as a base for more than the past decade. In

[Meguro11] the authors aim to build listening agents that can implement a human-like listening process. The developed dialogue control module is based on a POMDP which can learn an optimal dialogue control policy from dialogue data. The dialogue control module chooses the best system dialogue act at every dialogue point using the policy and the user dialogue act as input.

Another example of possible POMDP application domain could be the selection of a data compression algorithm, where POMDP provides a strategy to balance algorithm compression quality and computational time of a selected algorithm as an instance of a paid cost.

A variety of other elegant POMDP application types can be found in the published related research over the past decades. For more detailed insight we refer to the [CassServ98]. Some of recent application examples are wireless sensor networks [Chobsri08], developing navigation aids [Stankiewicz07], human-robot interaction [Doshi08], object grasping [Hsiao07], invasive species management [Haight10], trust/reputation reasoning [Regan05], machine vision [Darrell96], planetary rover control [Zilberstein02], searching for a moving target [Liu07;Krishnamurthy09; Li09; Aeron08; Fei10; He04; Hanselmann08], air traffic management [Kochenderfer08] as well as structural inspection [Ellis95], marketing [Aviv05; Rusmevichientong01] and inventory control [Treharne02].

In this thesis we are focusing on the diverse application domain of active sensing [Hoey07, Spaan10]. The following section gives the overview of the published research in the domain of our interest.

2.2 Related work

Existing work on decision-theoretic sensor management shows that there are still a lot of challenges especially in the context of mobile phones which have limited computational power or battery life, and for which the hidden state changes constantly, and the planning horizon is large. An effective solution to the limitations of mobile devices in energy and bandwidth is context-dependent selective sensing of the environment. Moreover, integrated or accessible sensors usually rang in quality and utility to the current task. Thus, adaptive sensing could be viewed as a resource management problem [Chong09], where dynamic sensor selection with respect to the context could maximize sensing performance. In this way developing a smart sensing strategy would allow saving battery and bandwidth by querying sensors that contribute to the task the most.

In this thesis we are focusing on dynamic or multistage resource management, which is a **nonmyopic** type of a planning scheme. In those kinds of schemes we are looking for optimal solution over the long-term planning horizon, while in **myopic** resource management the optimization is focusing on the short-term performance within the current decision time step.

The "active sensing" (smart sensing, selective or adaptive sensing) problem has recently became a subject for increasing research interest. The growing demand in effective sensor management is due to the development of modern sensing systems, which require capability of managing resources they have access to. Numerous operational constraints of the real world, such as power, bandwidth or time limitations, restrict simultaneous use of all resources all the time. Those justifications stimulate the growth of active research in the area of cost-sensitive data acquisition [Ji].

A typical solution for those kinds of tasks is a policy of the optimal sensor configuration schedule that will satisfy the system requirements of the demanded information and at the same time fit the operational constraints. From that point of view, the problem of sensor management could be naturally formulated within the field of decision–theoretic methods, which gives a rich variety of algorithmic choices for developing optimal planning policies.

The earliest applications of fully and partially observable Markovian processes to the task of sensor management go less than a decade back. An idea of utilizing a comprehensive mathematical framework for assessment and optimization of sensor and inter-sensor scheduling appeared in 1994 in a survey of Musick and Malhotra [Musick94]. At that time, applications of POMDP frameworks to the large-scale sensor management problems were confronted with the issue of intractability of existing exact solving methods, which forced the research in the area of approximate solutions.

Castañón in 1997 [Castañón97] made a first successful attempt to solve sensor management problems via POMDPs using policy rollout approximation. There he formulated the problem of dynamic scheduling of multi-mode sensor resources for classifying a large number of stationary objects. The POMDP-modeled optimal control in robot navigation systems by Evans and Krishnamurthy was published in 2001–2002 [Evans01; Krishnamurthy02]. They presented a near optimal finite dimensional algorithm for HMM sensor scheduling with additional complexity considering a nonlinear cost function of the information state.

Different approximation dynamic programming-based techniques to the optimal POMDP sensor management solution, including offline learning and rollout, are discussed in [Hero08]. A large class of reinforcement learning methods is presented in [Malhotra97; Cassandra98; Chong09] that use offline learning techniques such as offline simulation to explore the space of policies. The example of real-time rollout suboptimal policy approximation using online simulation was given in [Li09]. This problem approximation uses a simpler approximate model or reward function for the POMDP as a proxy for the PS problem and includes bandit and information gain approaches, discussed in the [Hero11].

Krishnamurthy proved an existence of threshold-based optimal single-sensor scheduling policy for a POMDP under special conditions with respect to the monotone likelihood ratio (MLR) ordering [Krishnamurthy07]. Like in [Krishnamurthy02], authors consider non-linear (in the belief state) reward functions. However, in this case we no longer have a property of optimal value function to be piecewise linear and convex. Thus, moving away from the standard POMDP setting, we have a risk of not being able to apply the majority of existing successful approximate algorithms that have been designed for standard POMDPs.

An example application of reinforcement learning for robotic active sensing is found in publication of Kwok and Fox [Kwok], where the sensing strategy is learned via least squares policy iteration. Although publication refers to the year 2003 before the latest breakthrough innovative approximation algorithms were established, real robot simulations indicated the reasonable behavior of the learned policy.

In [Ji07] an underwater sensing application of POMDP addresses the problem of multiaspect sensing on a single platform. There an observation model is learned from physical data, which increases policy reliability. However, technically the authors are solving a classification problem, which differs from our approach. More generally, the problem of cost-sensitive classification has been the subject of much research [Turney00; Chai; Greiner02], in which the objective is to select a subset of available features in order to minimize the cost of making a classification error. More recent work explicitly considers the combinatorial selection of features [Bilgic07], and has investigated theoretical properties of such a selection [Frazier10]. Cost-sensitive feature acquisition has also been applied to information mining from the internet [Kanani10].

One of the latest papers in Markov-optimal sensing policy [Wang] formulated a problem as a special simplified case of the POMDP model, in the Constraint MDP (CMDP) framework. In the designed CMDP model all sensors report observations with perfect accuracy. An energy budget constraint is implemented as a separate state of battery indicator or as a cost constraint applied on every sensor querying action. The paper suggests a computationally efficient algorithm to derive the optimal sensor sampling policy under the assumption that the user state transition is Markovian while minimizing user state estimation error under a given energy consumption budget. The constraint optimization problem is formulated as an infinite-horizon CMDP and solved via a corresponding Linear Programming problem. The obtained Markov-optimal sampling policy is compared to uniform (periodic) sampling applied on the collected user data. However, the notion of "Markov-optimal" sampling policy is not explicitly proved to be optimal there. More likely it is only an approximation of the optimal POMDP policy. There are 2 special conditions that hold for: i) the assumption of 100% accurate state detection; ii) a context-independent approach of state estimation (all states have to be verified each time when not all of them necessarily are needed to be verified). This makes the proposed model valid only within the above conditions.

[Krause05] addresses the problem of power consumption in context-aware mobile computing. The authors propose a selective sampling technique in order to reduce the number of required sensor readings to track the user activity. The objective is to increase the deployment lifetime of a wearable computing platform maintaining high prediction accuracy. The selective sampling problem is modeled as POMDP to determine whether the SVM classifier needs to be invoked to predict the user’s activity.

An alternative approach of the same active sensing problem was demonstrated in [Ji] but with the simple objective of classification. The authors integrate the class-dependent hidden Markov model (HMM) formulation into a POMDP as a myopic approximation to the non-myopic POMDP solution. There they use a HMM for explicit consideration of the sequential order in which the observations are made. The authors try to avoid the issue of undesirable repeated actions (querying of the same feature multiple times). This could be a valid realistic constraint given the real domain applications of health area where multiple health tests might be harmful to execute on the patient. But the drawback of using a HMM is the time cost of the learning phase for building the HMM classifier. For problems with very large sets of features learning phase could become a major problem time wise. However, experiments on the Pima Indian diabetes dataset demonstrate the advantage of myopic approximation behind the POMDP PBVI policy both in performance and policy computation time.

One of very few active-sensing papers in the area of chemical sensing is recently published in 2010 [Gosangi10]. There authors utilized a POMDP model formulation for developing an “active perception” strategy that allows to modulate the temperature profile of the Metal-oxide gas sensor to enhance sensitivity. The temperature optimization process is solved within the framework of sequential decision making under uncertainty where the system must balance the cost of applying additional temperature steps (sensing actions) against the risk of misclassifying the chemical analyte (classification actions). Underlying POMDP sensor dynamics are modeled as an input-output hidden Markov model (IOHMM), which is a generalization of the traditional HMM. The policy design is accomplished using same myopic policy as [Ji]. However, the policy is developed under assumption of a learned IOHMM model, which requires resources for training for each individual class.

Another example of using the POMDP model for taking into consideration the cost of querying sensors to make decisions about which sensors to get information from in which states is mentioned in the previous section [GOSSIP11].

POMDP approaches have been applied to many different sensing systems. One of the most active areas of application has been distributed multiple target tracking [Liu07]. In the family of chain graphical models [Krause09] authors are solving the problem of sensor placement for optimal coverage. However, the exact policy of sensor selection is only determined for only one sensor. Implementations of sensing systems managers using POMDPs and reinforcement learning could be found in projects like: multifunction multi-target radar [Krishnamurthy09], Multisensory multi-target tracking [Li09; Aeron08; Fei10], single-sensor one-target tracking [He04], passive radar [Hanselmann08], Wireless Sensor Networks [Chobsri08] and air traffic management [Kochenderfer08]. The general objective is to find an optimal schedule for one or multiple sensors so the information acquisition cost will be minimized.

[Spaan09] uses Symbolic Perseus for a single-target tracking. The POMDP model presented in [Spaan09] models the selection of a subset of cameras at each time step in order to minimize the uncertainty about the person’s true location. The proposed method selects k sensors each time frame and was successfully tested on a real-world network of 10 cameras. However, the authors do not incorporate the cost schema of the sensor calls in their model, so the utility is focused only on the coverage objective.

In the recently published paper [LaCasa] the authors discuss a context-aware sensing system for a hand-held device (prototype developed for Android platform), called La-Casa (Location and Context Aware Safety Assistant). The proposed assistant system uses multiple sources of local contextual information to reason about stochastic temporal events and makes decision theoretic choices about providing verbal prompts, visual aids, or other help to assist the user with dementia when needed. The system's controller is based on a POMDP. The system models uncertainty, learns about a person's patterns of behavior, and can reason decision theoretically about the costs of sensors (e.g. battery charge) and the relative costs of different types of assistance.

Even though POMDPs represent a very flexible non-myopic decision making model with partial observability, some existing work applies this concept in a limited manner. For example, in the area of autonomous robots [Guo03] still only the problem of cost-sensitive classification (cost of features) with misclassification cost is considered where the hidden state is fixed. The same limitation of a fixed hidden state is also present in [Sridharan10] where the system has to gather information about a static (fixed hidden state) scene in a computer vision setting.

For more detailed insight in the theory, algorithms, and applications of sensor management we refer to the [Hero11]. In [Liao09] one can find efficient sensor selection algorithms for solving both of sensor selection problems: maximum information gain within a budget limit and the trade-off between information gain and cost.

Figure 2.1 illustrates the conceptual connections between the general area of sensor management and scheduling, the framework of partially observed Markov decision process, existing POMDP solution algorithms, model compact representations and thesis research. In particular, in this thesis, we present an active-sensing approach to a cost-sensitive sensor selection for context-aware smartphone applications. We formulated the problem as a partially observable Markov decision process (POMDP), and generate non-myopic solutions from our model specifications using a point-based value iteration approach approximate planning algorithm Symbolic Perseus [Poupart05], that uses a structured representation of value functions and beliefs using algebraic decision diagrams (ADDS) [Poupart05].

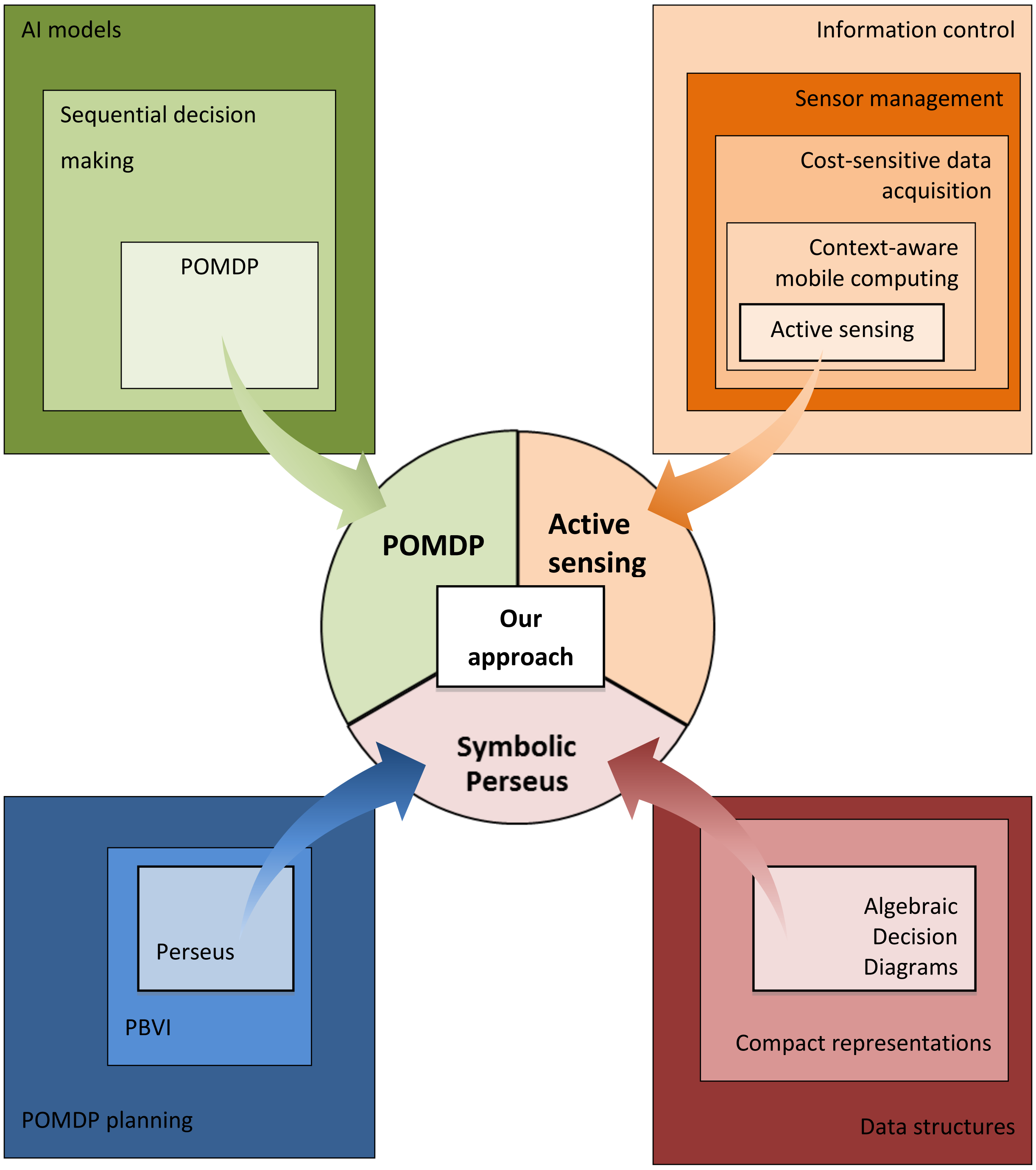


Figure 2.1. Conceptual connections between decision theory, information control, POMDP planning, structured representations and thesis research.

Chapter 3
Probabilistic Decision Making in Markovian Environments and algorithms

In the core of AI planning is an idea of developing a strategy to create the autonomous intelligent agent. Most often in a decision making problem the agent is specified by the set of actions he can choose from and goals expressed as specific states to be achieved. The world is usually represented as a set of states that the agent can occupy. The time is often considered to be discrete, i.e. counted as fixed time steps. Also it makes the planning less complicated if we make the evolutionary process of the agent-world interaction as Markovian. The process is called **Markovian** [Puterman05], or has the Markov Property, if the distribution over the future states of the process depends only on the current state and not past states. In this thesis we only consider working within the stochastic nondeterministic decision-making processes, with uncertainty about the next state of the world.

In this chapter we give the detailed information of the Markov Decision Process (MDP), and discuss some basic methods for MDP planning. We then derive the partially observable extension to the MDP, the Partially Observable Markov Decision Process (POMDP) and focus on a fastest of existing point-based value iteration methods of approximate POMDP planning - Symbolic Perseus.

3.1 Markov Decision Processes

3.1.1 Description / Model definition

The Markov Decision Process (MDP) framework is the classical approach to present and solve stochastic decision making problems [Bellman57]. We can define a MDP as a tuple $\langle S, A, T, R, \gamma \rangle$.

State space $\boldsymbol{S} = \{s_0, s_1, \dots, s_{\mathrm{n}}\}$ is a finite discrete set of all unique states that agent or environment can occupy (in some cases could also be infinite or continuous). An example of a state is the location of the tiger in a POMDP Tiger model [Appendix A]. In this thesis we will work with discrete models with a finite number of states.

Action space $\boldsymbol{A} = \{a_0, a_1, \dots, a_{\mathrm{m}}\}$ is a finite discrete set of all unique actions that the agent can execute (in some cases could also be infinite or continuous). At each time step an agent can execute only one action. An agent could decide to wait but this will be considered as a specific action with its own stochasticity of changing the world state. It is possible to specify the set of available actions for each state. But we are focusing on the model where we have one fixed set of actions uniformly for all states. Also in this thesis we assume that the action set is finite. An example of an action could be opening the door in a Tiger example.

Transition function *T* represents the stochastic nature of an action. We define the transition function T as a state-action transition table $T(s'|a,s) = \Pr : S \times A \times S \rightarrow [0,1]$, with $\Pr(s'|s,a)$ denoting the probability of moving from state s to s' when action a is taken;

$$\mathrm{T}(s'|a,s) = \Pr(s_{\mathrm{t+1}} = s'|s_{\mathrm{t}} = s,\ a_{\mathrm{t}} = a)\,\forall t,$$

where s_{t} is the state at time step t, a_{t} is the action taken at time step t, and s' is the future state at time $t+1$. Thus the transition function satisfies the Markov property, i.e. it predicts the future state probability distribution given just a state-action pair regardless the history.

$$\Pr\,(s_{\mathrm{t+1}} = s'|s_{\mathrm{t}}, a_{\mathrm{t}}, s_{\mathrm{t-1}}, a_{\mathrm{t-1}}, \dots, s_0, a_0,) = \Pr\,(s_{\mathrm{t+1}} = s'|s_{\mathrm{t}}, a_{\mathrm{t}}).$$

Reward is a function $\mathrm{R}(s,a) = S \times A \rightarrow \mathbb{R}$ that assigns a scalar value r_t for taking action a when in state s. The rationales for agent's actions are expressed in positive rewards (payoffs) and negative rewards (costs). Rewards are guidelines for an agent in terms of picking the best action to move from undesirable states to the target ones. That's why (in the classic POMDP formulation we are focusing on here) the reward function assigns a reward not to the state but to the pair {state, action to execute}. However, there exist models where reward could be assigned to only state, or {state, action, next action} [Puterman05].

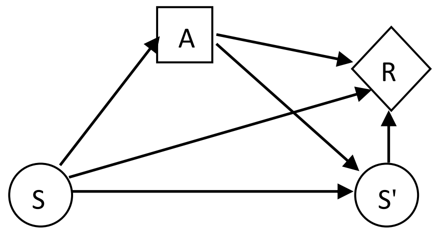


Figure 3.1. 1-step MDP as a dynamic decision diagram.

Discount - a numerical factor $\gamma \in [0,1]$ that discounts more the future reward of every next time step. The reward at time step t is discounted by the factor $\gamma^{(t-t_0)}$, where is to the start time. The objective of a discount factor is to assign a greater importance to the nearest future decisions as we are more certain in the nearest future because of the accumulative property of the uncertainty. The closer is γ to 1, the more valuable future decisions are. But if γ is smaller than 1 (close to 0), that makes agent more interested in maximizing local immediate utility, hence accomplishing short term goals. The process is guaranteed to converge as $\gamma^{(t-t_0)}$ is 0 in infinite time if $\gamma < 1$.

The objective of planning is to develop an action strategy that will maximize **the total discounted expected return** defined as following expectation expression (for finite horizon case):

$$E\left[\sum_{t=t_0}^{T} \gamma^{(t-t_0)} r_t\right],$$

where $r_{\mathrm{t}} = r(s_t, a_t)$ is the reward gained at time step t, y is a discount factor and t_0 and T are the start and end times of the process, respectively. For simplicity we consider the time to be finite within the thesis scope. However, the decision process could be run for infinite time.

A **policy** π is a measure of relevance of taking a particular action a being in a given state s:

$$\pi(s, a) = \Pr(a|s_t = s).$$

In other words the policy gives a probability distribution over all available actions for the current state.

A **deterministic policy** for MDP is a policy that assigns unique action each state, i.e. $\pi(s, a_i) = 1$ for a given action a_i, and $\pi(s, a) = 0, \forall \mathrm{a} \neq a_i$. In this thesis we consider only deterministic policies since they are more convenient to work with. Therefore, we are interested in policies of the form $\pi(s) \rightarrow a_\pi$, where we have a state-action mapping.

Let P be a set of some policies. An **optimal policy** $\pi^*(s) \in P$ is such a policy that no other policy from a policy set P could give a better return regardless the initial state s. One of the fundamental properties of MDP policies was proven by Bellman [Bellman57]. He showed that for any given MDP, there always exists a deterministic optimal policy π^*.

3.1.2 Planning

3.1.2.1 Bellman Equations

In order to compare different solutions for policies, provided by solving algorithms or methods, we need the measurement or the criteria in a form of expression. In MDP policy evaluation the notion of the **state-value function** $V^\pi(s)$ plays an important role. It is defined as follows:

$$V^\pi(s) = E_\pi[R_t|s_t = s] = E_\pi\left[\sum_{t=t_0}^{T} \gamma^{(t-t_0)} R(s_t, \pi(s_t))\right] = E_\pi\left[\sum_{t=t_0}^{T} \gamma^{(t-t_0)} r_t | s_t = s\right].$$

In other words the value function collects total discounted reward for all of the time steps of decision making started from initial state s and acting with respect to the associated policy π.

Then a **state-action value function** $Q^\pi(s, a)$ is defined as follows:

$$\begin{aligned} Q^\pi(s, a) &= E_\pi[R_t|s_t = s, a_t = a] \\ &= E_\pi\left[\sum_{t=t_0}^{T} \gamma^{(t-t_0)} R\big(s_t, \pi(s_t)\big)\right] \\ &= E_\pi\left[\sum_{k=0}^{T} \gamma^{(t-t_0)} r_t | s_t = s, a_t = a\right]. \end{aligned}$$

The state-action value function returns the value function given an additional condition of starting from initial state s, and following the policy π.

Now having the formal expression of the value of policy π over particular state s, we can define the optimal value function. As it was stated above, an optimal policy $\pi*$ is a policy that dominates above all other policies. We can formally define an optimal policy π^* as:

$$\pi^* = \arg\max_{\pi \in P} V^{\pi}(s).$$

Then the **optimal state-value function** $V^*(s)$ is simply the value function of an optimal policy:

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi \in P} V^{\pi}(s).$$

Similarly, the **optimal state-action value function** $Q^*(s,a)$ will be:

$$Q^*(s,a) = Q^{\pi^*}(s,a) = \max_{\pi \in P} Q^{\pi}(s,a).$$

Due to Bellman [Bellman57] the state-value function $V^{\pi}(s)$ recursively accumulates rewards with time. More precisely, the value function of agent acting from the state s under the policy π is the sum of immediate reward and discounted future value of acting from the next states w.r. to the state distribution specified by the transition:

$$V^{\pi}(s) = \mathrm{R}(s, a_{\pi}) + \gamma \textstyle\sum_{s' \in S} T(s'|s, a_{\pi}) V^{\pi}(s').$$

Thus, the optimal state-value function satisfies the same recursive equation:

$$V^*(s) = \max_{\pi \in P} [\mathrm{R}(s, a_{\pi}) + \gamma \textstyle\sum_{s' \in S} T(s'|s, a_{\pi}) V^*(s')],$$

and similarly this holds for the optimal state-action value function $Q^*(s,a)$:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in S} T(s'|s, a_{\pi}) V^*(s'),$$

$$V^*(s) = \max_{a \in A} Q^*(s,a).$$

These are known as the **Bellman equations** for the value function.

The optimal policy π^* now could be expressed via the optimal value function V^*:

$$\pi^*(s) = \operatorname*{argmax}_{a \in A} [\mathrm{R}(s, a_{\pi}) + \gamma \sum_{s' \in S} T(s'|s, a_{\pi}) V^*(s')].$$

This tells us that the optimality of a policy depends on a choice of best action that will maximize the combination of immediate reward and total discounted future return.

3.1.2.2 Value Iteration

Dynamic Programming methods have been widely applied for finding the optimal value function in the MDP framework. Popular examples of those are policy evaluation, policy iteration and value iteration [Bellman57, Howard60].

Value iteration [Bellman57] consists of two basic steps to compute a near optimal value function: **dynamic programming (DP) update** and **termination test**. **DP-update** is a step of computing a next value function from the current**.** It can be written as a **backup operation**:

$$V_{k+1}(s) = \max_{a \in A} [\mathrm{R}(s, a_\pi) + \gamma \textstyle\sum_{s' \in S} T(s'|s, a_\pi) V_k(s')].$$

The value iteration algorithm starts with the initialization of the value function. Then it repeatedly computes the future value and updates the total reward based on the backup equation. With every next iteration we approximate the outcome to the optimal value function, until we reach some stopping condition, such as a certain accuracy:

$$\max_{s \in S} |V_k(s) - V_{k-1}(s)| \leq \delta, \delta \geq 0.$$

This is a termination test expressed as the Bellman residual (maximum absolute difference between 2 consecutive approximations of the value function). If this condition is met, the value iteration terminates. Any time we compute the next approximation, we can get a policy π_{k+1} directly from the value function V_k:

$$\pi_{k+1}(s) = \operatorname*{argmax}_{a \in A} [\mathrm{R}(s, a_\pi) + \gamma \sum_{s' \in S} T(s'|s, a_\pi) V^*(s')].$$

It has been proved [Bellman57] that there exists a constant that bounds and approximation error if stopping condition is satisfied:

$$\max_{s \in S} |V_k(s) - V_{k-1}(s)| < \tfrac{2\gamma\delta}{1-\gamma}.$$

A policy π is defined to be **ϵ-optimal** if there exists a constant ϵ, s.t. $\max_{s \in S} |V_k(s) - V_{k-1}(s)| < \epsilon$. Hence, to compute the ϵ–optimal policy we have to run the value iteration backup operation until we have a convergence to the accuracy of $\frac{(1-\gamma)\epsilon}{2\gamma}$, which is:

$$\max_{s \in S} |V_k(s) - V_{k-1}(s)| < \tfrac{(1-\gamma)\epsilon}{2\gamma}.$$

This type of algorithm, when at any time step we can get the approximate solution with the known approximation error, is called an **anytime algorithm**.

The desirable ideal goal of stochastic planning is to compute the exact optimal policy. The only method theoretically proven to converge to the exact optimal solution in polynomial time is Linear Programming [Puterman05]. However, realistic application areas are of complexity that makes solving for the optimal policy not tractable. Therefore the research of the past decade for solving the MDP-based models has focused on the developing approximate algorithms rather than exact.

3.2 Partially Observable Markov Decision Processes

One of the greatest advantages of the POMDP is that the evolution of partially observable state is done considering the past actions and observations which makes the decision making process more reliable in terms of goal achievement than memory-less acting. Developing a long term strategy is an essential objective in modern robotics and mobile phone application domains. Several decent POMDP algorithmic surveys and PhD dissertations have been published in the past decade [Cassandra98; Zhang01; Shani07P; Poupart05; Shani12; Spaan12].

Partially Observable Decision Making Process models consist of a set of stochastic relational tables that manage the agent-world interaction. They are encoded in probabilistic transition tables and rewards of being in any state. The general purpose of many POMDP agents is to achieve a goal state (which is usually associated with the biggest reward) following an action strategy. During the interaction process with a stochastic environment, the agent is seeking a compromise between achieving long term and a short term goals due to the limitations of state uncertainty and time constraint.

Unless the action strategy is random, the chosen POMDP solver should compute the action plan before the agent will start acting. While performance of the agent is measured by achieved goals, the quality of strategy is optimized by the chosen measure function. This optimization criterion is a guideline to select the action (for each possible situation) that will deliver that maximum benefit. Rewards, horizon and discount factor are parameters that we can tune to direct the resulting action strategy. For example, assigning a large reward to some state will stimulate the solver to look for fast ways to reach this state and to avoid being in others for a long time [Spaan05].

Infinite horizon planning gives an agent a certain benefit in making very accurate decisions. When the agent is not under any time constraint, it can make as many as possible attempts to verify the current belief state and make more accurate decision.

During the process of interaction with the stochastic environment, the agent takes actions and observes the states, being not certain neither in the current state nor in the result of next action. Uncertainty in the future environmental state affected by the action is inherited from the MDP model. Uncertainty in the current state comes from the partial observability of the belief state due to the noisy observation signals. Hence, the POMDP model extends stochastic the MDP model by adding the observation influence on the belief state.

The sources of partial observability could be found in either the imperfection of sensor readings (represented as sensor "noise") or the existence of a disconnection between the observation and state spaces (not all sates are covered by observation variables) where the agent doesn't have access or ability to sense for special states.

Also the "perceptual aliasing" phenomena could become an additional difficulty when due to the perceptual limitations/ambiguity of the system some states could be indistinguishable but yet require different actions with respect to the best (optimal, most beneficial) strategy [Spaan05].

3.2.1 Framework

The POMDP model extends the MDP model by adding partial observability. More formally, a **Partially Observable Markov Decision Process (POMDP)** is a tuple $\langle S, A, T, R, Z, O, \hbar, y \rangle$. The components $S, A, T, R,$ and y are inherited from the MDP model. Together they form a tuple (S, A, T, R)

that is called the **underlying MDP**. Thus, the POMDP extensions are the observation set Z and the observation function O.

Observation set $\boldsymbol{Z} = \{o_0, o_1, \dots, o_L\}$ is the finite set of all possible observations that are accessible by the agent (in some cases could be also infinite or continuous). Observations are a toolset to verify the current environment state. Thus they have to be associated with a state set and with a particular action. Observation could be done during the action or could be an action itself. The stochastic association of the observation set with an action set and the state set is encapsulated in the state-observation function O.

The state-observation function $\boldsymbol{O}(o|s,a) = Pr : S \times A \times Z \rightarrow [0,1]$, with $Pr(o|a,s)$ denoting the probability of making observation o while the system occurs in the state s after taking action a.

$$O(o|s,a) = Pr(o_{t+1} = o | s_{\mathrm{t}} = s, a_{\mathrm{t}} = a), \forall t.$$

Note that in the case of MDP we have fully observability, thus, $S = \boldsymbol{Z}$.

Horizon $\hbar$ is a time frame for the agent to act. It defines a bound to a number of time steps or actions that the agent is allowed to take. If the agent has to reach a goal (the typical case – maximizing the long-term reward) in a finite number of time steps, then we say that the horizon is **finite**. It follows from Bellman's equation that in the case of a finite horizon the choice of the optimal action directly depends on the number of remaining time steps. The horizon length defines the goals that could be accomplished. Therefore the decision making depends on the current state observations and the remaining time steps. For example, in the case when there is little time left the agent will rather aim for short-term goals than long-term. In the case of the infinite horizon, time that is left is not an objective. In this case the only objective would be maximizing the infinite sum of discounted rewards. In this thesis we are focusing on the POMDP models with finite horizon and discount factor less than one.

In this thesis we consider all components of the POMDP to be stationary, i.e. not changing with time.

3.2.2 History and Belief States

The system actions cause stochastic state transitions, with different transitions being more or less rewarding (reflecting the relative utility of the states and actions). Unlike in a MDP, in a POMDP current states cannot be observed exactly, i.e. the agent has no certainty in the current state s_{t}. However, the agent can make observations at each step, which supplies an agent with some feedback knowledge from the environment. The stochastic observation model relates observable signals to the underlying state. In realistic domains the world's feedback contains a certain amount of noise, hence, is not fully reliable and insufficient for the full-state verification.

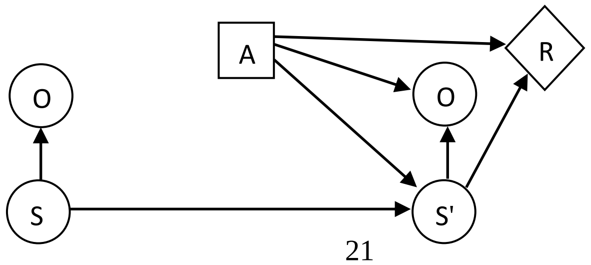


Figure 3.2. 1-step POMDP as a dynamic decision diagram.

The evolution epoch of a POMDP is usually split in a sequence of single time step chunks. At each of them a belief state is updated after the last action and the agent is rewarded accordingly. The belief update is defined by the transition model $T(s'|s,a)$ [Fig. 3.2]. The result of transition to a new state is completely defined by the previous state and executed action, which illustrates the Markovian property of the process. It is a classic POMDP model design when the observation of an environment is happening each time the agent is acting. However, in some models the observation could be an action-dependent occasion. There could be actions when the agent does not get any observations or actions targeted the environment observation exclusively. In the observational case, the agent updates the belief state according to the observation model $O(o|s,a)$.

One of the obvious ways to trace the agent's state is to keep track of the entire **history** of the agent's actions and observations. We define a history $\hbar_{\mathrm{t}}$ as

$$\hbar_{\mathrm{t}} = \{a_0, o_1, a_1, o_2, \ldots, a_{t-1}, o_t,\}.$$

In some cases this way of preserving history is efficient, when we can make it tractable by bounding the horizon in the case of not noisy observation function. However this way is not the most efficient for many tasks with a large state space and long horizon. A more compact way of history representation is maintaining a vector of a **belief state $\boldsymbol{b(s)}$.** The first advantage of such a representation is a significant reduction in the usage of memory, dealing with one vector instead of keeping the whole history. A second advantage is that the same belief state could be a result of more than one scenario of the decision making evolution. Thus, working with a belief state is easier than computing a policy over histories. A constantly updated belief state of the world carries stochastic environmental changes. At each time step the updated belief state is a distribution of probabilities between all states, where $0 \leq b(s) \leq 1\ \forall s \in S$. Due to the property $\sum_{s \in S} b(s) = 1$ the dimensionality of a continuous belief space ΔS could be dropped to ($|S|-1$) because we can uniquely express the current belief state of a system using only $|S|-1$ numbers.

The initial belief could be inferred from the context of the domain. If not, another approach is to assign the uniform probability distribution to the initial belief state so the agent will start planning without being biased towards a certain possible state. This assumption is done for developing the action strategy that will not be biased towards trajectories of belief states associated only with this state. Those traces could be too poor to represent the whole variety of reachable belief points, which we are more likely guaranteed to cover starting from the uniform belief state probability distribution.

Let τ be a probability function that transits current belief state b to a future belief state b'. After taking an action a at the belief state b and observing o, the probability of ending in some state $s', b'(s')$, is computed according to Bayes' rule as follows:

$$b'^{a}_{o}(s') = \tau(s'|b,a,o) = Pr(s'|o,a,b)$$
$$= \frac{Pr(o|s',a,b)\,Pr(s'|a,b)}{Pr(o|a,b)}$$

$$= \frac{Pr(o|s',a)\sum_{s\in S} Pr(s'|a,s)\, b(s)}{Pr(o|a,b)}$$

$$= \frac{O(o|s',a)\sum_{s\in S} T(s'|s,a) b(s)}{Pr(o|a,b)},$$

where $Pr(o|a,b) = \sum_{s'\in S} O(o|s',a)\sum_{s\in S} T(s'|s,a)b(s)$ serves for the normalization purpose.

3.2.3 Belief MDP

In POMDP policy a fixed belief state $b_t(s)$ could be viewed as a single state. More general, the POMDP policy works with a belief states in an exact way as the MDP works with a single state. Thus, the POMDP itself can be represented as a special case of MDP - the **Belief MDP** (MDP$_{belief}$) [Cassandra98]. We define MDP$_{belief}$ as a tuple $\{S, A, \tau, R_{belief}\}$, where:

- state space S is the belief space of the POMDP,
- action space A remains the same as the action space for the POMDP,
- transition function τ specifies the transition probability from a belief state b to the next state $b^{'}$ after action a:
- $\tau(o,a,b) = Pr\left(b^{'}|b,a\right) = \begin{cases} Pr(o|b,a) & \text{if } b^{'} = b_o^a \text{ for some } o, \\ 0 & \text{otherwize} \end{cases}$
- reward function $Rbelief$ specifies immediate reward given a belief state b and action a as a sum of MDP rewards weighted with respect to the state probability distribution b:

$$R_{belief}(b,a) = \sum_{s\in S} b(s)\mathrm{R}(s,a).$$

The illustrated transition from a POMDP to the belief space MDP is straightforward and transparent; thus, policies computed using either of the representations are equivalent. Since for a given MDP$_{belief}$ there always exists a deterministic optimal policy $\pi^*(b)$, we can say that there always exists an optimal deterministic policy for a finite POMDP under a finite horizon.

3.2.4 Policies and value functions

3.2.4.1 Policies

It was mentioned above [Subsection 3.1.1] that the outcome of planning is an action plan, that the agent will follow to succeed in a goal accomplishment starting from any belief state. This action plan is called a policy $\pi(b)$. While in the MDP case the policy is a state-action mapping, in the POMDP case we are working with the belief states, so we have to re-formulate the policy definition. POMDP deterministic **policy** is a map between belief states and action to take $\pi(b) \rightarrow a$, in order to maximize the expected gained rewards with respect to the chosen criterion. Different optimization criteria can be considered, [Littman96] such as the expected or worst case, sum of total or average, discounted or undiscounted rewards. In this thesis we use the most popular in the research literature expected total discounted reward as a measure to evaluate and select policies.

A deterministic policy $\pi(b)$ is **stationary** if it prescribes the same unique action for the same belief state. In other words, the decision about the next action choice is only a function of a current belief state, thus such policy is time-indifferent.

Similarly to MDP, we define a value of a finite horizon policy $\pi(b)$ as expected reward accumulated along the decision process as:

$$V^{\pi}(b_0) = E_{\pi}\left[\sum_{t=t_0}^{T} y^{(\mathrm{t-t_0})} R(b_t, \pi(b_t))\right] = E_{\pi}\left[\sum_{t=t_0}^{T} y^{(\mathrm{t-t_0})} \sum_{s\in S} b_t(s) R(s, \pi(b_t))\right],$$

which is the expected sum of the discounted rewards executing some policy π from the initial belief state b_0. Here, $b_t(s)$ a probability of being in a state s and $\pi(b_t)$ denotes the action associated in policy π with a belief distribution b_t.

3.2.4.2 Value Functions

POMDP value functions are much harder to track than those for an MDP, since they are much more saturated with information due to the large state space and complexity of value function and policy representations. The development of an action strategy could be described as an evolutionary process in time measured by time steps left to take. In case of only one time step left the agent is rewarded according to the single action it will choose to execute. When the horizon left is 2 the agent will make an observation between 2 actions, so the last action choice will directly depend on the result of observation. In general, any non-stationary t-step policy can be represented with a policy tree, where the depth of a tree will correspond to the policy horizon and the root to the first action to take [Cassandra98].

The expected discounted value of a policy tree π first depends on the initial belief state of the world. In the trivial case of 1-step left horizon the value of executing a single-action step policy from a state s defines by the corresponding reward scalar.

$$V^{\pi}(s) = \mathrm{R}(s, a_{\pi}),$$

where a_{π} is the action of a root of policy tree π. More generally, if π is a t-step policy tree, then we evaluate executing a t-step policy as a sum of immediate reward and the discounted expected future rewards:

$$V^{\pi}(s) = \mathrm{R}(s, a_{\pi}) + \gamma \sum_{s\prime\in S} T(s'|s, a_{\pi}) \sum_{o\in Z} O(o|s', a_{\pi}) V_{o_{\pi}}(s'),$$

where o_{π} is the $(t$ - 1)-step policy sub tree that corresponds to the observation o. The expected value of the future is a sum of values of executing a policy for all observations over all possible next states. The combination of resulting state and gained observation will determine the choice of policy sub tree.

Due to the state uncertainty; each time we are dealing with the probability distribution over all states. Thus, to evaluate the gained value from policy execution we have to sum up all values of executing the policy in each state.

$$V^{\pi}(b) = \sum_{s\in S} b(s)V^{\pi}(s).$$

A policy π which maximizes V^{π} over all given set of t-step policies P is called an **optimal policy** π^*. The corresponding optimal value function V^* guarantees the choice of a best (optimal) action strategy to maximize total gained reward:

$$V^*(b) = \max_{\pi\in P} V^{\pi}(b).$$

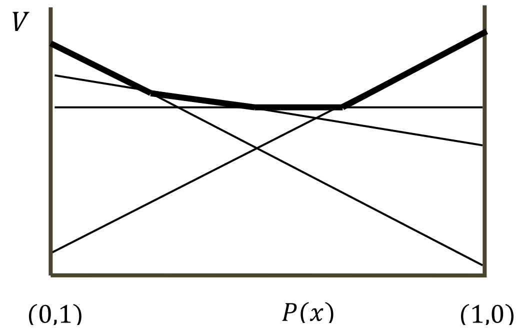


Figure 3.3 Geometric representation of a piece-wise optimal value function in a two-state POMDP belief space. The x-axis represents belief state, and the y-axis corresponds to the value of each belief. The value function consists of pieces of dominating α-vectors.

For the equations above it is easy to see that the value function is linear over the belief state vector $b(s)$. Thus, at each time step of policy executing it could be presented as a hyper plane over the belief state space. The optimal value function dominates over all value functions, which means, in the graphical representation the corresponding hyper plane will be positioned above all others. In case of a finite planning horizon the optimal value function V^* will consist of parts of dominating optimal value functions over different regions of belief space, so it will be piecewise linear and convex (PWLC) [Sondik73], and for infinite horizon tasks V^* can be approximated arbitrary well by a PWLC value function. At each time step, the dominating vector is associated with the best/optimal action to take [Sondik71]. Therefore, for any finite planning horizon t the value function is completely defined by the set of vectors $\Gamma_t = \{\alpha_0, \alpha_1, \ldots, \alpha_l\}$. These α-vectors are hyper planes over the continuous belief space, where each α-vector is associated with a specific action. Together they represent an approximation where the value for an α-vector is associated with the total return the agent would receive when executing the associated action at that belief with respect to the current time step.

The value of an α-vector at a current a belief state is simply an inner product of 2 vectors:

$$V^{\alpha}(b) = \sum_{s\in S} \alpha(s)b(s).$$

At a fixed time step t the current approximation of an optimal value function is made of the pieces of dominating α-vectors over different corresponding regions of a belief space:

$$V_t(b) = \max_{a\in\Gamma_t} \sum_{s\in S} \alpha(s)b(s),$$

where Γ_t is the set of α -vectors at time step t.

An α-vector α is **dominated** if for any belief point there exists an α-vector whose value is bigger:

$$\forall b \in \Delta S, \exists \alpha' \epsilon \Gamma \text{ s.t.} \sum_{s \in S} \alpha'(s) b(s) \geq \sum_{s \in S} \alpha(s) b(s),$$

Since a dominated α-vector does not dominate over any belief region, it will not figure in the approximation expression; hence, it will be useless. Thus, a dominated α-vector can be safely removed (**pruned**) without affecting the value function.

3.3 POMDP planning

POMDP solving, or **POMDP planning**, is the process of computing an action policy (optimal or approximate), for a designed POMDP model. This should not be mixed with a problem of formalizing or learning a POMDP model of an environment. Those problems are often addressed by implicit learning techniques like reinforcement learning, which aims to solve the problems of incomplete prior knowledge about the domain [Ross08]. One more area of research is a problem of formulating or coding POMDP model, where different representations or diagrams could serve for optimization.

Regarding the policy manufacturing process, POMDP algorithms are divided into two categories: **value iteration algorithms** (searching the space of value functions) and **policy iteration algorithms** (searching the space of policies) [Sondik73; Hansen98]. In each class, DP update could be computed exactly or approximately. Accordingly, a POMDP algorithm can be **exact** or **approximate** one.

The problem with the POMDP model is that, unlike as in MDP, we are uncertain in the current state, so we cannot solve for the exact optimal policy (with respect to the computational cost) for most practical problems of interest. For example, in finite-horizon POMDPs, the problem of finding optimal policies has shown to be PSPACE-complete [Papadimitriou], in infinite-horizon POMDPs the existence of exact solution is undecidable [Madani]. However, the near-optimal solution could be obtained via a series of approximations.

A variety of computational and design challenges for POMDP solving led to the appearance of different planning algorithms and techniques in past decades. In this section, we will briefly review some of the POMDP exact and approximate planning techniques. This will give us general idea on the evolution of existing solution techniques as well as outline the main computational problems. The insight to some classic exact algorithms is given in Section 3.3.1, followed by the section devoted to the approximate planning in Section 3.3.2. Special attention in this section is given to the branch of point-based solvers (Section 3.3.3-3.3.7) as this type of algorithm is used in this thesis.

3.3.1 Exact planning

As in the MDP case, a near-optimal value function for discounted POMDPs could be computed by a **value iteration** algorithm. Using the notion of the Belief MDP and the MDP backup step expression, we derive a backup operation equation for the POMDP value functions:

$$V_{k+1}(b) = \max_{a \in A}[R_{belief}(b,a) + \gamma \sum_{o \in Z} Pr(o|a,b) V_k(b'(s'))],$$

and similarly to an MDP model, our POMDP policy is derived directly from the value function at any time step:

$$\pi_{k+1}(b) = \underset{a \in A}{\operatorname{argmax}}[R_{belief}(b,a) + \gamma \sum_{o \in Z} Pr(o|a,b) V_k(b'(s'))].$$

Although a belief MDP is functionally equivalent to POMDP, applying the same MDP solving experience to a POMDP planning does not guarantee the same efficiency.

In case of value iteration, at each iteration solving is greatly affected by working with the entire continuous belief space, as well as by the low convergence speed in terms of the number of DP updates before meeting the termination condition [Shani07]. Those two problems are common for all near-optimal algorithms. The problem with a policy space is known as a **curse of dimensionality** [Shani07]. The problem with the excessive number of DP updates due to the size of observation space, action space and horizon length is known as a **curse of history** [Shani07].

Historically, exact planning algorithms were the first techniques proven to generate a POMDP policy [Feng05; Cassandra97; Sondik; Sondik71; Zhang96; Zhang01]. Sondik's **One-Pass algorithm** [Sondik; Sondik71] and Monahan's **Enumeration algorithm** [Monahan82] are some of the most basic plain solution schemas of adapting the MDP value iteration to the POMDP framework. The One-Pass algorithm dynamically builds a set of α-vectors from a set of α-vectors of previous iteration set moving backwards in time from the last time step to the first. For a single planning iteration at a horizon of t, the computation cost is $O(|A||Z||S|^2|\Gamma_t|^{|Z|})$, which exponentially blows the size of α-vectors set with the cardinality of an observation set and twice-exponentially with horizon [Poupart05].

The Enumeration algorithm extends this approach by adding the pruning stage, when we remove dominant α-vectors to shrink α-vector set to minimal representation. This takes away a significant amount of computation from DP backup stage, but the worst case is still exponential.

Among other exact planning algorithms the **Witness** [Littman96] algorithm uses a Q-functions to compute optimal value function. Using a Q-functions the α-vector set is constructed per each action and then combined to get the optimal set. The key computational advantage of the Witness algorithm is that the DP update operation takes only polynomial time with respect to the dimensionality of S, A, Z and the size of α-vectors set of previous iteration. However, the horizon still may give the exponential growth in number of α-vectors.

Also different kinds of heuristics to select belief points were tested to limit the test space to the potentially reachable regions [Zhang01].

Although the exact methods are well-developed, they still require a very high computational cost and deriving near optimal solutions for realistic domain POMDPs appeared to be computationally infeasible. This leads to the need of developing efficient approximation techniques.

3.3.2 Approximate planning

The last decade has seen a great variety of published approaches for approximating a value function for a POMDP. Lovejoy [Lovejoy93] proposed to explore a heuristic of the optimal value function of the underlying MDP to approximate the POMDP optimal value function V^*. Littman et. al. [Cassandra97] extended this idea by more accurate by approximating V^* using the set of vectors that corresponds to a Q-function of the underlying MDP. In this section, we describe some of the heuristics and the core of point-based planning methods.

Heuristic Approaches (MDP-based Approximations) Because of the fact that MDP policies are much easier to compute, it is convenient to use heuristics based on the underlying MDP. The most popular of those methods are **Most Likely State (MLS)** [Nourbakhsh95], **QMDP** [Littman95] **and Fast Informed Bound (FIB)** [Hauskrecht00V]. While MDP policy does not guarantee optimality when applied to the POMDP, it can be used as a valid heuristic guess, due to the fact that underlying belief MDP will still reflect POMDP stochastic in a simplified manner. The MLS heuristic constructs a policy simply picking the best action for most probable state in the current belief:

$$\pi_{MLS}(b) = \operatorname*{argmax}_{a \in A} Q_{MDP}\left(\operatorname*{argmax}_{s \in S} b(s), a\right).$$

As we can notice, this approach reaches its best performance in the situations when there is a leading state in the belief distribution. This is very likely to happen if the agent has access to accurate sensors. At the same time it means that in case of a low certainty in the belief state, this approach is not likely to produce a reliable policy. A slightly more complex heuristic is QMDP, which selects one single α -vector for each action, ($\alpha^a(s) = Q_{MDP}(s, a), \forall a \in A$) for the entire belief space at a time:

$$\pi_{QMDP}(b) = \operatorname*{argmax}_{a \in A} \textstyle\sum_{s \in S} b(s)\, Q_{MDP}(s, a).$$

The policy is developed with a consideration of the weighted sum over the belief state probability distribution; therefore, we are taking into account the uncertainty at the current state. However, this approach ignores the potential uncertainty in the observations. Also the nearest future decisions are not valued over the more distanced decisions, which means shortage of time does not play a role in decision making.

Hauskrecht's FIB heuristic integrates the observation noise into the update step. The produced heuristic selects the optimal action per state with respect to the expected observation.

$$\alpha_{t+1}^a = \mathrm{R}(s, a) + \gamma \sum_{o \in Z} \max_{\alpha_{t+1} \in \Gamma_t} \sum_{s' \in S} O(o|s', a) T(s'|s, a) \alpha_t\ (s').$$

Point-Based methods are a focus of this thesis. Point-based methods are a category of offline POMDP approximation techniques that recently gained popularity due to their superior efficiency over the exact methods. Modern point-based solvers are able to scale to solve for the volume of thousands of states, which is a significant breakthrough in comparison to the past, when solvers were not able to handle more than some dozen states. This breaks down the complexity of large state spaces, which is a significant computational issue in POMDP solving.

The novel approach of approximating belief space with selective belief point sampling became a key improvement opening the new family of algorithms. The main benefit of this approach is avoiding the exponential growth of the value function. Thus, much less computational power is required to solve for a large state-space POMDPs with long horizon.

There were many extensions proposed to the classic PB approach, focusing on either context of the domain or main aspects of the algorithm [Kaplow10]. In general, existing point-based algorithms differ in 3 main aspects: methods of belief selection and the order of value function updates and the set of important parameters (ex. the size of the belief space or the number of updates per iteration).

One can draw a correlation between the point-based methods and greedy based and heuristic-based methods. The reason is that point-based planners have greedy-based and heuristic-based methods as their parents and inherit some of their approaches. For example they inherit from the Grid-based approach the idea to run the DP update over only finite subset of belief nodes. Also various optimizations could be found such as heuristic initialization of value function or α-vector pruning.

In the next section will give the detailed description of core algorithmic components of point-based POMDP solvers.

3.3.3 Point-based POMDP solvers

Earlier we described the exact planning methods where the optimal value function is constructed from the generated set of α-vectors. Each α-vector is updated over the entire belief space. Point-based approaches optimize the update operation by considering only a selected finite subset of belief points. Below we give some processing details about one iteration of a point-based solver.

Let's define the belief update operation over the single belief point b:

$$b' = \alpha_t^b = \arg\max_{\alpha\in\Gamma_t} \sum_{s\in S} b(s)\alpha(s).$$

The backup operation is targeted to look up the α-vector that maximizes the given belief point b. Therefore here we are backing up the best α-vector from the set Γ_t for this belief point b.

The process of creation the new set of vectors starts with building the intermediate sets Γ_1^a (the immediate reward vectors that correspond to the immediately received rewards) and $\Gamma_t^{a,o}$ (plans of length $(t-1)$ based on the previous set of α-vectors Γ_{t-1}):

$$\Gamma_1^a = \{\alpha^a | \alpha^a(s) = R(s,a) \forall s\},$$

$$\Gamma_t^{a,o} = \{\alpha_t^{a,o} | \alpha_i^{a,o}(s) = \gamma \textstyle\sum_{s'\in S} T(s'|s,a)\, O(o|s',a)\alpha'_i\ (s'), \alpha'_i\ \in \Gamma_{t-1}\}.$$

Here Γ_1^a is a set of α-vectors that corresponds to the immediately received rewards for action a. $\Gamma_t^{a,o}$ corresponds to the taking action a and observing o and following the plans of length $t+1$ from previous iteration. The operations to compute these sets are the same as in the exact methods.

We now define the set of α-vectors Γ_b^a:

$$\Gamma_{t,b}^{a} = \Gamma_1^{a} + \sum_{o \in Z} \arg\max_{\alpha \in \Gamma_t^{a,o}} \sum_{s \in S} b(s)\alpha(s).$$

The intuition behind this is that we combine the one-step reward with the value of the best α-vector from $\Gamma_t^{a,o}$ per given belief state over all possible observations.

After computing $\Gamma_{t,b}^{a}$ given b and a, the full backup operation preserved only one α-vector from $\Gamma_{t,b}^{a}$ that maximizes the value at the current belief point b.

$$backup(b) = \arg\max_{\alpha \in \Gamma_{t,b}^{a}} \sum_{s \in S} b(s)\alpha(s).$$

In the point-based approach, building $\Gamma_{t,b}^{a}$ is specified per belief state b, instead of computing it for all possible belief points, as in the exact planning methods. The single iteration of t-horizon full backup computes for $O(|A||Z||S|^2|\Gamma_{t-1}|)$ time.

In the time constraint situations the stopping criterion could be time; in other cases, it is the number of iterations or the approximation accuracy. Note, that at any iteration we can extract the approximation value function.

3.3.4 Point-Based Value Iteration (PBVI)

Point-Based Value Iteration (PBVI) was proposed by Pineau at al. in 2003 [Pineau03P], where she introduced the idea of backing up only the finite subset of belief space, reachable from the initial belief state.

The central assumption of point-based planners is that a representative finite subset of belief space could be chosen so that the set will cover reachable belief regions with high approximation accuracy. Each iteration PBVI extends the beliefs subset by greedily choosing new reachable belief points that are as far as possible from the current belief points. New points are sampled stochastically from the POMDP model, thus are guaranteed to be reachable. At each iteration of belief point collection, new successor points are chosen in a way to improve worst-case density. Thus, the resulting coverage of belief space by collected reachable belief points in the limit will be exact.

Among other parameters that influence on the performance speed and quality of resulting policy, the most important are the size of selected belief subset and the number of iterations in the backup stage. As any other parameters, they provide a trade-off between computational cost and accuracy of result. For example, the approximation quality directly depends on the number of belief points participated in the process. However, the growth in the number of selected belief points slows down the performance of value update stage. The same influence comes from the number of backups.

3.3.5 Perseus

Perseus is a randomized variation of the PBVI that was published in 2005 by Spaan and Vlassis [Spaan05]. The proposed solution improves the computational efficiency in the value-iteration process by reduction in the number of belief updates per iteration without loss in the value function approximation quality. Instead of iterating the belief collection with belief update, Perseus builds a fixed set of belief points B at the beginning, by sampling trajectories of randomly uniformly selected actions at each step, starting from initial belief b_0.

The backups then are performed only on a randomized subset of B. First we initialize “unimproved” belief points set B' to the whole B. At each step in the backup iteration, we backup the randomly sampled belief point b and count it as “improved”. Then we remove from B' all belief points, which provide a better value with new α-vector. The process repeats until B' is empty (all points are “improved”). After emptying B', the whole iteration begins again until the stopping criterion is reached.

Since a single backup iteration potentially improves a value for many belief points in B, much fewer point-based backups are required for each iteration (particularly in the early stages) thus more iterations of belief point updates can be done. Overall, that improves the quality of the resulting value function.

3.3.6 Algebraic Decision Diagrams (ADDs) for Point-Based algorithms

As was mentioned before, researchers in the area of POMDP planning techniques were mainly struggling with two key computational complexity challenges: the curse of dimensionality [Pineau04], corresponding to the complexity of large state spaces, and the curse of history [Pineau04], related to the exponential complexity of policy and value function. Scalable POMDP algorithms can only be established when the cardinalities of belief, action and observation spaces do not blow up the required computational time exponentially, i.e. when both curses of intractability are resolved. The point-based approach mainly addresses the curse of dimensionality by restricting value function updates to be made only on the specifically selected belief point subset. However, even for POMDPs with only few states, the optimal value functions and the number of component α-vectors could be exponential. Thus the curse of history appears to be the most significant obstacle [Poupart05].

This section focuses on techniques for dealing with the complexity of policy and value function spaces. In particular we discuss integration of algebraic decision diagrams into the point-based iteration approach as a tool to compactly represent and efficiently manipulate policies and value functions.

One of the ways to tackle the second issue is to think of dimensionality reduction of vectors involved in main computations.

Factored models can efficiently represent POMDP components and reduce the computational complexity of various algebraic operations performed on vectors in the backing up iterations. A dimensionality reduction could be effectively done by exploiting a compact datastructure representations such as decision trees (DTs) [Boutilier96] or **algebraic decision diagrams** (ADDs) [Hansen00].

ADDs were successfully applied for compact representation of environment dynamics. Also ADDs were used for alpha-vector presentation for some point-based algorithms [Boutilier96; Hansen00; Poupart05, Shani08, Sim08]. The key advantage of using the ADD representation is a high efficiency in basic function sum and product operations; thus, implementation-wise those structures save a lot of computational time.

An Algebraic Decision Diagram [Bahar93] is an extension of Binary Decision Diagrams [Bryant86] that allows its branches to merge together, which provides a compact representation of decision trees. More specifically, ADD shortcuts repeating nodes in the tree structure, providing a unique minimal tree representation. An example of a decision tree transformed into an ADD is shown in Fig. 3.4.

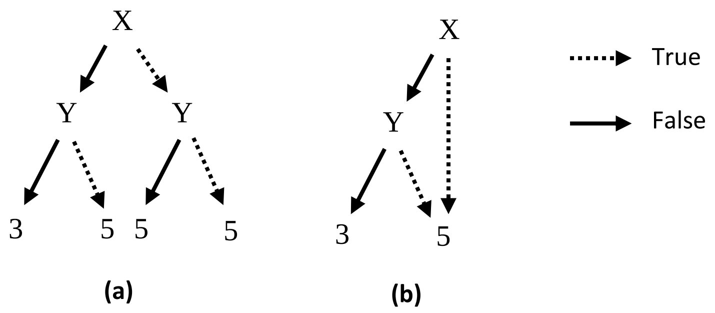


Figure 3.4. The decision tree (a) and the corresponding algebraic decision diagram (b).

ADDs achieve compact representation by aggregating together the identical entries of vectors or matrices utilizing conditional and context-specific independence of random variables and additive separability of functions. Thus, probability and utility tables in the POMDP are allowed to shrink up to the compact representation. The main benefit of using ADD is that the performance of basic arithmetic operations becomes more efficient because all identical entities participate in the operation only once. In the cases when there will be not a lot of identical values, the ADD could be compacted by aggregating similar entries [St-Aubin00; Feng01]. This will keep the size of ADD bounded.

It was shown that ADD-based algorithms scale-up better than algorithms that operate on flat structures [St-Aubin00]. Later, extending this approach to POMDP, Feng and Hansen implemented α-vector computation using compact ADD α-vector presentation [Hansen00]. Based on their work Poupart and Hoey et al. [Poupart05; Hoey07A] represented the belief state and α-vectors with ADD in an implementation of the Perseus point-based algorithm [Spaan05].

3.3.7 Symbolic Perseus

Symbolic Perseus is a point-based value iteration algorithm that uses ADDs as a data structure to represent its components. Symbolic Perseus was first introduced by Pascal Poupart in 2005 [Poupart05] and later implemented by Hoey at al. at 2007 [Hoey07A]. A key optimization of Symbolic Perseus is introducing a bound to the number of vectors representing the value function, which decreases the computational cost of backups operations with approximately no loss in the policy quality. Also a belief state approximation method is used to address the curse of dimensionality issue by merging states with values that differ by less than the Bellman error. This approximation is possible by breaking some correlations in probability distribution, which results in a compact factored representation of a belief state as a product of independent marginal of components. In the Symbolic implementation of Perseus [Shani07P] the author used this factored representation for belief states and ADDs for α-vectors, which resulted in potential savings in required memory and computations.

State transitions and observation stochastics are presented as a set of ADDs: one set to decode the transition probabilities for state variables in component i after performing an action a given all current state variables T; the other set to specify probabilities of observing o after taking action a.

Symbolic Perseus implements basic ADD operations in a more efficient way so the belief state backups and updates are running faster. The scalability of Symbolic Perseus was successfully tested on the factored POMDPs with up to 50 million states. An example of applying Symbolic Perseus for solving large size POMDPs with a specific factorization could be found in the following papers [Hoey07A, Hoey10, Boger05].

Realistic application domains demand scalable POMDP algorithms that would be robust to the two main challenges of curse of dimensionality and curse of history. Symbolic Perseus successfully overcomes the complexity of large state-space by selectively backing up in bounded numbers of reachable beliefs. The symbolic extension of Perseus also solves the curse of dimensionality by compactly representing alpha-vectors and belief states with ADDs. The efficient classic value iteration implementation with ADD would be found in Hoey et al. [Hoey99] and Hansen et al. [Hansen00]. The curse of history is solved by bounding the number of α-vectors for value function. Hence, the symbolic version of Perseus algorithm effectively overcomes some both sources of intractability, and it is what we use in this thesis.

Lately the research in the point-based direction of value-iteration algorithms succeeded with publication of HSVI - Symbolic Heuristic Search Value Iteration for Factored POMDPs [Sim08]. This algorithm is an extension of HSVI and Symbolic Perseus and demonstrates a better performance than its presenters.

Chapter 4
POMDPs for sensor selection: active sensing model

4.1 Active sensing motivation

"One important facet of the POMDP approach is that there is no distinction drawn between actions taken to change the state of the world and actions taken to gain information. This is important because, in general, every action has both types of effect. Stopping to ask questions may delay the robot's arrival at the goal or spend extra energy; moving forward may give the robot information that it is in a dead-end because of the resulting crash.

Thus, from the POMDP perspective, optimal performance involves something akin to a "value of information" calculation, only more complex; the agent chooses between actions based on the amount of information they provide, the amount of reward they produce, and how they change the state of the world." [Cassandra98]

We begin this chapter with the quote from the often cited Littman and Cassandra paper, where some classic approaches for solving POMDPs are discussed. The problem, outlined in the quote, describes the complicated choice of action that the agent has to perform each time, given that actions differ in their resulting effect on the world state, the belief of the state and the benefit gained with respect to the planning horizon. In this thesis we investigate this tradeoff in the search of the optimal strategy, in the situations when the "value of information" becomes a significant, maybe a critical, criterion of an action choice The approach proposed in this thesis targets a specific area of a context-sensitive sensor selection for sequential decision making under uncertainty, when the dynamic sensor selection is done with respect to the agent's goals and the trade-off of information quality and cost. The fundamental idea behind the active sensing approach is to separate the perceptual and transitional (state-changing) agent abilities, enabling dynamically selective sensing potential, that will optimize the expected utility in a planning horizon perspective and meet the system energy constraints at the same time.

The proposed active sensing approach is motivated from many existing problems in the sensor planning area. For example, if a robot is designed to act autonomously in a complicated environment, it has to have the access to all sorts of necessary contextual information. In most real-world scenarios, a single sensor is not enough to assist this requirement. Thus, modern robots that were targeted for multi-tasking are most likely equipped with different specific sensors. In this architecture of multi-streaming perceptual ability, the sensor planning problem requires an effective resource planning strategy for efficient goal achievement. The planning paradigm here could be viewed as a 2-level problem, where at the first level there will be a question of whether the current information in the system is enough for decision making or the agent has to reassure the state by using sensor resources. If there is a lack of information, in those states the second level would particularly address the problem of sensor choice with maximum information gain within a budget limit.

In this thesis we will be comparing two approaches for POMDP modelling. In the first, it is assumed that all sensor data is always available at every step, and the full cost is thereby incurred at every time step. That is, the sensor queries are not considered separately from any other state-changing actions. We will call this the passive sensing (PS) approach. For instance, in target tracking systems, the agent usually operates under the condition of simultaneously running sonars and scanners ready to deliver the information any time. In the second approach, each sensor can be queried independently of any state transitions or other sensor queries, thereby allowing the POMDP to select observations itself based on its needs. We will call this approach the active sensing (AS) approach. This is done by explicitly separating out the sensor query actions from the state transition actions, which allows avoiding paying for sensor querying in the situations when it is not required by the system.

The PS assumption has some important drawbacks on the two orthogonal aspects: conceptual and algorithmic. Those drawbacks make the PS approach less beneficial when applied in large-scale realistic domains. Conceptually, as a rule, in real domain applications there is always a cost associated with information acquisition, expressed either in the energy consumption, acquisition time, bandwidth channel load or signal processing computational effort. From the algorithmic point of view, the latest point-based algorithms are still struggling with computational time, which blows up with the observation space [Kaplow10]. Querying all sensors at each action immediately blows up the alpha-vector space, as we have to consider all possible observations. Even though the observation space can be compressed (without any loss) to be equal to the number of alpha vectors from the previous step [Poupart05], generating a policy for the real-domain large models is computationally challenging. The active sensing approach relaxes this issue, because it considers each observation only once (in the sensor query action), while the PS approach considers each observation in all actions.

One of motivating factors in the active sensing (AS) approach refers to the value of available information. Suppose that within the sensor network the robot has access to different types of sensors simultaneously, but not all of the available features are required or could be useful in the current decision. Some of the available sensor resources may not deliver any useful information, and so have no contributive effect at the moment or may even be redundant (not associated with any states or be currently off). In the situation when not all queried sensors are contributing to the decision making process, it is not useful for the system to query all of them, thereby paying penalties for processing time and/or processing power. For example, the GPS sensor in a mobile sensing context would be queried even if the main concern of a system will be to check the weather. Furthermore, as sensors and the sensed context information are highly correlated, there are situations when context information, when inferred from different sensors, may have overlap or even conflicts at the reasoning level. For example, we can have 2 different sensors of temperature, one of which is more accurate. But if the system is querying both at the same time, the information will overlap as they carry the same functionality - they both report the same temperature context. In such a situation the agent could save energy by choosing and querying only one of the two sensors which will be the optimal one for the current moment. Note that the optimal sensor in this case is not always the most accurate one. In this thesis we address the active sensing as a sensor selection problem.

The strength of the AS approach is that by separating out the sensing-only actions, we allow dynamic active sensing for uncertainty reduction, adapted to the information acquisition rates. On the system management level, the action plan developed under this approach allows for effective information resource planning under any possible constraints associated with the sensing or data delivery. In PS models, the agent performs more information gathering (from all available resources), which potentially could lead to bandwidth overload and computational burden on the processing side, when applied to the large real-world sensor networks. Treating sensing as a separate action, we open a new level of flexibility, in which we can consider the real information acquisition efforts expressed as sensor query costs. This makes the AS approach more applicable to the target domain of context-sensitive decision making under uncertainty.

Active sensing is especially beneficial in a sequential decision-making model. The active perceptual ability gives informational support for the decision analysis, which can be extremely helpful in choosing the best decision. When evaluating the next action against all the objectives, the agent can choose to gather information that will be not only useful in the current moment of decision making, but also will greatly improve the result in the long term perspective. Furthermore, in cases where the state-transition actions (Fig. 4.1) have much heavier direct impact on the utility function than sensing actions, it is better to perform a series of a state-verification sensing actions before any state-transition action in order to maximize utility. In that case we can say that looking only at the state-transition actions, active sensing approximates the policy to the QMDP policy, prescribing the state-transition actions to the almost certain belief states. Overall, the active sensing approach allows utilizing the additional state-acquisition actions in the policy separately from the state-transition actions without the loss in the quality of the decision making.

There could be cases when the active sensing approach would result in an action policy identical to the passive sensing approach. Those situations are possible when every decision is influential, and all of the information sources are valid (connected to belief space), useful (participate in the policy), not mutually exclusive (no duplicates in functionality) and are required to verify the belief state. Thus, the active sensing approach would prescribe sequentially querying all available sensors before performing each state-transition action. This scenario is similar to the passive sensing approach of querying all sensors with each action. In this case, there is no evident benefit in active sensing (note that at the same time there is no loss either). However, the advantage could come in two ways. First, in saving off-line time on policy generation, as it is demonstrated in our experimental part [Chapter 5]. Secondly, the active sensing approach allows certain flexibility, by separating information acquisition and state-transition on the logical modeling level.

4.2 Modeling active sensing

When we apply the POMDP model to the sensor selection problem, we need to augment the model to deal with the fact that observations are not always immediately present, but rather that the system must choose which sensors to derive readings from. The proposed AS approach does not change the functionality of the POMDP, i.e. it does not integrate any additional information gathering capacities, which the agent did not have in the PS model. Basically, all same sensors and system actions are still available at every step. The only difference is in the logical separation of sensing from acting and wrapping it in the form of stand-alone action.

Any PS model can be transformed to an equivalent AS model by performing three specific changes with observations, actions and the discount factor as follows:

1. Observations become unobserved unless the action explicitly queries them, so the information update is disabled for state-transition actions. If there were any costs associated with observation gathering, they also will be removed from the action's rewards accordingly.
2. For each available observation in the system an observation action is created with the corresponding observation function and a reward equal to the query cost. Under these new actions, the states remain the same so no state transition happens. Clearly, one must ensure that any sensor query actions already present in the PS model are not repeated in the AS model.
3. If the PS model were discounted, then the discount factor has to be taken care of properly in the AS modification. Conceptually, it is assumed that in the PS model all observations are queried each time the agent acts, while in the AS model each observation query is an independent action, separate from state-transition actions. Thus, one action of the PS model corresponds to N+1 actions of the AS model, where N is the number of available sensors in the system. Therefore, the correct way to discount the AS model is to add an action-dependent discount factor, when the sensor query actions do not get discounted in the AS model, assuming they do not take significant time. In Section 4.4.2, we describe our implementation of this modification, where we explicitly force the AS model to query all sensors in a sequence between state transitions, and control that one PS action has to be discounted by the same amount as N+1 AS. This is an equivalent method of making the two models consistent. In practice, the PS model will normally carry a discount for each sensor query that is dependent on the amount of time it takes to query the sensor.

POMDPs for PS and AS are shown in Figure 4.1 as dynamic decision networks, and described more precisely below. Each network in Fig. 4.1 consists of chance nodes for states S and observations O, decision nodes for actions A and utility nodes for rewards R and C. Arrows represent the probabilistic dependencies between variables. As in the PS POMDP, the world state S describes the (unobservable) state of the world.

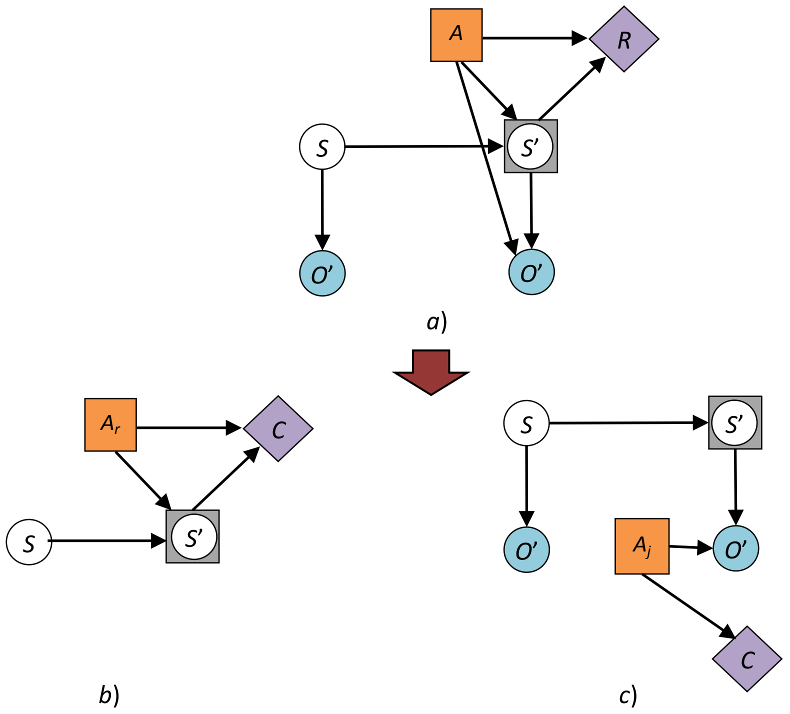


Figure 4.1. General POMDP functional flow for PS (a) and AS (b and c) POMDP formulations. The AS actions are of two types: state-transition (b), that change the state of the world, or information acquisition (c), that query sensors.

The top diagram (a) demonstrates the PS model, where the POMDP actions are conditioned on both the states and observations. In the AS model there are 2 sets of actions. One set (b) is only conditioning the states, and the other (c) is only conditioning the observations.

The state-related actions are the usual actions in the POMDP that change the state of the world according to some transition dynamics $Pr(s'|s,a)$. The sensor-related actions "fill in" the sensor reading O_i for the sensor i (one action corresponds to one sensor/observation). Thus, $A \in A_r \cup A_j$, where A_r is the set of actions in the PS POMDP model, and $A_j \equiv \{A_1, A_2, A_3 \dots A_{|O|}\}$ are the sensor-related actions, one for each sensor. Thus, we have that

$$Pr(S' |S,A) = \begin{cases} \delta(S' = S) & \text{if } A \epsilon A_j, \\ Pr(S'|S,A) & \text{if } A \epsilon A_r. \end{cases}$$

The state remains the same if the action is sensor-related (although in general it may change due to exogenous events, we ignore these events for simplicity here). The second case ($A \epsilon A_r$) shows that the state dynamics are as in the PS POMDP if the action is state-related.

For the observation function, we have that

$$Pr(O_i|S,A) = \begin{cases} Pr(O_i|S) \text{ if } A \epsilon A_j, \\ \delta(O_i = N/A) \text{ if } A \epsilon A_r. \end{cases}$$

Here, the first case shows, that the observation function for the i-th sensor is the same as in the PS POMDP when the action is the sensor-related action for sensor i. The second case shows that the observation from the sensor will not be available if the action is not sensor-related (including state-related and any actions related to querying other sensors). For convenience, we model this by augmenting each sensor with a value of “N/A”, meaning that the sensor is not measuring anything (or the measurements are being ignored). One could also simply have a constant observation function and then make the observation variable unobserved for any action that does not explicitly query it, but this complicates the implementation somewhat.

The reward function $R(S,A)$ is the usual reward function in the underlying POMDP, $R_r(S,A)$, for state-related actions; for each sensor-related action the cost $C(A_j)$ is unique corresponding to the sensor.

4.3 Sensor quality/accuracy trade-off

Each time when an agent is making a choice of whether to execute an action or pick a sensor to query, it is facing two conflicting goals. One the one hand, having perceptual abilities, the agent can potentially use them to reduce the uncertainty in the current belief state by asking for the most accurate information. On the other hand, the cost effort of getting this high quality information negatively affects the total utility, so the agent will want to lower the spending on the sensor acquisition. Thus, aiming at an optimal long-term goal, the agent needs a policy of actions for all possible belief states, in order to make a long-term decision about the cost/benefit trade-off. This kind of a tradeoff is not rare for many real world application domains. The value of the proposed AS approach is that it suggests an effective and elegant solution for this type of problem.

In order to test the AS approach, we would like to see how it compares with the PS approach as more and more sensors are required. Our hypothesis is that it will become increasingly beneficial to do active sensing. However, to make a legitimate comparison between the AS and PS approaches, we have to carefully control usefulness and information validity of the modeled sensors. We are going to do this by constructing a series of artificial models with increasing numbers of sensors, but we have to make sure we aren’t just adding useless sensors, thereby artificially penalising the PS approach. One way to do this is to consider a valid existing sensor in the model and to replicate it with different parameters, so we will have numerous different sensors properly associated with the state space. However, each of the modeled sensors has to be useful at some point of the decision making process. That means that for every sensor there will exist the corresponding area in the belief space, where the uncertainty level will make the agent query this sensor to verify the state before performing any state-transition action. Hence, the belief regions of such states will be mapped in the policy to the corresponding sensor query actions to fulfill or compensate the missing information. In that case we will be able to make a valid comparison of PS and AS approaches, where no artificial disadvantage will be placed on the PS approach through the inclusion of any useless sensors.

To create a variety of artificial sensors, we assume that the sensor quality/cost trade-off is static in both time and context, and depends only on the device or network characteristics. In the most straightforward way, we can fix all sensor parameters to be equal, except the quality. Then the cost of the sensor query will be strongly correlated to how trustworthy the sensor is. Logically, the noisier the sensor readings are, the less reliable is the information. Hence, the less utility the agent gains from querying this sensor. For example, in the classic tiger problem [Section 4.5.1] we may have a very accurate sensor that reports tiger's location with 95% accuracy, and a less reliable sensor with an accuracy of 80%. At the same time, the first higher-quality sensor will cost more due to the larger energy consumption associated with higher bit rate and computational effort spent on better audio-processing techniques. Taking the negative utilities of the sensors as their costs, we will assign the lower costs to those sensors whose readings have less accuracy in our experiments.

To maintain the usability constraint specified above, we have to resolve the trade-off between sensor accuracy and cost for the required number of sensors, such that each sensor would be useful in the near-optimal policy. The technique, specified in Appendix B, allows us to generate pairs of accuracy and the corresponding cost to make many copies of the same sensor, so that each of them will be useful. We use this technique only to generate the artificial sequence of models to study the general principles and verify the proposed idea by looking at the behavior of policies for different number of sensors. Following this technique, it is possible to construct POMDP models with an arbitrary number of useful sensors with the same functionality, by replicating a single sensor with different generated accuracies and costs. Thus, the experiments, performed on those models, would be valid for analysis of AS approach for the sensor selection problem.

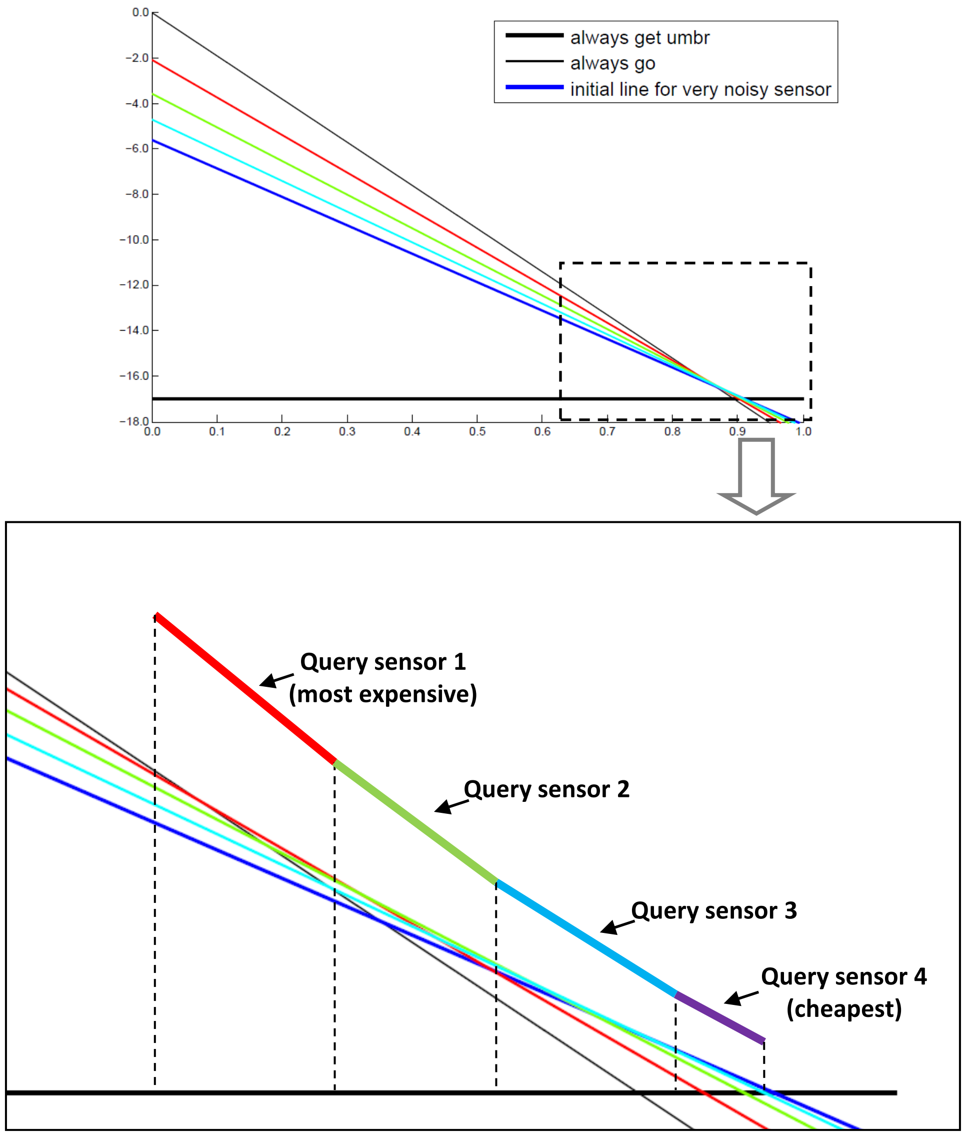


Figure 4.2. Alpha vectors of two-step horizon policy for the simplified coffee delivery problem [Appendix B] with 4 sensors. The 2D belief space of rain is depicted on the *X*-axis as a continuous interval [0,1], where 0 corresponds to the belief state of no rain, and 1 to the rain state accordingly. *Y*-axis gives a total reward. Sensor query alpha vectors are highlighted with colors, dominating over each other in a sequence, starting from the cheapest one in the bottom (the purple line of the corresponding alpha-vector) to the most expensive one on top (the red line of the corresponding alpha-vector).

Suppose, we have a simple problem, when the robot needs to decide whether he has to take an umbrella or just go, based on the sensor information about current weather situation. Assume that the policy prescribes to take an umbrella if the sensor reports rain and go otherwise. If the robot has a choice of weather sensor to query, his decision will depend on the current certainty in the weather. Figure 4.2 illustrates the case of 4 available sensors with different characteristics of cost and accuracy (the example details can be found at the end of Appendix B, Table B.3).

In particular, Figure 4.3 shows a family of 4 sensors, represented by the corresponding colored alpha-vectors of the sensor query actions (for the problem, specified in the Appendix B). The sensors are generated by applying the replication technique [Appendix B], so each of them takes a place in the near-optimal policy. From the geometrical point of view that means that each of the sensor query alpha-vectors dominates all other alpha-vectors in some region of belief space. The objective behind the action choice is to avoid getting wet. The conditional Table B.2 and the following generic alpha-vector line equation justify the geometric arrangement of the sensor lines. Notice that the "purple" alpha-vector for the cheapest sensor dominates over the belief region closest to the critical belief in rain, after which the robot will be certain enough to rather take the umbrella without querying any sensors. Being in that region, the robot requires the minimum of information to reassure that it is raining, and thus make the fastest decision if raining got confirmed by the sensor. So calling the noisiest and cheapest sensor will be enough at this point to support the state analysis and assist the decision making process. The critical belief point here is the point of the intersection of the purple line with the horizontal alpha-vector of "always get umbrella" action. Symmetrically, the most expensive "red" sensor will be queried when the agent has the least certainty in rain near the critical lowest belief point (the intersection of the red sensor alpha vector with the "always go" alpha-vector). Before this critical belief point the robot will be sure enough, that there is no rain, so he will just go without worrying about getting wet. But being in the belief state right before this point the robot would rather use the most expensive but accurate sensor, as risk of rain is far more important than loss on the sensor cost.

The demonstrated sensor replication approach is formulated in the framework of the simple POMDP problem. This method of sensor generation is not directly applicable for the longer sequential models. However it was used in this thesis with the reasonable results of generating useful sensors, as illustrated below.

Figure 4.3 illustrates the usefulness of sensor selection actions via graphical representation. Another way of sensors validity verification is simulation. Below is the Table 4.1 that demonstrates the simulation results performed on the policy for the coffee delivery AS model [Section 4.5.2]. The table shows some selective examples of belief states of rain when the policy prompted a sensor query action. In each row of the table there is a belief state of rain, the name of the sensor, corresponding to the prompted action, and the sensor characteristics of cost and accuracy. (For convenience of representation, some steps related to the state-transition action are ommited). During the simulation process we can observe that at different belief states the AS policy prescribes to query different sensors that are more beneficial at the moment with respect to the cost/accuracy characteristics. Note, that in the states with higher probability of rain, the policy allows to use cheaper less accurate sensors, while in the cases of lower probability of rain, the verification is done with more accurate sensors. The objective behind that choice is that the agent makes his best decision when he is very certain in the current belief state. Thus, fighting the uncertainty, the agent will choose the most effective information source for the current situation. In the case of low certainty it will be the most accurate sensor, which will shift the belief state to the corner of certainty for the shortest time. The prompting action strategy is smartly adapted to the current uncertainty. This gives evidence of the usefulness of duplicated sensors and validity of experimental setup.

Belief state: rain - yes | no		Best action by policy	Sensor cost/accuracy
0.5	0.5	From the initial state: Get umbrella -> move -> buy coffee -> deliver coffee ->	
0.5183	0.4817	Query sensor 4	0.5/0.7
		…	
0.3121	0.6879	Query sensor 2	0.9855/0.7698
		…	
0.2508	0.7592	Query sensor 1	2.4390/0.9558

Table 4.1. Simulation for the discounted coffee delivery AS model with 4 costly sensors. Two left columns represent the binary belief state of the rain variable. The middle column shows the best action selected by the SP policy. The right column gives the sensor specifications of cost and accuracy for the sensors recommended to query by the policy.

For the tiger problem all alpha-vectors corresponding to sensor query actions would be horizontal due to the symmetric setup of the problem. The penalties and rewards for opening doors are set up equally for all 4 doors. So the optimal policy will always have only one sensor to query if it needs the information. Thus, adding noisier sensor only brings the useless sensors to the system, so the AS approach gets a larger benefit and the experimental results confirms that by showing more clear picture of AS approach dominance [Figure 5.5]. In other problems, not all of generated sensors appeared on the optimal policy.

4.4 Approach behind the experiments

As mentioned above, in the experimental section our goal is to evaluate the proposed active sensing (AS) approach by comparing with the passive sensing (PS) model. We support the evaluation by series of experiments with the three POMDP problems (described fully in the Section 4.5). For each POMDP problem, we compare policies generated for the PS and AS versions with different parameter settings (discount factor, with or without sensor costs, policy generator type). The AS approach aims at solving the problem of dynamic sensor selection in the environments with many sensors. Therefore, to compare the behavior of the AS approach vs. the PS approach, we study the performance of the PS and AS policies as a function of the number of sensors.

The key difference between the PS and AS versions is that in the PS model the agent's perceptual ability is not separated from the state transitions. All sensors are forced to be queried at each system action. The observation space is factored so each observation variable is separate.

The new AS versions are modified from the PS version by adding the new set of sensing actions, specified such that they do not influence on the state transition as described in Section 4.2. In order to simplify the implementation, we also merge the observation variables in the AS versions: all observation variables are united to a single one and that their values become values of the new single observation variable.

$$\{O_1(o_{11}, o_{12}, \dots), O_2(o_{21}, o_{22}, \dots), \dots\} \rightarrow O_{united} = \{o_{11}, o_{12}, \dots, o_{21}, o_{22}, \dots\}.$$

This modification is done for the convenience purposes and is not affecting the policy quality, as we do not change the quantative characteristics of the model. Merging observations in one single variable leaves the cardinality of the observation values the same.

In order to minimise notation, we will use the same capital letters (e.g. S, O) to represent subsets of these state and observation variables, such that $S \equiv \{S_1, S_2, S_3 \dots\}$ and $O \equiv \{O_1, O_2, O_3, \dots\}$. Then, each state or observation variable can take on one value from a set of possibilities: $S_1 \in \{S_{11}, S_{12}, S_{13} \dots\}$ and $O_1 \in \{o_{11}, o_{12}, \dots,\}$.

Except as specified above, all other parameters of the AS and PS versions remain identical. Below we discuss in more detail the experimental setup, specifically, the sensor scaling approach, the discount factor and policy generators considered for comparison.

4.4.1 Global parameter settings

Sensors modeling. In our experiment set we study behaviors of the PS and AS policies with respect to increasing number of sensors. To simulate a growing sensor variety, we extend the sensor set in the PS model, and then create the AS modification by adding new sensing actions. Within the experiments, each sensor is determined by its name, accuracy and query cost. In terms of modeling, adding a new sensor to the model is done by specifying the new observation variable, its values, the corresponding observation function and the associated cost of sensor call. Conceptually, we imitate the sensor variety by replicating one sensor with different characteristic parameters, so all replicated sensors are valid, unique and differing by noise and cost parameters. The noise/cost trade-off is controlled to make more accurate sensors more costly. Note that, for our experiments we are generating versions of models with the increasing number of meaningful sensors, where all sensors actually deliver valuable information that contributes to the decision making and appear in the generated policy as corresponding querying actions [Section 4.3]. The technique that can be applied to construct an arbitrary number of copies of one sensor in the system and guarantee the usefulness of the generated sensors, can be found in Appendix B. In general, for each POMDP problem we experiment with the policies for models with up to 10 duplicated sensors in each model. Sensors are either costly or could be accessed for free. In the case of costly sensors, for each combination of parameter settings [Table 4.2], we generate 5 models with 2,4,6,8 and 10 duplicated sensors for the extended tiger and PS coffee problems, and with 2,3,4,5 and 6 duplicated sensors for the extended coffee problem. For the PS approach models, the sum of costs for all available sensors is added to each state-transition action, as all sensors are queried with each action. In the AS models, new query sensor actions are assigned with the associated sensor cost and do not influence on the state transition, at the same time all state-transition actions do not make any observations; thus, their reward functions are not increased by sensing costs.

Discount factors. In our policy evaluation we perform 3 sets of experiments: where all actions in both PS and AS versions are discounted by factors of 1, 0.97 and a case when AS model is discounted by 0.97 and all basic system actions in the PS model are discounted by 0.97^{N+1} where N is the number of sensors that are queried at each action.

The discount factor of 1 corresponds to the case when the future reward is undiscounted. This is a special case for Symbolic Perseus due to the specifics in algorithmic design. The Perseus randomized value iteration technique proceeds by computing the sequence of finite horizon optimal value functions by doing partial point-based backups at the reachable belief states. In the partial point-based backup, for each witness point from the set of witness belief states, the support vector is computed only if this belief point was not improved by any new vector compared to the previous backup. The termination of the process depends on the magnitude of improvement in the next generated near-optimal policy. Termination is reached when the maximum difference between two consecutive value functions is less than some convergence value. The discounted future makes the process converge as each next value function will differ by a smaller more and more discounted value with time. In the case of undiscounted future, the process of improving the policy will not converge with time, as each new value function will be significantly better that the previously computed and not a lot of belief points will be backed-up. As a result, at any horizon the near-optimal policy is likely quite poor. That's why the Symbolic Perseus solver does not work for the undiscounted models (first case) in general, and especially for the AS approach [Section 5.5.3].

For the discounted cases, if all actions in the PS and AS models are equally discounted by 0.97 (second case), the PS model is unfairly favored. When the AS model is discounted by 0.97 for any sensing or system action, the PS model agent is discounted equally for querying all sensors and performing the system action at once. Obviously, the PS model is discounted unfairly less, because to perform the same activity, the AS model agent has to separately query all sensors and perform the system action. Thus, the correct discount factor setup to compare AS and PS versions is to have an action-dependent discount factor, when the sensor query actions do not get discounted in the AS model. We implement this idea by equally discounting each action in the PS POMDP model by the same discount factor of 0.97^{N+1}, if N is the number of sensors to be called per each action.

Sensor costs. For each of 3 variants of discount factor we consider two cost settings: sensor cost free models and models with the sensor cost included.

Thus, per each discount factor setting, we have 4 types of cost/model settings (see Table 4.2):

a) PS version - all sensors are free;

b) PS version - each sensor has its own cost;

c) AS version - all sensors are free;

d) AS version - each sensor has its own cost;

Sensor cost	Model type	Discount =1	Discount = 0.97	Discount =
Free sensors (cost = 0)	PS model	PS-disc:1-free	PS-disc:0.97-free	PS-disc:0.97N-free
	AS model	AS-disc:1-free	AS-disc:0.97-free	AS-disc:0.97N-free (same as AS-disc:0.97-free)
Costly sensors	PS model	PS-disc:1-costly	PS-disc:0.97-costly	PS-disc:0.97N-costly
	AS model	AS-disc:1-costly	AS-disc:0.97-costly	AS-disc:0.97N-costly (same as AS-disc:0.97-costly)

Table 4.2. Legends for experimental settings of cost and discount factors.

For each combination of cost, discount factor and model type from the Table 4.2, we generate 5 files with 2,4,6,8 and 10 duplicated sensors for the extended tiger and PS coffee problems, and with 2,3,4,5 and 6 duplicated sensors for the extended coffee problem.

Types of policies: we are mainly interested in comparing PS and AS proposed approaches for policies generated by the Symbolic Perseus solver. In addition to that, the Symbolic Perseus package also allows to evaluate a QMDP policy, using it as simulator [Littman95]. In this case, we ignore the partial observability of the model and solve the underlying MDP using value iteration. We also perform evaluation of each model against a random policy to use in analysis as the worst case bound.

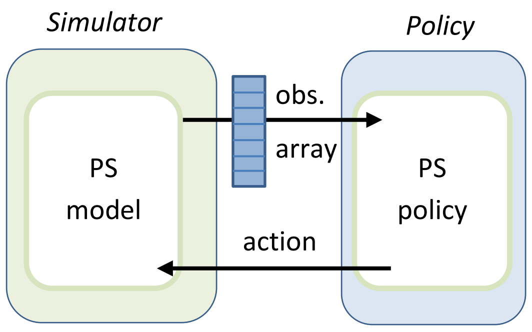


Figure 4.3. PS-PS: Evaluating PS model policy vs. PS model.

4.4.2 Evaluation scenarios

To compare the policies, generated for all settings, specified above, we are running 3 evaluation scenarios, where we are testing policies against the models that are running as simulators:

1) **PS-PS** (evaluating PS model policy vs. PS model): we run the PS POMDP model on the simulator, and test different policies (SP, QMDP, Random) against that simulator (Fig. 4.3). Each time step after each action, the simulator sends all observations as an array for the belief update. In case of costly sensors, the model is penalized by the cost of all sensors together.

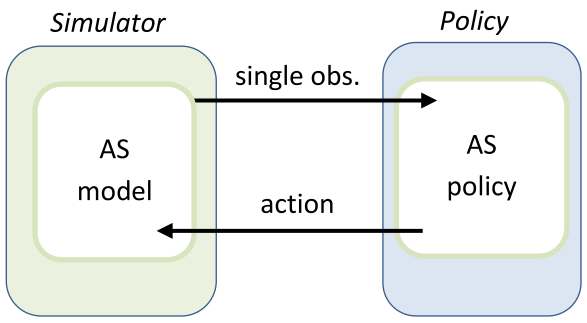


Figure 4.4. AS-AS: Evaluating AS policy vs. AS model.

2) **AS-AS** (evaluating AS policy vs. AS model): we run the AS POMDP model on the simulator, and evaluate different policies (SP, QMDP, Random) AS policies against that simulator. This time the simulator sends one observation per single sensor query action [Fig. 4.4], and N/A for the state-transition actions.

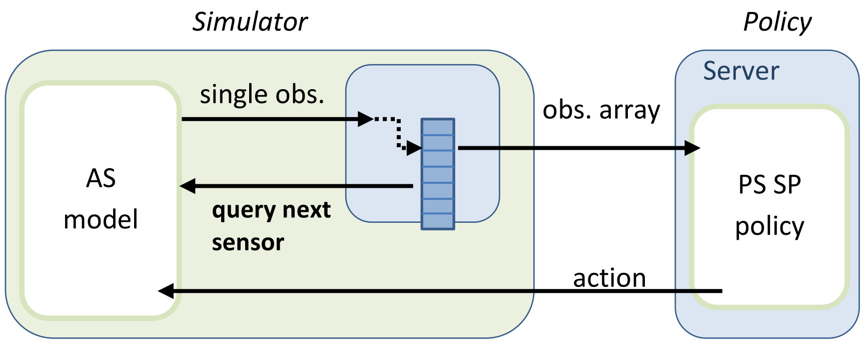


Figure 4.5. Server: Evaluating PS model policy vs. active sensing model.

3) **PS-AS**: (evaluating PS SP policy vs. AS model): we are running the AS POMDP as the simulator, against the PS POMDP SP policy only [Fig. 4.5]. In this case we apply the PS model policy each time after the new model is forced to perform a sequence of all sensor query actions after taking a basic state-transition action. The simulator accumulates the returned sensor readings to collect the full complete observation set from all possible query actions. This set is then used to obtain the next state-transition action from the PS policy. Essentially, we are mapping between the new action+observation space in the AS model and the PS POMDP.

Evaluation scenario 3 could also be done in another way. We can run the evaluation scenario 2 and force the AS POMDP to have a policy when it will has to query every single sensor in a sequence one at a time. As we do not want to manually create such a policy (too complicated), we have to set up the model in such a way that the solver will output the policy we are interested in. This would require changing the model variables, transitions and costs specifically to make the required policy. In particular, this leads to the introduction of another set of variables – one per each sensor – that will be the flags of whether the sensor has been queried, as well as assigning the large penalty for state-transition actions until all the sensors are queried. However, the results of experiments would not be obviously clean and transparent, so the third scenario is the "cleaner", but equivalent, way of evaluation.

For the last experiment setup, we have to configure the environment to test the PS model policy on the new AS model. In this case policy evaluation is not as straightforward as it was in previous two cases, due to the difference in the action set and actions specifics between models. The PS model will prompt the next action only after all observations are available. Thus, we have to force the AS model to collect all observations before the PS model policy makes a decision on the future action.

We are running PS model with its policy and the AS simulator simultaneously, keeping them in separate Java processes. The two running models are exchanging information via a broker program that is running on a PHP socket server. The broker is responsible for scheduling the dialog of requesting and sending information between two Java processes with running models. Both models maintain their own belief state. The AS model simulates sequential querying of all sensors, and, upon the forming a full array of observations, the simulator sends them to the PS model via the broker. The PS model updates its belief state and answers to the AS model with the next action according to the PS SP policy. The simulator processes the incoming action and then the cycle repeats [Fig. 4.5]. The reward is computed and accumulated on the side of the simulator.

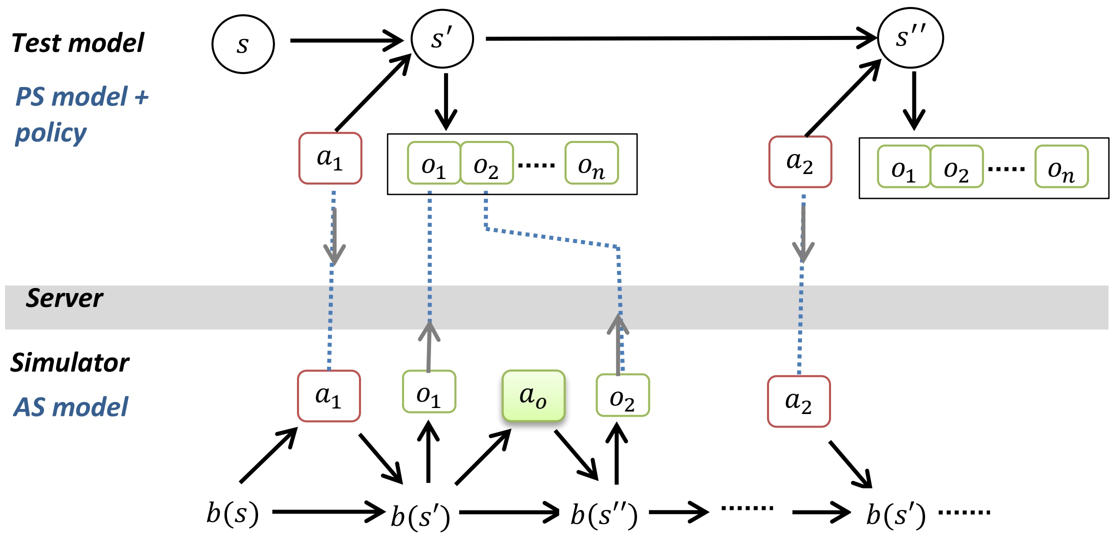


Figure 4.6. The simulating environment for server scenario. Communication schema between the PS model SP policy (is running as separate Java process) and AS model (is running as simulator) during the evaluation process.

As a summary, the following settings were used for experiments [Table 4.3]:

Setting parameter	Setting values
POMDP models	Extended tiger; coffee delivery; extended coffee delivery
Approaches	PS; AS
Discount factor	1; 0.97; 0.97^{N+1} for PS approach
Sensor costs	Free sensors; costly sensors
Policy generators	SP; QMDP; Random
Number of sensors	{2,3,4,5,6} (extended coffee delivery problem); {2,4,6,8,10} (coffee delivery and extended tiger problems)
Evaluation scenarios	PS-PS; AS-AS; PS-AS

Table 4.3. Experimental parameters settings for POMDP models.

4.5 Problem description

Below are descriptions of three problems used for experiments in this thesis.

4.5.1 The extended partially observable tiger problem

The tiger problem is a classic toy problem in the POMDP research literature [Cassandra98, Hoey07]. In this problem an agent has to open one of the two closed doors when there is a tiger behind one and the reward behind the other. The agent is interested to estimate tiger's location in order to avoid opening the door with tiger. There is a penalty assigned to opening the tiger's door and a reward for the other. To estimate the tiger's location the agent can acquire this information by listening. However, listening has its own cost as well as the challenge that incoming information is noisy.

The problem that we use in this thesis is an extension of classical tiger model. Here we consider that they are three locations for an agent to be in: a hall and two rooms. The agent starts in the hall and can move into one of the two rooms. In each room there are two doors: left and right. There are five actions that the agent can take: open left door, open right door, listen, move to room one or move to room two. In each room the agent can make an observation whether the sound is coming from the left or right door. In the hall, the listen action gives information about which room has the door with the tiger. The cost of listen action is 1. The agent could not hear any sound if tiger is in the other room and observation is not available for all actions except the listen one. The reward function constructed the way that the optimal strategy is {listen -> move -> listen -> open the door} action strategy. The reward for opening the door without tiger is 100, the punishment for opening the door with tiger is -50. The listen action has no effect on the agent's or tiger's location. Also agent doesn't make any observations when it moves or opens the doors. Moving actions shift the agent's location to the target room with probability of 1. The opening of any door resets the situation to the initial position in the hall when tiger is equally probable behind all doors. For the full details see the corresponding SPUDD file in Appendix A.

4.5.2 The partially observable coffee delivery problem

In this problem (first appeared in by [Boutilier96], also mentioned in [Poupart05]) an agent is a robot that uses battery, computational and physical resources to obtain and deliver coffee to a single user. The user is located in the office and the robot can move from an office location to a local coffee shop across the street to get the coffee and deliver to a user. There is a possibility of rain between office and coffee shop, so the robot can get wet if it doesn't carry the umbrella. Getting the umbrella is an action the robot can take in the office only. Getting wet is highly undesirable for the robot.

The state space consists of 6 Boolean variables referring to whether: user has coffee (*huc*); robot is wet (*w*); robot has coffee (*hrc*); it is raining outside (*r*); the robot has an umbrella (*u*); robot is in the office of at the coffee shop.

The robot cannot directly observe the rain, but only detect if the pavement is wet with the accuracy of 0.9. Based on the observations about the rain the robot can get an umbrella. The robot's goal is to deliver a coffee when the user doesn't have any and to avoid getting wet.

The initial state of the world is when the user has no coffee, robot is in the office, not wet, it doesn't have an umbrella, robot doesn't have the coffee to deliver, rain is equally possibly to be outside.

There is a variety of actions that robot can do: it moves (*move*), delivers coffee to user (*delc*), gets the umbrella (*getu*), and buys coffee (*buyc*). However robot can only take the umbrella if it is in the office. Also there is an action of not doing anything (*do nothing*), when the exogenous events may affect some states. In this action all state variables remain the same except *huc* and *r*. The world's stochasticity for those states is specified in the corresponding conditional probability tables (CPT). There we have dynamically changing rain probability between no rain and the heavy rain, and user slowly drinks coffee with time.

The robot is rewarded by 25 when the user has coffee, and it is penalized by 20 if it gets wet. There is also a cost of 15 for getting an umbrella, a little costs of moving (depending on the robot's location it is either 0.2 for the office and 0.1 for the shop) and buying a coffee (0.3 if the robot is in the shop). For the full details see the corresponding SPUDD file in Appendix A.

4.5.3 The extended partially observable coffee delivery problem

The extended version of partially observable coffee problem is the result of merging the weather problem published in [GOSSIP] with the classical coffee delivery problem specified above [Boutilier96]. It is enriched with 5 new states and 6 new observation variables.

In this extended version the robot can check the battery level, observe if the user has coffee or if robot has coffee, and gather information about environmental properties. The rain state from the classic version of coffee model is extended to the weather state *weather {moist; rain; comfort}*. Also *batt {high; low; dead}* reports the state of the battery. Three new environmental factors are introduced as underlying states: atmospheric pressure *PR*, relative humidity *RH*, and current temperature *CT*, each taking on values *{high; low; normal}*. Those states fuse together to provide prediction for rain and keep the robot more updated about the weather situation.

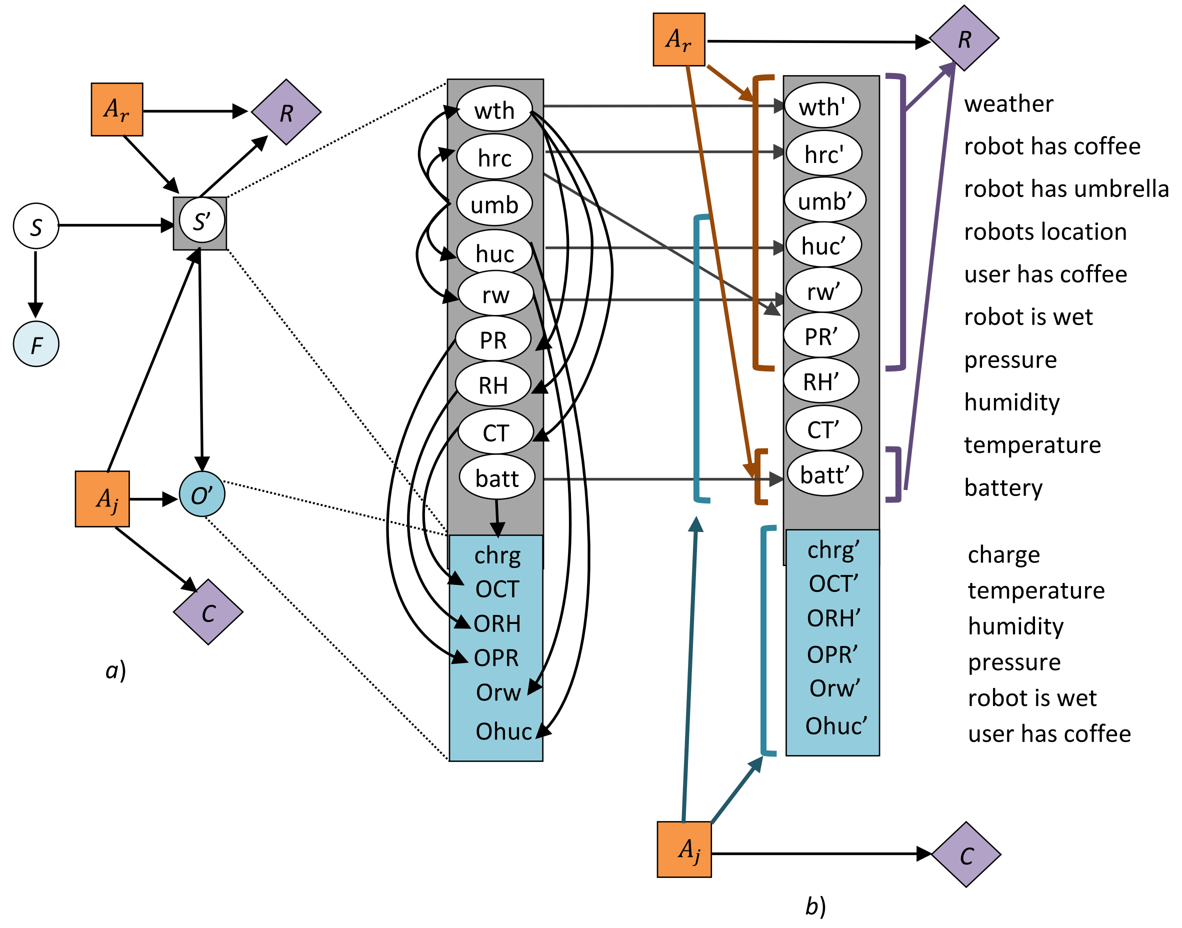


Figure 4.2. Two-time slices of a) general POMDP; b) a factored POMDP for the extended coffee delivery problem. The system actions are either to query any sensor one at a time, or to prompt an action the robot should take.

New available observations are whether or not the user has coffee, the robot is wet, the robot battery is enough, the battery charge is zero or one or two, and environmental properties of pressure, humidity and temperature. The sensor readings about these factors update corresponding states in the state set.

The system can also wait, which will recharge the battery. The reward function preserves the coffee delivery problem objectives and also adds a penalty for the battery running dead. The transition dynamics are such that the weather is expected to change to a different state with probability 0.01 in every time step, but otherwise stays the same. The battery discharges slowly for prompting actions, and quickly for sensing actions. The weather causes a variety of probabilistic changes in the environmental factors, and these are reported by the sensors with some noise.

Chapter 5
Experimental environment and results evaluation

To verify that the proposed ideas of AS approach presented in this thesis carry valid applicability potential to real world domains, we performed a series of experiments as a proof of concept. In the previous chapter we spoke about parameter settings and evaluation scenarios for the experiments we conducted [Table 4.3]. In this chapter we give the detailed overview of the experimental computational environment, selected performance measures and present the results and the analysis.

5.1 Computational environment

Summary of experimental setup. We implement a set of simulations to evaluate the policies generated for the three POMDP problems [Section 4.5] for different parameter setups and policy types [Table 4.3]. Our goal is to validate the expectations that the AS approach has a beneficial position with respect to the PS one. We model the selected experimental problems [Section 4.5] in a POMDP framework. We prepare 2 versions of POMDP problems: the PS and corresponding AS, for different values of discount factor and sensor costs. For each POMDP version we gradually scale up the number of available sensors, using the scaling technique [Section 4.3]. Thus, to simulate the sensor variety for each combination of other parameters we model 5 files with increasing sensor variety, adding noisier, cheaper sensors. We base our analysis on the 3 types of policies, generated for prepared POMDPs – SP, QMDP and Random. We are mainly interested in the SP policies, while QMDP and Random policies still contribute to the analysis, as they demonstrate approximate and the worst case policy performance respectively. We evaluate the proposed ideas by looking at the correlation between the performance of POMDP policies and the number of available sensors in the model. We give highlights of the results in the following sections, and refer the reader to Appendix C for the full results.

Quantitative characteristics of the source POMDP models. The convenience of point-based POMDP algorithms is that the running time could be managed by tuning numerous parameters, for example, the size of the belief set, and exploring different heuristics for faster convergence, for example, QMDP sampling of belief space. Decreasing the number of rounds, iterations, belief set size or lowering the bound for the alpha-vector set size, we can speed up the running time. However, there is always a risk of loss in the quality of the generated policy. In Table 5.1 there is a general statistics of state, action and observation spaces over the PS POMDP problems considered for experiments in this thesis. The last column indicates the number of belief states used in policy generation.

Model	States	Actions	Observations	Belief States
Tiger extended	2	5	2,4,6,8,10	100
Coffee delivery	6	5	2,4,6,8,10	2000
Coffee extended	10	5	7+2,3,4,5,6	1000

Table 5.1. Chart of experimental domains. Quantitave characteristics of PS POMDP models.

Software characteristics. We compute the required policies for the experimental POMDPs using the Symbolic Perseus package [Poupart05] re-implemented by Hoey in Java [SPJ link]. It implements factored structured point-based value iteration solution technique on the base of the Perseus algorithm [Spaan05], combined with Algebraic Decision Diagrams [Bahar93] as a data structure for efficient computations. Policies computed with the Symbolic Perseus solver are generated in 2 rounds, 100 steps simulations, with 30 backup iterations of Perseus per round, iterating over the maximum of 100 alpha vectors. Sampling of the belief space is done by using a QMDP heuristic for 1000-3000 belief points [Table 5.1].

The average discounted return is computed by simulating the policy on the server platform with the following characteristics: 4x AMD Opteron^{TM} 6282 SE CPU with 2.6 GHz clock frequency, 100 GB RAM of physical memory maximum, OS Ubuntu 12.04, Open JDK IcedTea 1.11.3 implementing JRE 1.6.0.

5.2 Performance measures

Since we are not going to compare different POMDP solving algorithms, the evaluation process will not meet many usual difficulties such as comparing multiple different innovations at once or identifying the most influential factor on the performance [Kaplow10, Shani07]. Hence, we can use the traditional evaluation measures, popular in the target research area, without loss of analysis quality. In this thesis we are aiming to evaluate a new AS approach against the PS one in the POMDP modeling, using the Symbolic Perseus POMDP solver package as policy generator and simulator. Our main focus of experiments is comparing policies generated by the Symbolic Perseus solver. Inter-policy comparison is done with QMDP and random policies, using QMDP and random policy generators available within the Symbolic Perseus package.

Average reward. The traditional measure for performance of the POMDP policy evaluation is the comparison of value functions. Average discounted reward measure is a classical evaluation approach for value functions. In our experiments we accumulate the average discounted reward over the rounds of simulations. The discounted reward is gathered at each time step, based upon the ground-truth state and the POMDP reward function. In those simulations the agent-environment interaction is processed over a number of steps forming one trial. Numerous trials then are averaged to obtain the averaged discounted reward.

CPU%. The percentage of CPU used is also a good indicator of the computational load of the algorithm over different models. Since all experiments are performed on the same computational environments (the same platform, hardware and software) the comparison results are not subject to the used implementation or work station characteristics.

We have measured the CPU% during the policy generations of SP algorithm taking samples every 20 CPU clock times [Table C.3]. It was noticed that CPU% and physical memory usage are fluctuating a lot at the early stages of the policy generation, but are stabilizing over time. The typical pattern of CPU% usage for one policy computation is shown at the Fig. 5.1).

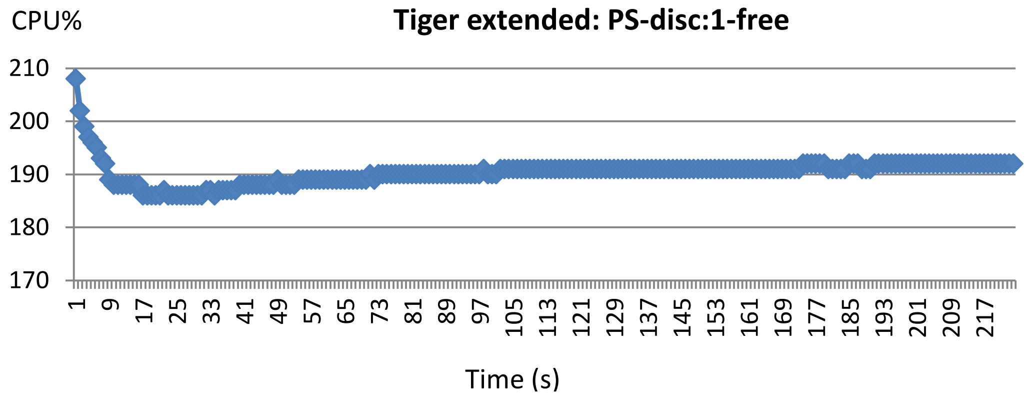


Figure 5.1. Fluctuations CPU% usage during the SP policy generation for the undiscounted tiger extended model with 2 duplicated cost-free sensors (PS approach).

For the CPU% evaluation we have used the highest registered CPU loads during the policy generation as they reflect the computation requests demanded by the SP algorithm. The average of CPU% numbers will not be representative enough because of uneven processor load on different policy generation steps as well as because of the significantly different policy generation time for new and PS models. The maximum CPU load per policy generation epoch is selected as the unique characteristic to compare PS and AS approaches with respect to the computational efforts spent on the policy generation.

Policy generation time. Generation time is also very informative in measuring and comparing performance of the proposed AS approach. Although the clock time is a subjective measure due to the possible inter-process communication influence when the performance could be slowed down by the side process, the CPU time would be an objective marker, given that the comparison is done within the same hardware and platform computational environment. In our experiments we found a significant difference in the time demanded for the policy generation of the PS POMDP models and for AS models [Table C.1][Section 5.3.2].

We have also measured the dynamics in the **resident set size** for the policies computation [Table C.2], but significant fluctuations in the memory usage during one policy generation do not form a reliable base for comparison [ex. Figure 5.2].

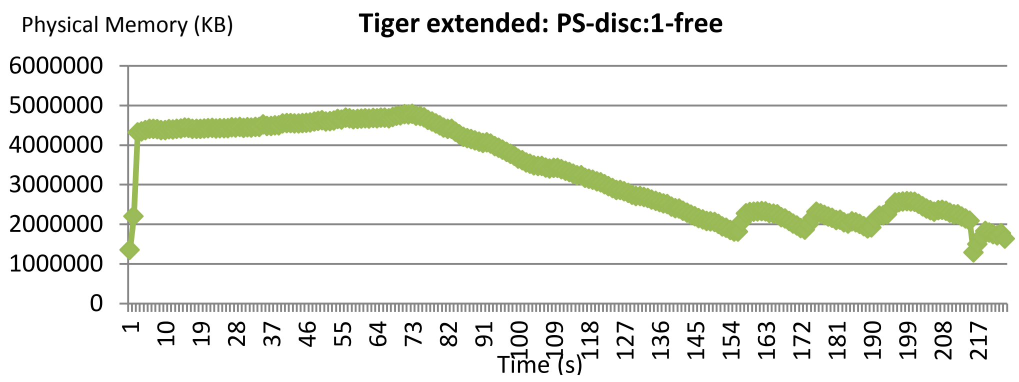


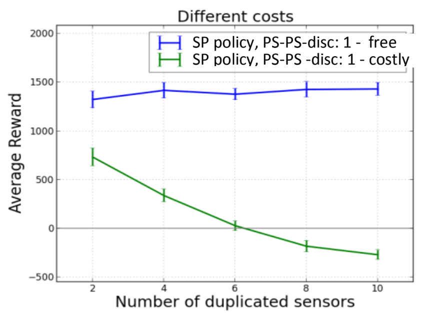
Figure 5.2. Fluctuations in physical memory usage (residential set size, KB) during the SP policy generation for the undiscounted extended tiger model with 2 duplicated free sensors (PS approach).

Together those measures give a perspective picture, reporting not only the quality of the generated policies, but also indication the computational effort connected to the presented approaches.

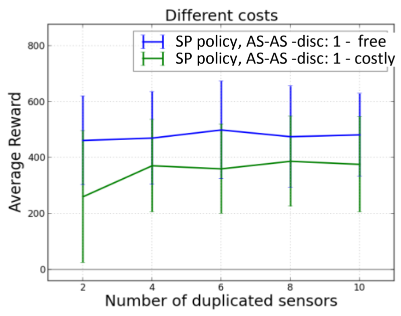
5.3 Results and analysis

5.3.1 Average reward

The average reward, as a classic measure for performance evaluation of policies, is the most informative source for validation of proposed ideas within the POMDP framework. The generated graphs from experimental data compare the behavior of different policies for various combinations of



a)



b)

Figure 5.3. Comparison of undiscounted SP policies for the tiger problem for free and costly sensors. Performance is measure in average discounted rewards for increasing amount of sensors for PS-PS (a) and AS –AS (b) evaluation scenarios.

parameters, where the *X*-axis is the number of duplicated sensors per model, the *Y*-axis displays the associated expected average reward. The detailed overview of the parameter choice is done in the Chapter 4.4.1. For full results see Appendix C, Tables C.4 –C.6.

Cost influence. Experiments indicate that the models with the free sensors reasonably achieve higher reward. The lines that correspond to the policies for models with free sensors have bigger reward and are located above the lines that correspond to the policies of costly sensors models [Fig. 5.3]. This behavior is valid for any combination of discount factor, model type or policy type.

Figure 5.3 gives a comparative picture of the cost influence on the behavior of undiscounted policies for the PS-PS (left chart) and AS-AS (right chart) **evaluation scenarios** applied to the tiger POMDP problem. For the PS approach with the free sensors (blue line) the rewards remain approximately constant, when the costly sensors (green line) make reward decrease with the sensor set growth. In the AS approach there is no increased loss in reward with more sensors involved, even if they are costly, which demonstrates the scalability of active sensing approach.

In Figure 5.4, the discounted coffee delivery SP policies are compared for free and costly sensors. The left chart corresponds to the PS approach and the right chart compares the AS policies. Note that for the AS approach the average reward range is higher than for the PS version of model. In the discounted models for the PS approach we see the exponentially decreasing trend on both sensor free and costly policies, however the discounted costly sensors policies are decreasing much faster. In particular, in the left chart the costly sensor green line decreases exponentially rapidly with the growth in the number of sensors, as the agent pays more for querying all sensors with each action. The “free” sensors line in the left PS chart also decreases because we add to the model new sensors and the system got penalized being forced to query more sensors, which slows the process down, so we see the decrease as a consequence of the accumulating discount factor.

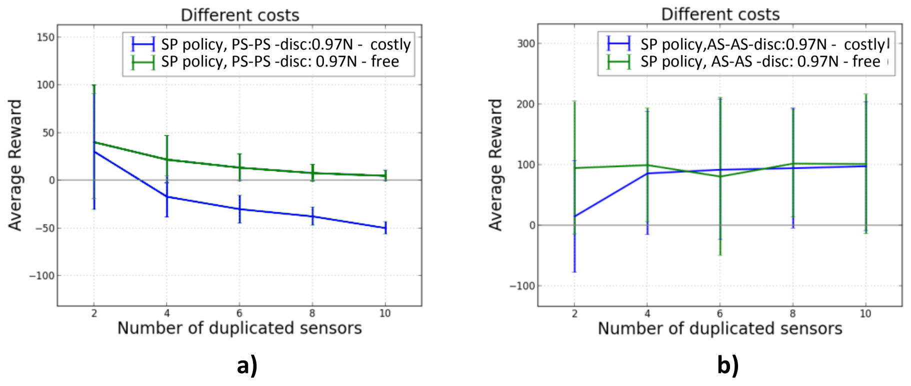
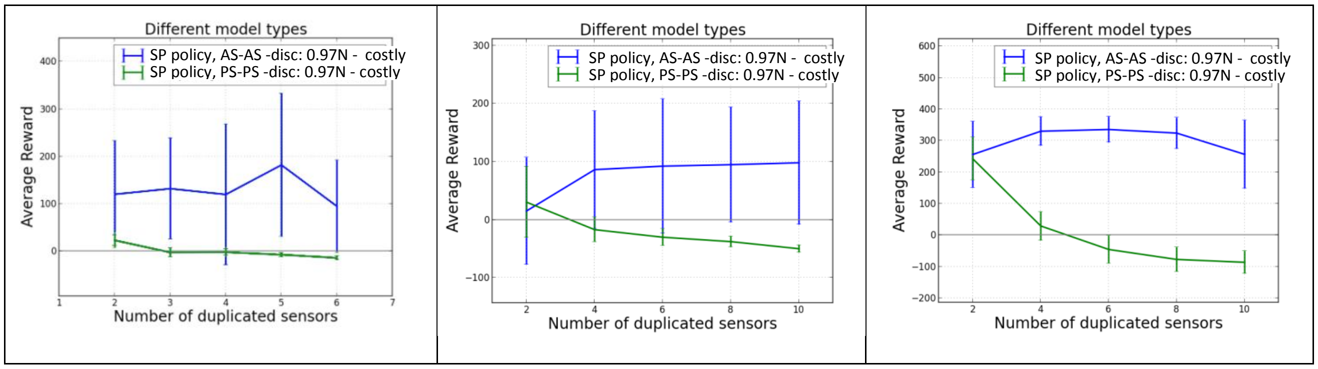


Figure 5.4. Comparison of discounted SP policies for the coffee delivery problem for free and costly sensors. Performance is measure in discounted rewards for increasing amount of sensors for PS-PS (a) and AS –AS (b) evaluation scenarios.

The right chart for the AS policies again demonstrates the scalability of proposed approach, as both policies behaviors are not negatively affected by the growth in the cardinality of the sensor set in the model. When the green “free sensors” line remains constant, the costly sensors blue line approaches the free sensors line fast. The agent can choose between information acquisition and system actions, so with each sensing action the cost is played only for the single sensor call. Moreover, the extension of the sensor set was done by adding the cheaper sensors; thus, we can see that the blue line of the average reward for the costly sensors is slightly increasing with time, as the agent can pick cheaper sensors and still fulfill the information needs.

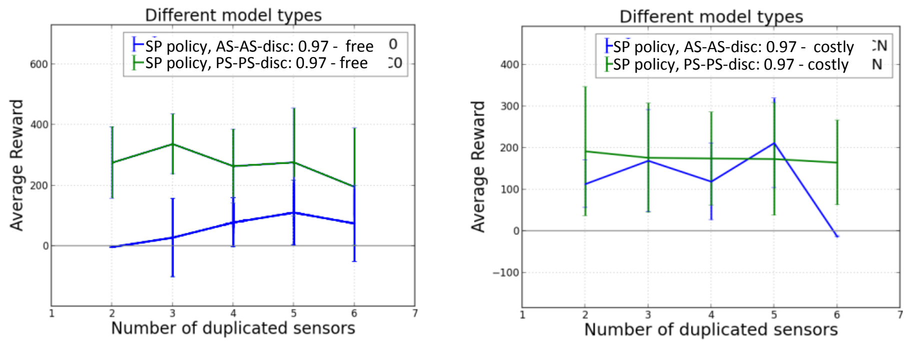
Comparing models. The focus model-comparing experimental results are shown in the charts of Figure 5.5, where we compare AS approach with the PS one for the SP policies for all 3 problems from Section 4.5. Parameter settings used for these results are the closest to reflect the reality and the most fair for a valid comparison. More precisely, here the discount factor is set up properly to discount the PS model for all sensor queries [Chapter 4.4.1] and sensors are not free. In all three charts the lines corresponding to the AS average reward, dominate the PS policies. Thus, we can conclude that for the proper discount factor setting in case of costly sensors the AS approach overperforms the PS one in terms of the expected average discounted reward of SP policy.



a) b) c)

Figure 5.5. SP policies are compared for PS-PS and AS-AS evaluation scenarios for 3 discounted problems with costly sensors: extended coffee delivery (a), classic coffee delivery (b), extended tiger (c) POMDP problems.

If both AS and PS models are discounted by 0.97, then the PS approach gets the advantage of asking for all sensors for single discount for this action, when the AS approach will be discounted for each sensor query separately, thus, get less reward [Chapter 4.4.1]. This is well illustrated for example by the Figure 5.6. Here the behavior of SP policies for coffee extended problem demonstrates the dominant position of the PS approach for costly (right chart) and free (left chart) sensors due to the unfair comparison (for full results see Appendix C Table C.6).

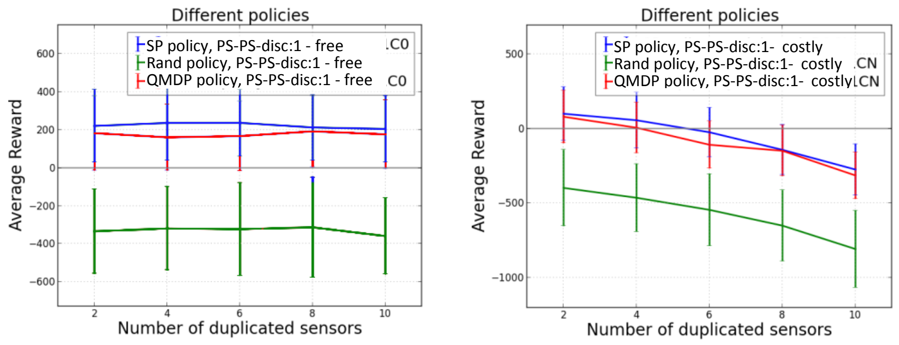


a) **b)**

Figure 5.6. Equally discounted SP policies are compared for PS-PS and AS-AS evaluation scenarios for the extended coffee delivery problem with free (a) and costly (b) sensors.

Comparing policies. For the fixed model type, discount factor and cost settings we compare SP, QMDP and Random policies as well as SP PS policy applied on the AS model. As expected, across the experiments, the Random policies deliver the worst results.

For the undiscounted models when all sensors are free, the SP policy of PS approach does not demonstrate a significant advantage, which is shown for example, in the Figure 5.7 (a). As it was explained in the Chapter 4, the Symbolic Perseus does not produce an effective policy for discount factor 1 due to the algorithmic design. The same is observed for the costly sensors [Fig. 5.7 (b)]. Here the cost assign to sensors make the rewards of PS model go rapidly down for all policies.



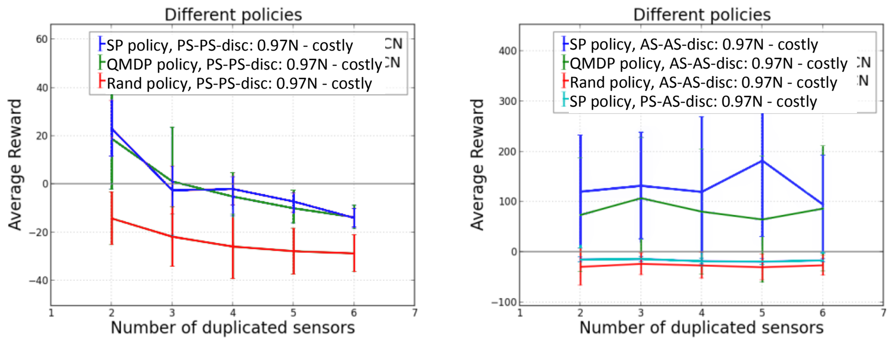
a) **b)**

Figure 5.7. SP, QMDP and Random policies for the PS-PS evaluation scenario, compared for the undiscounted coffee delivery problem with free sensors (a) and costly sensors (b).

In the cases where we evaluate PS models, we can see that SP policies could be effectively approximated by the QMDP policies for any discounts and costs [ex., Fig. 5.7, Fig. 5.8 (a)], which does not hold for the AS models [ex., Fig. 5.8 (b)].

In Figure 5.8, the fairly discounted policies are compared for costly sensors for the PS (left chart) and AS (right chart) approaches. We notice the decrease in the reward with the growth of available sensors for the discounted PS models, while the AS policies demonstrate approximately constant behavior. This pattern of ability to scale without loss in the expected reward of the proposed approach remains across all the conducted experiments (for full results see Appendix C Table C.5).

In case of discounted models, the dominance of AS approach is very observable for the discount set up of 0.97^{N+1}. The same patterns could be observed in the Table C.5 where we combine different policies and approaches to compare for the fixed discount factor and cost settings. For the proper comparison setup when PS model is discounted for every observation that each action contains, we see the dominating SP policy of the AS approach.



a) **b)**

Figure 5.8. SP, QMDP and Random policies for PS-PS (a) and AS-AS, PS-AS (b) evaluation scenarios, compared for the discounted extended coffee delivery problem with costly sensors.

In most of the cases QMDP policy appears slightly better than Random policy. The poor quality of QMDP policy approximations is explained by absence of an information acquisition strategy. Since the QMDP approach is solving for underlying MDP, the information acquisition actions are not considered for the policy and the perceptual-oriented side of any approach is not explored. Those heuristics are also proved to be ineffective for belief set initializing in the target domain of AS. The experimental results of Kaplow on the Dialogue system are indicating poor performance of using MDP Heuristic Collection method for the belief points collection [Kaplow10]. The reason is that under the MDP Heuristic belief regions corresponding to observation actions are unreachable for search due to the exclusive state-transition action orientation. However, the quality of the developed policy in the AS domain depends on how successful we can explore the advantage of ability to gather information before performing usual system actions. That means how well we sample the critical belief points where the decision of effective state acquisition would give the best performance. That’s why the choice of belief sampling heuristic holds a significant importance in the domain of AS.

5.3.2 Policy generation time

In the Table C.1 [Appendix C] there are 6 graphs of dependency of policy calculation time of SP on the number of sensors for different discount factors and sensor cost settings for the coffee extended problem. On each graph the 2 lines correspond to the PS and AS models. It is noticeable, that for any combination of cost and discount factors the dependency of time required for policy calculation for the PS POMDP model is approximately exponential in the number of duplicated sensors, while the AS time shows nearly-constant behavior. The typical example of this kind of behavior is presented on the Figure 5.9, where we measure the time spent by the Symbolic Perseus solver on policy generation for the discounted extended coffee delivery problem with costly sensors, with growing number of sensors. The blue line corresponds to the PS version of the coffee extended problem discounted by the factor 0.97^{N+1} (where N is the number of sensors), and the red line corresponds to the AS version discounted by 0.97.

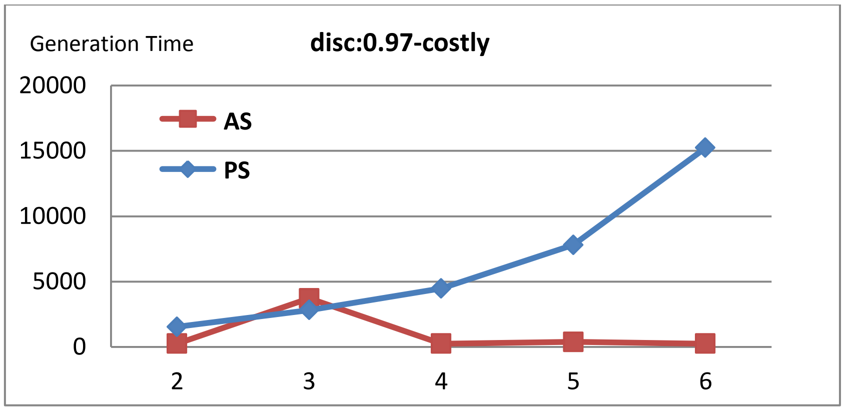


Figure 5.9. Comparison chart of SP policy generation time for the PS (blue line) and AS (red line) approaches, applied for the discounted coffee extended POMDPs with costly sensors. *X*-axes correspond to the number of sensors, *Y*-axes corresponds to the SP solver running time of policy generation.

This demonstrates a property of scalability of the AS approach with respect to the number of sensors, as the growth in the number of sensors is not affecting the SP policy calculation time which gives a great potential of the proposed approach to be applied in the real-world complex domain problems.

5.3.3 Used CPU%

We did not notice a strong correlation between the CPU usage and the number of scaled duplicated sensors in the model for both AS and PS approaches. However, the AS approach appeared to require less computational power during the SP policy calculation period.

In Figure 5.10 the red line shows the dependency of maximum CPU% usage during the SP policy generation for the discounted coffee extended problem with costly sensors. The blue line that corresponds to the PS SP policies dominates the red line for all tested sensor numbers. In addition, for the PS model policies, the CPU usage is increasing with sensor variety. This shows the advantage of the proposed idea computationally, as less power is demanded for the SP AS policy generation on one hand, and the computational power needs are not influenced by the sensor set extension on the other hand (for full result see Appendix C Table C.3).

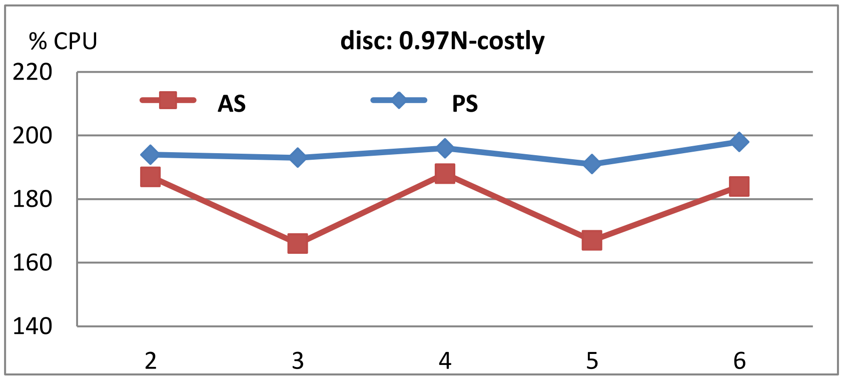


Figure 5.10. Comparison chart of the picks of CPU usage (%) in SP policy calculation for the PS (blue line) and AS (red line) approaches, applied for the discounted coffee extended POMDPs with costly sensors. *X*-axes correspond to the number of sensors, *Y*-axes corresponds to the maximum CPU% demanded by SP solver during each policy generation.

Chapter 6
Conclusion

In our research we are studying informative POMDPs, which allow performing observation-gathering actions aside from the state-transition ones. In other words, in uncertain situations the agent could choose to support its belief state with the related portion of information by querying sensors or perform a state-transition action. The information is becoming a fully functioning model component with the acquisition actions logically separated from the state-transition actions, to consider as an independent action set in the decision making process. This opens a new dimension of effective planning under uncertainty with the freedom of choice between perceptual and transition actions. With the ability of dynamic selective sensing of the environment the agent has a great potential to significantly decrease the payoffs of signal acquisition, streaming and processing on one side, and make more accurate successful decisions having more reliable state knowledge.

In this thesis we present an approach to sensor selection for smart phone applications. We formulated the problem as a partially observable Markov decision process (POMDP). The POMDP allows for fusion of information from partially available sensors, and provides a theoretical framework in which the relative utilities of sensing actions can be traded off against those of user goals and preferences. We have showed a non-myopic solution to the problem using a state of the art approximate planning algorithm Symbolic Perseus. We have tested our method on a 3 small example domains, comparing different policy types, discount factors and cost settings. The experimental results demonstrated in Chapter 5, proved that the proposed approach delivers a better policy in the situation of costly sensors, while at the same time provides the advantage of faster policy computation with less memory usage.

For the experiments in our model we make the assumption that the state space is discretized, relatively small and the planning horizon is finite. In order to keep the problem computationally feasible we did not exceed the state space of 20 states with low level of state value granularity. Also we suppose that the model state transitions are already given. However, POMDP model learning is a separate area of research which is not in the focus of this thesis.

Also in our experimental setups we are making sensors with overlapping sensor information as they all are replications of one sensor but with different accuracies and costs. From one side this makes the experimental setup and following results transparent for sanity check and analysis. For the sensor replication we describe a supportive justification of validity of modeled sensors, using the special replication technique [Appendix B] and performing simulations [Chapter 4.3]. The situations of mutually-exclusive sensors or possible sensor collisions were not considered for the experiments.

Considering the time spent on querying the sensor, the AS model is beneficial if the sensor queries are taking a long time (as it usually happens in practice), as the PS model will be slowed down at each action by querying all sensors all the time. However, the PS model may have an advantage if the sensor queries are fast and cheap to do, but the POMDP updates are slow and expensive. In such a case, the POMDP updates will be the limiting factor and the AS model will pay a high price if it has to query many sensors. One of the further research could be done on the investigation the case where we also include a cost for doing an update (so an additional cost on every single action), and use an action-dependent discount factor for the AS model.

This research could be extended by working towards general-purpose models for applications on smart phones that are connected to large networks of sensors. These networks could include other smart phones on a participating network, embedded environmental sensors, or information mined from the internet. The extended robot coffee delivery model, presented in the appendix section, describes how any application described by a POMDP can be combined with any number of sensors to produce a utility-sensitive or goal-directed sensor selection POMDP.

Some of the major challenges remaining in order to scale this to large sensor networks are:

1. The computation of even an approximate POMDP policy scales poorly with the number of observations and actions, and so as more sensors are added, our problems will become more challenging due to the increased size of both observation and action spaces. We tackle this problem by using state [Poupart05, Spaan05] and observation [Hoey05] aggregation methods.

2. The effective policy initialization could be used to boost the performance of the computed policy. As we are working with the domain where the initial sampling of the belief set affects the resulting policy, the sampling methods that are sensitive to the specifics of the AS domain could be very beneficial. For example, instead of using the simple worst-case policy initialization, the blind policy was proposed by Hauskrecht [Hauskrecht97]. In the domains with the information acquisition robot abilities blind policy iteration effectively explores the belief space with respect to the information acquisition areas [Kaplow10].

3. The set of sensors available at any time may have dynamicly changing quality/cost characteristics, as well as the set itself may not be stationary, and we need some way to rapidly re-plan based on the currently available set. Online algorithms for planning in large POMDPs seem a promising avenue in this regard.

4. The rapid specification of novel applications as POMDPs is one of our current areas of research that we believe will be relevant to this domain [Hoey11]. The next step could be providing the functionality of the POMDP planning and sensor selection algorithms as “black box” tools for smart-phone application developers who can then leverage the abilities of the POMDP to plan for sensor measurements that are relevant.

5. The elicitation of preferences about the relative costs of sensor readings, application goals, etc. is a challenging problem that needs to be addressed. However, we can engineer the preferences, or learn them by observing behaviour. Challenges and issues associated with actual specific implementation could be studied via set of experiments with real devices such as mobile phones.

Appendix A. Problems SPUDD format

SPUDD ("Stochastic Planning using Decision Diagrams") is a special format for POMDP model representation that is used as input for SP package [SPJ link]. The *.spudd* text file contains explicitly specified state variables, observations and corresponding observation functions, the actions and corresponding transition matrixes, reward functions, a discount factor and a threshold. The stochastic transitions are encoded in ADD structure on a skim semantic base(*).

SPUDD: "Tiger" extended model

*// **pos** - position of a tiger left or right side in the room 1 or room 2*
*// **me** - user's position: in the hall (**h**), in room 1 (**r1**) or in room 2 (**r2**)*
*// observations are listed as tiger is to the left side (**lft**), right (**rt**),*
*//no sound if tiger is in the other room (**noth**) and **NA** is observation is not available.*
// the reward function is so the optimal thing to do is the listen, move, listen, open
*//actions: **listen, move_r1,move_r2, openleft, openright***

```
(variables
        (pos   r1_left  r1_right r2_left  r2_right)
        (me h r1 r2))
(observations
        (tigp  lft  rt noth NA))

dd tigpOF
(me'  (h        (pos'    (r1_left  (tigp'    (lft (0.85))         (rt (0.15)) (noth (0.0)) (NA (0.0))))
                         (r1_right (tigp'    (lft (0.85))         (rt (0.15)) (noth (0.0))(NA (0.0)) ))
                         (r2_left  (tigp'    (lft (0.15))         (rt (0.85)) (noth (0.0)) (NA (0.0))))
                         (r2_right (tigp'    (lft (0.15))         (rt (0.85)) (noth (0.0)) (NA (0.0))))))
        (r1       (pos'    (r1_left  (tigp'    (lft (0.85))         (rt (0.15)) (noth (0.0)) (NA (0.0))))
                         (r1_right (tigp'    (lft (0.15))         (rt (0.85)) (noth (0.0)) (NA (0.0))))
                         (r2_left (tigp'     (lft (0.05))         (rt (0.05)) (noth (0.9)) (NA (0.0))))
                         (r2_right (tigp'    (lft (0.05))         (rt (0.05)) (noth (0.9)) (NA (0.0))))))
        (r2       (pos'    (r1_left (tigp'     (lft (0.05))         (rt (0.05)) (noth (0.9)) (NA (0.0))))
                         (r1_right (tigp'    (lft (0.05))         (rt (0.05)) (noth (0.9)) (NA (0.0))))
                         (r2_left  (tigp'    (lft (0.85))         (rt (0.15)) (noth (0.0)) (NA (0.0))))
                         (r2_right (tigp'    (lft (0.15))         (rt (0.85)) (noth (0.0)) (NA (0.0)))))))
enddd

unnormalised
init (me      (h        (1.0)) (r1         (0.0)) (r2 (0.0)))

action listen
        pos      (SAMEpos)
        me       (SAMEme)
observe
```

```
        tigp        (tigpOF)
endobserve
cost    (1.0)
endaction
```

* Detailed SPUDD specification is available at https://cs.uwaterloo.ca/~jhoey/research/spudd/index.php

```
action move_r1
        pos         (SAMEpos)
        me          (me       (h          (me'      (h (0.0)) (r1(1.0)) (r2(0.0))))
                                                    (r1  (mer1))
                                                    (r2  (mer2)))
observe
         tigp       (tigpNA)
endobserve
cost    (0.0)
endaction

action move_r2
        pos         (SAMEpos)
        me          (me       (h          (me'       (h (0.0)) (r1(0.0)) (r2(1.0)) ))
                                                    (r1  (mer1))
                                                    (r2  (mer2)))
observe
         tigp       (tigpNA)
endobserve
cost    (0.0)
endaction

action openleft
        pos         (0.5)
        me          (me' (h (1.0)) (r1(0.0)) (r2(0.0)) )
observe
         tigp       (tigpNA)
endobserve
cost     (me        (h (pos  (r1_left (0))  (r1_right(0)) (r2_left(0))  (r2_right(0)) ) )
                    (r1(pos  (r1_left (50))  (r1_right(-100)) (r2_left(0))  (r2_right(0)) ) )
                    (r2(pos  (r1_left (0))  (r1_right(0)) (r2_left(50))  (r2_right(-100)) ) ))

endaction
action openright
        pos         (0.5)
        me          (me' (h (1.0)) (r1(0.0)) (r2(0.0)) )
observe
         tigp       (tigpNA)
endobserve
cost     (me        (h (pos  (r1_left (0))  (r1_right(0)) (r2_left(0))  (r2_right(0)) ) )
                    (r1(pos  (r1_left (-100))  (r1_right(50)) (r2_left(0))  (r2_right(0)) ) )
                    (r2(pos  (r1_left (0))  (r1_right(0)) (r2_left(-100))  (r2_right(50)) ) ))
endaction

discount 0.95
tolerance 0.001
```

SPUDD: “Coffee delivery robot” model

```
(variables
        (huc yes no)            // huc: user has coffee
        (w drenched dry)        // w  : robot is wet
        (hrc yes no)            // hrc: robot has coffee
        (r heavy no)            // r  : raining outside?
        (u yes no)              // u  : robot has umbrella
        (l office shop)         // l  : robot's location
)

(observations       (wetp   wet dry))   // observations are whether or not the pavement is wet
init [* (huc   (yes    (0.0))   (no     (1.0))) // initial belief is a product of marginal beliefs
        (w     (drenched (0.0)) (dry    (1.0)))
        (hrc   (yes    (0.0))   (no     (1.0)))
        (r     (heavy  (0.5)) (no       (0.5)))
        (u     (yes    (0.0))   (no     (1.0)))
        (l     (office (1.0))   (shop   (0.0)))  ]

// some CPTS may be unnormalized -  this means all CPTs will be automatically normalized when read in
// unnormalised user keeps coffee if it has it
// example of an unnormalized CPT -  hucno is a predefined dd of the form <xx><yy>
// where <xx> is a variable name and <yy> is a value that variable can take on. whenever xx' has value
// the dd <xx><yy> then is 1  yy, zero otherwise.
```

```
dd defaulthuc
(huc (yes      (huc'    (yes (9.0)) (no (1.0))))
               (no      (hucno)))
enddd
dd moveloc                                  // location changes if robot moves
        (l     (office  (l'       (office   (1.00E-5)) (shop (9.000E-5))))
               (shop    (l'       (office   (9.0E+3)) (shop   (1E+3)))))
enddd
dd defaultrain                              // state of rain can progress from no <-> light <-> heavy
        (r     (heavy   (r' (heavy (0.9))  (no (0.1))))
               (no      (r' (heavy (0.1))  (no (0.9)))))
enddd
dd evenwetp
        (wetp'   (dry (0.5)) (wet (0.5)))
enddd
dd wetpOF
        (r'      (heavy   (wetpwet))
                 (no      (wetp'   (wet (0.1))    (dry (0.9)))))
enddd

action nothing                  // do nothing - exogenous events may affect huc and r
        huc       (defaulthuc)
        hrc (SAMEhrc) // SAME<xx> where <xx> is a variable name are pre-defined to be the CPT
        w       (SAMEw)         // that is 1 whenever xx is the same as xx', 0 otherwise.
        r       (defaultrain)
        u       (SAMEu)
        l       (SAMEl)
        observe
                wetp    (wetpOF)
        endobserve
      endaction
// move to shop if in office and vice-versa
// robot can get wet along the way if it's raining and robot has no umbrella
// this works less well if battery is low and doesn't work at all if battery is dead
// its uphill from office->shop so the cost is a little more (wear on motors)
    action move
        huc     (defaulthuc)
        hrc     (SAMEhrc)
        w       (w      (drenched       (wdrenched))
                        (dry            (r      (heavy  (u      (yes    (w'(drenched    (0.01))
                                                                        (dry            (0.99))))
                                                                (no     (w'(drenched    (0.99))
                                                                        (dry            (0.01)))))
                                                (no     (w'     (drenched       (0.0))
                                                                (dry            (1.0)))))))
        r       (defaultrain)
        u       (SAMEu)
        l       (moveloc)
        observe
                wetp    (wetpOF)
        endobserve
        cost    (l      (office (0.2))
                        (shop   (0.1)))

    endaction

    action delc             // robot delivers coffee to user
    huc (huc    (yes    (huc'   (yes    (0.90)) (no     (0.10))))
                        (no     (hrc    (yes    (l      (office (huc'   (yes    (0.80)) (no  (0.20))))
                                                        (shop   (hucno))))
                                        (no     (hucno)))))
        hrc     (hrc    (yes    (l      (office (hrc'   (yes    (0.10)) (no     (0.90))))
                                        (shop   (hrc'   (yes    (0.80)) (no     (0.20))))))
                        (no     (hrcno)))
        w       (SAMEw)
        r       (defaultrain)
        u       (uno)
        l       (SAMEl)
        observe
                wetp    (wetpOF)
        endobserve
    endaction
    action getu                     // robot gets the umbrella
        huc     (defaulthuc)
        hrc     (SAMEhrc)
        w       (SAMEw)
        r       (defaultrain)
        u       (u      (yes    (uyes))
                        (no     (l      (office (u'     (yes    (0.90)) (no     (0.10))))
                                        (shop   (uno)))))
        l       (SAMEl)
        observe
                wetp    (wetpOF)
        endobserve
        cost    (15.0)
      endaction
// robot buys coffee carefully, only if the robot is in the shop
// the robot can get wet here because of coffee spillage on receiving coffee
// this costs more in computation power, but the probability of getting wet is smaller than in buyc_fast
action buyc
        huc     (defaulthuc)
        hrc     (hrc    (yes    (hrcyes))
                        (no     (l      (office (hrcno))
                                        (shop   (hrcyes)))))
        w       (SAMEw)
        r       (defaultrain)
        u       (SAMEu)
        l       (SAMEl)
        observe
                wetp    (wetpOF)
        endobserve
        cost    (l      (office (0.0))
                        (shop   (0.3)))
    endaction

    reward [+  (huc     (yes    (25.0))
                        (no     (0.0)))
                (w      (drenched (-20.0))
                        (dry    (0.0)))]

    discount 0.95
    tolerance 0.001
```

SPUDD: “Coffee delivery robot” extended model

```
(variables
        (huc yes no)                    // huc: user has coffee
        (w drenched dry)                // w  : robot is wet
        (hrc yes no)                    // hrc: robot has coffee
        (u yes no)                      // u: robot has umbrella
        (l office shop)                 // l: robot's location
        (batt     dead low high)        // batt : battery state
        (weather moist rain comf)       // weather states: moisture, rain, comfortable
        (PR H N L)                      // PR: pressure
        (RH H N L)                      // RH: humidity
        (CT H N L)                      // CT: temperature
)

 (observations
                (wetp  wet  dry )
                (o_huc  yes no)
                (o_w     yes no)
                (OPR    H N L)
                (ORH    H N L)
                (OCT    H N L)
                (charge zero one two)
)

init [* (huc      (yes     (0.0))    (no      (1.0)))
        (w        (drenched          (0.0))    (dry      (1.0)))
        (hrc      (yes     (0.0))    (no      (1.0)))
        (u        (yes     (0.0))    (no      (1.0)))
        (l        (office  (1.0))    (shop    (0.0)))
        (batt     (dead    (0.0))    (low     (0.0))    (high     (1.0)))
        (weather (moist(0.1)) (rain(0.1)) (comf(0.8)))
        (PR (H(0.0)) (N(1.0)) (L(0.0)))
        (RH (H(0.0)) (N(1.0)) (L(0.0)))
        (CT (H(0.0)) (N(1.0)) (L(0.0)))
   ]

unnormalised

dd wetp_OF
(wetp'
 (wet (weather'
  (moist(0.2))  (rain(0.98))  (comf(0.0)))
) (dry (weather'
  (moist(0.8))  (rain(0.02))  (comf(1.0)))
) )
enddd

dd o_hucOF                  (huc'  (yes     (o_hucyes))
                  (no       (o_hucno)))
enddd

dd o_wOF
  (w'     (drenched        (o_wyes))
                  (dry      (o_wno)))
enddd

dd chargeOF
         (batt'    (dead    (chargezero))
                  (low      (charge'(zero     (0.1))    (one      (0.9))    (two      (0.0))))
                  (high     (charge'(zero     (0.0))    (one      (0.2))    (two      (0.8)))))
enddd

dd sensorNoise
  (0.13)
enddd

dd sensorAccuracy
  (0.87)
enddd

dd weather_change_const
  (0.01)
enddd

dd OPROF
(PR'   (H   (OPR'   (H (sensorAccuracy))   (N  (sensorNoise))   (L (sensorNoise))  ))
     (L   (OPR'   (H (sensorNoise))   (N  (sensorNoise))   (L (sensorAccuracy)) ))
     (N   (OPR'   (H (sensorNoise))   (N  (sensorAccuracy))   (L (sensorNoise))  )))
enddd

dd OCTOF
(CT'   (H   (OCT'   (H (sensorAccuracy))   (N  (sensorNoise))   (L (sensorNoise))  ))
     (L   (OCT'   (H (sensorNoise))   (N  (sensorNoise))   (L (sensorAccuracy))  ))
     (N   (OCT'   (H (sensorNoise))   (N  (sensorAccuracy))   (L (sensorNoise))  )))
enddd

dd ORHOF
(RH'   (H   (ORH'   (H (sensorAccuracy))   (N  (sensorNoise))   (L (sensorNoise))  ))
     (L   (ORH'   (H (sensorNoise))   (N  (sensorNoise))   (L (sensorAccuracy))  ))
     (N   (ORH'   (H (sensorNoise))   (N  (sensorAccuracy))   (L (sensorNoise))  )))
enddd

//STATE DYNAMICS
dd RHweather
   (weather' (moist (RH' (H (0.98)) (L (0.01)) (N(0.01))))
          (comf (RH' (H (0.2)) (L (0.2)) (N(0.6))))
          (rain (RH' (H (0.8)) (L (0.1)) (N(0.1)))))
enddd
```

```
dd PRweather
(weather'    (moist( PR' (H (0.3)) (L (0.3)) (N(0.3))))
             (comf (PR' (H (0.2)) (L (0.2)) (N(0.6))))
             (rain (PR' (H (0.01)) (L (0.98)) (N(0.01)))))
enddd

dd CTweather
(weather' (moist(CT' (H (0.98)) (L (0.01)) (N(0.01))))
          (comf (CT' (H (0.2))  (L (0.2)) (N(0.6))))
          (rain (CT' (H (0.8)) (L (0.1)) (N(0.1)))))
enddd

dd defaulthuc
(huc      (yes    (huc'      (yes      (9.0)) (no          (1.0))))
                  (no        (hucno)))
enddd

dd moveloc
          (l        (office   (l'       (office   (1.00E-5)) (shop          (9.000E-5))))
                    (shop     (l'       (office   (9.0E+3)) (shop   (1E+3)))))
enddd

dd batt_discharge_slow
          (batt     (dead     (batt'    (dead     (1.0))    (low      (0.0))    (high     (0.0))))
                              (low      (batt'    (dead     (0.2))    (low      (0.8))    (high     (0.0))))
                              (high     (batt'    (dead     (0.0))    (low      (0.01))   (high     (0.99)))))
enddd


dd batt_discharge_fast
                    (batt     (dead     (batt'    (dead     (1.0))    (low      (0.0))    (high     (0.0))))
                              (low      (batt'    (dead     (0.5))    (low      (0.5))    (high     (0.0))))
                              (high     (batt'    (dead     (0.1))    (low      (0.4))    (high     (0.5)))))
enddd


action wait
          huc       (defaulthuc)
          hrc       (SAMEhrc)
          w         (SAMEw)
          u         (SAMEu)
          l         (SAMEl)
          weather    [+  (SAMEweather)  (weather_change_const)]
   RH (RHweather)
   PR (PRweather)
   CT (CTweather)
   batt   (batt_discharge_slow)
          observe
                    wetp (wetp_OF )
                    o_huc   (o_hucOF)
                    o_w     (o_wOF)
```

```
                ORH    (ORHOF)
                OPR    (OPROF)
                OCT    (OCTOF)
                charge  (chargeOF)
        endobserve
        cost (weather   (moist (-4.0))
                                        (rain      (0.0))
                                        (comf  (0.0))
                )
endaction

action move
        huc       (defaulthuc)
        hrc       (SAMEhrc)
        w         (w        (drenched (wdrenched))
                            (dry       (weather (moist (w'       (drenched         (0.2))
                                                                          (dry                  (0.8))))
                                                (rain   (u (yes (w' (drenched       (0.01))
                                                                         (dry                  (0.99))))
                                                            (no (w'  (drenched         (0.99))
                                                                         (dry                  (0.01))))))
                                                (comf  (wdry)))))
        u         (SAMEu)
        l         (moveloc)
        weather   [+  (SAMEweather)  (weather_change_const)]
   RH (RHweather)
   PR (PRweather)
   CT (CTweather)
   batt   (batt_discharge_slow)
        observe
                 wetp (wetp_OF )
                o_huc   (o_hucOF)
                o_w      (o_wOF)
                ORH    (ORHOF)
                OPR    (OPROF)
                OCT    (OCTOF)
                charge  (chargeOF)
        endobserve
        cost      [+(l       (office   (0.2))
                              (shop    (0.1)))
                  (weather           (moist (0.2))
                                (rain  (u (yes (0.0)) (no(10.0))))
                                (comf  (0.0)))
                              ]
endaction

action delc
        huc       (huc      (yes      (hrc      (yes      (l          (office   (hucyes))
                                                                          (shop   (defaulthuc))))
                                                                         (no                   (defaulthuc))))
                                      (no                 (hrc      (yes      (l          (office   (hucyes))
                                                            (shop   (defaulthuc))))
                                           (no      (defaulthuc)))))
      hrc       (hrc      (yes      (l          (office   (hrcno))
                                      (shop   (hrc'      (yes      (0.8)) (no          (0.2))))))
                                     (no        (hrcno)))

      w         (SAMEw)
      u         (uno)
      l         (SAMEl)
      weather   [+  (SAMEweather)  (weather_change_const)]
  RH (RHweather)
  PR (PRweather)
  CT (CTweather)
  batt  (batt_discharge_slow)
      observe
                 wetp (wetp_OF )
            o_huc   (o_hucOF)
            o_w     (o_wOF)
            ORH    (ORHOF)
            OPR    (OPROF)
            OCT    (OCTOF)
            charge  (chargeOF)
      endobserve
      cost (huc         (yes      (5.0))
                        (no       (0.0)))
endaction

action getu
      huc       (defaulthuc)
      hrc       (SAMEhrc)
      w         (SAMEw)
      u         (u        (yes      (uyes))
                          (no       (l          (office   (uyes))   //'       (yes      (0.90)) (no         (0.10))))
                                                (shop     (uno)))))
      l         (SAMEl)
      weather   [+  (SAMEweather)  (weather_change_const)]
  RH (RHweather)
  PR (PRweather)
  CT (CTweather)
  batt  (batt_discharge_slow)
      observe
            wetp (wetp_OF )
            o_huc   (o_hucOF)
            o_w     (o_wOF)
            ORH    (ORHOF)
            OPR    (OPROF)
            OCT    (OCTOF)
            charge  (chargeOF)
      endobserve
      cost      [+(4.0)
      (weather          (moist  (0.0))
                                      (rain  (-5.0))
                                      (comf   (0.0)))
                               ]
endaction

action buyc
      huc      (defaulthuc)
      hrc      (hrc      (yes      (hrcyes))
                         (no       (l        (office   (hrcno))
                                                (shop     (hrcyes)))))
      w        (SAMEw)
      u        (SAMEu)
      l        (SAMEl)
      weather    [+  (SAMEweather)  (weather_change_const)]
  RH (RHweather)
  PR (PRweather)
  CT (CTweather)
  batt   (batt_discharge_slow)
      observe
            wetp (wetp_OF )
            o_huc   (o_hucOF)
            o_w     (o_wOF)
            ORH     (ORHOF)
            OPR     (OPROF)
            OCT     (OCTOF)
            charge  (chargeOF)
      endobserve
      cost     [+(l      (office   (3.0))
                         (shop     (0.0)))
                         (huc      (yes      (4.0))
                         (no       (0.0)))]
endaction

reward [+[+    (huc      (yes      (25.0))
                                                (no       (0.0)))
                            (w        (drenched (-15.0))
                                      (dry      (0.0)))]
                            (batt     (dead     (-13.0))
                                                (low      (0.0))
                                                (high     (0.0)))
]

discount 0.95
tolerance 0.001
```

Appendix B. Sensor replication technique

Below we describe a methodology to generate an arbitrary number of useful sensors on the simple example of POMDP problem. This technique allows to solve the sensor cost/accuracy tradeoff and generate the set of cost-accuracy pairs for the required number of sensors. Thus, one initial sensor could be replicated with new parameters to scale the number of useful meaningful sensors in the system.

Suppose we have a simple problem, when the robot needs to decide whether he should take an umbrella or just go, based on the sensor information about current weather situation. If it is rainy outside, the robot has to take an umbrella not to get wet. Suppose, there is a sensor that the robot can query to verify the current weather. Then the corresponding POMDP problem will have one state variable with 2 values (*rain* {*yes, no*}), one observation variable (*is_raining* {*yes, no*}), one state-observation action (*query_sensor*) and 2 state-transition actions (*go* and *take_umbrella*). There is a cost *c* (>0) associated with a query of rain sensor, as well as there is a cost of taking umbrella *tu* (>0) and cost of getting wet *gw* (*gw*>*tu*). Suppose, the sensor reports the rain observation with an accuracy of d, $d \in [0.6,1]$ and with a noise of $(1-d)$ accordingly [Table B.1].

State: rain	*Observation*: is_raining = yes
T	*d*
F	1-*d*

Table B.1 Conditional probability table for rain sensor.

If the robot has only one action attempt left to perform, he can base his decision only on the information about current belief state with respect to the immediate reward he will get from actions: $V_0(s) = \mathrm{R}(s,a)$.

Let *x* = *Pr* (*rain*=*yes*) be a probability of rain, then the belief space could be represented as the [0,1] interval of *x* values. Then the value of an α-vector at a current a belief state is simply an inner product of 2 vectors: $V^{\alpha}(b) = \sum_{s\in S} V(s)b(s)$. The α-vectors, corresponding to the 2 actions will be:

$$V_{go}(b) = x \cdot (-gw) + (1-x) \cdot 0 = -x \cdot gw.$$

$$V_{take\ umbrella}(b) = x \cdot (-tu) + (1-x) \cdot (-tu) = -tu.$$

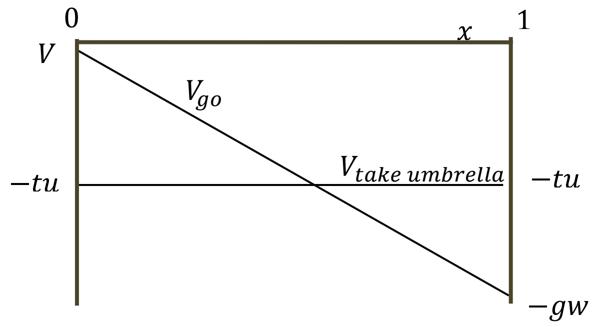


Figure B.1 Geometric representation of the optimal value function for one-step horizon. The *X*-axis represents the belief state (probability of rain), and the *Y*-axis corresponds to the value.

The geometric representation of those α-vectors is shown on the Figure B.1. The optimal value function will consist 2 of pieces of dominating α-vectors, corresponding to the most rewarded action at the current belief region. If the chances of rain are lower (left side of belief interval) then the robot will decide to go, in the higher risk of rain (right side of belief interval) the best action would be to take umbrella.

When the action horizon is 2, the robot gets a choice to make a state-observation action to support the following state-transition action. Suppose the policy prescribes to query the sensor and act according to the observations. In particular, the strategy is to take the umbrella if the sensor reports rain and go otherwise. Assuming that the observation action is not influencing on the state transition, the value function corresponding to the observation action will be the sum of rewards gained for all possible observations for the current belief [Table B.2]:

rain	sensor	Value
T	T	$x \cdot d \cdot (-c - tu)$
F	F	$(1 - x) \cdot d \cdot (-c)$
T	F	$x \cdot (1 - d) \cdot (-c - gw)$
F	T	$(1 - x) \cdot (1 - d) \cdot (-c - tu)$

Table B.2 Conditional probability table for the value of 2-step horizon policy.

The equation of the α-vector line, corresponding to the sensor query action, will be the following:

$$x \cdot d \cdot (-c - tu) + x \cdot (1 - d) \cdot (-c - gw) + (1 - x) \cdot (1 - d) \cdot (-c - tu) + +(1 - x) \cdot d \cdot (-c)$$
$$= x \cdot (d \cdot gw - 2 \cdot d \cdot tu - gw + tu) + d \cdot tu - c - tu.$$

At a fixed time step t the current approximation of an optimal value function is made of the pieces of dominating α-vectors over different corresponding regions of a belief space:

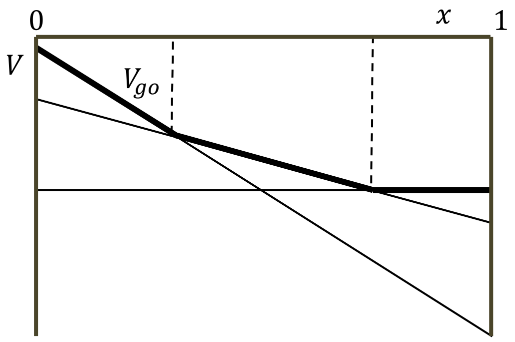


Figure B.2 Geometric representation of the optimal value function for the 2 –step horizon policy. The *X*-axis represents the belief state (probability of rain), and the *Y*-axis corresponds to the value.

Note, that in the middle of the belief space, in the least certain region, the dominating α-vector refers to the sensor query action. Thus, this action appears in the optimal policy.

The usefulness of the sensor is defined not only by existing in the model or being accessible, but also by the appearing of the corresponding query action in the optimal policy. It shows that the functionality of this sensor fulfills the informational system needs in some situations of uncertainty mapped in the belief regions where the policy will prompt the sensor call.

Suppose we have a goal to construct an arbitrary number of valid sensors in the system. In other words the corresponding sensor query action is guaranteed to be optimal at some belief space region. In the POMDP model, each sensor query action is uniquely defined by the accuracy resolution of the sensor, specified in the observation function, and the associated information acquisition cost. Geometrically the position of the corresponding α-vector is defined by the coefficients from the line equation, which expresses the relationship between sensor accuracy and cost and gives a value for the action at each belief point of the belief space:

$$V_{querry\ sensor}: y = x \cdot (d \cdot gw - 2 \cdot d \cdot tu - gw + tu) + d \cdot tu - c - tu \tag{1}$$

Equations of state-transition actions of taking umbrella and going will be:

$$V_{go}: \ y = -tu \tag{2}$$

$$V_{take\ umbrella}: y = -gw \cdot x \tag{3}$$

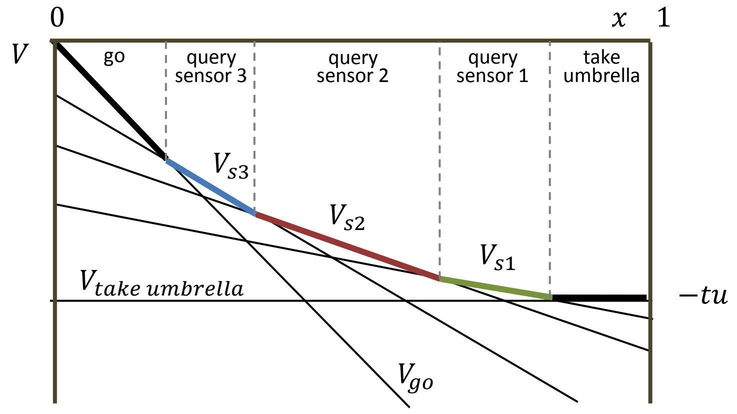


Figure B.3 Stair-case representation of α-vectors of meaningful sensor.

To construct *n* sensors we need to setup the cost and accuracy characteristics for *n* sensors so that the corresponding vector lines will be arranged to have a region of dominance, hence will appear in the piece-wise optimal value function. One of the ways to organize this kind of arrangement is to make alpha-vectors to be tangent to the to a circle arc. In that case, lines will have sequential "stair-case" dominance over each other [Fig. B.3, B.4]. The sensor characteristics inferred from equations of those lines will

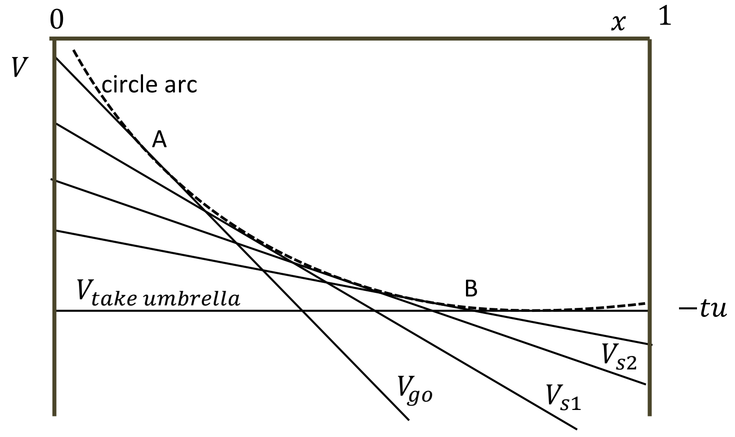


Figure B.4 Stair-case order α-vectors when they tangent the circle arc.

define the required useful sensors.

Thus, we have to find *n* pairs of sensor accuracies and sensor costs, so the corresponding alpha vectors will touch a circle curve.

Suppose that V_0 is the initial sensor line for the noisiest sensor ($c = c_0, d = d_0$). Let (x_1, y_1) be the point of intersection of V_0 and V_{go}, (x_2, y_2) be the point of intersection of V_0 and $V_{take\ umbrella}$ [Fig. B.5]. then, from equations (1) and (3) we can express coordinates (x_1, y_1) as:

$(x_1, y_1) = V_0 \cap V_{go}\ :\ x \cdot (d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu) + d_0 \cdot tu - c_0 - tu\ =\ -gw \cdot x,$

$x \cdot (d_0 \cdot gw - 2 \cdot d_0 \cdot tu + tu) + d_0 \cdot tu - c_0 - tu\ =\ 0.$

$$x_1 = \frac{-d_0 \cdot tu + c_0 + tu}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu + tu} \qquad (4)$$

$$y_1 = -gw \cdot x_1 \qquad (5)$$

Similarly, from (1) and (2) we can find (x_2, y_2):

$(x_2, y_2) = V_0 \cap V_{take\ umbrella}$:

$x \cdot (d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu) + d_0 \cdot tu - c_0 - tu\ \ =\ -tu,$

$x \cdot (d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu) + d_0 \cdot tu - c_0 =\ 0.$

$$x_2 = \frac{c_0 - d_0 \cdot tu}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu} \qquad (6)$$

$$y_2 = -tu \qquad (7)$$

Having (4,5,6,7) we can find the distance between (x_1, y_1) and (x_2, y_2).

$$dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \qquad (8)$$

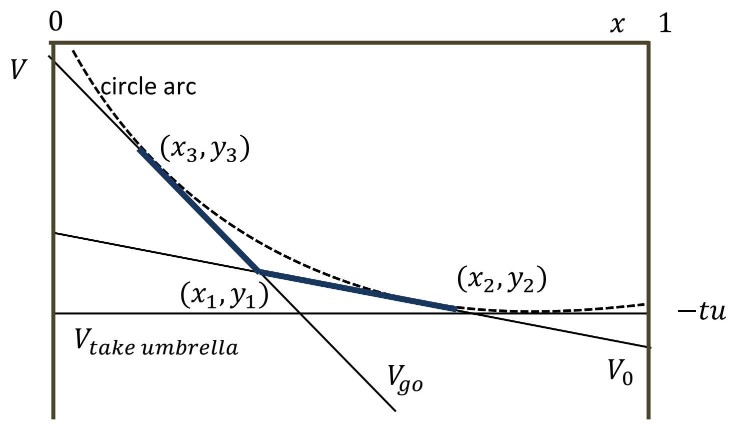


Figure B.5 Two touch points of a circle arc equally distanced from the intersection (x_1, y_1) of "go" and "take umbrella" α-vector lines.

To find the circle equation, let's assume that there are two points that the circle arc will go through. Suppose that the circle touches V_0 at the point (x_2, y_2), and V_{go} at the point (x_3, y_3), distanced equally (by $dist$) from the point (x_1, y_1) [Fig. B.5]. From (3, 4, 5, 8):

$$x_3 = x_1 - \frac{dist}{\sqrt{1+gw^2}}, \tag{9}$$

$$y_3 = -gw \cdot x_3 \tag{10}$$

The center of a circle will be an intersection of 2 lines l_1 and l_2, that are perpendicular to V_0 and V_{go} at the points (x_2, y_2) and (x_3, y_3) accordingly [Fig. B.6].

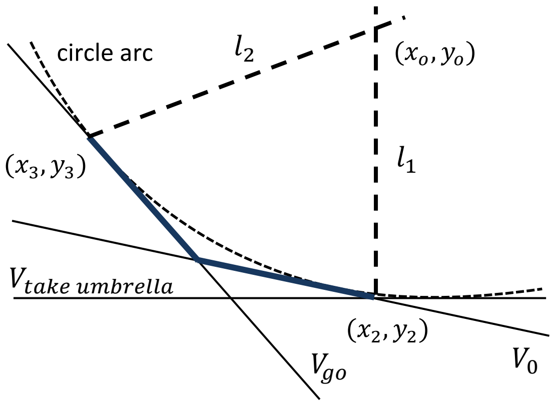


Figure B.6 The center of a circle (x_o, y_o) as a cross-point of perpendiculars to the V_{go} and V_0 α-vectors.

For (x_0, y_0) we need to find the l_1 and l_2 equations first. Since l_1 is perpendicular to V_0 from equation (1), the l_1 slope is $k = -\frac{1}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu}$. From the general equation of a line $y = k \cdot x + C$, the coefficient C is:

$$C = y_2 + \frac{x_2}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu}.$$

Thus, the line equation for l_1 is:

$$y = -\frac{x}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu} + y_2 + \frac{x_2}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu}.$$

Similarly, the line equation for l_2 is:

$$y = \frac{x}{gw} + y_3 - gw.$$

Then, the center of a circle (x_o, y_o) is the intersection point of l_1 and l_2 lines:

$$\frac{x_0}{gw} + y_3 - \frac{x_3}{gw} = -\frac{x_0}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu} + y_2 + \frac{x_2}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu},$$

$$x_o = \frac{y_2 + \frac{x_2}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu} - y_3 + \frac{x_3}{gw}}{\frac{1}{gw} + \frac{1}{d_0 \cdot gw - 2 \cdot d_0 \cdot tu - gw + tu}}, \quad (11)$$

$$y_o = \frac{x_o}{gw} + y_3 - \frac{x_3}{gw} \quad (12)$$

The circle radius *r* is a distance between the center (x_o, y_o) and (x_2, y_2), using (6, 7, 11, 12):

$$r = \sqrt{(x_o - x_2)^2 + (y_o - y_2)^2} \quad (13)$$

Let α-vector lines of new n sensors touch the circle at the points (x_i, y_i), with abscises $x_i \in [x_3, x_2]$ that are equally distanced from each other:

$$x_i = x_3 + i \cdot \frac{x_2 - x_3}{n}, \ 1 \le i \le n - 1, \quad (14)$$

and ordinates, that are found from the general circle equation, using (11, 12, 13, 14):

$$y_i = y_o + \sqrt{(r^2 - (x_i - x_o)^2)} \quad (15)$$

Those α-vector lines are tangents to the circle, thus, are perpendicular to the radius. The radius is a line that goes through the center of a circle and a point of tangency (x_i, y_i). Thus, the slope of radius line is equal to $\frac{y_i - y_o}{x_i - x_o}$. Then the slope of α-vector line will be $-\frac{y_i - y_o}{x_i - x_o}$. Given x_i (13) and a slope, we can find the free coefficient from the general equation of a line:

$$C = y_i + x_i \cdot \frac{x_i - x_o}{y_i - y_o} \quad (16)$$

From (15) and (16) the equations of a new sensor α-vector lines will be:

$$y = x \cdot \frac{x_o - x_i}{y_i - y_o} + y_i + x_i \cdot \frac{x_i - x_o}{y_i - y_o} \quad (17)$$

Comparing (1) and (17) that stand for the same line, we have:

$$y \;=\; x \cdot (d_i \cdot gw - 2 \cdot d_i \cdot tu - gw + tu) + d_i \cdot tu - c_i - tu \;.$$

So, $d_i \cdot gw - 2 \cdot d_i \cdot tu - gw + tu \;= \frac{x_o - x_i}{y_i - y_o}$, and $d_i \cdot tu - c_i - tu \;=\; y_i + x_i \cdot \frac{x_i - x_o}{y_i - y_o}$,

Thus, sensor accuracy and costs are found from the following equations:

$$d_i \;= \frac{\frac{x_o - x_i}{y_i - y_o} + gw - tu}{gw - 2 \cdot tu} \tag{18}$$

$$c_i \;=\; -\left(y_i + x_i \cdot \frac{x_i - x_o}{y_i - y_o} - d_i \cdot tu + tu\right)\;, \tag{19}$$

where tu, gw are specified in the POMDP, c_0, d_0 are given and y_i, x_i, y_o, x_o are known from (14, 15, 11, 12) respectively.

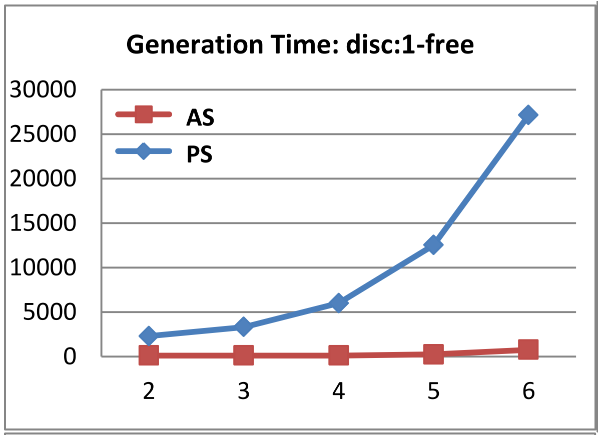
Thus, we can calculate n pairs of accuracy and cost, such that the corresponding alpha-vector will appear in the optimal policy and guarantee the sensor usefulness.

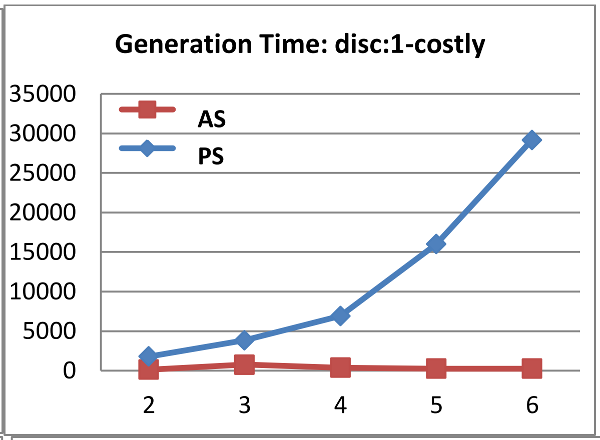
Below is the table of four sensor's cost-accuracy pairs, generated with the described approach with the following parameters: c_0=0.5, d_0=0.7, n=4, tu =17, gw =19.

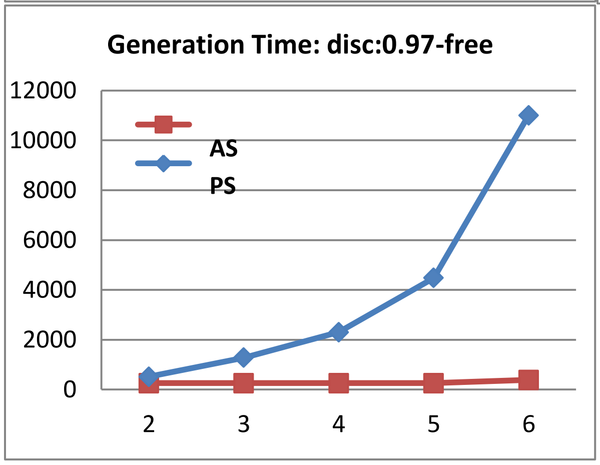
cost	1.5266	1.0544	0.7311	0.5000
accuracy	0.9663	0.8515	0.7664	0.7000

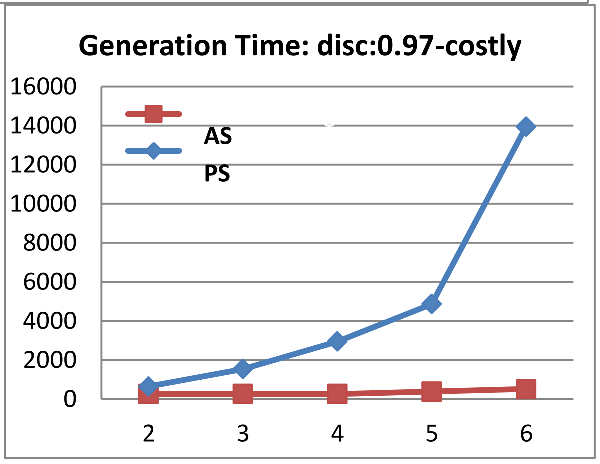
Table B.3 Generated sensor's cost-accuracy pairs.

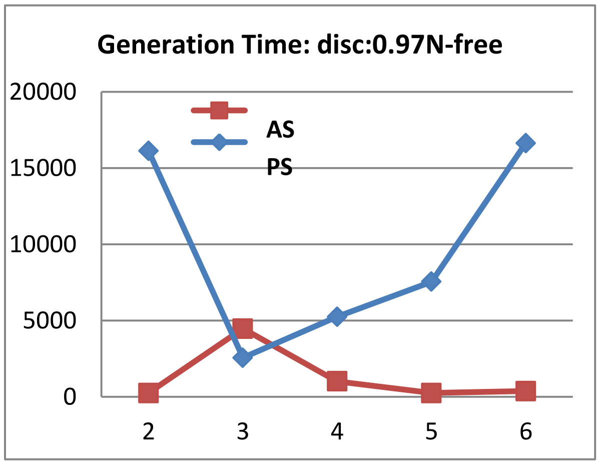
Appendix C. Experimental results











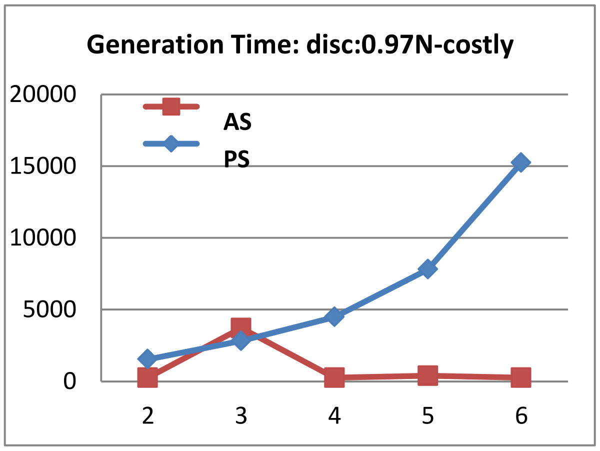


Table C.1. Dependency of policy generation time of SP on the number of sensors for different discount factors and sensor cost settings in coffee extended problem.

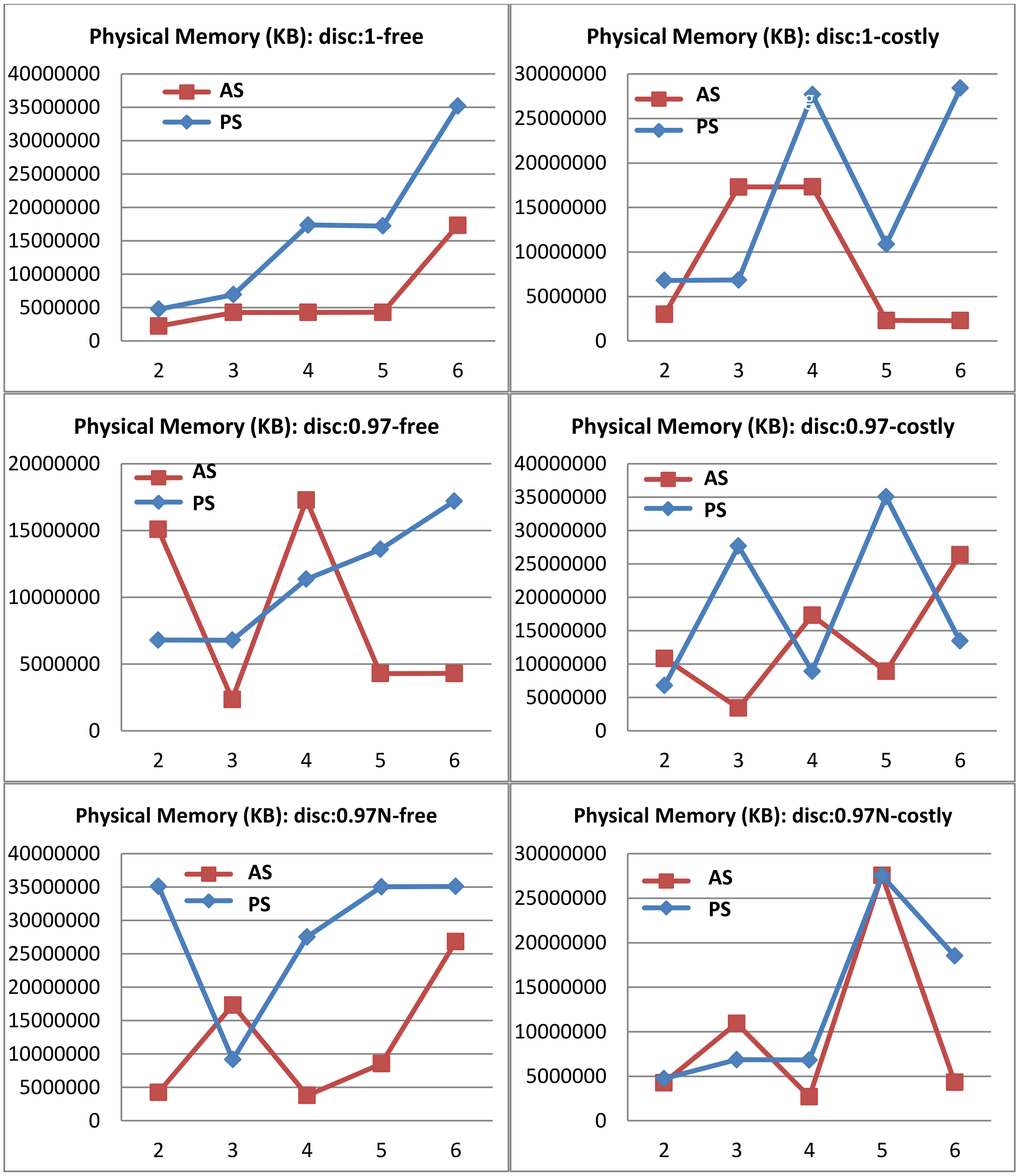


Table C.2. Dependency of maximum of total physical memory usage (KB) during policy calculation of SP on the number of sensors for different discount factors and sensor cost settings in coffee extended problem.

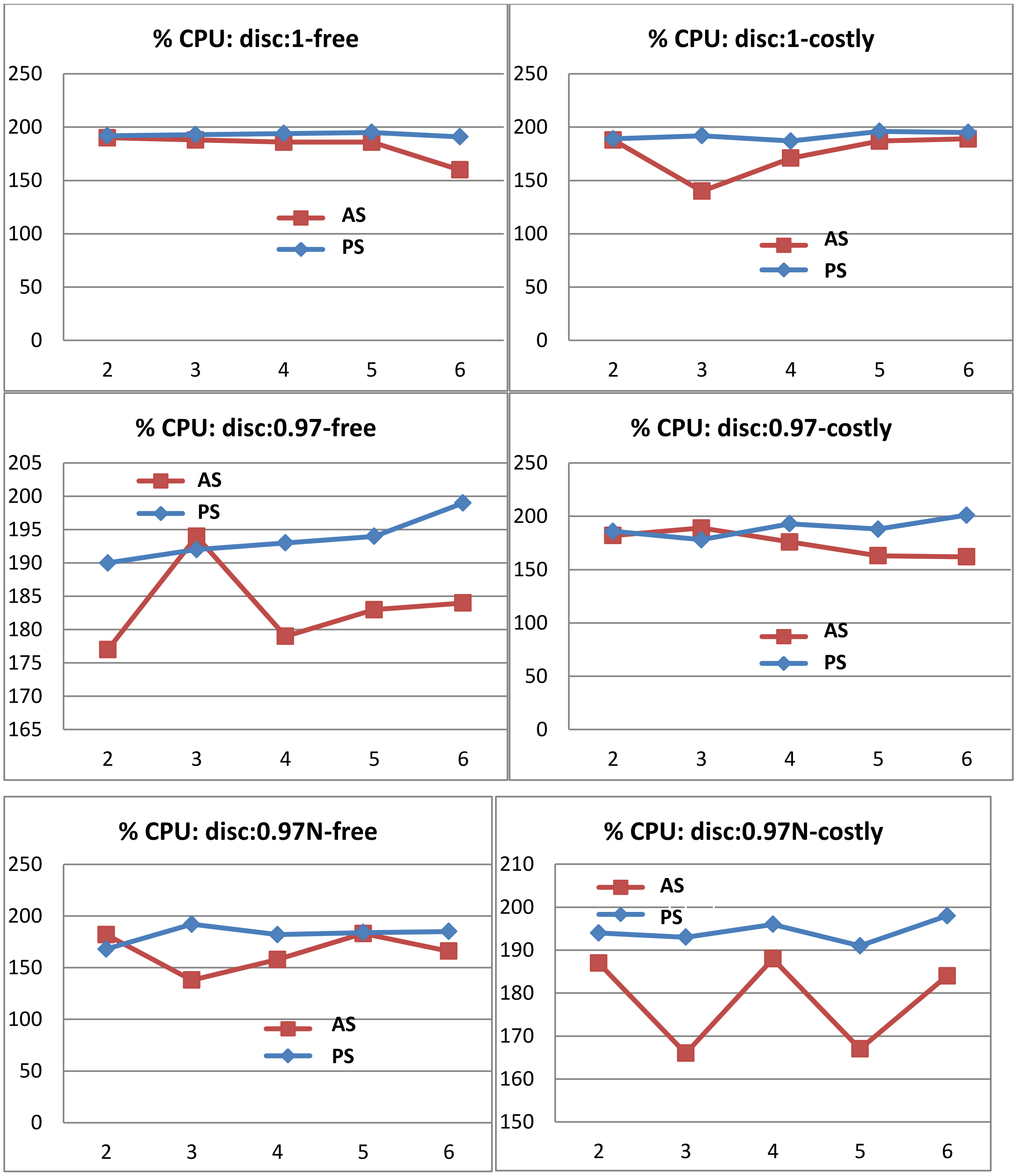
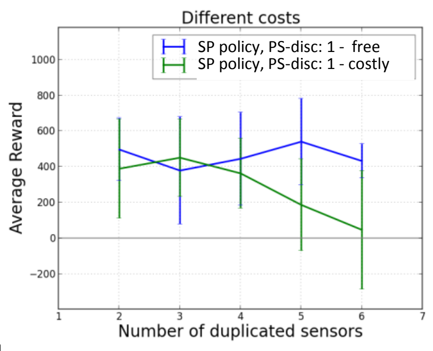
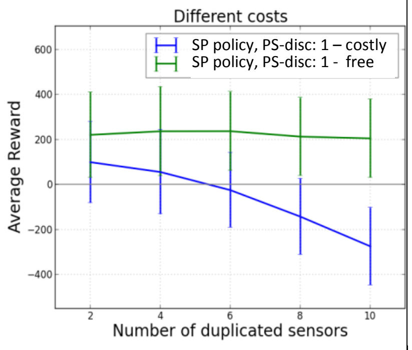


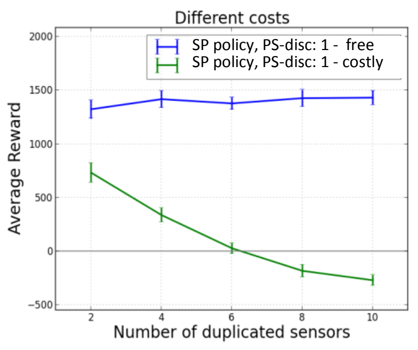
Table C.3. Dependency of CPU usage (%) during policy calculation of SP on the number of sensors for different discount factors and sensor cost settings in the coffee extended problem.

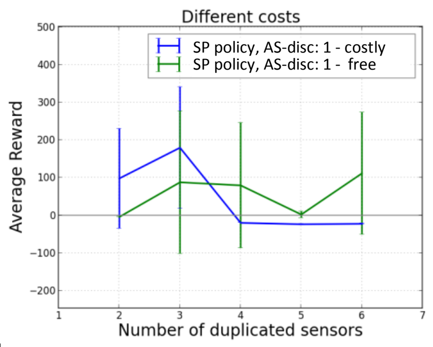
Different costs

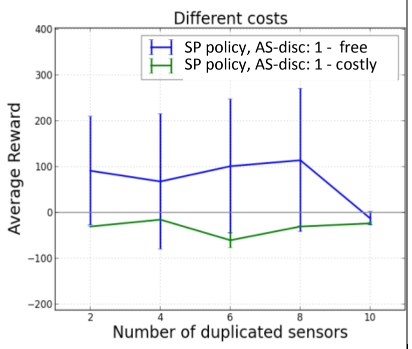
Coffee extended problem **Coffee problem** **Tiger problem**

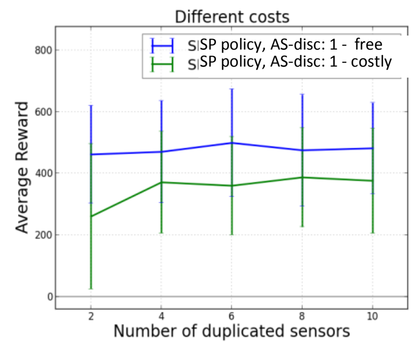


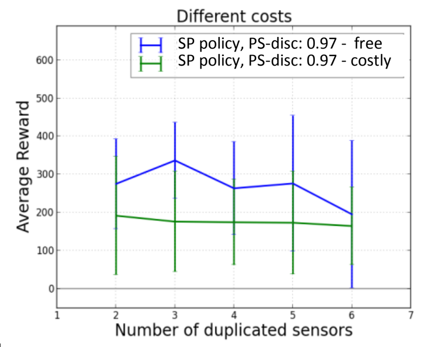


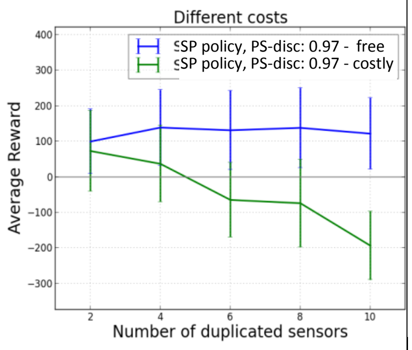


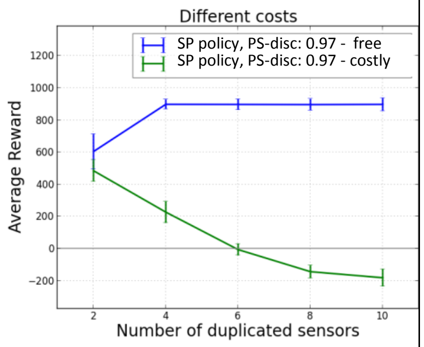












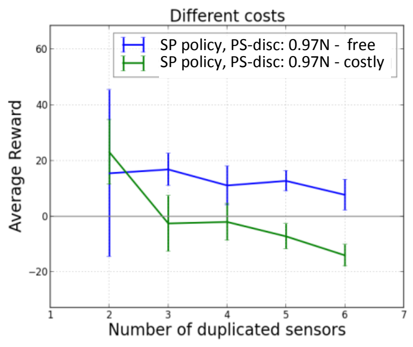
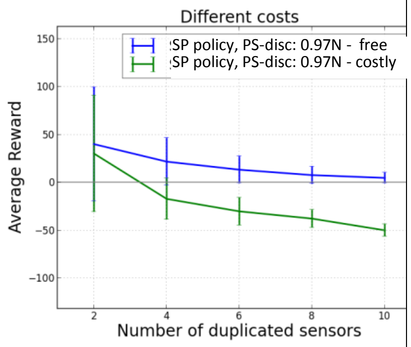
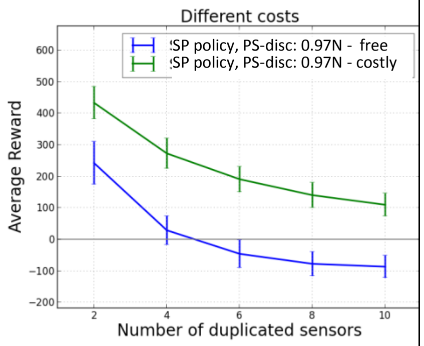
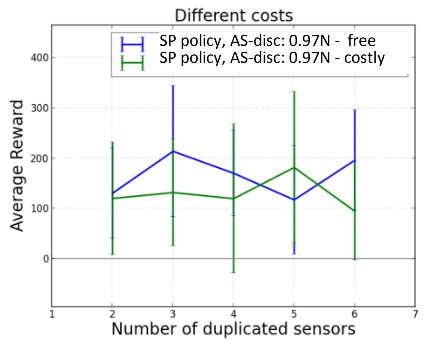
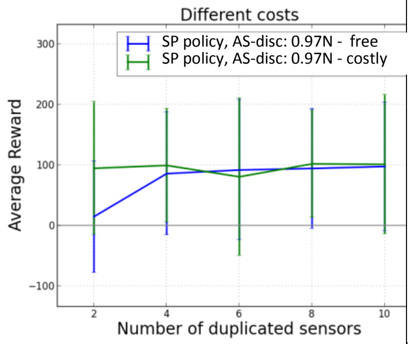
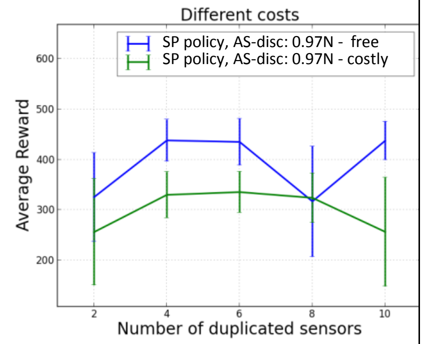
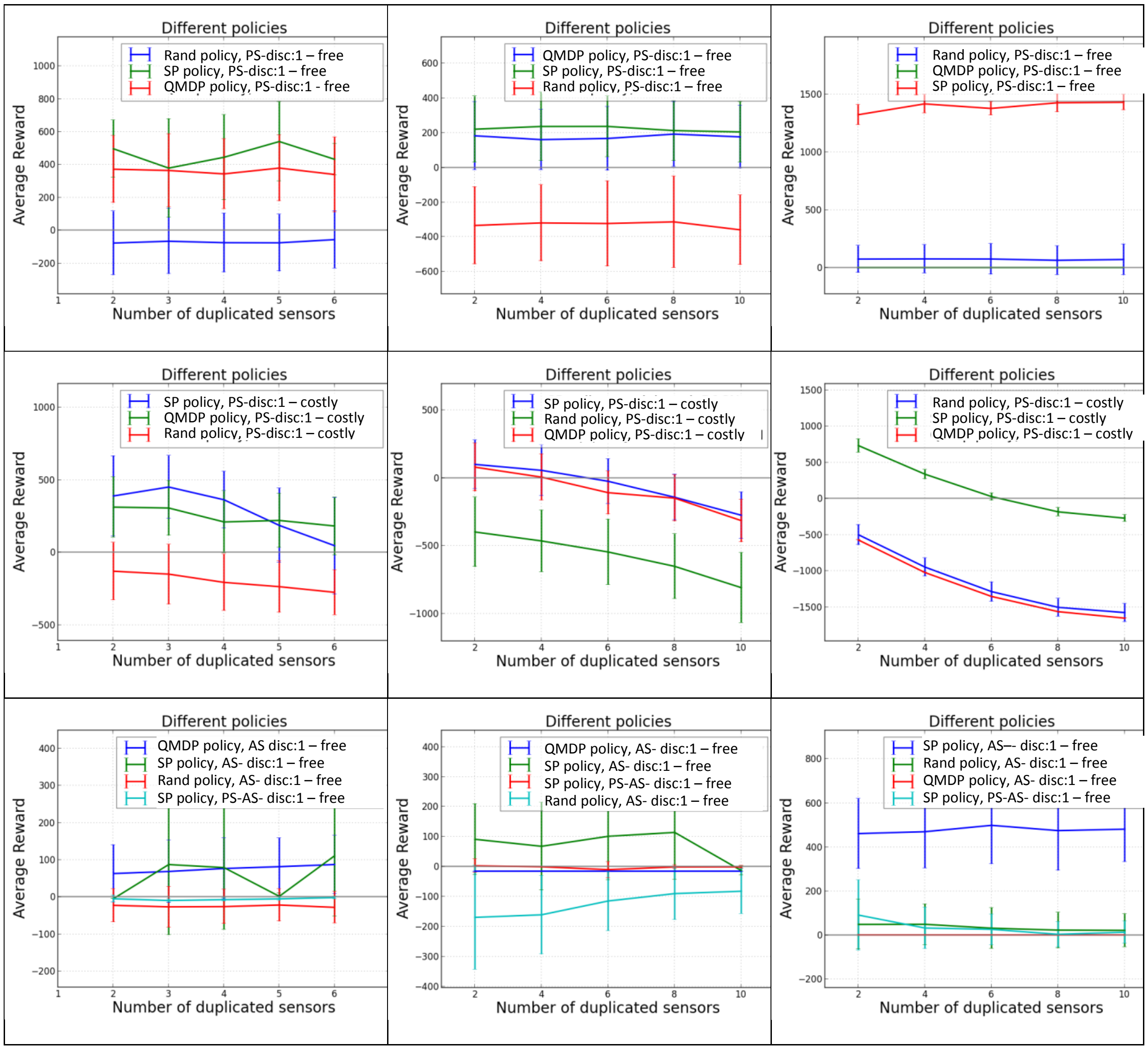
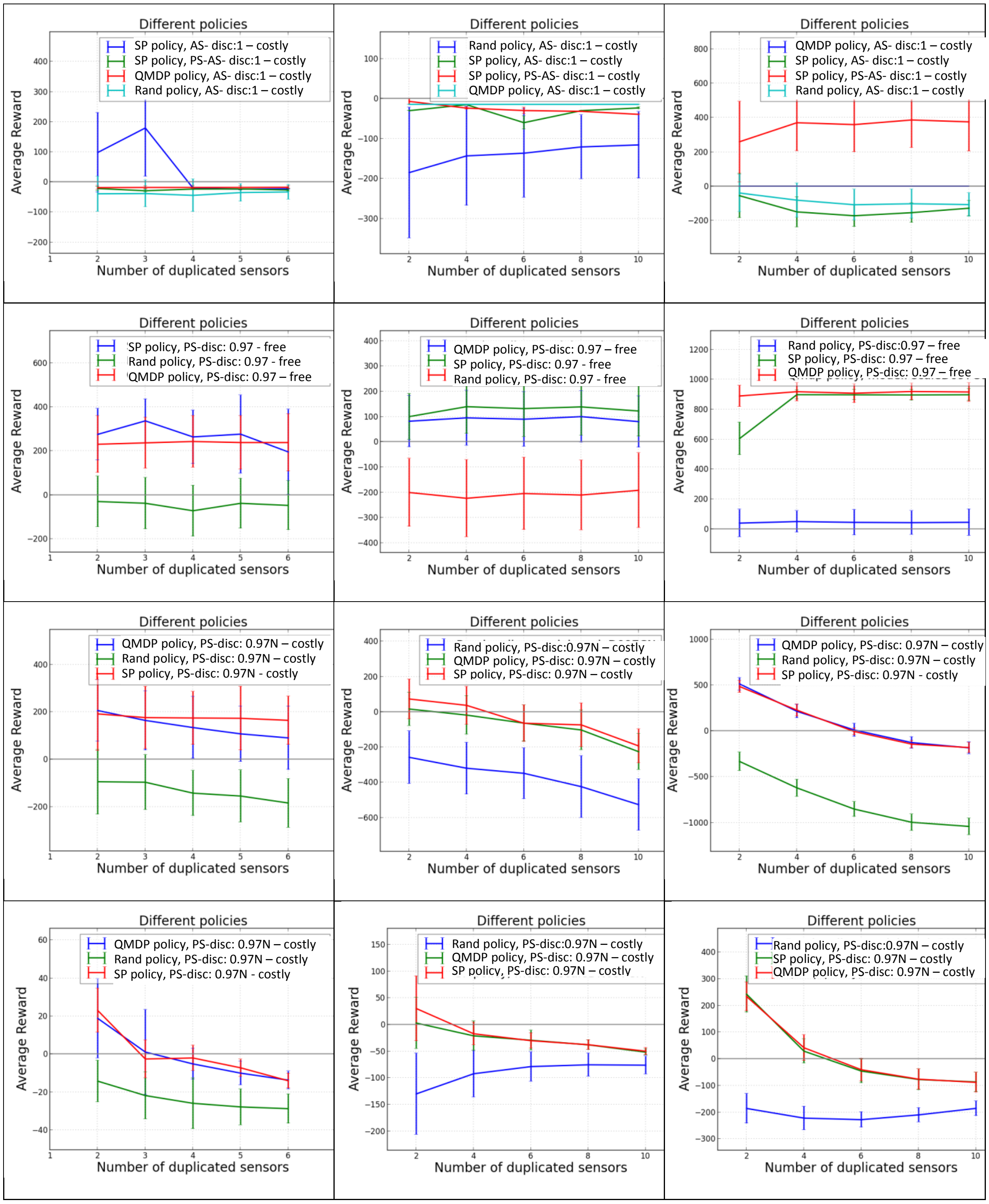
		
		

Table C.4. Graphs of comparing SP policies for free and costly sensors for 3 POMDP problems: extended coffee delivery problem (1st column), classic coffee delivery problem (2nd column), and extended tiger problem (3rd column). Each row corresponds to the unique combination of model type (the active sensing – "sense" or the PS – "scale") and discount factor (1, 0.97 or 0.97^{N+1} for the PS model) specified in the legend of each chart. Each cell gives a comparison of costly and free sensor SP policies in terms of average reward (*Y*-axes) along the increasing number of duplicated sensors (*X*-axis). Policies are generated for the fixed unique combination of model type and discount factor per graph. For example, senseDN097CO corresponds to the cost-free active sensing models with discount factor of 0.97.

Different policies

Coffee extended problem **Coffee problem** **Tiger problem**




Different policies
Average Reward
Number of duplicated sensors
SP policy, AS- disc:1 – costly
SP policy, PS-AS- disc:1 – costly
QMDP policy, AS- disc:1 – costly
Rand policy, AS- disc:1 – costly
Rand policy, AS- disc:1 – costly
SP policy, AS- disc:1 – costly
SP policy, PS-AS- disc:1 – costly
QMDP policy, AS- disc:1 – costly
QMDP policy, AS- disc:1 – costly
SP policy, AS- disc:1 – costly
SP policy, PS-AS- disc:1 – costly
Rand policy, AS- disc:1 – costly
SP policy, PS-disc: 0.97 - free
Rand policy, PS-disc: 0.97 - free
QMDP policy, PS-disc: 0.97 – free
QMDP policy, PS-disc: 0.97 – free
SP policy, PS-disc: 0.97 - free
Rand policy, PS-disc: 0.97 - free
Rand policy, PS-disc:0.97 – free
SP policy, PS-disc: 0.97 – free
QMDP policy, PS-disc: 0.97 – free
QMDP policy, PS-disc: 0.97N – costly
Rand policy, PS-disc: 0.97N – costly
SP policy, PS-disc: 0.97N - costly
Rand policy, PS-disc:0.97N – costly
QMDP policy, PS-disc: 0.97N – costly
SP policy, PS-disc: 0.97N – costly
QMDP policy, PS-disc: 0.97N – costly
Rand policy, PS-disc: 0.97N – costly
SP policy, PS-disc: 0.97N - costly
QMDP policy, PS-disc: 0.97N – costly
Rand policy, PS-disc: 0.97N – costly
SP policy, PS-disc: 0.97N - costly
Rand policy, PS-disc:0.97N – costly
QMDP policy, PS-disc: 0.97N – costly
SP policy, PS-disc: 0.97N – costly
Rand policy, PS-disc:0.97N – costly
SP policy, PS-disc: 0.97N – costly
QMDP policy, PS-disc: 0.97N – costly

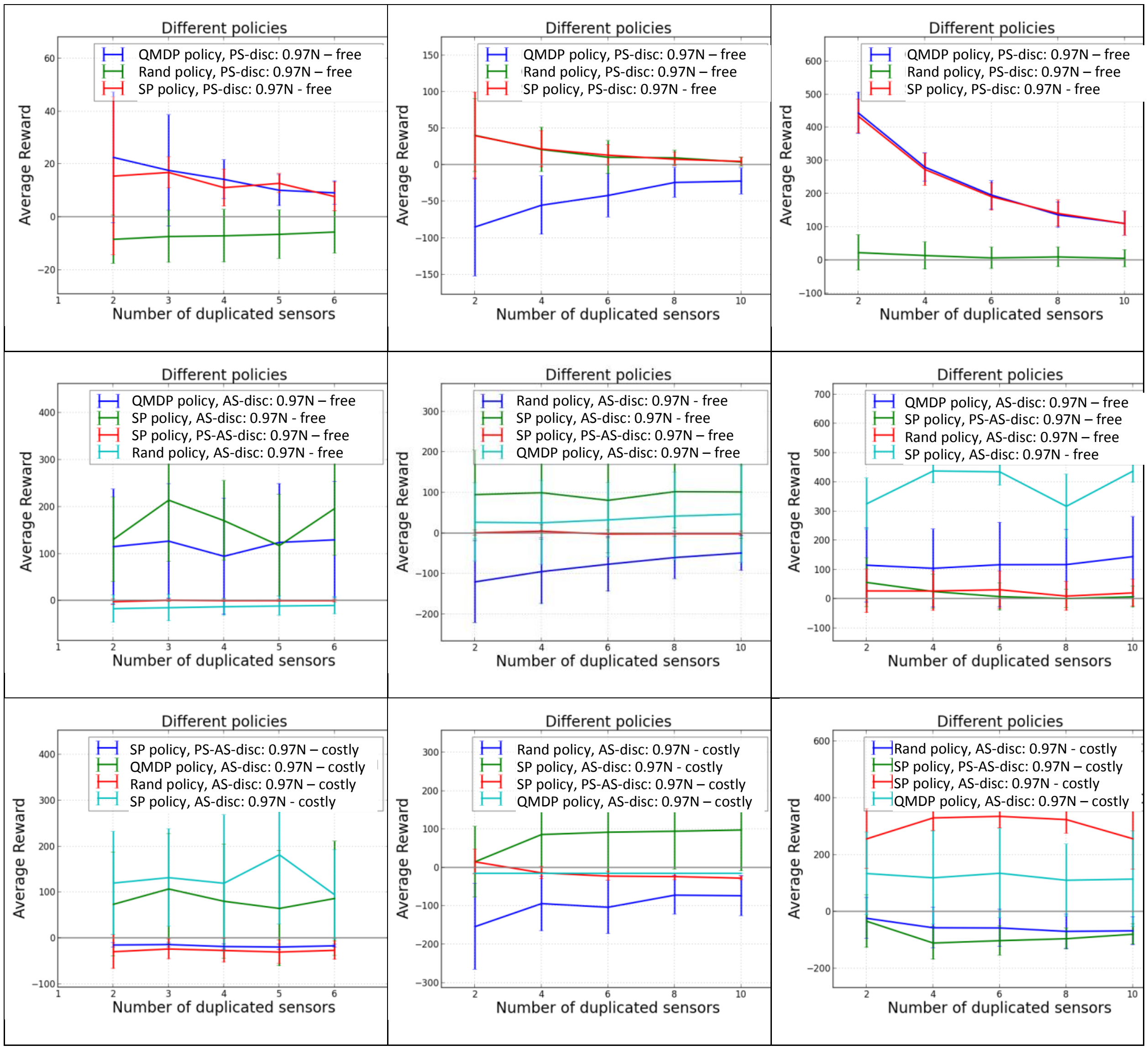
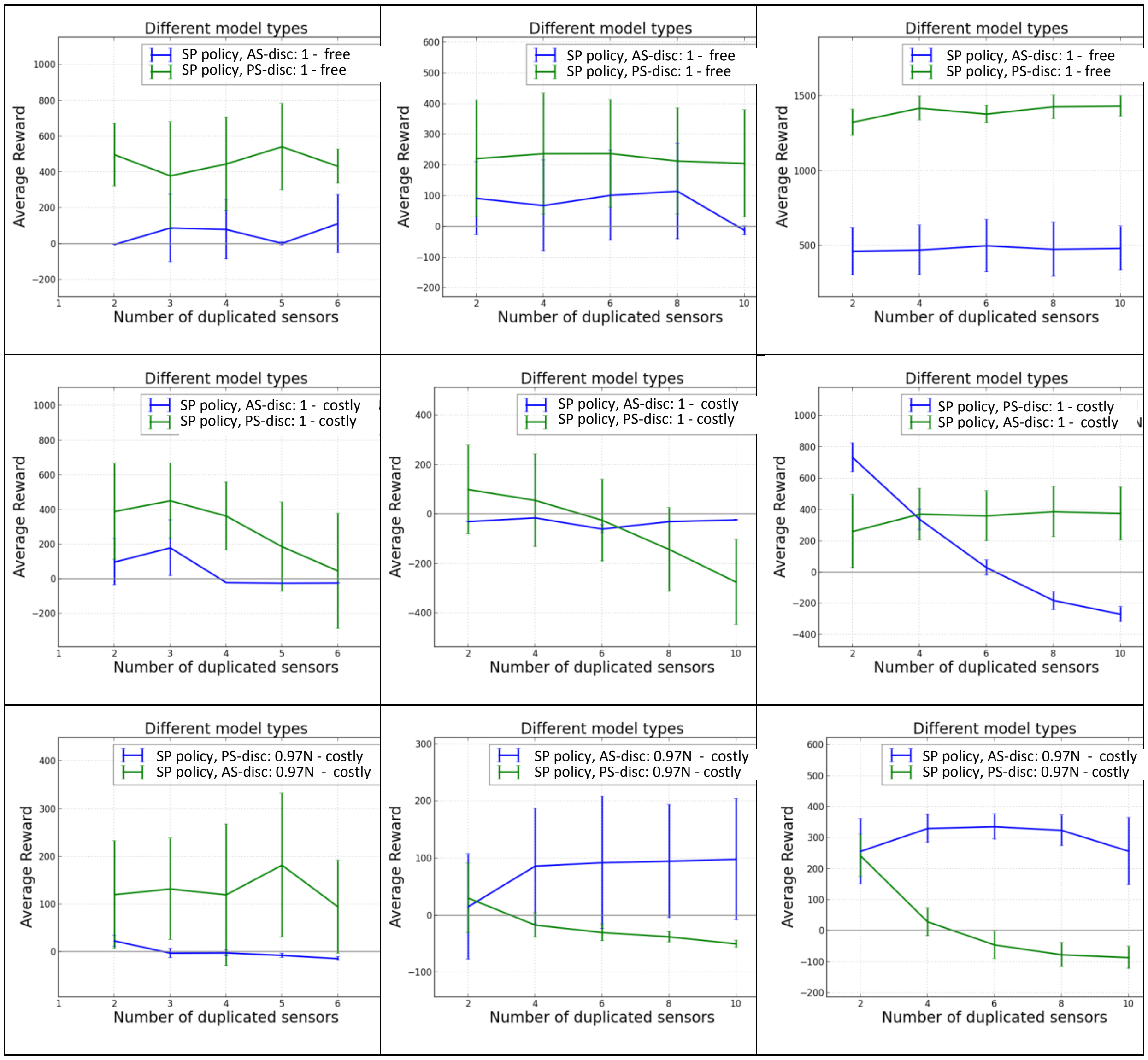


Table C.5. Graphs of comparing SP, QMDP and Random policies for 3 POMDP problems: extended coffee delivery problem (1st column), classic coffee delivery problem (2nd column), extended tiger problem (3rd column). Each row corresponds to the unique combination of model type (AS or PS), discount factor (1, 0.97 or 0.97^{N} for the PS model) and sensor cost (free or costly sensors) parameter settings, specified in the legend of each graph. Each cell graph gives a comparison of different policies in terms of average reward (*Y*-axes) along the increasing number of duplicated sensors (*X*-axis). Policies are generated for the fixed unique combination of the model type, discount factor and sensor cost per graph.

Different models

Coffee extended problem Coffee problem Tiger problem



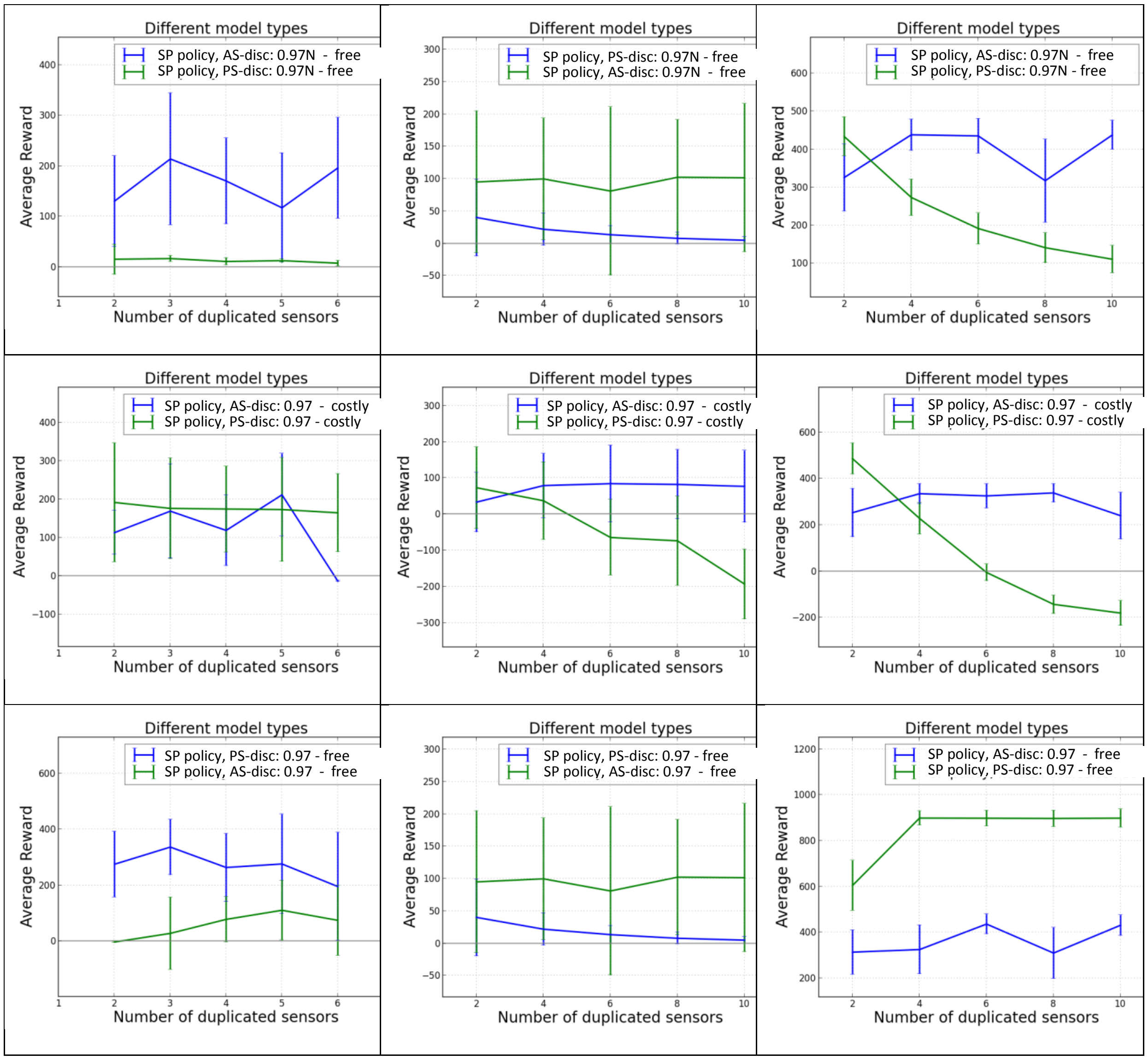


Table C.6. Graphs of comparing active sensing and original SP policies for 3 POMDP problems: extended coffee delivery problem (1st column), classic coffee delivery problem (2nd column), extended tiger problem (3rd column). Each row corresponds to the unique combination of discount factor (1, 0.97 or 0.97*ᴺ* for the original model) and sensor cost (free or costly sensors) parameter settings, specified in the legend of each graph. Each cell graph gives a comparison of SP policies for proposed active sensing and original approaches in terms of average reward (*Y*-axes) along the increasing number of duplicated sensors (*X*-axis). Policies are generated for the fixed unique combination of the discount factor and sensor cost per each graph.

Bibliography

[Aeron08] Aeron S., Saligrama V., Castañon D.A.: Efficient Sensor Management Policies for Distributed Target Tracking in Multihop Sensor Networks. *IEEE Transactions on Signal Processing,* 56(6), pp. 2562-2574 (2008)

[Aviv05] Aviv Y., Pazgal A.: A partially observed Markov decision process for dynamic pricing. *Management Science*, 51(9), pp. 1400–1416, (2005)

[Bahar93] Bahar R.I., Frohm E.A., Gaona C.M., Hachtel G.D., Macii E., Pardo A., Somenzi F.: Algebraic Decision Diagrams and Their Applications. *In Proc. on the Int. Conf. on CAD, IEEE /ACM*, pp. 188-191, (1993)

[Bellman57] Bellman R.: Dynamic programming. *In Princeton University Press*, (1957)

[Bertsekas96] Bertsekas D. P., Tsitsiklis J. N.: Neuro-dynamic programming. *Athena Scientific,* Belmont, MA, (1996)

[Bilgic07] Bilgic M., Voila G. L.: Efficient feature-value acquisition for classification. *In Proc. AAAI*, (2007)

[Boger05] Boger J., Poupart P., Hoey J., Boutilier C., Fernie G., Mihailidis A.: A Decision-Theoretic Approach to Task Assistance for Persons with Dementia. *In Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 1293-1299, (2005)

[Bonet02] Bonet B.: An epsilon-optimal grid-based algorithm for partially observable Markov decision processes. *In Proc. of the Int. Conf. on Machine Learning (ICML),* pp. 51–58, (2002)

[Boutilier96] Boutilier C., Poole D.: Computing optimal policies for partially observable decision processes using compact representations. *In Proc. of the National Conf. on Artificial Intelligence (AAAI),* 13, pp. 1168-1175, (1996)

[Brafman97] Brafman R.I.: A heuristic variable-grid solution method for POMDPs. *In Proc. of the Fourteenth National Conference on Artificial Intelligence*, pp. 727-733, (1997)

[Braziunas04] Braziunas D., Boutilier C.: Stochastic local search for POMDP controllers. In Proc. of the Nineteenth National Conference on Artificial Intelligence, pages 690-696, (2004)

[Bryant86] Bryant, R.E.: Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, *35* (8), pp. 677-691, (1986)

[Cassandra97] Cassandra A.R., Littman M.L., Zhang N.L.: Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. *In Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI),* pp. 54–61, (1997)

[Cassandra98] Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101, pp. 99-134, (1998)

[CassServ98] Cassandra, A.R.: A Survey of Partially-Observable Markov Decision Process Applications. *AAAI Fall Symposium*, (1998) [Castañón97] Castañón D.A.: Approximate dynamic programming for sensor management. *In Proc. of the IEEE Conf. Decision Control*, 2, pp. 1202–1207, (1997)

[Chai] Chai, X., Deng, L., Yang., Q.: Test-cost sensitive naive Bayes classification. *In Proc. of the Int. Conf. on Data Mining (ICDM)*, (2004)

[Chobsri08] Chobsri, S., Usaha, W.**:** A POMDP framework for data acquisition in wireless sensor networks. *The Fifth International Conference in Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON 2008),* (2008)

[Chong09] Chong E.K.P., Kreucher C., Hero A.: Partially observable Markov decision process approximations for adaptive sensing," *Discrete Event Syst.*, 19(3), pp. 377–422, (2009)

[Darrell96] Darrell T., Pentland A.: Active gesture recognition using partially observable Markov decision processes. *In Proc. of the Int. Conf. on Pattern Recognition (ISPR),* 13, (1996)

[Dean95] Dean T., Kaelbling L.P., Kirman J., Nicholson A.: Planning Under Time Constraints in Stochastic Domains. *Artificial Intelligence*, 76(1-2), pp. 35-74, (1995)

[Doshi08] Doshi F., Roy N.: The permutable POMDP: fast solutions to POMDPs for preference elicitation. *In Proc. of the Int. Conf. on Autonomous Agents and Multi Agent Systems (AAMAS),* (2008)

[Ellis95] Ellis J.H., Jiang M., Corotis R.: Inspection, maintenance, and repair with partial observability. *Journal of Infrastructure Systems*, 1(2): pp. 92–99, (1995)

[Evans01] J. S. Evans, V. Krishnamurthy, Optimal sensor scheduling for hidden Markov model state estimation, *Int. J. Contr.*, 74(18), pp. 1737–1742, (2001)

[Fei10] X. Fei, A. Boukerche, F. R. Yu: A POMDP Based K-Coverage Dynamic Scheduling Protocol for Wireless Sensor Networks. *GLOBECOM*, pp. 1-5, (2010)

[Feng01] Feng Z., Hansen E.A.: Approximate planning for factored POMDPs. In Proc. of the European Conf. on Planning (ECP), 6, (2001)

[Feng05] Feng Z., Zilberstein S.: Efficient maximization in solving POMDPs. *In Proc. of the National Conf. on Artificial Intelligence (AAAI),* 20, pp. 975–980, (2005)

[Fikes71] Fikes R.E., Nilsson N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4), pp. 189-208, (1971)

[Frazier10] Frazier P.I., Powell W.B.: Paradoxes in learning and the marginal value of information. *Decision Analysis*, 7(4):378–403, (2010)

[Gosangi10] Gosangi, R., Gutierrez-Osuna, R.: Active temperature programming for metal-oxide chemoresistors. *IEEE Sensors Journal*, 10(3), (2010)

[GOSSIP11] Koltunova, V., Hoey, J., Grzes, M.: Goal-oriented sensor selection for intelligent phones: (GOSSIP). *In Proc. of the SAGAware '11, ACM*, (2011) [Greiner02] Greiner R., Grove A. J., Roth D.: Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139, pp.137–174, (2002)

[Guo03] Guo A.: Decision-theoretic active sensing for autonomous agents. *In Proc. AAMAS*, 2003

[Haddawy98] Haddawy P., Hanks S.: Utility Models for Goal-Directed, Decision-Theoretic Planners. *Computational Intelligence*, 14(3), pp. 392-429, (1998)

[Haight10] Haight R.G., Polasky S.: Optimal control of an invasive species with imperfect information about the level of infestation. *Resource and Energy Economics In Press*, Corrected Proof, (2010)

[Hanselmann08] Hanselmann T., Morelande M., Moran B., Sarunic P.: Sensor scheduling for multiple target tracking and detection using passive measurements. *In Proc. Int. Conf. Inform. Fusion*, pp. 1–8, (2008)

[Hansen00] Hansen E.A., Feng Z.: Dynamic programming for POMDPs using a factored state representation. *In Proc. of the Int. Conf. on AI Planning Systems (ICAIPS),* 5, pp. 130-139, (2000)

[Hansen98] Hansen E.A.: Solving POMDPs by searching in policy space. *In Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI),* pp. 211–219, (1998)

[Hauskrecht97] Hauskrecht M.: Incremental methods for computing bounds in partially observable Markov decision processes. *In proc. of AAAI/IAAI*, pp. 734–739, (1997)

[Hauskrecht00] Hauskrecht M., Fraser H.: Planning treatment of ischemic heart disease with partially observable Markov decision processes. *Artificial Intelligence in Medicine,* 18, pp. 221–244, (2000)

[Hauskrecht00V] Hauskrecht M.: Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 13, pp. 33–94, (2000)

[He04] He Y., Chong E.K.P.: Sensor Scheduling for Target Tracking in Sensor Networks. *In proc. of the 43rd IEEE Conference on Decision and Control, IEEE*, pp.743-748, (2004)

[Hero08] Hero A., Castañón D. A., Cochran D., Kastella K.: Foundations and Applications of Sensor Management. New York: *Springer*, (2008)

[Hero11] Hero A., Cochran D.: Sensor Management: Past, Present, and Future. *IEEE Sensors Journal*, 11(12), (2011)

[Hoey05] Hoey J., Poupart P.: Solving POMDPs with continuous or large discrete observation spaces. *In Proc. of IJCAI,* (2005)

[Hoey07] Hoey J., Little J.J.: Value-directed human behavior analysis from video using partially observable Markov decision processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7), pp.1–15, (2007) [Hoey07A] Hoey J., von Bertoldi A., Poupart P., Mihailidis A.: Assisting Persons with Dementia during Handwashing Using a Partially Observable Markov Decision Process. *In Proc. of the Int. Conf. on Vision Systems (ICVS),* (2007)

[Hoey08] Mihailidis A., Boger J.N., Candido M., Hoey J.: The COACH prompting system to assist older adults with dementia through handwashing: An efficacy study. *BMC Geriatrics*, 8(28), (2008)

[Hoey10] Hoey J., Poupart P., von Bertoldi A., Craig T., Boutilier C., Mihailidis A.: Automated Handwashing Assistance for Persons with Dementia Using Video and a Partially Observable Markov Decision Process. *Computer Vision and Image Understanding (CVIU),* 114(5), pp. 503-519, (2010)

[Hoey11] Hoey, J., Plotz, T., Jackson, D., Monk, A., Pham, C., Olivier, P.: Rapid specification and automated generation of prompting systems to assist people with dementia. *Pervasive and Mobile Computing,* 7(3), (2011)

[Hoey99] Hoey J., St-Aubin R., Hu A., Boutilier C.: SPUDD: Stochastic planning using decision diagrams. *In Proc. of the Conf. on Uncertainty in Artificial Intelligence (CUAI),* 15, pp. 279-288, (1999)

[Horvitz88] Horvitz E.J., Breese J.S., Henrion M.: Decision theory in expert systems and artificial intelligence. *Int. Journal of Approximate Reasoning*, 2, pp. 247-302, (1988)

[Howard60] Howard R.A.: Dynamic Programming and Markov Processes. *MIT Press*, Cambridge, Massachusetts, (1960)

[Hsiao07] Hsiao K., Kaelbling L., Lozano-Perez T.: Grasping pomdps. *In Proc. of the Int. Conf. on Robotics and Automation (ICRA), IEEE,* pp. 4685–4692, (2007)

[Hu96] Hu C., Lovejoy W.S., Shafer S.L.: Comparison of some suboptimal control policies in medical drug therapy. *Operations Research*, 44(5), pp. 696–709, (1996)

[Ji] Ji, S., Carin, L.: Cost-sensitive feature acquisition and classification. *Pattern Recognition*, 40, pp. 1474–1485, (2007)

[Ji07] Ji S., Parr R., Carin L.: Nonmyopic multiaspect sensing with partially observable Markov decision processes. *IEEE Trans. Signal Process.*, 55(6), pp. 2720–2730, 2007

[Kanani10] Kanani P., McCallum A., Hu S.: Resource bounded information extraction: Acquiring missing feature values on demand. *In Proc. PAKDD,* 2010

[Kaplow10] Kaplow R.: Point-based POMDP Solvers: Survey and Comparative Analysis, Master's Thesis, McGill University, 2010

[Kochenderfer08] Kochenderfer M. J., Espindle L. P., Kuchar J. K., Griffith J. D.: A comprehensive aircraft encounter model of the national airspace system. *Lincoln Lab. J.*, 17(2), pp. 41–53, (2008)

[Krause05] Krause A., Ihmig M., Rankin E., Leong D., Gupta S., Siewiorek D.P., Smailagic A., Deisher M., Sengupta U.: Trading off prediction accuracy and power consumption for context-aware wearable computing. *In Proc. of the 9th International Symposium on Wearable Computers*, IEEE Computer Society Press, pp. 20–26, (2005)

[Krause09] Krause, A., Guestrin, C.: Optimal value of information in graphical models. *Journal of Artificial Intelligent Research (JAIR),* 35, pp.557–591, (2009)

[Krishnamurthy02] V. Krishnamurthy, Algorithms for optimal scheduling and management of hidden Markov model sensors, *IEEE Trans. Signal Process.*, 50(6), pp. 1382–1397, (2002)

[Krishnamurthy07] Krishnamurthy V., Djonin D. V.: Structured threshold policies for dynamic sensor scheduling - a partially observed markov decision process approach. *IEEE Transactions on Signal Processing,* 55(10), pp. 4938–4957, (2007)

[Krishnamurthy09] Krishnamurthy V., Djonin D.V.: Optimal threshold policies for multivariate POMDPs in radar resource management. *IEEE Trans. on Signal Processing*, 57(10), 3954–3969, (2009)

[Kurniawati08] Kurniawati H., Hsu D., Lee W.: SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. *In Proc. of the Robotics: Science and Systems (RSS),* (2008)

[Kwok] Kwok, C., Fox, D.: Reinforcement Learning for Sensing Strategies. *In Proc. of the Intelligent Robots and Systems (IROS),* (2004)

[LaCasa] Hoey, J., Yang, X., Favela, J.: Decision theoretic, context aware safety assistance for persons who wander. *Int. Workshop on Ubiquitous Health and Wellness (UbiHealth 2012),* (2012)

[Li09] Li Y., Krakow L.W., Chong E.K.P., Groom K.N.: Approximate stochastic dynamic programming for sensor scheduling to track multiple targets. *Digital Signal Process.*, 19(6), pp. 978–989, (2009)

[Liao09] Liao W., William Q. Ji, Wallace A.: Approximate Nonmyopic Sensor Selection via Submodularity and Partitioning. *IEEE Transactions on Systems, Man, and Cybernetics, Part A,* 39(4), 782-794 (2009)

[Lin92] Lin L.-J., Mitchell T. M.: Memory approaches to reinforcement learning in non-Markovian domains. *Technical Report CMU-CS-92-138,* (1992)

[Littman95] Littman M.L., Cassandra A.R., Kaelbling L.P.: Learning policies for partially observable environments: Scaling up. *In Proc. of the Int. Conf. on Machine Learning (ICML),* pp. 362–370, (1995)

[Littman96] Littman M.L.: Algorithms for Sequential Decision Making. PhD thesis, Department of Computer Science, Brown University, (1996)

[Liu07] Liu J., Chu M., Reich J. E.: Multitarget tracking in distributed sensor networks. *IEEE Signal Process. Mag.*, 24(3), pp. 36–46, 2007

[LoPresti04] LoPresti, E.F., Mihailidis, A., Kirsch, N.: Assistive technology for cognitive rehabilitation: State of the art. *Neur. Rehab.,* 14(1/2), pp. 5–39, (2004) [Lovejoy91] Lovejoy, W.S.: A survey of algorithmic methods for partially observed Markov decision processes. *Annals of Operations Research,* 28(1), pp. 47-65, (1991)

[Lovejoy91Gb] Lovejoy W. S.: Computationally feasible bounds for partially observed Markov decision processes. *Operations Research*, 39, pp. 162-175, (1991)

[Lovejoy93] Lovejoy, W.S.: Suboptimal Policies with Bounds for Parameter Adaptive Decision Processes. *Operations Research*, 41, pp. 583-599, (1993)

[Madani] Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and infinite horizon partially observable decision problems. *In Proc. of the National Conf. on Artificial Intelligence (AAAI),* 16, pp. 541–548, (1999)

[Malhotra97] Malhotra R., Blasch E.P., Johnson J. D.: Learning sensor-detection policies. In *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, 2, pp. 769–776, (1997)

[Meguro11] Meguro T., Minami Y., Higashinaka R., Dohsaka K.: Wizard of Oz Evaluation of Listening-oriented Dialogue Control using POMDP. *In Proc. of the Automatic Speech Recognition and Understanding (ASRU), IEEE,* (2011)

[Meuleau99] Meuleau, Kim K.-E., Kaelbling L., Cassandra A. R.: Solving POMDPs by searching the space of finite policies. *In Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI),* pp. 417–426, (1999)

[Monahan82] Monahan G.E.: A survey of partially observable Markov decision processes: Theory, models and algorithms. *Manage. Sci.,* vol. 28(1), pp. 1–16, (1982)

[Murphy00] Murphy K.: A Survey of POMDP Solution Techniques. *Comp. Sci. Div.,* UC Berkeley, Tech. Rep., (2000)

[Musick94] S. Musick, R. Malhotra, Chasing the elusive sensor manager. In *Proc. IEEE Nat. Aerosp. Electron. Conf. (NAECON)*, pp. 606–613, (1994)

[Nourbakhsh95] Nourbakhsh I.R., Powers R., Birchfield S.: DERVISH - an office-navigating robot. *AI Magazine*, 16(2), pp. 53–60, (1995)

[Papadimitriou] Papadimitriou, C.H., Tsitsiklis, J.N.: The complexity of Markov decision processes. *Mathematics of Operations Research,* 12(3), pp. 441–450, (1987)

[Pham] Pham, C., Olivier, P.: Slice&dice: Recognizing food preparation activities using embedded accelerometers. *In Proc. of the European Conf. on Artificial Intelligence (ECAI),* (2009)

[Pineau03] Pineau J., Montemerlo M., Pollack M., Roy N., Thrun S.: Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems*, 42(3–4), pp. 271–281, (2003)

[Pineau03P] Pineau J., Gordon G., Thrun S.: Point-based value iteration: An anytime algorithm for POMDPs. *In Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI),* pp. 1025–1032, (2003) [Pineau04] Pineau J.: Tractable Planning Under Uncertainty: Exploiting Structure. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburg, 2004.

[Pineau05] Pineau, J., Gordon, G.: POMDP planning for robust robot control. *In Proc. of the Int. Symposium on Robotics Research (ISRR)*, 28, pp. 69–82, (2005)

[Poupart05] Poupart P.: Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes. PhD thesis, University of Toronto, (2005)

[Poupart11] Poupart P., Kim K.E., Kim D.: Closing the gap: Improved bounds on optimal POMDP solutions. *In: International Conference on Planning and Scheduling* (ICAPS) (2011)

[Puterman05] Puterman, M. L.: Markov Decision Processes, J. Wiley and Sons, New York, NY, (2005)

[Regan05] Regan, K., Cohen, R., Poupart, P.: The advisor-POMDP: A principled approach to trust through reputation in electronic markets. *In Proc. of the Third Annual Conf. on Privacy, Security and Trust (PST),* pp.121-130, (2005)

[Ross08] Ross S., Chaib-draa B., Pineau J.: Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. *In Proc. of the Int. Conf. on Robotics & Automation (ICRA), IEEE,* pp. 2845–2851, (2008)

[Roy00] Roy N., Pineau J., Thrun S.: Spoken dialogue management using probabilistic reasoning. *In Proc. of the 38th Annual Meeting of the Association for Computational Linguistics (ACL2000),* (2000)

[Rusmevichientong01] Rusmevichientong P., Van Roy B.: A tractable POMDP for a class of sequencing problems. *In Proc. of the Uncertainty in Artificial Intelligence (UAI),* (2001)

[Shani07] Shani G., Brafman R., Shimony S.: Forward search value iteration for POMDPs. *In Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI),* (2007)

[Shani07P] Shani G.: Learning and Solving Partially Observable Markov Decision Processes. PhD thesis, Ben-Gurion University of the Negev, (2007)

[Shani08] Shani, G., Poupart, P., Brafman, R.I., Shimony, S. E.: Efficient ADD operations for point-based algorithms. *In Proc. of the Int. Conf. on Automated Scheduling and Planning (ICAPS),* pp. 330–337, (2008)

[Shani12] Shani G., Pineau J., Kaplow R.: A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent Systems,* (2012)

[Shatkay] Shatkay, H., Kaelbling, L.P.: Learning topological maps with weak local odometric information. *In Proc. of the Fifteenth Int. Joint Conf. on Artificial Intelligence (IJCAI),* (1997)

[Sim08] Sim, H.S., Kim, K.E., Kim, J.H., Chang, D.S., Koo, M.W.: Symbolic heuristic search value iteration for factored POMDPs. *In Proc. of the National Conf. on Artificial Intelligence (AAAI),* pp. 1088–1093, (2008) [Simmons98] Simmons, R., Koenig, S.: Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models. *In Artificial Intelligence Based Mobile Robotics*: Case Studies of Successful Robot Systems, D. Kortenkamp, R. Bonasso and R. Murphy (editor), pp. 91-122, (1998)

[Smith04] Smith T., Simmons R.: Heuristic search value iteration for POMDPs. *In Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI),* (2004)

[Sondik71] Sondik, E.J.: The Optimal Control of Partially Observable Markov Decision Processes. PhD thesis, Stanford University, (1971)

[Sondik73] Smallwood, R.D., Sondik, E.J.: The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21, pp. 1071–1088, (1973)

[Spaan04] Spaan, M.T.J., Vlassis, N.: A point-based POMDP algorithm for robot planning. *In Proc. of the Int. Conf. on Robotics and Automation (ICRA), IEEE,* (2004)

[Spaan05] Spaan, M.T.J., Vlassis, N.: Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24(1), pp. 195-220, (2005)

[Spaan09] Spaan M.T.J., Lima P.U.: A decision-theoretic approach to dynamic sensor selection in camera networks. *In Proc. of the ICAPS,* pp. 279--304, (2009)

[Spaan10] Spaan M.T.J., Veiga T.S., Lima P.U.: Active cooperative perception in network robot systems using POMDPs. *In Proc. of the Int. Conf. on Intelligent Robots and Systems (ICIRS),* (2010)

[Spaan12] Spaan M.T.J.: Partially Observable Markov Decision Processes. *To appear in M. A. Wiering and M. van Otterlo, editors, Reinforcement Learning: State of the Art,* Springer Verlag, (2012)

[Sridharan10] Sridharan M., Wyatt J., Dearden R.: Planning to see: A hierarchical approach to planning visual actions on a robot using POMDPs. *Art. Intell. J.,* 174(11), (2010)

[Stankiewicz07] Stankiewicz B., Cassandra A., McCabe M., Weathers W.: Development and evaluation of a Bayesian low-vision navigation aid. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions,* 37(6), pp. 970–983, (2007)

[St-Aubin00] St-Aubin, R., Hoey, J., Boutilier, C.: APRICODD: Approximate policy construction using decision diagrams. *In Advances in Neural Information Proc. Systems (NIPS),* pp. 1089-1095, (2000)

[Theocharous02] Theocharous G., Mahadevan S.: Approximate planning with hierarchical partially observable Markov decision processes for robot navigation. *In Proc. of the Int. Conf. on Robotics and Automation (ICRA), IEEE,* (2002)

[Treharne02] Treharne, J.T., Sox C.R.: Adaptive Inventory Control for Non-Stationary Random Demand. *Management Science,* 48(5), pp. 607-624, (2002)

[Turney00] Turney P.: Types of cost in inductive concept learning. *In Workshop on Cost-Sensitive Learning,* (2000)

[Wang] Wang, Y., Krishnamachari, B., Zhao, Q., Annavaram, M.: Markov-optimal sensing policy for user state estimation in mobile devices. *In Proc. of the Int. Conf. on Information Proc. in Sensor Networks, ACM/IEEE*, pp. 268-278, (2010)

[Wang10] Wang, C., Khardon, R.: Relational partially observable mdps. *In Proc. of the National Conf. on Artificial Intelligence (AAAI)*, (2010)

[White91] White C.C.: A survey of solution techniques for the partially observed Markov decision process. *Annals of Operations Research*, 32, pp. 215–230, (1991)

[Williams07] Williams, J.D., Young, S.: Partially observable Markov decision processes for spoken dialog systems. *Computer Speech & Language*, 21(2), pp. 393–422, (2007)

[Williams92] Williams R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, pp. 229–256, (1992)

[Zhang01] Zhang N.L., Zhang W.: Speeding up the convergence of value iteration in partially observable Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 14, pp. 29–51, (2001)

[Zhang01] Zhang W.: Algorithms for Partially Observable Markov Decision Processes, Phd thesis, Hong Kong University of Science and Technology, Hong Kong (2001)

[Zhang96] Zhang N.L., Liu W.: Planning in stochastic domains: Problem characteristics and approximation. *Technical Report HKUST-CS96-31*, Department of Computer Science, Hong Kong University of Science and Technology, (1996)

[Zhou01] Zhou R., Hansen A.: An improved grid-based approximation algorithm for POMDPs. *In Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, pp. 707–716, (2001)

[Zilberstein02] Zilberstein, S., Washington, R., Bernstein, D.S., Mouaddib, A.: Decision theoretic control of planetary rovers. *In M. Beetz, et al., (Eds.), Lecture Notes in Artificial Intelligence (LNAI)*, 2466, pp. 270-289, (2002)

[SPJ link] Java Symbolic Perseus package by J. Hoey
https://cs.uwaterloo.ca/~jhoey/research/spudd/symbolicPerseusJava.tar.gz