Master in Artificial Intelligence (UPC-URV-UB)

Master of Science Thesis

A POMDP APPROACH TO THE HIDE AND SEEK GAME

Chryso Georgaraki

Advisor: Dr. René Alquézar Mancho

January 13, 2012

ABSTRACT

Partially observable Markov decision processes (POMDPs) provide an elegant mathematical framework for modeling complex decision and planning problems in uncertain and dynamic environments. They have been successfully applied to various robotic tasks. The modeling advantage of POMDPs, however, comes at a price exact methods for solving them are computationally very expensive and thus applicable in practice only to simple problems. A major challenge is to scale up POMDP algorithms for more complex robotic systems. Our goal is to make an autonomous mobile robot to learn and play the children's game hide and seek with opponent a human agent. Motion planning in uncertain and dynamic environments is an essential capability for autonomous robots. We focus on an efficient point-based POMDP algorithm, SARSOP, that exploits the notion of optimally reachable belief spaces to improve computational efficiency. Moreover we explore the mixed observability MDPs (MOMDPs) model, a special class of POMDPs. Robotic systems often have mixed observability: even when a robots state is not fully observable, some components of the state may still be fully observable. Exploiting this, we use the factored model, proposed in the literature, to represent separately the fully and partially observable components of a robots state and derive a compact lower dimensional representation of its belief space. We then use this factored representation in conjunction with the point-based algorithm to compute approximate POMDP solutions. Experiments show that on our problem, the

new algorithm is many times faster than a leading point-based POMDP algorithm without important losses in the quality of the solution.

Keywords: POMDP, MOMDP, robot, hide and seek, artificial intelligence, uncertainty, planning, SARSOP, approximate, policy.

ACKNOWLEDGMENTS

There are many people I need to thank for all the support and guidance I have been given for this thesis.

First and foremost, I would like to thank my supervisor, Dr. Renè Alquèzar, who I've had the pleasure of meeting from my first day in UPC. Throughout the last years, he provided encouragement, sound advice, good teaching, good company, and lots of good ideas. I am really grateful that I had the opportunity to work with him these past months. With his enthousiasm, his inspiration, and his continuous help and guidance made my work easy and mostly fun. His contribution was invaluable. His guidance helped me in all the time of my studies and completing this thesis. I could not have imagined having a better advisor and mentor for my Master study.

I would also like to express my sincere gratitude to the PhD candidate, Alex Goldhoorn, for his endless help, constructive comments and for his important support throughout this work. With out him, none of this would have been possible.

My deepest gratitude goes to my family for their unflagging love and support throughout my life; This thesis is simply impossible without them. My parents, Giorgos and Maria Georgaraki, made everything possible for me with their continuous support in every level, unconditional love and patience. My brother, Dr. Konstantinos Georgarakis and my sister Natasa Georgaraki, who have always been beside me, supporting and encouraging me in every step in my life.

I also would like to thank my friends for their support, their faith in me and their valuable help in the experimental phase of this work. Special thanks to Alex Rodriguez for all his great help and patience during this Master Thesis.

I have had a wonderful time working at the Institute of Robotics and Industrial Informatics (IRI) these past months, and I would like to thank the Director Alberto Sanfeliu as well as all the people in the lab of Mobile Robotics.

Contents

Page

Page

List of Tables

List of Figures

Figure **Page**

Chapter 1

Introduction

One way in which animals and artificial systems acquire complex behaviors is by learning to obtain rewards and to avoid punishments. Reinforcement learning theory is a formal computational model of this type of learning. RL is a popular branch of artificial Intelligence and it addresses many objectives with major economic impact like marketing, power plant control, bio-reactors, Vehicle Routing, flying control, finance, computer games and robotics.

RL theory comprises a formal framework for describing an agent interacting with its environment, together with a family of learning algorithms, and analyses of their performance. In the standard framework for RL, a learning agent-an animal or perhaps a robot-repeatedly observes the state of its environment, and then chooses and performs an action. Performing the action changes the state of the world, and the agent also obtains an immediate numeric payoff as a result. Positive payoffs are rewards and negative payoffs are punishments. The agent must learn to choose actions so as to maximize a long-term sum or average of the future payoffs it will receive.

The agents payoffs are subjective; in the sense that they are the agents own evaluation of its experience. The ability to evaluate its experience in this way is an essential part of the agents prior knowledge. The payoffs define what the agent is

trying to achieve; what the agent needs to learn is how to choose actions to obtain high payoffs. For this framework to be sensible, the agent must be able to visit the same or at least similar states on many occasions, to take advantage of learning.

A standard assumption is that the agent is able to observe those aspects of the state of the world that are relevant to deciding what to do. This assumption is convenient rather than plausible, since it leads to great simplification of the RL problem.

In real-world scenarios when the agent is a robot not only we have to deal with the limited observability of the agent due to physical obstacles but also with the uncertainty in the robot motion and the sensor observations. Robotic planning becomes even harder when different parts of the environment appear similar to the sensor system of the robot. Such "perceptual aliasing" is common in office environments where robots are often employed. In these partially observable domains a robot needs to explicitly reason with uncertainty in order to successfully carry out a given task, e.g. navigating through an office to deliver mail.

Partially Observable Markovian Decision Processes (POMDPs) are used to model intelligent agents in uncertain and dynamic environments. The agent observes its environment, develops a belief state (a probabilistic state estimate) and chooses an action to maximize the expected future reward. POMDPs are powerful because all environments are uncertain to varying degrees. Consequently, POMDPs more closely model reality. They have been successfully applied to various robotic tasks, but unfortunately, creating a policy is currently intractable because the number of belief states grows with the number of actual states and the number of actions taken. Intuitively speaking, looking one time step deeper into the future requires considering each possible action and each possible observation. As such, the cost of computing the solution for a POMDP grows exponentially

with the desired planning horizon.

A major challenge is to scale up POMDP algorithms and ease the computational complexity so that it would be adequate for more complex robotic systems.

Robotic systems often have mixed observability: even when a robot's state is not fully observable, some components of the state may still be fully observable. This is exploited by models with mixed observability, MOMDPs, that use a factored model to represent separately the fully and partially observable components of a robot's state and derive a compact lower-dimensional representation of its belief space. Separating fully and partially observable state components using a factored model opens up several opportunities to improve the efficiency of POMDP algorithms.

The intention of this work is to have an autonomous mobile robot playing the hide and seek game with an opponent human agent. To achieve that we define the problem as a POMDP model and we attain an approximate optimal policy for the robot, using some state-of-the-art solvers. We also investigate the adequacy of MOMDP model for this game and compare to the model learned using POMDPs.

1.1 Motivation

The idea of an autonomous mobile robot interacting with human(s) by playing the hide and seek game, was born at the Institute of Robotics in Barcelona (IRI - Insitut de Robòtica & Informàtica Industrial). It was proposed as a new Experiment on the CEEDS project [1]. The Collective Experience of Empathic Data Systems (CEEDS) project advances a novel integrated technology that supports the experiencing, analyzing and understanding of massive datasets. The purpose of the experiment proposed is to develop a testbed for integration of CEEDS modules. The experiment will make use of the Barcelona Robot Lab infrastructure that includes mobile robots in an urban pedestrian setting and a distributed cam-

era network. The aim of the experiment is to enable a human to learn and discover human-robot interaction strategies through his/her embodiment into a robot using the CXIM machine. As a sample application, a hide-and-seek situation will be solved, in which other humans hide from the robot at the experimental site, and the user in the CXIM must find them. The game poses relevant challenges from the perspective of human-robot interaction such as people detection, recognition and tracking of human activity, prediction and anticipation to human behavior. With respect to the CEEDs paradigm, the experiment poses relevant challenges for real-time data representation through a limited communications channel, and for learning and categorization of human behavior from multi-sensory data patterns.

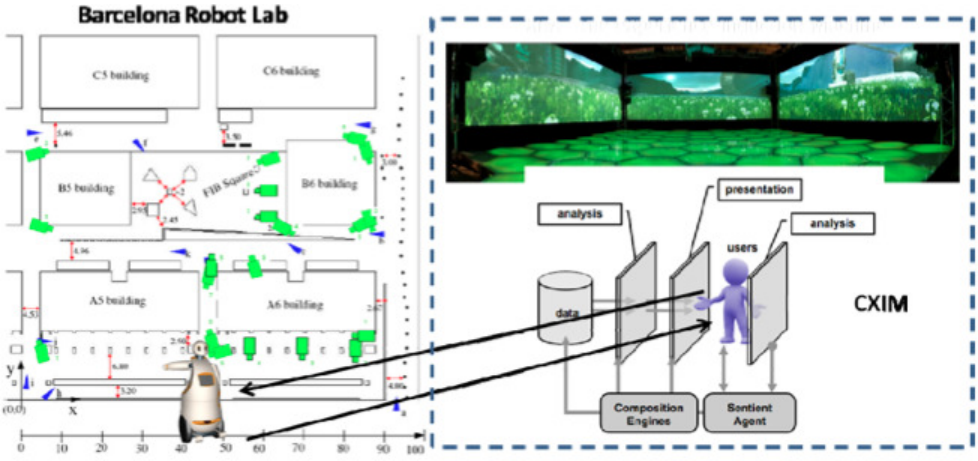


Figure 1: The hide-and-seek experiment. Map of the enviroment - Barcelona Robot Lab and the tele-operation centre - CXIM.

In this work, we aim to have an automated seeker playing the hide-and-seek game, as good as a human would play. Our objective is to calculate, at the end, an approximate optimal policy for the role of the seeker. In order to do that we focused in defining, learning and testing POMDP (more precisely, MOMDP) models for the hide-and-seek game. The game, in this thesis, will be played in simulation on a grid of n x m cells, including some obstacles that limit observability and mobility,

of two agents: a seeker (robot) and a hider (human). Even though a good policy for the hider could also be computed using POMDPs, we will focus on computing good policies for the seeker.

A simulation tool for the hide and seek game will be designed and implemented in order to estimate possible behaviours of the human agent from experience (by playing the game in the simulator) and to assess the quality of the learned robot policies. To this end, a graphical user interface will be designed using the Qt graphics library for Linux.

1.2 State of the Art

As said before, a popular approach to artificial intelligence is to model an agent and its interaction with its environment through actions, perceptions, and rewards. Intelligent agents should choose actions after every perception, such that their long-term reward is maximized. A well-defined framework for this interaction is the partially observable Markov decision process (POMDP) model. Unfortunately solving POMDPs is an intractable problem [2] mainly due to the fact that exact solutions rely on computing a policy over the entire belief-space, which is a simplex of dimension equal to the number of states in the underlying Markov decision process (MDP).

Markov decision process models (MDPs) have proven to be useful in a variety of sequential planning applications where it is crucial to account for uncertainty in the process [3]. The success of the application of MDP can be attributed to the existence of efficient algorithms for finding optimal solutions for MDP models [3].

The partially observable MDP model (POMDP) generalizes the MDP model to allow for even more forms of uncertainty to be accounted for in the process. However, POMDPs are inherently hard to solve using value iteration given the two key challenges: the curse of dimensionality [4], whereby planners must handle large

state spaces, and the curse of history [5] whereby the number of possible histories to be considered grows exponentially with the planning horizon. For these reasons, researchers have explored different ways of refining the value iteration algorithm. Numerous authors have proposed exact methods with various pruning strategies ([6], [4]).

However, given the limited scalability of these exact algorithms, the focus has shifted toward approximate solutions. One set of approximations relies on point-based value iteration to build a fixed-size value function, ([7], [8], [5]) and is typically executed offline. Another set of approximations relies on tree search to construct a local policy for a single belief state, and is typically executed online.

Offline algorithms take a POMDP problem as input and compute a value function from which an associated policy is extracted before the agent is deployed in the environment. These algorithms have to face the difficulty of planning for all possible situations a priori, only knowing the initial belief state at which the agent will be deployed. On the other hand, online algorithms take a POMDP specification and the agents current belief state as input and compute a single action while the agent is in the environment. The advantage here is that the agent knows what belief state it is currently in and can therefore plan for immediate contingencies. However, online planning is generally required to meet real-time constraints and this can be difficult for large POMDPs.

Next we overview some of the most important point-based methods that were proposed in the past years and intend to overcome the complexity challenges by careful selection of belief points, by choosing points that will support a more accurate value function for the POMDP. These methods also attempt to generalize from these individual points, by using the methods developed for exact POMDP planning to give good value function approximations for points in the belief space

that were not chosen.

Theocharous and Kaelbling [9] explore the idea of taking advantage of the fact that for most POMDP problems, a large proportion of the belief space is not experienced, but in combination with the notion of temporally extended actions (macro-actions). They propose and investigate a model-based reinforcement learning algorithm over grid-points in belief space, which uses macro-actions and Monte Carlo updates of the Q-values. The algorithm was applied to large scale robot navigation and demonstrates that with macro-actions an agent experiences a significantly smaller part of the belief space than with simple primitive actions. In addition, learning is faster because an agent can look further into the future and propagate values of belief points faster. Finally, well designed macros, such as macros that can easily take an agent from a high entropy belief state to a low entropy belief state (e.g. go down the corridor, which will reach a junction where the agent can localize), enable agents to perform information gathering.

Point-based planners draw from the grid-based methods in how they approximate via choosing only a subset of belief nodes over which to iterate. These planners are also influenced by the heuristic-based methods. There have been a variety of point-based POMDP planners proposed in the literature. These planners differ in several places, including the methods for collecting points, updating points, and other optimizations. Some state-of-the-art solvers include Perseus, HSVI, FSVI, SARSOP, etc.

Spaan and Vlassis presented the Perseus algorithm in 2004 [8]. Perseus introduces a method to reduce the number of belief updates required at each iteration, while still improving the value function for a large number of beliefs. With less belief point updates per step, there is room for many more iterations of belief point updates, for a given time constraint. Perseus is a randomized approximate value

iteration algorithm, to planning in a robotic context, which features large state spaces and high dimensional observations. Experimental results indicate that the algorithm can successfully handle large POMDPs, and is very competitive to other algorithms both in terms of solution quality as well as speed; in several benchmark problems it is at least one order of magnitude faster than other state-of-the-art algorithms.

Point-based Value Iteration, PBVI, published in 2003 [5], was an early point-based approximate solver. Pineau et al. introduced the notion of only applying the backup operation on a finite subset $B \subset S$ where all $b \in B$ are reachable from the initial belief b_0. Building on this insight, they presented empirical results demonstrating the successful performance of the algorithm on a large (870states) robot domain called Tag, inspired by the game of lasertag. PBVI selects a small set of representative belief points and iteratively applies value updates to those points. The point-based update is significantly more efficient than an exact update (quadratic vs. exponential), and because it updates both value and value gradient, it generalizes better to unexplored beliefs than interpolation-type grid-based approximations which only update the value. In addition, exploiting an insight from policy search methods and MDP exploration, PBVI uses explorative stochastic trajectories to select belief points, thus reducing the number of belief points necessary to find a good solution. Finally, the theoretical analysis of PBVI shows that it is guaranteed to have bounded error.

The HSVI, Heuristic Search Value Iteration, was published in 2004 [10] and was updated in 2005 [11]. HSVI returns a policy and a provable bound on its regret with respect to the optimal policy. HSVI gets its power by combining two well-known techniques: attention-focusing search heuristics and piecewise linear convex representations of the value function. Its soundness and convergence have

been proven. HSVI uses heuristics based on upper and lower bounds of the value function to collect beliefs. The algorithm was introduced as an improvement over the state of the art in POMDP solving, as using the heuristics to guide the search in belief space would lead to critical beliefs faster than PBVI. HSVI maintains both a lower bound ($\underline{V}$) and an upper bound ($\overline{V}$) over the optimal value function. It traverses the belief space using a heuristic based on both bounds and performs backups over the observed belief points in reversed order. Unfortunately, using $\overline{V}$ slows down the algorithm considerably as updating $\overline{V}$ and computing upper bound projections ($\overline{V}(b)$) are computationally intensive. However, this heuristic pays off, as the overall performance of HSVI is better than other methods.

Forward Search Value Iteration (FSVI) was introduced in 2007 [12] . This point-based algorithm suggests a method for belief point selection, and for ordering backups which does not use an upper bound. Rather, it uses a search heuristic based on the underlying MDP. Using MDP information is a well known technique, especially as an initial guess for the value function (starting with the Q_{MDP} method of [13]), but did not lead so far to very successful algorithms. This algorithm, Forward Search Value Iteration (FSVI), traverses together both the belief space and the underlying MDP space. Actions are selected based on the optimal policy for the MDP, and are applied to the belief space as well, thus directing the simulation towards the rewards of the domain. In this sense, this approach strongly utilizes the underlying MDP policy. This algorithm was tested on all the domains on which HSVI was tested, and it converges faster than HSVI and scales up better, allowing the solution of certain problems for which HSVI fails to converge within reasonable time.

SARSOP [14] was introduced in 2008 as an improvement over HSVI, by modifying the sampling approach to sample near $R^*(b_0)$, the subset of belief points

reachable from b_0 which are explored under optimal action selection. Of course, knowing $R^*(b_0)$ exactly is impossible, since it requires the exact POMDP solution. The algorithm iteratively approximates $R^*(b_0)$ by first using the current estimate to update the value function bounds, $\underline{V}(b)$ and $\overline{V}(b)$, and then using the updated bounds to recompute $R^*(b_0)$. The SARSOP sampling method is similar to the one proposed in HSVI, but adds the notion of selective deep sampling. In the next chapter we will present in more detail the SARSOP algorithm since this is the one we use to complete our task.

Chapter 2

Theoretical Background

2.1 POMDP

A POMDP models an agent taking a sequence of actions under uncertainty to maximize its total reward. Formally a discrete state, infinite-horizon, discounted POMDP is specified as a tuple $(S, A, O, T, Z, R, \gamma)$, where S is a set of states, A is a set of actions, and O is a set of observations.

In each time step, the agent takes an action $a \in A$ and moves from a state $s \in S$ to $s' \in S$. Owing to the uncertainty in action, the end state s' is described as a conditional probability function $T(s, a, s') = \Pr(s' \mid s, a)$, which gives the probability that the agent lies in s', after taking action a in state s. The agent then makes an observation on the end state s'. Owing to the uncertainty in observation, the observation result $o \in O$ is again described as a conditional probability function $Z(s', a, o) = \Pr(o \mid s', a)$ for $s' \in S$ and $a \in A$. See figure 2 for an illustration.

To elicit desirable agent behavior, we define a suitable reward function $R(s, a)$. In each step, the agent receives a real valued reward $R(s, a)$, if it is in state $s \in S$ and takes action $a \in A$. The agent's goal is to maximize its expected total reward by choosing a suitable sequence of actions. When the sequence of actions has infinite length, we typically specify a discount factor $\gamma \in (0, 1)$ so that the total

reward is finite and the problem is well defined. In this case, the expected total reward is given by $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$, where s_t and a_t denote the agent's state and action at time t, respectively.

For POMDPs, planning means computing an optimal policy that maximizes the agent's expected total reward. In the more familiar case where the agent's state is fully observable, a policy prescribes an action, given the agent's current state. However, a POMDP agent's state is partially observable and not known exactly. So we rely on the concept of a belief. A POMDP policy $\pi : \mathrm{B} \rightarrow A$ maps a belief $b \in B$, which is a probability distribution over S, to the prescribed action $a \in A$.

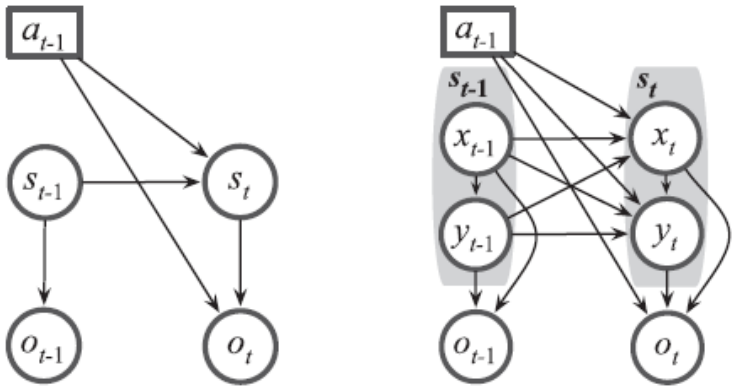


Figure 2: The POMDP model (left) and the MOMDP model (right). A MOMDP state s is factored into two variables: $s = (x, \mathrm{y})$, where x is fully observable and y is partially observable.

A policy π induces a value function V, which specifies the expected total reward $V(b)$ of executing π starting from b. It is known that V^*, the value function for an optimal policy π^*, can be approximated arbitrarily closely by a piecewise-linear, convex function

$$V(b) = \max_{a \in \Gamma} \{a \cdot b\} \tag{1}$$

where b is a vector representing a belief and Γ is a finite set of vectors called

α-vectors. Each α-vector is associated with an action, and the policy can be executed by selecting the action corresponding to the best α-vector at the current belief b, using Equation (1). So a policy can be represented as a set Γ of α-vectors. Policy computation, which, in this case, involves the construction of Γ, is usually performed offline.

Given a policy π, the control of the agent's actions is performed online in real time. It repeatedly executes two steps. The first step is action selection. If the agent's current belief is b, it then takes the action $a = \pi(b)$, according to the given policy π. The second step is belief update. After taking an action a and receiving an observation o, the agent updates its belief:

$$b'(s') = \tau(b, a, o) = \eta Z(s', a, o) \sum_{s \in S} T(s, a, s') b(s), \tag{2}$$

where η is a normalizing constant. The process then repeats.

Equation (2) shows that a POMDP can be viewed as a belief-space MDP. Each state of this MDP represents a belief, resulting in a continuous state space. The transition function can be easily constructed using equation (2).

2.2 MOMDP

Classical POMDPs are essentially an indirectly observable MDPs, because the information about the state is obtained indirectly through instant observations. On the other hand, in an MDP the current state is directly observed at each step. MOMDPs propose a middle-ground scenario, where some of the state variables can be directly observed-namely visible variables-and the remaining ones are hidden variables. Factored POMDPs present the state s (and possibly the observation o) as a vector of variables in a view to exploit the underlying structure of the problem as typically done in probabilistic graphical models. Following a similar idea, in MOMDPs, the fully observable state components are represented as a

single state variable x, while the partially observable components are represented as another state variable y. Thus, (x, y) specifies the complete system state, and the state space is factored as $S = X \times \mathrm{Y}$, where X is the space of all values for x and Y is the space of all values for y.

Formally a MOMDP model is specified as a tuple $(X, \mathrm{Y}, A, O, T_x, T_\mathrm{y}, Z, R, \gamma)$. The transition function

$$T_x(x, \mathrm{y}, a, x') = \Pr(x' \mid x, \mathrm{y}, a)$$

gives the probability that the fully observable state variable has value x' if the robot takes action a in state (x, y), and

$$T_\mathrm{y}(x, \mathrm{y}, a, x', \mathrm{y}') = \Pr(\mathrm{y}' \mid x, \mathrm{y}, a, x')$$

gives the probability that the partially observable state variable has value y' if the robot takes action a in state (x, y) and the fully observable state variable has resulting value x'. Compared with the POMDP model, the MOMDP model has a factored state space $X \times \mathrm{Y}$, with transition functions T_x and T_y, while other aspects remain the same. See Figure 2 for a comparison. As mentioned before, a POMDP can be viewed as a MDP with continuous (belief) states. Correspondingly, a MOMDP can be viewed as a hybrid MDP with both discrete and continuous states. The discrete state variable corresponds to x, and the continuous state variable corresponds to b_y, the belief over y.

So far, the changes introduced by the MOMDP model seem mostly notational. The computational advantages become clear when we consider the belief space B. Since x is fully observable and known exactly, we only need to maintain a belief b_y, a probability distribution on y. Any belief $b \in B$ on the complete system state $s = (x, \mathrm{y})$ is then represented as (x, b_y).

Consider a simple example, in which an agent's state is specified by two binary variables (x, y). The full belief space is a 3-simplex, i.e. a tetrahedron, because the

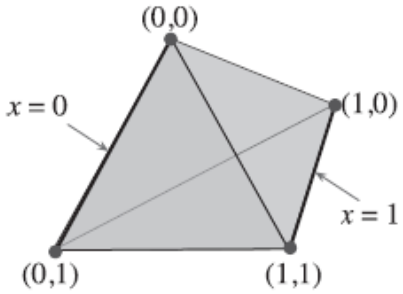


Figure 3: The belief space for two binary variables (x, y) is a tetrahedron. If x is fully observable, then all beliefs lie on two edges of the tetrahedron, corresponding to $x = 0$ and $x = 1$.

probabilities of all of the states must sum up to 1. Each corner of the tetrahedron represents a belief that the agent is in any of the four states (0,0), (0,1), (1,0), (1,1) with probability 1. The center of the tetrahedron represents the belief that the system is in the four states with equal probabilities. Now if x is fully observable, then all beliefs must lie on one of the two opposite edges of the tetrahedron. So B is in fact a union of two line segments and not the full tetrahedron (Figure 3).

In general, let B_{y} denote the space of all beliefs on the partially observable state variable y.We associate with each value x of the fully observable state variable a belief space for y: $B_{\mathrm{y}}(x) = \{(x, b_{\mathrm{y}}) \mid (b_{\mathrm{y}} \in B_{\mathrm{y}})\}$. Here B_{y} is a subspace in B, and B is a union of these subspaces: $\cup_{x \in X} B_{\mathrm{y}}(x)$. While B has $\mid X \mid \times \mid \mathrm{Y} \mid$ dimensions, where $\mid X \mid$ and $\mid \mathrm{Y} \mid$ are the number of states in X and Y, each $B_{\mathrm{y}}(x)$ has only $\mid \mathrm{Y} \mid$ dimensions. Effectively the high-dimensional space B is represented as a union of lower-dimensional subspaces. When the uncertainty of a system is small, specifically, when $\mid \mathrm{Y} \mid$ is small, the MOMDP model leads to dramatic improvement in computational efficiency, due to the reduced dimensionality of belief space representation and the resulting implications for policy representation and computation.

2.2.1 MOMDP Policies

As mentioned in Section 2.1, a POMDP policy can be represented as a value function $V(b) = \max_{a \in \Gamma}\{a \cdot b\}$, where Γ is a set of α-vectors. In a MOMDP, a belief is given by (x, b_{y}), and B is represented as a union of subspaces $B_{\mathrm{y}}(x)$ for $x \in X$. Correspondingly, a MOMDP value function $V(x, b_{\mathrm{y}})$ is represented as a collection of α-vector sets: $\{\Gamma_{\mathrm{y}}(x) \mid x \in X\}$, where for each x, $\Gamma_{\mathrm{y}}(x)$ is a set of α-vectors defined over $B_{\mathrm{y}}(x)$. To evaluate $V(x, b_{\mathrm{y}})$, we first use the value of x as an index to find the right α-vector set and then find the maximum α-vector from the set:

$$V(x, b_{\mathrm{y}}) = \max_{a \in \Gamma_{\mathrm{y}}(x)} \{a \cdot b_{\mathrm{y}}\} \tag{3}$$

It is not difficult to see that any value function $V(b) = \max_{a \in \Gamma}\{a \cdot b\}$ can be represented in this new form. Geometrically, each α-vector set $\Gamma_{\mathrm{y}}(x)$ represents a restriction of $V(b)$ to the subspace $B_{\mathrm{y}}(x)$, obtained by restricting the domain of $V(b)$ from B to $B_{\mathrm{y}}(x)$. In MOMDP policy computation, we compute only these restrictions in lower-dimensional subspaces $B_{\mathrm{y}}(x)$ for $x \in X$, because B is simply a union of these subspaces.

A comparison of Equation (1) and Equation (3) indicates that Equation (3) results in faster policy execution, because action selection can be performed more efficiently. In a MOMDP value function, the α-vectors are partitioned into groups according to the value of x. We only need to calculate the maximum over $\Gamma_{\mathrm{y}}(x)$, which is potentially much smaller than Γ.

In summary, by factoring out the fully and partially observable state variables, a MOMDP model reveals the internal structure of the belief space as a union of lower-dimensional subspaces. We want to exploit this structure and perform ally operations on beliefs and value functions in these lower dimensional subspaces rather than the original belief space.

2.3 SARSOP

SARSOP stands for Successive Approximations of the Reachable Space under Optimal Policies, and was introduced in 2008 [14] as an improvement over HSVI [11]. In this section we take a closer look at the SARSOP algorithm since is the one that we use for our experiments. SARSOP iterates over three main functions, SAMPLE, BACKUP, and PRUNE. A sketch is shown in Algorithm 1.

Algorithm 1 SARSOP

1: Initialize the set Γ of α-vectors, representing the lower bound $\underline{V}$ on the optimal value function V^*. Initialize the upper bound $\overline{V}$ on V^*.
2: Insert the initial belief point b_0 as the root of the tree T_R.
3: **repeat:**
4: $SAMPLE(T_R, \Gamma)$.
5: Choose a subset of nodes from T_R. For each chosen node b, BACKUP(T_R, Γ, b).
6: $PRUNE(T_R, \Gamma)$.
7: **until** termination conditions are satisfied.
8: **return** Γ.

Like all point-based algorithms, SARSOP samples a set of points from the belief space. The sampled points form a tree T_R (Figure 4). Each node of T_R represents a sampled point. As there is no confusion, we use the same symbol b to denote both a sampled point and its corresponding node in T_R. The root of T_R is the initial belief point b_0. To sample a new point b' we pick a node b from T_R as well as an action $a \in A$ and an observation $o \in O$ according to suitable probability distributions or heuristics. Then, b' is computed using the formula:

$$b'(s') = \tau(b, a, o) = \eta Z(s', a, o) \sum_s T(s, a, s') b(s) \tag{4}$$

where η is a normalization constant, and insert b into T_R as a child of b. Clearly, every point sampled this way is reachable from b_0. If we apply all possible sequences of actions and observations, the set of nodes in T_R is exactly R. The key is, of course, to avoid doing so and focus the sampling, instead, on R^*.

To achieve this, SARSOP maintains both a lower bound $\underline{V}$ and an upper bound $\overline{V}$ on the optimal value function V^*. The set Γ of α-vectors represents a piecewise-linear approximation to $\underline{V}$ and is also a lower bound when suitably initialized, using, e.g. a fixed action policy. For the upper bound $\overline{V}$, SARSOP uses the sawtooth approximation. The upper bound can be initialized in various ways, using the MDP or the Fast Informed Bound technique. SARSOP uses the upper and the lower bounds to bias sampling towards R^*.

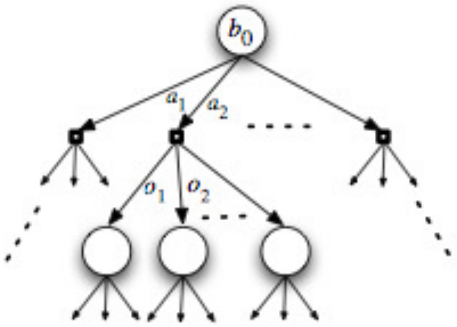


Figure 4: The belief tree T_R rooted at b_0.

2.3.1 Backup

Next, backup at selected nodes in T_R is performed. A backup operation at a node b collates the information in the children of b and propagates it back to b. We perform the standard α-vector backup (Algorithm 2), with the value function approximation represented as a set Γ of α-vectors. The value function approximation at b, obtained from the α-vector backup, is the same as that from the Bellman backup. However, the Bellman backup propagates only the value, while the α-vector backup propagates the gradient of the value function approximation along with the value to obtain a global approximation over the entire belief space rather than a local approximation at b. Invocation of SAMPLE and BACKUP

generates new sampled points and α-vectors. However, not all of them are useful for constructing an optimal policy and are pruned to improve computational efficiency.

Algorithm 2 Perform α-vector backup at a node b of T_R.

BACKUP(T_R, Γ, b)
1: For all $a \in A$, $o \in O$, $\alpha_{a,o} \leftarrow arg\max_{\alpha \in \Gamma}(\alpha \cdot \tau(b, a, o))$
2: For all $a \in A, s \in S, \alpha_a(s) \leftarrow R(s, a) + \gamma \sum_{o,s'} T(s, a, s')Z(s', a, o)\alpha_{a,o}(s')$.
3: $\alpha' \leftarrow arg\max_{a \in A}(\alpha_a \cdot b)$
4: Insert $\alpha' into \Gamma$.

SARSOP is an anytime algorithm that returns the best policy found within a pre-specified amount of time. It gradually reduces the gap ε between the upper and lower bounds on the value function at b_0, until it reaches either a pre-specified gap size or the time limit.

2.3.2 Sampling

To sample new belief points, SARSOP sets a target gap size ε between the upper and lower bound at the root b_0 of T_R and traverses a single path down T_R by choosing at each node the action with the highest upper bound and the observation that makes the largest contribution to the gap at the root of T_R. This is the same action and observation selection strategy used in HSVI2 [19]. The sampling path is terminated under suitable conditions. Together, the strategies for action and observation selection and the choice of termination conditions control the resulting sampling distribution.

Although the SARSOP sampling method is similar to the one proposed in HSVI2, adds the important notion of selective deep sampling. SARSOPs selective deep sampling continues down sampling paths deeper than the HSVI2 collection method when doing so would likely lead to improvements in the lower bound value at belief nodes earlier in the search.

a) *Selective deep sampling:* As each backup operation chooses the action that maximizes the expected reward, improvements in lower bounds are quickly propagated to the root when nodes with high expected rewards are found. This not only directly improves the policy but also provides information to stop sampling more quickly in regions that are likely outside R^*. In contrast, upper bounds cannot be propagated beyond a node until the upper bounds for all the actions at the node are sufficiently improved. Finding the best action is not enough. Thus it gives preference to lower bound improvements and continue down a sampling path beyond the node with a gap of $\gamma^{-t}\epsilon$, if predicted that doing so likely leads to improvement in the lower bound at the root. To make such a predication, conceptually we predict the optimum value $V^*(b)$ at a node b and propagate the predicted value $\hat{V}$ up towards the root. If $\hat{V}$ improves the lower bound at the root, we expand b and then repeat the procedure at the next selected node down the sampling path. Otherwise, we proceed to check the gap termination criterion described below. To predict the optimal value $V^*(b)$, a simple learning technique is used. The beliefs are clustered according to suitable features and use previously computed values of beliefs in the same cluster as b to predict the value of b. This allows us to learn which parts of the belief space is worth exploring.

b) *termination criterion:* If the prediction shows no improvement of the lower bound at the root, it uses the target gap size ε at the root to decide whether to terminate the sampling path and avoid sampling in regions unlikely to be in R^*. As mentioned earlier, the straightforward way of achieving the target gap size ε between the upper and lower bounds at the root of T_R is to require a gap size $\gamma^{-t}\epsilon$ for all leaves of T_R. However, it is in fact sufficient to ensure that the condition is satisfied somewhere along all the paths from the root to the leaves, rather than at the leaves themselves. This has the advantage of leveraging information globally

from the other parts of T_R to terminate a sampling path as early as possible and thus improving computational efficiency.

The combination of selective deep sampling and the gap termination criterion leads to an effective sampling strategy that goes deep into T_R when needed. This avoids unnecessarily sampling in R/R^* and gives a better approximation to R^*.

2.3.3 Pruning

The efficiency of backup operations, which take up a significant fraction of the total computation time, depends significantly on the size of the set Γ of α-vectors. To improve computational efficiency, existing point-based algorithms usually prune an α-vector from Γ if it is dominated by others over the entire belief space B. The notion of optimally reachable space suggests an alternative and more aggressive pruning technique: ideally, we want to prune an α-vector if it is dominated by other α-vectors over R^*, rather than B. Since R^* is potentially much smaller than B, this may substantially reduce the size of Γ and improve the efficiency of the backup operations and thus the overall algorithm. As R^* is not known in advance, we use B, the set of all sampled belief points contained in T_R, as an approximation.

2.4 Computing MOMDP Policies Algorithm

The algorithm used to compute a MOMDP policy is a combination of the MOMDP model and SARSOP algorithm and was presented at 2010 [15]. The new algorithm is actually very similar to SARSOP so in this section we will only overview the parts that need to be altered. So in order to solve a MOMDP, the main modifications required in SARSOP concern the belief update operation and the backup operation.

As in Sarsop, after initialization, the algorithm iterates over three functions, SAMPLE, BACKUP, and PRUNE.

2.4.1 Sampling

The sampled points form a tree T_R (figure 4). Each node of T_R represents a sampled point in R. In the following, we use the notation (x, b_{y}) to denote both a sampled point and its corresponding node in T_R. The root of T_R is the initial belief point (x_0, b_{y0}).

To sample new belief points, we start from the root of T_R and traverse a single path down. At each node along the path, we choose α with the highest upper bound and choose x' and o that make the largest contribution to the gap ϵ between the upper and the lower bounds at the root of T_R. New tree nodes are created if necessary. To do so, if at a node (x, b_y), we choose a, x', and o, a new belief on y is computed:

$$b'_{\mathrm{y}}(y') = \tau(x, b_{\mathrm{y}}, a, x', o) = \eta Z(x', y', a, o) \times \sum_{y \in \mathrm{y}} T_x(x, y, a, x') T_{\mathrm{y}}(x, y, a, x', y') b_{\mathrm{y}}(y) \tag{5}$$

where η is a normalization constant. A new node (x', b'_{y}) is then inserted into T_R as a child of (x, b_{y}). Clearly, every point sampled this way is reachable from $(x_0, b_{\mathrm{y}0})$. By carefully choosing a, x' and o based on the upper and lower bounds, we can keep the sampled belief points near R^*.

When the sampling path ends under a suitable set of conditions, we go up back to the root of T_R along the same path and perform backup at each node along the way.

2.4.2 Backup

A backup operation at a node (x, b_{y}) collates the value function information in the children of $(x, b_{\mathrm{y}}))$ and propagates it back to (x, b_{y}). The operations are performed on both the lower and the upper bounds. For the lower bound, we

perform α-vector backup (Algorithm 3). A new α-vector resulting from the backup operation at (x, b_{y}) is inserted into $\Gamma_{\mathrm{y}}(x)$, the set of α-vectors associated with observed state value x. For the upper bound backup at (x, b_{y}), we perform the standard Bellman update to get a new belief-value pair and insert it into $\Upsilon_{\mathrm{y}}(x)$.

Algorithm 3 α-vector backup at a node (x, b_{y}) of T_R

BACKUP$(T_R, \Gamma, (x, b_{\mathrm{y}}))$

1: For all $a \in A$, $x' \in X$, $o \in O$,

$$\alpha_{a,x',o} \leftarrow arg \max_{\alpha \in \Gamma_{\mathrm{y}}(x')} (\alpha \cdot \tau(x, b_{\mathrm{y}}, a, x', o))$$

.

2: For all $a \in A$, $y \in Y$,

$$\alpha_a(y) \leftarrow R(x, y, a) + \gamma \sum_{x', o, y'} (T_x(x, y, a, x') \times T_{\mathrm{y}}(x, y, a, x', y') Z(x', y', a, o) \alpha_{a,x',o}(y'))$$

3: $\alpha' \leftarrow arg \max_{a \in A}(\alpha_a \cdot b_{\mathrm{y}})$.

4: Insert α' into $\Gamma_{\mathrm{y}}(x)$.

2.4.3 Pruning

Invocation of SAMPLE and BACKUP generates new sampled points and α-vectors. However, not all of them are useful for constructing an optimal policy and are pruned to improve computational efficiency. We iterate through the α-vector sets in the collection $\{\Gamma_{\mathrm{y}}(x) | x \in X\}$ and prune any α-vector in $\Gamma_{\mathrm{y}}(x)$ that does not dominate the rest at some sampled point (x, b_{y}), where $b_{\mathrm{y}} \in B_{\mathrm{y}}(x)$.

2.4.4 Computational Efficiency

Modifying the belief update and backup operations does not affect the convergence property of SARSOP. The altered algorithm above, provides the same theoretical guarantee as the original SARSOP algorithm.

MOMDPs allow the belief space B to be represented as a union of low-dimensional subspaces $B_{\mathrm{y}}(x)$ for $x \in X$. This brings substantial computational

advantages. Specifically, the efficiency gain of this algorithm comes mainly from BACKUP and PRUNE, where α-vectors are processed. In a MOMDP, α-vectors have length $|Y|$, while in the corresponding POMDP, α-vectors have length$|X||Y|$. As a result, all operations on α-vectors in BACKUP and PRUNE are faster by a factor of $|X|$ in our algorithm. Furthermore, in a MOMDP, α-vectors are divided into disjoint sets $\Gamma_{\mathrm{y}}(x)$ for $x \in X$. When we need to find the best α-vector, e.g., in line 1 of Algorithm 3, we only do so within $\Gamma_{\mathrm{y}}(x)$ for some fixed x rather than over all α-vectors, as in a POMDP.

Chapter 3

Problem Description

The objective of this work is to obtain an autonomous mobile agent which will learn to play the hide and seek game. The hide and seek game involves 2 agents - a hider and a seeker - placed on a grid environment. Assuming the hider is a human player, the problem is then for the seeker to keep track of the hider and catch him. This is a trivial problem of planning under uncertainty. To solve this problem we use Partially observable Markov decision processes (POMDPs) -and MOMDPs- which provide a rich mathematical framework for solving such planning problems, with several applications in robotics. The goal is to learn to play hide and seek and thereby handling the different challenges. In this chapter a definition of the problem is given and a MOMDP model as described by [15] and [16]. A similar hide-and-seek experiment has been discussed in [16].

3.1 Game Specification

The *Hide and Seek* game which we play is actually a version of the standard hide and seek game. In the standard game there are at least two players; one seeker and one or more hiders. They are both placed at the *base* of the game. The game starts with the seeker counting to ten with eyes closed while the hider(s) choose a place to hide. When the counting is finished, the seeker searches for the hider(s)

and tries to catch them, while the hider(s) try to reach the base avoiding being caught by the seeker.

In this version of the Hide and Seek game we consider the beginning of the game to be the moment after the seeker has finished counting. This would mean that the game begins with the seeker next to the base and the hider in the position it chooses to hide itself. The seeker wins if it approaches the hider sufficiently and "catches" it. As in the standard vrsion, the hider wins if it has not been caught by the seeker and reaches the base. But also, if no player has won within the maximum predefined time, the game result is a *tie*. Last, the games are placed on a grid enviroment which contains a base (cell), free space-in which the players can move- and obstacles-which make part of the space not accessible by the players and limit their visibility. There are available five different map configurations.

3.1.1 Limitations

As a first approach we will simplify the problem by putting some restrictions:

- First of all the game will be played in simulation on a grid of $n \times m$ cells.
- On each time step (Δt), both the hider and seeker can move 1 cell in straight or diagonal direction, or stay in the same cell. Obviously this also depends on the map, for example there should not be an obstacle or wall in the target cell.
- The maximum velocity of both the hider and seeker is 1 grid cell per Δt.
- All players know the complete map of the environment.
- All players know their position on the map and the position of all other players that are visible to them.
- When both players 'win' at the same time step t, then the seeker wins.

3.1.2 Triangle Strategy

Before starting real games, a simple but wise strategy, to play the hide and seek game, as well for the hider as the seeker, has been defined. This strategy takes into account the limitations as mentioned before and an environment *without obstacles*. In this hide and seek version we have three important distances as shown in Figure 5:

1. d_{sh}: the distance between the seeker and hider.
2. d_{sb}: the distance between the seeker and the base.
3. d_{hb}: the distance between the hider and the base.

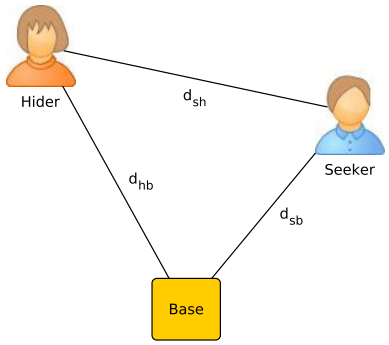


Figure 5: The hide and seek triangle shows the dependencies of the distances between the: *seeker, hider* and *base*.

For the seeker it is important to approach and recognize the hider, thus to minimize d_{sh}. At the same time he should make sure that the hider does not reach the base, which in an environment without obstacles and with the same maximum speed for all players, means that he should be closer to the base than the hider: $d_{hb} > d_{sb}$. For the hider the optimal strategy is almost the opposite. The hider wants to go to the base, thus minimize d_{hb}, and wants to be closer to the base than the seeker: $d_{hb} < d_{sb}$.

3.2 Model Definition

Next we define the problem as a MOMDP model and notice the main differences from the corresponding POMDP model.

The general MOMDP model is defined as a tuple:

$$\langle X, Y, A, O_x, O_y, T_x, T_y, Z, R, \gamma \rangle \tag{6}$$

Notice that for the corresponding POMDP model the tuple would be $\langle S, A, O, T, Z, R, \gamma \rangle$.

Next we will discuss the different sets and functions of this model.

3.2.1 States: X, Y

States contain variables, of which some are completely observable, these are $x \in X$, and some partly observable: $y \in Y$. The full state will be thus: $s = (x, y)$. Both states are defined as:

$$x = (x_s, y_s), y = (x_h, y_h) \tag{7}$$

where (x_s, y_s) and (x_h, y_h) are the positions of the seeker and hider respectively. We can already define the final states, which are all the states where the seeker or the hider wins:

$$S_{\text{winS}} = \{(x, y) | x = (x_s, y_s) \in X, y = (x_h, y_h) \in Y, (x_s = x_h \wedge y_s = y_h)\} \tag{8}$$

$$S_{\text{winH}} = \{(x, y) | x \in X, y = (x_h, y_h) \in Y, (x_h = x_{\text{base}} \wedge y_h = y_{\text{base}})\} \setminus S_{\text{winS}} \tag{9}$$

We should make some notes to these equations:

- In equation 9 we subtract the set S_{winS}, because the hider does not win when the seeker already wins for a certain state.

- A state is a list of variables, of which the fully visible variables are in X and the partially visible variables are in Y, thus the full list of states is: $S = X \times Y$.

- Another factor which ends the game is the time, as soon as the game time passed ($t > t_{\text{game}}$) and no final state has been reached, then the game ends in a tie.

Notice that in the POMDP model there is no separation of the seeker and hider variable in the state. Each state, $s \in S$ is defined by the wholistic view of the map at the moment.

3.2.2 Actions: A

The space of actions A is only defined for the seeker, but in principle the hider's actions will be the same. The player can take an action, $a \in A$, by moving in one of the eight directions (like a king in a chess board) or staying in the same position:

halt, move north, move north-east, move east, etc.

Figure 6 shows the movements which the player can do at each time step, where the grid is self-centred, thus the player is located at cell 0. Due to the maximum speed of 1 cell per time step, the player can only move to one neighbouring cell or can stay at the same location.

1	2	3
8	0	4
7	6	5

Figure 6: A local grid of a player where it is located at cell 0. The player can move to any of its neighbouring cells or it can stay the current cell.

3.2.3 Observations: O_x, O_y

The observation variables can be divided in a fully visible part (O_x) and a partly visible part (O_y) like introduced in [16], where $O_x = X$, and $O_y = Y \cup \{hidden\}$. The *hidden* observation should only be possible when the hider is not visible to the seeker, due to an obstacle or a limited field of view.

3.2.4 Transition functions: T_x, T_y

In an MOMDP two transition functions are to be defined, one for the fully observable states (T_x) and one for the partially observable states (T_y):

$$T_x = p(x'|x, y, a) = \begin{cases} 1 & \text{if } x \to x' \text{ is possible with action } a \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$T_y = p(y'|x, y, a, x') = \begin{cases} 1/n & \text{if } y \to y' \text{ is possible} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

We should note for (10) that each action already defines the movement direction (or no movement) and because we assume a constant speed the next state is known. For (11) n refers to the number of possible movements, which depends on the map, because obstacles will reduce the number of possible movements. It is clear that the probability should be 0 if it is not possible to go directly from a state y to another y' (due to speed limitations or due to the limitations of the environment), however calculating the probability of the reachable states may be done in several ways:

1. Opponent has random movement, i.e. the probability is $1/n$.

2. Define a policy for the opponent, using the *triangle strategy*.

3. Learn the transition probabilities from doing games (in simulation).

3.2.5 Observation Probability: Z_x, Z_y

The observation probabilities are also split for fully and partially observability (like in [16]):

$$Z_x = p(o_x|a, x', y', o_y) = \begin{cases} 1 & \text{if } o_x = x' \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Thus Z_x is only 1 if the observation of the own position (o_x) is equal to the real position (x), since we said that the own position is fully observable. The partially observable observation refers to the location of the hider:

$$Z_y = p(o_y|a, x', y') = \begin{cases} 1 & \text{if } o_y = y' \text{ and } y' \text{ is visible from } x' \text{ OR} \\ & o_y = \textit{hidden} \text{ and } y' \text{ is not visible from } x' \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Here we need a function which tells us if the hider on state y' is visible from state x'. This depends on (1) if there is any obstacle between those states and (2) if y' is in the field of view of the seeker.

3.2.6 Rewards: R

The reward should reflect that the optimal policy lets the seeker win. A reward function is a reward value given when being in a state and doing an action, but here we first ignore the action for the reward since the most important are the final states for which the robot will receive a reward and after which the game will end. The goal of the seeker is to approach the hider sufficiently ($d_{sh} < \epsilon$ or simply $d_{sh} = 0$ for the discrete case) which should give the maximum reward; in the contrary when the hider wins (i.e. is on the base $d_{hb} = 0$) the reward should be minimum. There are several approaches of a reward function:

1. The reward could be set only for the final states: $\forall_{s \in S_{\text{winS}}} R(s) = r_{\text{winS}}$ and $\forall_{s \in S_{\text{winH}}} R(s) = r_{\text{winH}}$ where obviously $r_{\text{winH}} \ll r_{\text{winS}}$, $r_{\text{winS}} > 0$, $r_{\text{winH}} < 0$.

2. The problem of only giving a reward in the final states is that it might take the learning algorithm a long time to converge, therefore we can give rewards

for all states which increases the probability for the seeker to win. A good strategy to follow is the previously introduced *triangle strategy* where we try to minimize the distance to the hider, but at the same time we must be sure that the hider does not arrive to the base before the seeker. The reward function is:

$$R_{\text{seeker}} = (l - d_{sh})w - d_{sh}(1 - w) = lw - d_{sh} \tag{14}$$

Where

$$w = \begin{cases} 1 & \text{if } d_{hb} > d_{sb} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

and l is the maximum distance, which in this case is the number of states ($l = nm$). Since each state represents the position of the seeker (x_s, y_s) and the hider (x_h, y_h) we can calculate the distances. For the distances any obstacles should be taken into account.

Although we have no plans to learn the game from the hider, we could define the reward for the hider in a similar manner:

$$R_{\text{hider}} = (l - d_{hb})w' - (l - d_{sh})(1 - w') \tag{16}$$

where

$$w' = \begin{cases} 1 & \text{if } d_{hb} < d_{sb} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

3. The problem of the previous approach is that the reduction of the reward is really abrupt when the hider goes from being further to the base than the seeker ($d_{hb} > d_{sb}$) to being closer ($d_{hb} < d_{sb}$). We can take as direct reward the negative distance between the seeker and hider, and we add the relative distance to the base of both players with a certain factor (ψ): $R_{\text{seeker}} = -d_{sh} + \psi(d_{hb} - d_{sb})$.

The goal is to find the best policy to play the hide and seek game as seeker, therefore we should also explore different reward functions.

3.2.7 Experimental Procedure

Since we defined the problem as a MOMDP/POMDP problem we are ready to continue with the experimental procedure.

First step is to build a simulation application, where we can represent the problem and play the actual game with human and automated agents. The final goal would be of course to have an automated seeker that learns to play the game quite well - even better than a human. In order to reach that goal we decided to use the SARSOP solver and learn an approximate optimal policy for the seeker's game. The input to the solver would be the model, as defined above. We test the learner for both types of models Partially and Mixed Observability MDPs. In order to obtain better results and a a better policy we decided to calculate the Transition Probabilities using data from real games between human-experts. These steps are described in chapter 5. In chapter 4 we describe the simulation tool developed and demonstrate a game example.

Chapter 4

The Simulation

4.1 Description

As said before, to simulate the problem we created a simple application as shown in the figure 7 below. Figure 7 shows the grid environment of the game. It

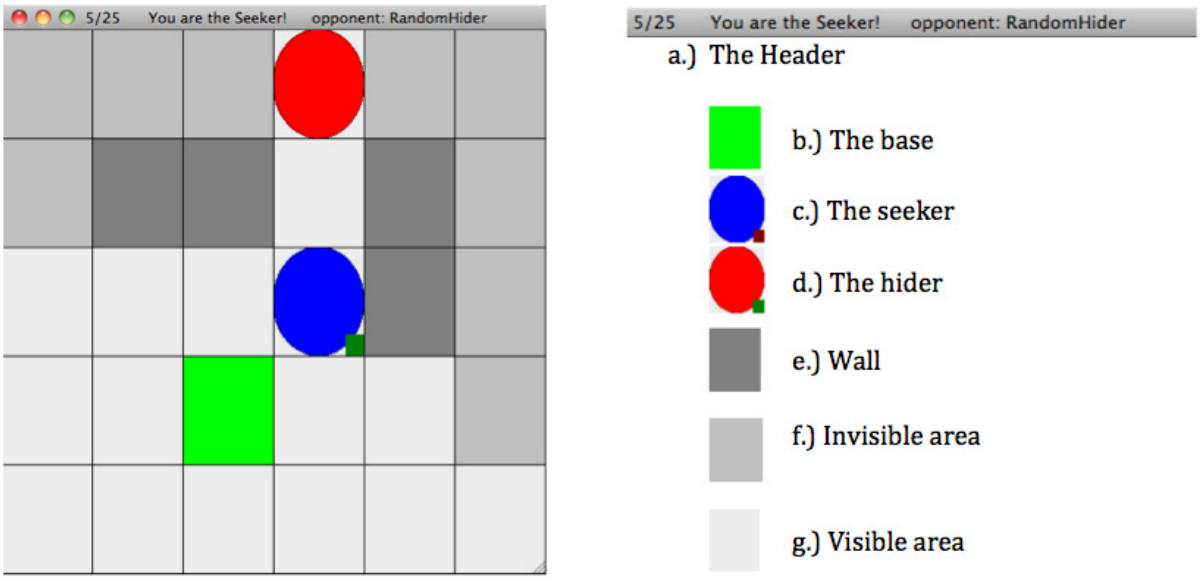


Figure 7: The hide-and-seek simulator.

is a 6×5 grid and it consists of the two players (hider and seeker), a base, one -or more-obstacles, and the invisible space -depending on the point of view. The

two players move around the grid, trying to reach their respective goals. To move around, they choose one of the 9 available actions (towards the 8-directions and halt). Obviously, the agents face limitations due to the size of the grid (they can not move outside of the borders), due to the obstacles laying in the grid (the wall area is not accessible so they cannot mount an obstacle), and the limited visibility in some space which is also caused by the walls. The visibility for each player is calculated with ray-tracing algorithm.

1. ***The Header:*** (figure 7.a.)displays information about the game that is currently played. On the left corner there is a counter showing how many moves have been made out of the maximum number of moves for the game. e.g. 5/25 this was the fifth move out of 25. In the middle of the header, the role of the player is displayed, e.g. "You are the Hider" or "You are the Seeker". At the right, the opponent's name is displayed. In the game of the figure 7 the Seeker is playing against the "RandomHider".

2. ***The Base:*** (figure 7.b.) is the green cell of the figure . The base is a key-area for the game since it is the area the hider wants to reach in order to win the game and at the same time the area that the seeker "protects" from the hider. The seeker's initial position is always next to the base.

3. ***Wall:*** (figure 7.e.)Walls in the game are cells that cannot be mounted by any player and moreover reduce the player's visibility.

4. ***Invisible area:*** (figure 7.f.) In light gray the player sees the area that is currently not visible by him. In this sense, the areas behind walls are invisible and the player does not have direct information over them.

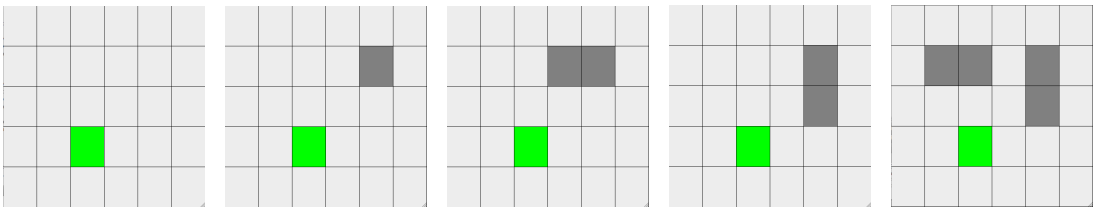
5. ***Visible area:***(figure 7.g.) The white area is the area where the player can see and access without limitations.

6. ***The Hider:*** (figure 7.d.)The player with the role of the hider is always of red color. The hider's objective is to reach the base without being caught by the seeker. The seeker catches the hider when they are both placed on the same cell. The little rectangle on the bottom right corner, is a semaphore, that shows whether the player can take the next action or not. If the semaphore is green that means that the player can take an action. If the semaphore is red the player should wait.

7. ***The Seeker:*** (figure 7.c.)The player with the role of seeker is always of blue color. The seeker's objective is to catch the hider before he reaches the base. The seeker catches the hider when both are placed on the same cell. The little rectangle on the bottom right corner, just like for the hider, is a semaphore, that shows whether the player can take the next action or not. If the semaphore is green that means that the player can take an action. If the semaphore is red the player should wait.

4.1.1 Available Maps & Actions

For this problem we have used 5 different topologies (as shown in Figure 8). These maps are very simple, yet computing the optimal policy for them is not trivial because it depends on the hider's strategy.



(a) Map1 **(b)** Map2 **(c)** Map3 **(d)** Map4 **(e)** Map5

Figure 8: The different topologies of the simulation.

As said before, the available actions that the players can take are nine. These include moving (always one step) towards the 8 directions, as shown in the figure 9, and staying at the same place (halt). The player is not allowed to take any action that results to moving out of boundaries or mount a wall, depending on the topology of the currently played game.

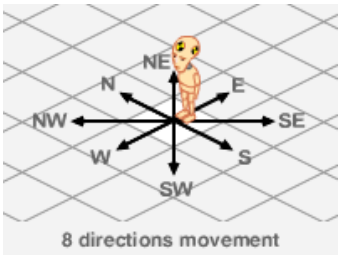


Figure 9: 8-directional movement

4.1.2 Players

The simulation can be played between human players but we have also defined automated players for the role of the hider. This means that human seeker can play against an automated hider. We used two different strategies for the hider:

1. a random strategy, where the hider randomly walks in the grid environment without considering the seeker position or the player's objective.

2. a "smarter" strategy, where the hider chooses his next action considering the triangle strategy, as described in section 3.2.6. In more detail, the hider at each game step ,that he has to take an action, he gets all the possible next positions and scores each one of them. At the end chooses to go to the one with the highest score. The scoring function is similar to the equation 18. The score is assigned to a position p of the grid which is a candidate next position for the hider. The position with the highest score is the one that in the end will actually be the next hider position.

$$score(p) = L - d_{hb} + \gamma d_{hs} + N \tag{18}$$

where L is the maximum distance, in this case we define it as the number of positions $L = rows \times columns$, d_{hb} is the distance between hider and the base, γ is a discount factor, d_{hs} is the distance between the hider and the seeker

and N is zero-mean Gaussian Noise with 1 variance. For the distances of the hider-base and hider-seeker we don't use the euclidean distance between them, but we calculate the shortest path.

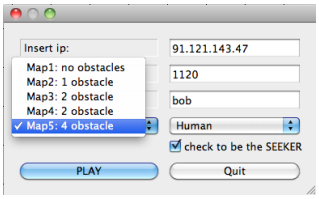
For more information on the simulation you can visit http://alterhost.org/hideandseek/

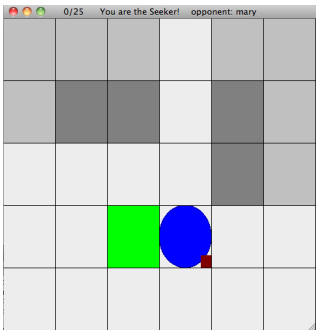
4.2 Game Example

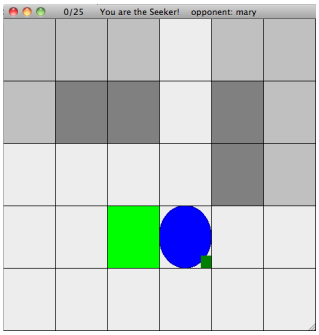
In this section we will give a small example of a game played with the hide and seek simulation. The figures on the left side of the page is the seeker and on the right side of the page is the hider. In the middle you can find some brief explanation of the game. Each row of figures represents the state of the game at each step.

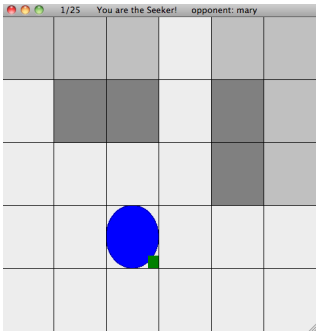
The first step of the game is to choose the configuration. Both of the players insert the ip and port of the server, their usernames, choose the map topology they prefer, their opponent and their role(seeker checkbox). As soon as both hit "play" the game begins. The seeker is already placed on the grid but the hider needs to select (with a mouse click) his initial position. The seeker waits, not being able to move(activation square is red), untill the hider selects the initial position. Then they can both proceed to their first action. Note that the activation square is green when they are allowed to take the next action.

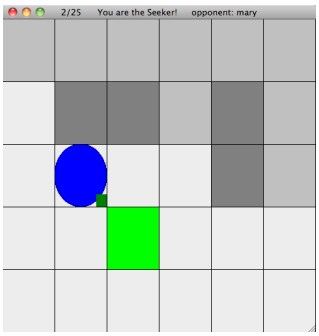
SEEKER











1. Seeker is Bob. Introduces ip/-port of the server. Selects map (map 5), opponent (human player) and role (check the seeker box). The Hide is Mary. Introduces ip/-port of the server. Selects map (map 5), opponent (human player) and role (uncheck the seeker box).

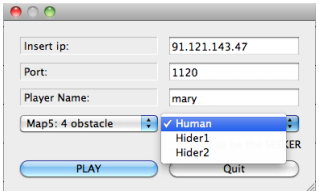
2. The game starts. The seeker is already placed next to the base. He cannot take an action until the hider initializes herself.

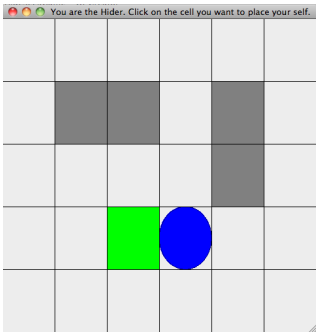
3.Mary sees the grid and chooses her position (mouse click). Her initial position is out of the sight of Bob. Now they can both take their first action (activation square=green)

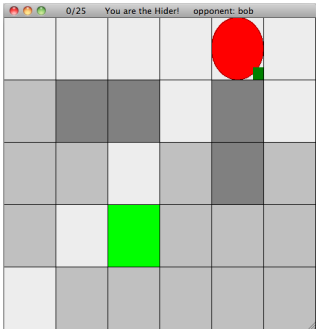
4. Mary moves East-South and Bob moves West towards the base. Still they cannot see each other.

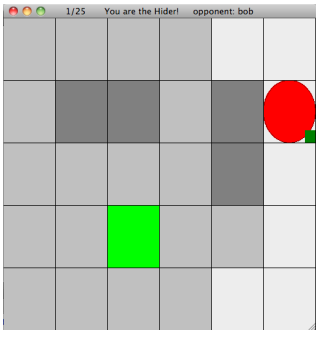
5.Mary moves South and Bob moves North-West. Still they cannot see each other.

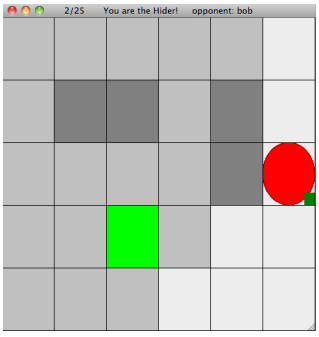
HIDER



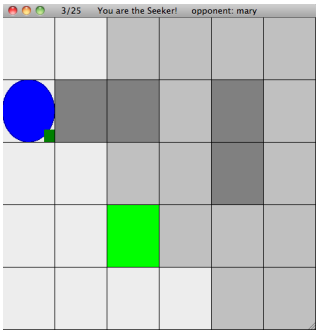


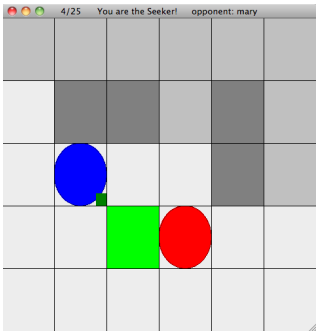


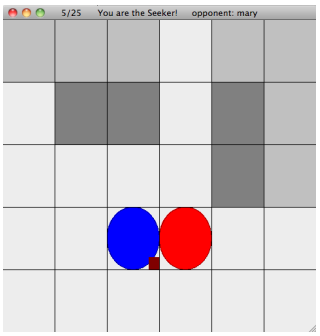


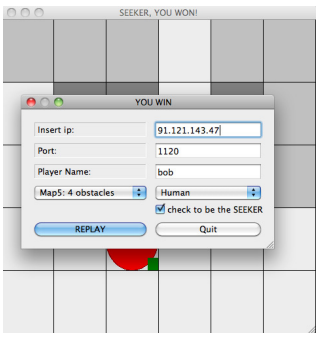


SEEKER









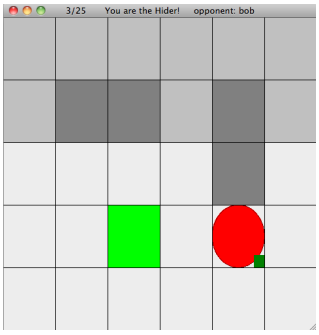
6. Bob keeps moving North-West to gain better visibility behind that wall. Mary decides to make her move towards the base and moves West.

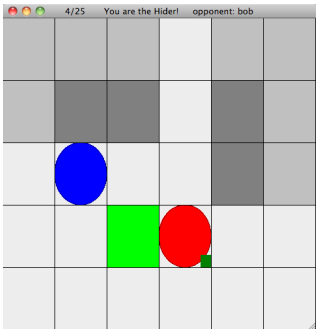
7. Since Bob did not have any luck on that part of the map returns close to the base with a South-East movement, while Mary head straight to the base (moving West).

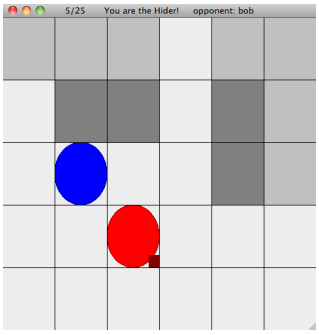
8. After the previous action (4th action) they can now see each other. So Bob mobes again South-East and he gets on the base while at the same step Mary also moves West and gets on the base.

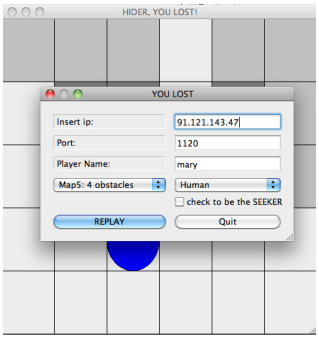
9. Since both got to the base at the same time, rules say that Bob (seeker) wins and Mary loses. The popup window appears again and the players can replay the game with same configuration, replay the game changing the configuration or quiting the game.

HIDER









4.3 Tools and Architecture

The simulation is developed in C++, with Qt creator under Ubuntu-Linux operating system. Qt is a cross-platform application and UI framework. It includes a cross-platform class library, integrated development tools and a cross-platform IDE. Using Qt, one can write web-enabled applications once and deploy them across many desktop and embedded operating systems without rewriting the source code.

The idea is to have an application that people would be able to play on their computers without much effort. This is why we chose to build this application on a server-client architecture. In this way we can have the server and the clients running on different machines. In a client- server topology, each computer sends its player input to the server, which then updates the state of the players in the session and sends the results back to the clients. The game simulation runs entirely on a single computer that has been designated as the server. Client computers send their player inputs to the server, which then sends the resulting game state back to each client. Game logic only ever runs on the server: the clients are effectively functioning as dumb terminals, responsible only for reading input and rendering the game world as described by the server. This of course excludes the automated smart hider, described in section 4.1.2. The automated player is a client which unlike the other clients, does not render the world and usually is located where the server is located. It is not just a dumb terminal, since it performs several calculations in order to decide its own actions.

Server: This process represents the world. It is actually the brain of the game where all the major decisions are being made; for example whether the players lose or win. The server is the one to handle the gameplay aspects and the communication channel between the clients. It is set to listen on a specific port

and as soon as it accepts two new connection requests (from the two clients), the game starts. For the communication with the clients, the server uses sockets over tcp protocol. For each game played the server maintains two sockets corresponding one for each client. Through these sockets, it sends to the clients the new state of the game at each step and receives from them their actions.

Client: The client application, also a standalone application, is the application installed at the user end. It presents the graphical interface of the game and allows the user to introduce his action through the keyboard and mouse input. The graphical interface shows the current map of the game and the players placed on this configuration. For each player the environment differs according to its visibility at the moment. The visibility of the player at each step of the game is calculated with the ray-tracing algorithm. A brief explanation of the ray-tracing algorithm version that we used, can be found in Appendix A.

A special version of the client is the automated player. This client is not handled by a human player and it contains two automated hiders. The client is usually placed on the same machine as the server and is being called by the server when a player wants to play against him. The communication interface of the automated client is exactly the same with that of the normal client's. Their differences is that the automated client unlike the normal client does not display a graphical interface of the game and is not being handled by a human user, but it calculates by itself the next action.

Communication: As already mentioned, the communication between the server and the client is made using sockets over tcp protocol. The communication is predefined and the server exchange messages with the clients to keep them informed about the state of the game. The message sequence is presented in figure 10.

The server which is listening on a certain port, accepts two new connection requests from two clients, it establishes the connections and as soon as it receives the player's information from both clients it sends them the information about the environment that is the topology of the game, the initial positions of the players, the role (hider/seeker) that was given to the client. As we saw at the example in section 4.2, the hider at the beginning of the game can choose his hiding place. So next the hider sends to the server his initial position. The server sends the initial position of the hider to the seeker and then both of the clients can send their first action. When both of them have sent their actions then the server sends them their opponent's new position which defines the state of the game. The players continue sending new actions until they get a response that contains an ending status, informing the player whether he lost or won. After this, the clients disconnect from the server and exit. The server keeps on listening for new connection requests. Note that the requirement of our game that wants both of the players to have the same velocity is taken care of by the server's message handling.

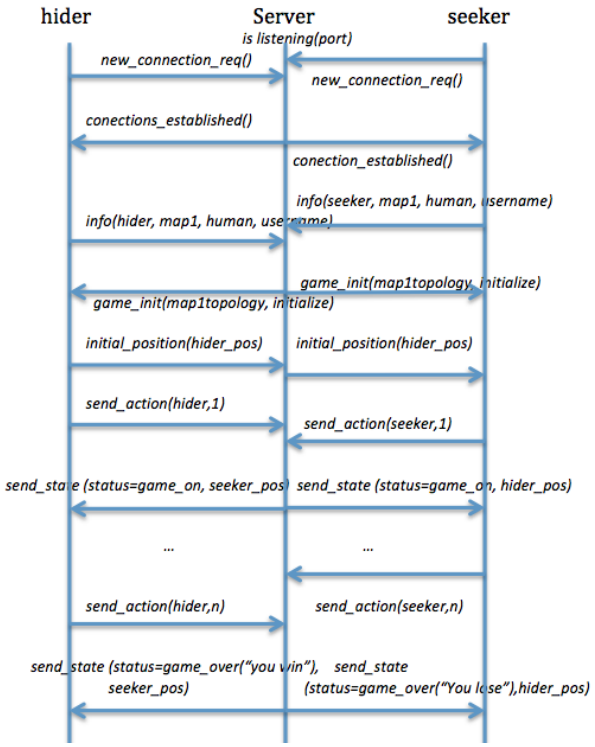


Figure 10: The communication protocol between server and clients(hider/seeker).

Chapter 5

Game Data Handling and PreProcessing

The game simulation, described in the previous chapter, was used in order to collect game data from human players. We invited people from different places to install the game client and play the game. We recorded the information of the games played and used them to compute the transition probabilities of the model (section 5.1). Next, we used that model as input to the POMDP solver and attained the learned policy for the seeker(section 5.2.2). With the computed strategy of the seeker we performed experiments to compare the way of playing of the automated seeker, SeekroBot, with respect to the playing of the human "experts". (chapter 6)

5.1 Game Data

The information recorded for each game was introduced into a MYSQL database. We used the MySQL database because it is a high performance, open source database and will make the data easier to handle. The database schema is shown in figure 11. For each game, the information we store is: the general information of the game which is the names of the players, the map configuration, the role given (who is the hider and who is the seeker), the ending status of the game and the number of the steps required for the game, and the game flow of

each game which contains the state of the world at each step. The game step is defined as the time interval between the actions taken by the players. That is, when both of the players performed their first action the first step is done, when they both perform the second action then the second step is done and so on. All this information is conviniently handled by the schema below.

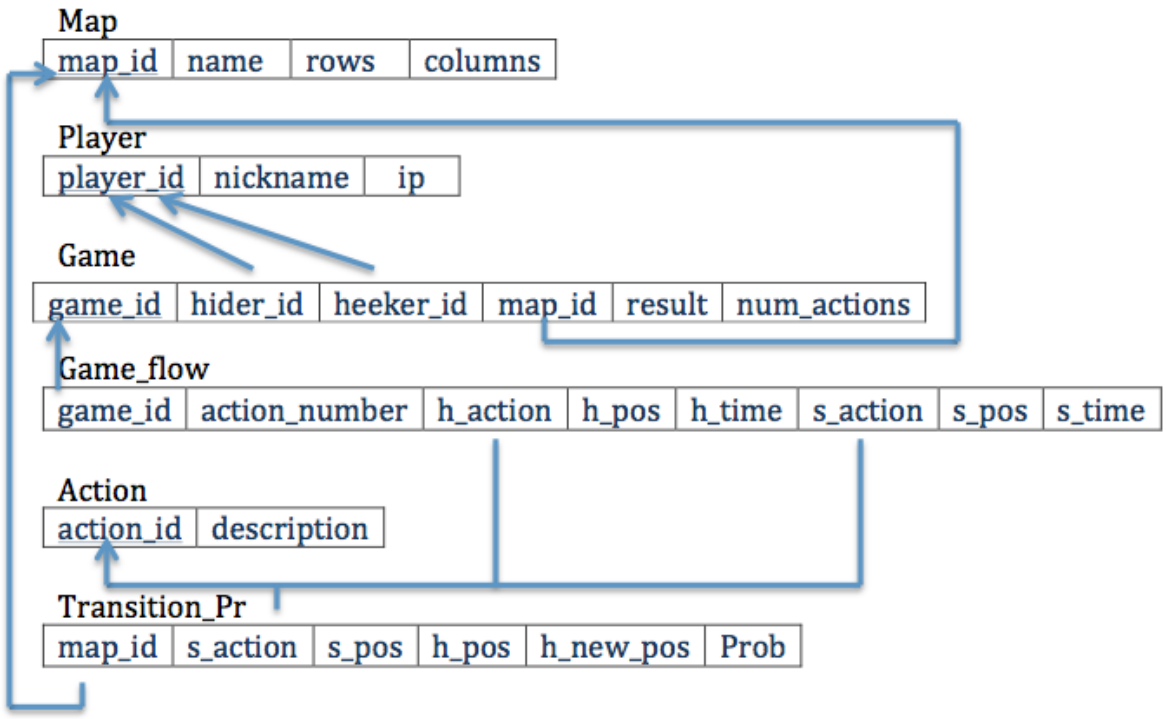


Figure 11: The hide-and-seek database schema.

The transition probabilities that we calculated, at the end, are stored in the *trasitions_Pr* table and each record presents the probability of the hider to be in a new position (*h_new_pos*) on the next state given that on the current state the hider is in position (*h_pos*) , the seeker is in position (*s_pos*), and the seeker chose action (*s_action*), for a certain map with (*map_id*). Note that for this calculation we included only the games played by human hiders or by the automated SmartHider. We exclude the games with the automated RandomHider since it is rather pointless

to base the calculations on random movement.

The amount of data that we were able to use finally was of the order of 4000 records for the $game\_flow$ table. The transitions calculated from the data aqcuired are of the order of 10% of all possible transitions. This percentage is rather small because the players tend to move around the "important areas" mostly, which is around the base and the obstacles, leaving the "colder areas", the ones far way from the base, unexplored. The transition combinations that were not found in the recorded data, were treated with a uniform distribution over the probabilities.

5.2 The Solver - APPL

The solver we used, named APPL - Approximate POMDP Planning Toolkitn and developed at the National University of Singapore, is a C++ implementation of the SARSOP algorithm [14], using the factored MOMDP representation [17]. It takes as input a POMDP model in the POMDP or POMDPX file format and produces a policy file. It also contains a simple simulator for evaluating the quality of the computed policy. A simulation can be found in Appendix B.

The POMDPX file that is used as the input of the solver, describes the MOMDP model of the problem. In our case it describes the model we defined in chapter 3.2. The transition probabilities that were calculated as explained in section 5.1, are loaded in this file along with the rest of the information that defines the model. In the next sections we overview the POMDPX and the POLICYX file format structure.

5.2.1 Input: File Format Structure

A PomdpX document consists of a header and a PomdpX root element which in turn contains child elements, as shown in Example 1 below. The first line of the document is an XML processing instruction which defines that the document

adheres to the XML 1.0 standard and that the encoding of the document is ISO-8859-1. Other encodings such as UTF-8 are also possible.

Example 1: A PomdpX document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pomdpx version="0.1" id="HideAndSeek"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="pomdpx.xsd">
      <Description>             . . . </Description>
      <Discount>                      </Discount>
      <Variable>                      </Variable>
      <InitialStateBelief>            </InitialStateBelief>
      <StateTransitionFunction>       </StateTransitionFunction>
      <ObsFunction>                   </ObsFunction>
      <RewardFunction>                </RewardFunction>
</pomdpx>
```

<pomdpx> Tag

Continuing with the example above, the second line contains the root-element of a PomdpX document, the pomdpx element, which has the following attributes:

- *version*
- *id* - optional name for the specified model.
- *xmlns:xsi* - defines xsi as the XML Schema namespace.
- *xsi:noNamespaceSchemaLocation* - this is where we put our XML Schema definition, *pomdpx.xsd*. The PomdpX input should be validated with this schema to ensure well-formedness.

The conventional ordering of the child elements is *Description*, *Discount*, *Variable* and thereafter: *InitialStateBelief*, *StateTransitionFunction*, *ObsFunction* and

RewardFunction. However this ordering is not strictly required and one may permute their orderings. *Description* is an optional, short description of the specified model. The other child elements specify the POMDP tuple $(S, A, O, T, Z, R, \gamma)$ and the initial belief b_0.

In general these elements should all be present, and each can appear only once. *ObsFunction* may be omitted if there are no observation variables in the model. Similarly, *InitialBeliefState* may be omitted if all state variables are fully observed (for example an MDP model). *PomdpX's* child elements are described in greater detail in the following subsections.

<Description> Tag

This is an optional tag that one may provide to give a brief description of the specified problem.

<Discount> Tag

This specifies the discount factor γ. It has to be a real-valued number, for example for our Hide and Seek problem, we will be using a discount factor of 0.95.

<Variable> Tag

The state, action and observation variables which factorize the state S, action A, and observation O spaces are declared within the Variable element. Reward variables, R are also declared here.

Each state variable is declared with the <StateVar> tag. It contains the following attributes:

- *vnamePrev* - identifier for the variables start state.
- *vnameCurr* - identifier for the variables end state.
- *fullyObs* - set to true if the variable is fully observed. The default is false

which implies that it is partially observed.

The possible values that a variable can assume are either specified with regards to the <*NumValues*> or <*ValueEnum*> tags.

The observation and action variables are declared similarly, with the <*ObsVar*> and <*ActionVar*> tags respectively. Both require the attribute *vname* which serves as the identifier for the variable. The possible values that an observation or action can assume can also be specified with either <*NumValues*> or <*ValueEnum*>.

Finally, reward variables are declared with the <*RewardVar*> tags which must contain the *vname* attribute. The *vname* serves as an identifier for the reward variable. The <*RewardVar*> is an empty XML tag and no values are specified.

<InitialStateBelief> Tag

This is an optional tag. It specifies the initial belief b_0, and may be omitted if all state variables are fully observed. The PomdpX format allows the initial belief to be specified as multiple multiplicative factors, with each <*CondProb*> tag specifying one of these factors.

The <*CondProb*> tag has no attributes and requires the following three children tags:

- <*Var*> - identifies the factor being specified. Only identifiers declared as *vnamePrev* of state variables are allowed here.

- <*Parent*> - the set of conditioning variables. Only the keyword null and identifiers declared as *vnamePrev* of state variables are allowed here. Note however that PomdpX only allows certain combinations of *vnamePrev* identifiers to be specified in the <*Var*> and <*Parent*> tags. The keyword null may be used to signify the absence of any conditioning variables.

- <*Parameter*> - specifies the actual probabilities in the factor.

<StateTransitionFunction> Tag

This specifies the transition function T , which in general is the multiplicative result of the individual transition functions of each state variable in the model. Each <*CondProb*> tag specifies the transition function for each state variable.

<ObsFunction> Tag

This specifies the observation function Z, which in general is the multiplicative result of the individual observation functions of each observation variable in the model. Each <*CondProb*> tag specifies one of these individual observation functions.

For each <*CondProb*> element, the identifier within the <*Var*> tags identifies the observation variable whose observation function is being specified. The identifiers within the <*Parent*> tags identifies the conditioning variables in the observation function. Identifiers that appear within the <*Var*> tags must be identifiers which had been declared as the *vname* attribute of observation variables. Identifiers that appear within the <*Parent*> tags must be identifiers which had been declared as the *vnameCurr* attribute of state variables, or the *vname* attribute of action variables.

<RewardFunction> Tag

This specifies the reward function R, which in general is the additive result of the individual reward functions of each reward variable in the model. Each <*Func*> tag specifies one of these individual reward functions.

Similar to the <*CondProd*> tag, the<*Func*> tag has no attributes and requires the following three children tags to be defined:

- <*Var*> -this identifies the reward variable whose reward function is being

specified. Only identifiers that had been declared as the *vname* attribute of reward variables may appear here.

- *<Parent>* - this identifies the domain of the reward function. All identifiers declared as *vnamePrev* or *vnameCurr* attributes of state variables, *vname* attribute of action variables or *vname* attribute of observation variables are allowed here.

- *<Parameter>* - specifies the actual values in the function.

5.2.2 Output: File Format Structure

The output of the solver is a PolicyX file. PolicyX is an XML file format for specifying POMDP and MOMDP policy computed by the offline solver. In this section we overview the PolicyX format. The Example 2 below, presents a PolicyX document sample.

Example 2: A PolicyX document.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Policy version="0.1" type="value" model="HideAndSeek.pomdpx"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="policyx.xsd">
   <AlphaVector vectorLength="2" numObsValue="3" numVectors="6">
       <Vector action="1" obsValue="0">9.5 9.5 </Vector>
       <Vector action="3" obsValue="0">19.025 -0.975 </Vector>
       <Vector action="2" obsValue="0">18.0737 9.025 </Vector>
       <Vector action="1" obsValue="1">10 10 </Vector>
       <Vector action="0" obsValue="1">17.1701 8.57375 </Vector>
       <Vector action="3" obsValue="2">0 0 </Vector>
   </AlphaVector>
</Policy>
```

The first line of the document is an XML processing instruction which defines that the document adheres to the XML 1.0 standard and that the encoding of the document is ISO-8859-1. Other encodings such as UTF-8 are also possible. It is followed by the root element <Policy> tag that specifies the policy.

<Policy> Tag

The second line of the document is the <Policy> tag, the root element of the PolicyX file. The <Policy> tag has the following attributes:

- <version>- version of PolicyX
- <type> - type of the policy. Either "value" for alpha vector policy or "graph" for policy graph
- <model> - Optional. Stores the file name of the model that this policy solves.
- <xsi:noNamespaceSchemaLocation> - The location of the XML schema for PolicyX file format, policyx.xsd. PolicyX documents should be validated with this schema to ensure correctness.

The <Policy> tag has either an alpha vector policy specified by <AlphaVector> tag or a policy graph specified by <DAG> tag as its child element.

<AlphaVector> Tag

This tag contains the pool of alpha vectors that form the policy as child elements. Its child elements can be either <Vector> tag or <SparseVector> tag. A mixture of both <Vector> tag and <SparseVector> tag is allowed. The <AlphaVector> tag has the following attributes:

Input: File Format Structure

- *vectorLength* - The number of entries in each vector.
- *numObsValue* - The number of observed states for (MOMDP) mixed observability problems or "1" for POMDP problems.
- *numVectors* - The total number of alpha vectors in this policy. This attribute is optional.

<Vector> Tag

The <Vector> tag specifies a single alpha vector as a list of floating point numbers within the tag. The number of floating point entries in the tag must be the same as *vectorLength* specified earlier in the <AlphaVector> tag. On top of the floating point entries, the following attributes also need to be specified:

- *action* - the action number associated with this alpha vector
- $obsValue$ - the observed state number associated with this alpha vector or 0 if the problem is POMDP.

<SparseVector> Tag

This tag specifies a single alpha vector as a list of index, value pair in its children elements. Entries whose index are not specified within the tag are assumed to be zero. This is useful when many of the entries in the alpha vector are zero. The <SparseVector> tag has the same attributes as <Vector> tag.

<Entry> Tag

This tag is the child element of <SparseVector> tag. Within the <Entry> tag, it contains index and value of an entry in an alpha vector.

<DAG> Tag

This tag specifies the policy graph. It is not implemented yet.

A Policy file for Hide and Seek problem with alpha vectors specified in dense format using <Vector> tags, can be found in Appendix ?.

Chapter 6

Experimental Results

This section describes the experimental procedure followed and the results obtained during this work.

After the definition of the model, we present it in a PomdpX format, as described in the previous chapter, and using the solver and its components we were able to compute an approximate optimal policy for the role of the seeker.

In detail, we defined different PomdpX files for each map, for each model (MOMDP/POMDP) and also for two different cases of transition probabilities. The first considers the transition probabilities that were calculated from the data of real human games and the other was built applying a uniform distribution over the transition probabilities. Using these files we were able to learn the corresponding policies. The computation results are shown in tables 1 and 2 .

For each calculated policy Table 1 presents the computation time required in seconds, the precision reached (difference between upper and lower bounds of the value function for the optimal policy), and the estimated expected total rewards for the MOMDP model, while Table 2 presents the same information for the POMDPs.

Map	MOMDP					
	Uniform Trans Prob			Real Trans Prob		
	Time(s)	Precision	Reward	Time(s)	Precision	Reward
Map1	1.31	0.000991	654.275	0.7	0.785167	653.747
Map2	7.73	0.000996	655.485	7.08	0.799379	654.903
Map3	8.51	0.000998	659.142	5.59	1.44731	658.303
Map4	7.48	0.000980	659.223	8.42	1.39021	658.417
Map5	19.28	0.000980	664.881	4.76	2.38466	663.575

Table 1: Results of policy computation using MOMDP models

Map	POMDP					
	Uniform Trans Prob			Real Trans Prob		
	Time(s)	Precision	Reward	Time(s)	Precision	Reward
Map1	382.75	0.000986	654.275	107.16	0.756137	653.753
Map2	1090.15	0.000996	655.485	257.97	0.830996	654.873
Map3	10431.20	0.000999	659.142	473.64	1.31992	658.415
Map4	7824.96	0.000998	659.223	227.38	1.34085	658.479
Map5	32753.70	0.000996	664.881	810.46	2.62077	663.357

Table 2: Results of policy computation using POMDP models

Comparing the two tables (1, 2) we can observe the clear difference on computation time between the MOMDP and POMDP models. Even though the precision reached is similar for the corresponding cases of the models, the computation times for the POMDP case are much higher in comparison with the corresponding times of MOMDP. This confirms the conclusion in [15] that the MOMDP approach improves the performance of the point-based POMDP algorithm by many times.

The policies calculated using the uniform transition probabilities seem to always reach the target percision (0.001), while the calculations using the real data probabilities do not reach the target and the precision increases as the map topology becomes "harder", with more obstacles laying in the grid.

For the calculations with real probabilities on the MOMDP model we used a memory limit (of 4G) and we aqcuired the policy calculated before this limit was exceeded. We noticed that even though the calculations were running for several minutes (between 30-90 mins depending on topology) before the limit exceeded, the final precisions were reached on the times as shown in Table 1. For the calculations with real probabilities on the POMDP model, the memory limit was not enough, so we introduced also a time limit. We run the solver for 30mins but the final precisions were reached much earlier as shown in Table 2.

The fact that the precision of the policies using real data never reach the target but with the uniform data the target is reached needs further investigation.

Map	HUMAN					
	RandomHider			SmartHider		
	Win	Lose	Tie	Win	Lose	Tie
Map1	79.5	14.3	6.2	92	2.7	5.3
Map2	92.8	7.2	-	91.5	5	3.5
Map3	76.1	7.5	16.4	81.8	12.5	5.7
Map4	77.8	13.9	8.3	83.9	10.7	5.4
Map5	76.3	15.8	7.9	84.3	7.85	7.85

Table 3: Winning percentages on human games against the two automated hiders, RandomHider and SmartHider.

Next step would be to evaluate the "quality" of the policies aquired. In order to do that, we introduced these policies into the simulation (Chapter 3) and through an automated player, SeekroBot, we compare these strategies against the strategies the human players followed. More specific, we compute the winning/losing/tie percentages of all the games that humans played (in the seeker role) against RandomHider and SmartHider and compare them to the winning/losing/tie percentages of the games that SeekroBot played (using the policies) against the same hiders.

Map	SEEKROBOT (uniform trans. prob.)					
	RandomHider			SmartHider		
	Win	Lose	Tie	Win	Lose	Tie
Map1	41	-	59	99	-	1
Map2	35	-	65	98	-	2
Map3	50	1	49	99	-	1
Map4	44	1	55	98	-	2
Map5	44	-	56	95	2	3

Table 4: Winning percentages of the games played by SeekroBot, using Uniform Transition Probabilities, against the two automated hiders, RandomHider and SmartHider.

First we analyze the games humans played against the automated hiders (RandomHider and SmartHider). In Table 3 the winning/losing/tie percentages over the overall games played are presented. The results are presented in percentages but the actual games played for each map and against each hider opponent are of the order of 100

	SEEKROBOT (real trans. prob.)					
	RandomHider			*SmartHider*		
Map	Win	Lose	Tie	Win	Lose	Tie
Map1	45	-	55	100	-	-
Map2	38	-	62	98	-	2
Map3	41	1	58	95	2	3
Map4	45	-	55	97	-	3
Map5	54	1	45	99	-	1

Table 5: Winning percentages of the games played by SeekroBot, using Real Data Transition Probabilities, against the two automated hiders, RandomHider and SmartHider.

In Tables 4 and 5 we present the results of SeekroBot playing against the Random and Smart hider. SeekroBot played 2000 games using the policies calculated. So for every map and every hider opponent, SeekroBot played exactly 100 games. The amount of games was defined in this way so it would be of the same order as the games humans played. At this point, the games played by SeekroBot use only the policies learned from the MOMDP model. The policies acquired from the two

models are quite similar, if not identical, and to make sure of that we compared the corresponding policies of the two models using the solver's simulation component. An explanation on this comparison and an example are given in Appendix B.

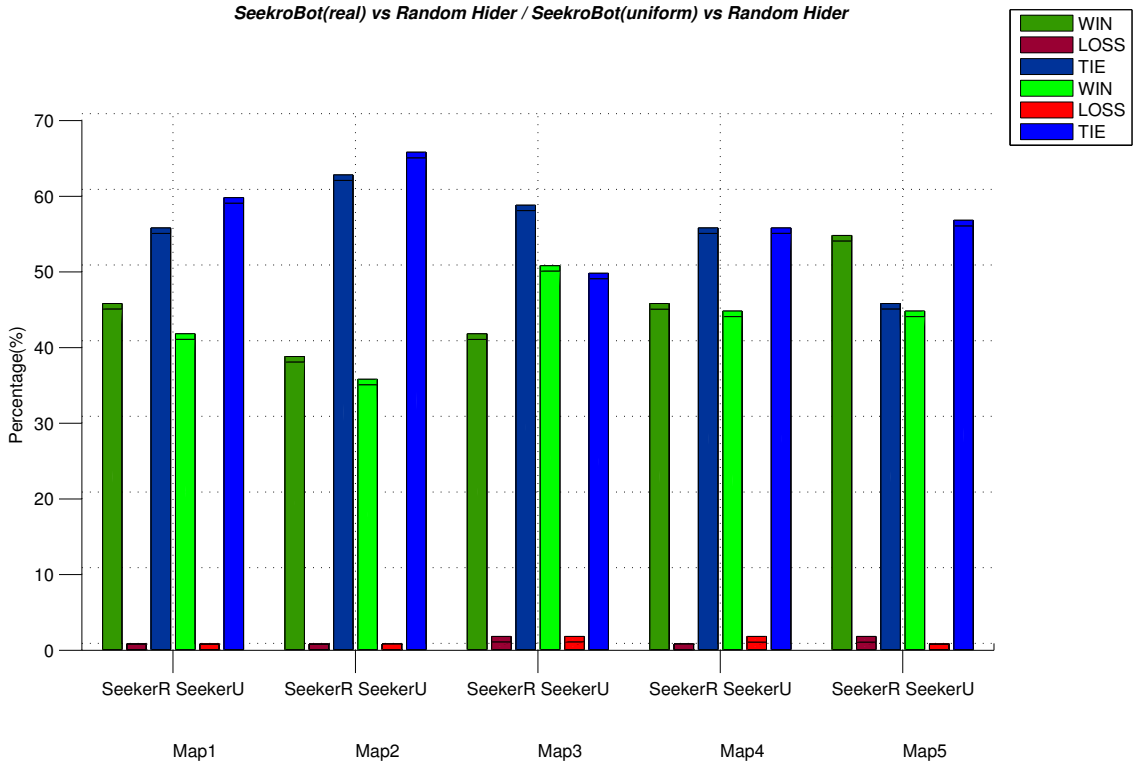


Figure 12: The winning results for the SeekroBot against the RandomHider. On each pair of column sets, the left one represents the Seekrobot womputed with real transition probabilities and the righ the one computed with uniform transition probabilities.

Table 4 shows the results of the SeekroBot, using the policies learned from Uniform probabilities, playing against the Random and Smart Hiders. As we can notice, SeekroBot, performs very well, with almost never losing. We observe that against the RandomHider the percentage of the tie is significally higher than the respective percentage of the Smart Hider. This happens because the SmartHider's strategy forces him to pursue its objective and move towards the base, so the SeekroBot playing in an almost optimal way can track its opponent down and win.

Unlike the RandomHider, which only wonders randomly in the grid without having an objective in mind.

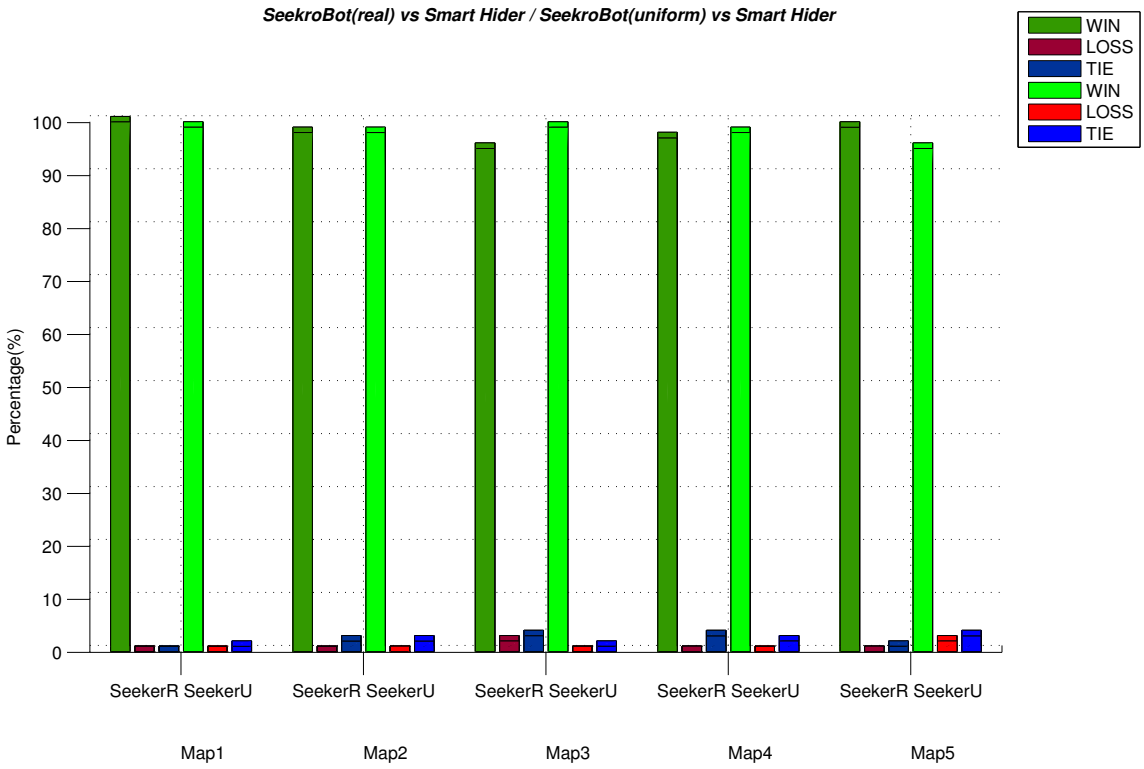


Figure 13: The winning results for the SeekroBot against the SmartHider. On each pair of column sets, the left one represents the Seekrobot womputed with real transition probabilities and the righ the one computed with uniform transition probabilities.

Table 5 shows the winning results, in percentages, of the SeekroBot, using the Real Probabilities for learning the policies, in games against the Random and Smart Hiders. The observations are fairly the same as the ones made in Table 4, since the SeekroBot performs equally well. One could argue that the results from the real transition probabilities games are slihgtly better from the games played with the policies learned using the uniform probabilities (figures 12 and 13), but this is rather risky since the differences are minor and the data set size quite small. In order to clarify this point further investigation would be required with a larger

data set of games and a larger data set of real transitions.

In the next histograms a comparison between the human and the SeekroBot game results, is presented. For every map, there are two sets of triple columns representing every player's win/loss/tie statistics. The left set of columns are the human's stats and the right are those of the SeekroBot.

The histogram of figure 14 represent the win stats of the human players against the SmartHider in comparison to the SeekroBot which policy was calculated with probabilities from real data, against also the SmartHider. We observe that SeekroBot has better performance than the human players. The winning percentage is higher in every map and the loss percentage significantly lower.

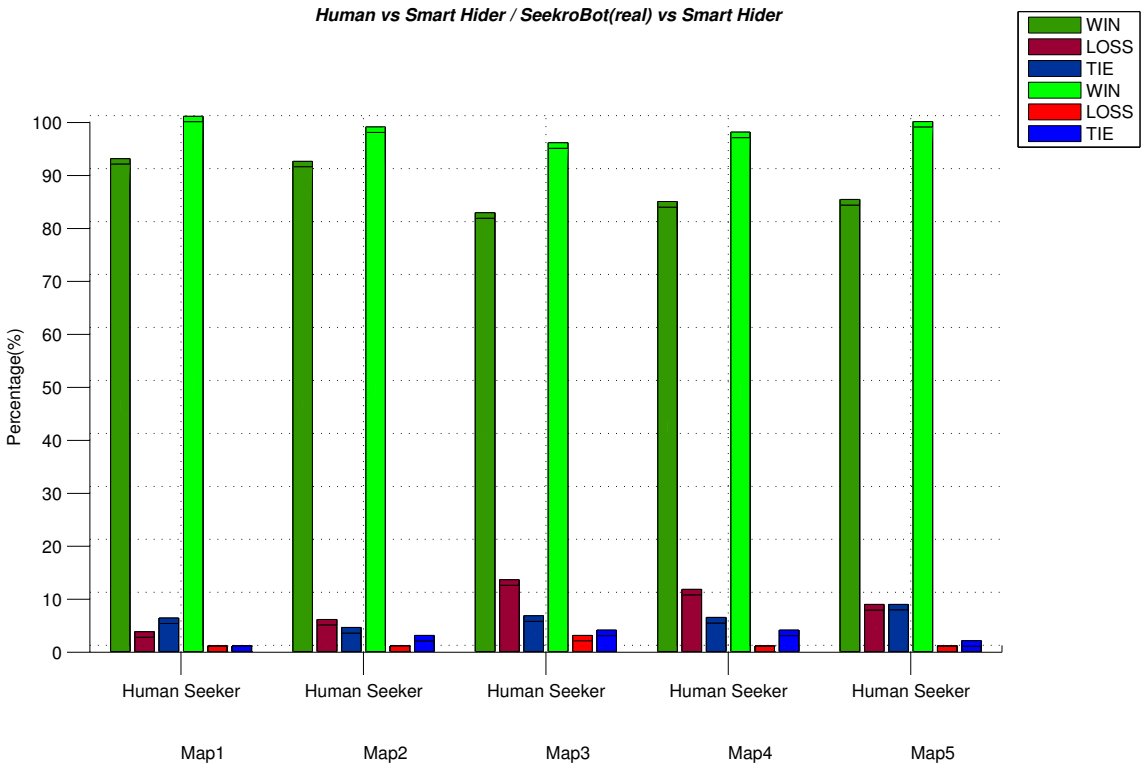


Figure 14: Winning results for human and the SeekroBot with real probabilities against the SmartHider.

The same observations can be made for the comparison of human with the

uniform Seekrobot (figure 15). Seekrobot again performs better than humans and its superiority increases as the map gets harder. This is probably caused beacause human are often prone to mistakes, something that obviously is not true for SeekroBot.

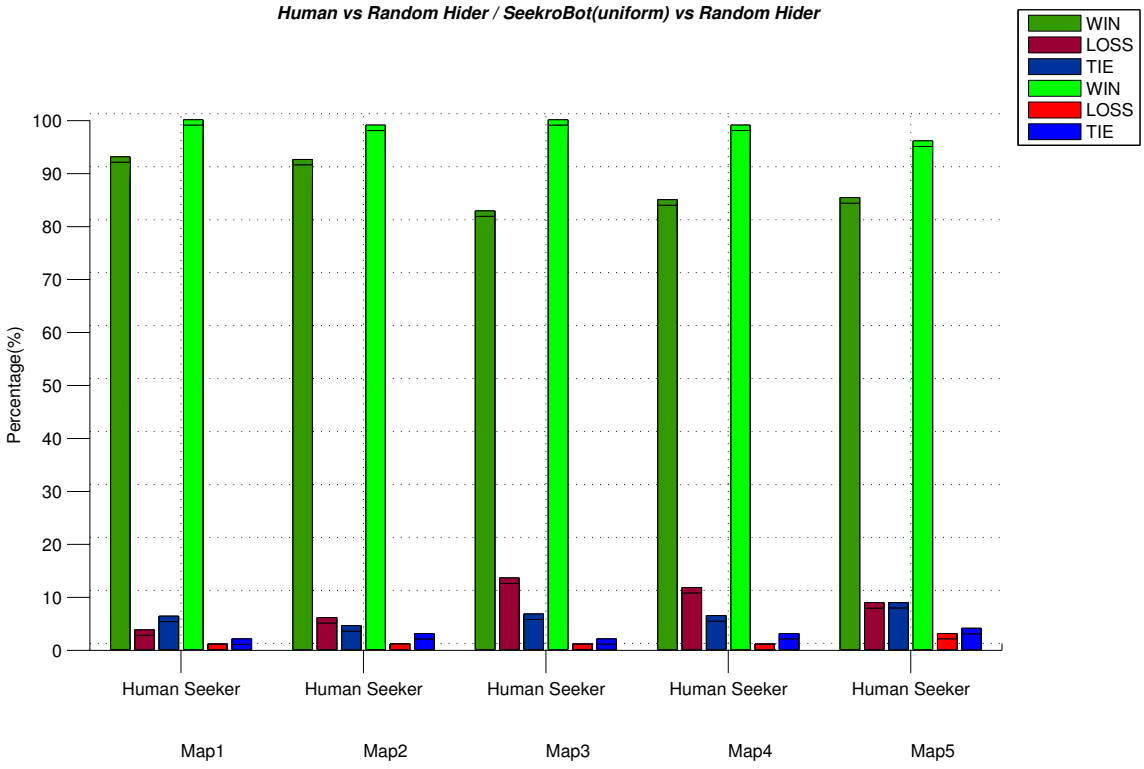


Figure 15: Winning results for human and the SeekroBot with uniform probabilities against the SmartHider.

The situation changes when humans and SeekroBot fight against the RandomHider (figures 16, 17). The human's winning pecentage is much higher than the SeekroBot's. Humans though have a loss percentage much higher than the SeekroBot since SeekroBot has almost never lost. The interesting part of these histograms is the significantly high percentage of tie that the SeekroBot has. This didn't happen in the cases were the SeekroBot was playing against the SmartHider. This is because SeekroBot plays rather defensive, reassuring first the base and not

going after his opponent, while the RandomHider just moves in a random way along the grid without taking into consideration the base or the seeker.

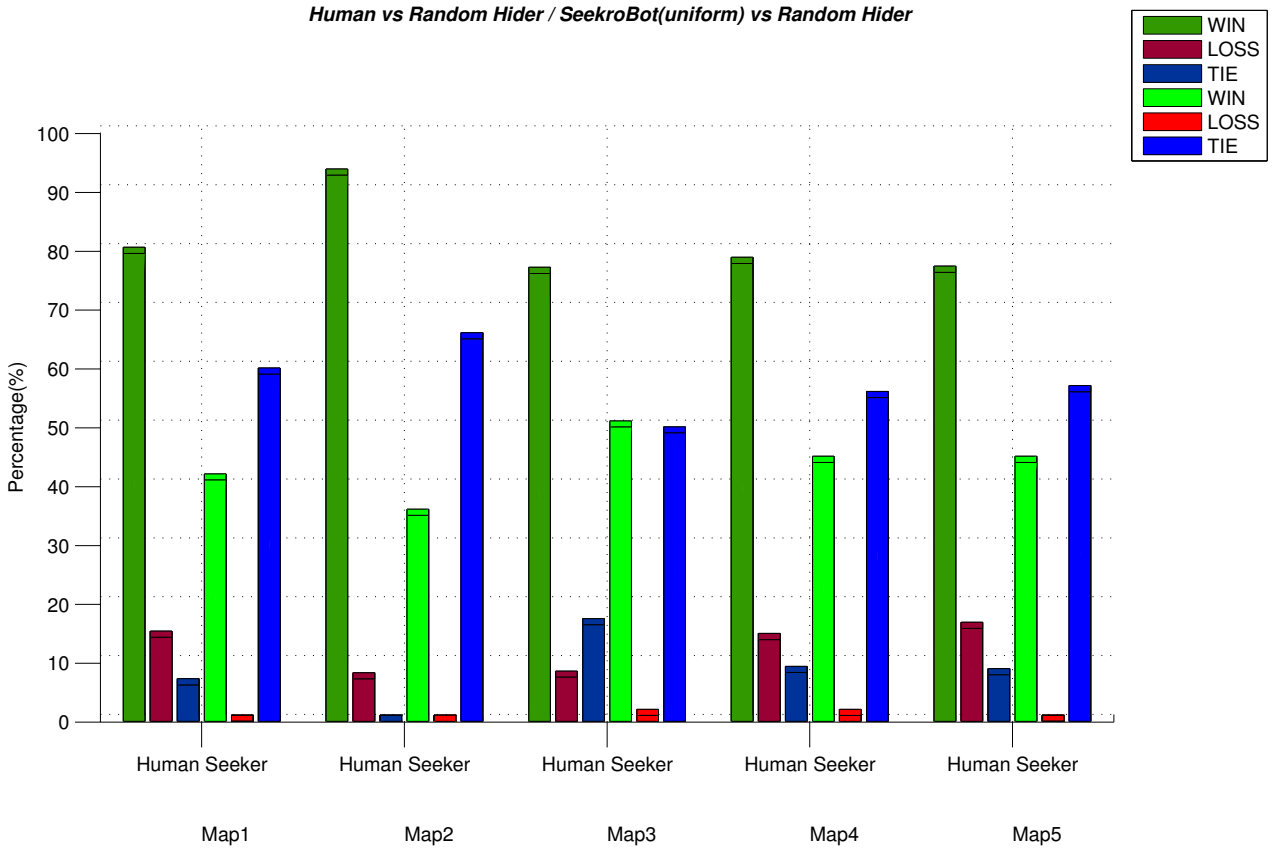


Figure 16: Winning results for human and the SeekroBot with uniform probabilities against the RandomHider.

Summarizing, we can say that Seekrobot is now an expert player of the hide and seek game. He ouperformed human players when playing against the SmartHider. In the case of games against the RandomHider, human players performed globally a little bit better at the expense of losing more times. SeekroBot almost never lost.

We have not been able yet to directly confront SeekroBot against human players just by lack of time, at least to collect meaningful statistics, but we foresee

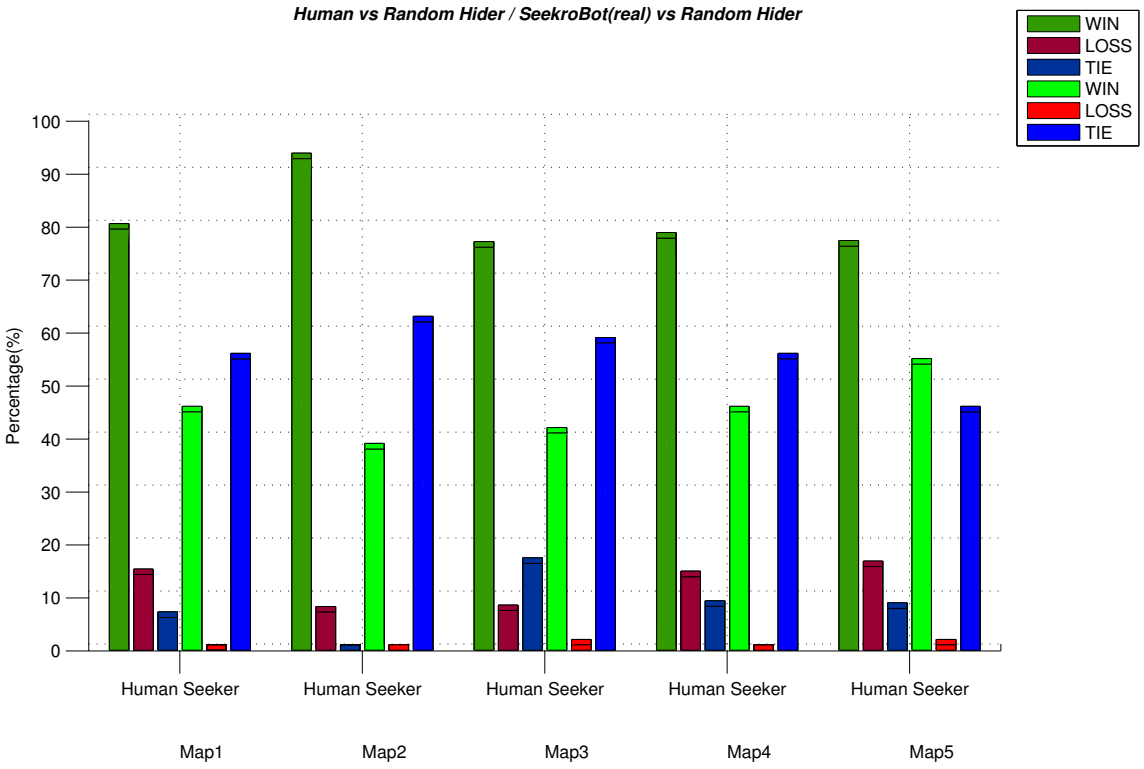


Figure 17: Winning results for human and the SeekroBot with real probabilities against the RandomHider.

Chapter 7

Conclusions and Future Work

We studied in this project a state-of-the-art approximate point-based algorithm for solving POMDPs [14] and its variation for solving MOMDPs [17]. We demonstrated how to apply it to planning in a robotic context, specifically in the Hide and Seek game. For the simple maps tested in simulation, the learned policies showed very good performance, superior in many cases to human players. In order to test the learned policies and to collect the data required to define the POMDP and MOMDP models from human played games, we implemented a graphical application for simulating the game based on client-server architecture.

POMDPs have been successfully used for motion planning under uncertainty in various robotic tasks ([5], [11] and [10], [18]). Point-based algorithms have greatly improved the speed of POMDP solution by sampling from the reachable space. Even so, a major challenge remaining is to scale up POMDP algorithms for complex robotic systems. Exploiting the fact that many robotic systems have mixed observability, the MOMDP approach uses a factored model to separate the fully and partially observable components of a robot's state. This work applies and tests the MOMDP model in a hide and seek simulation. Our experiments show that on our task, the MOMDP approach improves the performance of a leading point-based POMDP algorithm by many times. Furthermore, even when a robot

does not have obvious fully observable state components, it still can be modeled as a MOMDP by reparameterizing the robot's state space. Ten years ago, the best POMDP algorithm could solve POMDPs with a dozen states. Five years ago, a point-based algorithm solved a POMDP with almost 900 states, and it was a major achievement. Nowadays, POMDPs with hundreds of states can often be solved in seconds, and much larger POMDPs can be solved in reasonable time.

However, scaling up the approach presented here to a considerably larger grid and more obstacles in the Hide and Seek game, as in a more realistic environment, may be hard. Future work goes into this research direction trying to exploit both online planning [19] and hierarchical decompositions of the task[20].

Appendix A

Ray Tracing Algorithm

Each player at the game of hide and seek has his own perspective of the game since he has a different point of view. In order to compute everything thats visible from a single viewpoint we use the ray tracing algorithm. For every position of the map in respect to the players position it performs a line-of-sight check and identifies all grid squares that intersect the line segment. If any square is solid, the line of sight is blocked.

We will explain the ray tracing algorithm with an example.

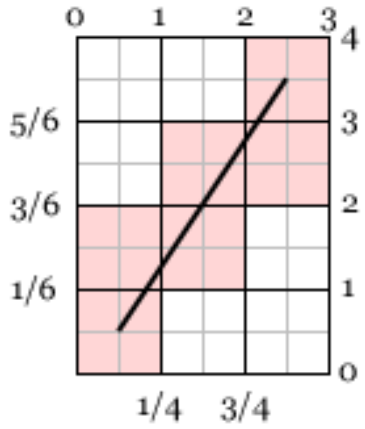


Figure A.18: The ray-tracing algorithm.

The numbers at top and right are the grid coordinates; the line segment goes

from (0.5, 0.5) to (2.5, 3.5). We want to identify all the squares marked in pink.

In pseudocode the algorithm is as follows:

Algorithm 4 ray-tracing

1: Start at the square containing the line segments starting endpoint.
2: For the number of intersected squares:
3: If the next intersection is with a vertical grid line:
4: Move one square horizontally toward the other endpoint
5: Else:
6: Move one square vertically toward the other endpoint).

The number of squares intersected by the line is fairly easy to compute; its one (for the starting square) plus the number of horizontal and vertical grid line crossings.

As we move along the line, we can track where we are as a fraction of the lines total length. Call this variable t. In the example, wed start with t = 0 in the lower left corner. The first horizontal grid intersection is at 1/4, while the first vertical intersection is at 1/6. Since the vertical one is closer, we step up one square and advance t to 1/6. Now the next vertical intersection is at 3/6, while the next horizontal intersection is still at 1/4, so we move horizontally, advancing t to 1/4, and so forth.

The fraction of the line between grid intersections in the horizontal direction is simply one over the lines horizontal extent. Likewise, the line fraction between vertical grid intersections is one over the lines vertical extent. These values go to infinity when the line is exactly horizontal or exactly vertical. Fortunately, IEEE 754 floating-point (used by all modern computers) can represent positive or negative infinity exactly and compute correctly with them, so the divisions by zero work fine.

Appendix B

Policy Comparison

APPL toolkit contains a simulation component that is used to get a simulation path for a policy. The simulator can also estimate the expected total reward (ETR) for a policy by generating many simulation paths and computing the average reward of these paths. In order to compare the policies of the two models, POMDP and MOMDP, we generated several simulation paths and compare them as long as their ETR. The simulation paths are created with random observations. In order to ensure that we have the same pseudo-random array, we input as an argument of the execution a fixed seed number. This reassures that in every execution the list of random observations will be the same. As result of these comparisons we concluded that the policies learned from the two models are -if not exactly- almost the same. Below we present the path trace of the policies learned with MOMDP and POMDP model, in map 3, with the real game transition probabilities. Notice that the actions chosen for each step are the same for both policies.

In the path traces below, X is the position of the seeker. For example, 18 is the index position of the 28 existing states in map 3. belief (over Y), is the belief probabilities of the seeker, that is, in which state does the seeker belief that the hider is positioned. A is the action taken, R is the reward received, and O is the observation.

The MOMDP path trace	The POMDP path trace.
>>> begin	>>> begin
X: 18	X: 0
belief (over Y): 0:0.037037	belief (over Y): 504:0.037037
1:0.037037 2:0.037037	505:0.037037 506:0.037037
3:0.037037 4:0.037037	507:0.037037 508:0.037037
5:0.037037 6:0.037037	509:0.037037 510:0.037037
7:0.037037 8:0.037037	511:0.037037 512:0.037037
9:0.037037 10:0.037037	513:0.037037 514:0.037037
11:0.037037 12:0.037037	515:0.037037 516:0.037037
13:0.037037 14:0.037037	517:0.037037 518:0.037037
15:0.037037 16:0.037037	519:0.037037 520:0.037037
17:0.037037 19:0.037037	521:0.037037 523:0.037037
20:0.037037 21:0.037037	524:0.037037 525:0.037037
22:0.037037 23:0.037037	526:0.037037 527:0.037037
24:0.037037 25:0.037037	528:0.037037 529:0.037037
26:0.037037 27:0.037037	530:0.037037 531:0.037037
A: 0	A: 0
R: 27.8519	R: 27.8519
X: 18	X: 0
O: 14	O: 14
belief (over Y): 14:1	belief (over Y): 518:1
A: 2	A: 2
R: 28	R: 28
X: 13	X: 0
O: 21	O: 21

continue...

```
belief (over Y): 21:1
A: 4
R: 28
X: 20
O: 15
belief (over Y): 15:1
A: 1
R: 29
X: 14
O: 20
belief (over Y): 20:1
A: 6
R: 29
X: 19
O: 20
belief (over Y): 20:1
A: 0
R: 29
X: 19
O: 21
belief (over Y): 21:1
A: 3
R: 28
X: 20
O: 21
```

continue...

```
belief (over Y): 385:1
A: 4
R: 28
X: 0
O: 15
belief (over Y): 575:1
A: 1
R: 29
X: 0
O: 20
belief (over Y): 412:1
A: 6
R: 29
X: 0
O: 20
belief (over Y): 552:1
A: 0
R: 29
X: 0
O: 21
belief (over Y): 553:1
A: 3
R: 28
X: 0
O: 21
```

continue...

```
belief (over Y): 21:1
A: 0
R: 29
X: 20
O: 27
belief (over Y): 27:1
A: 5
R: 29
X: 26
O: 26
belief (over Y): 26:1
A: 8
R: 60
X: 19
O: 25
belief (over Y): 25:1
A: 7
R: 29
X: 18
O: 19
belief (over Y): 19:1
A: 0
R: 29
X: 18
O: 25
```

continue...

```
belief (over Y): 581:1
A: 0
R: 29
X: 0
O: 27
belief (over Y): 587:1
A: 5
R: 29
X: 0
O: 26
belief (over Y): 754:1
A: 8
R: 60
X: 0
O: 25
belief (over Y): 557:1
A: 7
R: 29
X: 0
O: 19
belief (over Y): 523:1
A: 0
R: 29
X: 0
O: 25
```

MOMDP Simulation		POMDP Simulation	
Simulating ...		Simulating ...	
action selection : one-step look ahead		action selection: one-step look ahead	
#Simulations	Exp Total Reward	#Simulations	Exp Total Reward
10	664.988	10	666.066
20	666.478	20	666.918
30	661.563	30	661.062
40	663.593	40	663.683
50	662.158	50	662.165
60	658.636	60	659.086
70	657.882	70	657.891
80	656.96	80	656.426
90	656.259	90	655.556
100	657.602	100	656.961
Finishing ...		Finishing ...	
#Sims	ETR | 95% Conf. Interval	#Sims	ETR | 95% Conf. Interval
100	657.602 (650.849, 664.356)	100	656.961 (650.163, 663.758)

List of References

[1] UPC, IRI. "Ceeds: The collective experience of empathic data systems." Sept. 2010. [Online]. Available: http://www.iri.upc.edu/project/show/103

[2] C. H. Papadimitriou and J. N. Tsitsiklis, "The complexity of markov decision processes," *Mathematics of operations research*, vol. 12, pp. 441–450, 1987.

[3] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[4] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *ARTIFICIAL INTELLIGENCE*, vol. 101, pp. 99–134, 1998.

[5] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for pomdps," 2003.

[6] Z. Feng and S. Zilberstein, "Region-based incremental pruning for pomdps," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, ser. UAI '04. Arlington, Virginia, United States: AUAI Press, 2004, pp. 146–153. [Online]. Available: http://dl.acm.org/citation.cfm?id=1036843.1036861

[7] K. Poon, "Master's thesis on a fast heuristic algorithm for decision-theoretic planning," 2001.

[8] M. T. J. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for pomdps," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[9] G. Theocharous and L. P. Kaelbling, "Approximate planning in pomdps with macro-actions," in *In Proceedings of Advances in Neural Information Processing Systems 16*, 2003.

[10] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, ser. UAI '04. Arlington, Virginia, United States: AUAI Press, 2004, pp. 520–527. [Online]. Available: http://dl.acm.org/citation.cfm?id=1036843.1036906

[11] T. Smith and R. G. Simmons, "Point-based POMDP algorithms: Improved analysis and implementation," in *Proc. Int. Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[12] G. Shani, R. I. Brafman, and S. E. Shimony, "Forward search value iteration for pomdps," in *In IJCAI*. IJCAI, 2007.

[13] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Learning policies for partially observable environments: Scaling up," 1995.

[14] H. Kurniawati, D. Hsu, and W. S. Lee, "Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces," 2008.

[15] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee, "Planning under uncertainty for robotic tasks with mixed observability," *Int. J. Rob. Res.*, vol. 29, pp. 1053–1068, July 2010. [Online]. Available: http://dx.doi.org/10.1177/0278364910369861

[16] M. Araya-Lopez, V. Thomas, O. Buffet, and F. Charpillet, "A closer look at momdps," in *Proceedings of the 2010 22nd IEEE International Conference on Tools with Artificial Intelligence - Volume 02*, ser. ICTAI '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 197–204. [Online]. Available: http://dx.doi.org/10.1109/ICTAI.2010.101

[17] S. Ong, S. Png, D. Hsu, and W. Lee, "Pomdps for robotic tasks with mixed observability," in *Proceedings of Robotics: Science and Systems V (RSS09)*, 2009.

[18] H. K., L. Kaelbling, and Lozano-Perez, "Grasping pomdps," in *In Proc. IEEE Int. Conf. on Robotics and Automation*, 2007, pp. 4485–4692.

[19] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online planning algorithms for pomdps," *Journal of Artificial Intelligence Research (JAIR)*, vol. 32, pp. 663–704, 2008.

[20] E. A. Hansen, "Dynamic programming for partially observable stochastic games," in *IN PROCEEDINGS OF THE NINETEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, 2004, pp. 709–715.