The pairwise heuristic

A method for treating uncertainty in planning and robot localization

by

Koosha Khalvati

B.Sc., The University of Tehran, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

December 2012

Abstract

One of the major challenges in the field of Artificial Intelligence is dealing with uncertainty. Finding the optimal solution in the presence of uncertainty is computationally quite costly. This makes it impossible to solve large problems. In this thesis, we propose a new heuristic, named the pairwise heuristic, which efficiently finds a near-optimal solution for such problems. The pairwise heuristic is based on optimal solutions for the pairs of states. For each pair, it solves the problem assuming that the uncertainty exists only between the two states of the pair. A greedy online strategy uses these solutions to solve the main problem. We tested the pairwise heuristic on two problems where uncertainty plays a major role, i.e., localization and planning under uncertainty. Our achievements in connection with both problems are novel in their respective fields. In the field of localization, we have developed an efficient method to localize a robot in any kind of environment in a fully autonomous way. In the field of planning under uncertainty, our method finds a near-optimal solution in a time shorter than the time required by any other current method in the field.

Preface

Chapter 3, Sections 1.1, 2.1, 5.1, 6.1, and Appendix A are based on "Active Robot Localization with Macro Actions," Koosha Khalvati and Alan K.Mackworth, In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS)*, 2012

Table of Contents

List of Tables

List of Figures

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Alan K. Mackworth. It was an honor to work with him during my graduate studies. His invaluable guidance, endless support, and encouragement were instrumental in the past two years. This thesis would not have been possible without his tremendous help. I would also like to thank Professor David Poole for reviewing this thesis and for his insightful comments.

Many thanks to my friends, Saghar Mirbagheri, Azin and Negin Jahan Afrooz, and Mohammad Hossein Momenzadeh for their support and love. These two years would have been dreadful without their presence.

I owe my deepest gratitude to my family especially my beloved sister, Roxana. She is always there when it matters and that means a lot to me.

Chapter 1

Introduction

There has been tremendous progress in the field of Artificial Intelligence in the last fifty years. Several optimal and efficient methods have been proposed and many problems have been solved. However, it appears as though Artificial Intelligence is more successful in theoretical problems than in solving real world challenges. For example, there is no widely-used home robot yet, autonomous vehicles are still being tested and rescue robots are not used in disasters very often. One of the underlying reasons for this slow progress is the existence of uncertainty in the real world. This uncertainty is inevitable and ongoing due to the noise in actuators and sensors, and the use of limited sensors [31]. Uncertainty makes the decision making process much more complex. Finding an optimal or at least a near optimal solution efficiently for a problem with uncertainty is one of the major challenges in the Artificial Intelligence field in the recent years. In this thesis, we introduce a new heuristic that can be used to find near optimal solutions efficiently in two kinds of problems where uncertainty plays a major role, namely robot localization and planning under uncertainty. Our focus is on the perceptual ambiguity causing the agent's unawareness of its actual state. Our heuristic, named the pairwise heuristic, finds the optimal solution of the problem for each pair of states assuming that the actual state is one of the members of the pair with equal probability. It means that the problem is solved in the case that uncertainty exists only between two states. A simple greedy algorithm can use the pairwise heuristic to solve the main problem efficiently.

1.1 Localization

Localization is the task of determining a robot's position in a known environment. In this problem, the robot knows the possible observations in different places of the environment, but it is unaware of its own location. It should find out its position by comparing current observation with possible observations in different places. The main issue is that sometimes one observation may arise in different places due to the symmetry in the environment

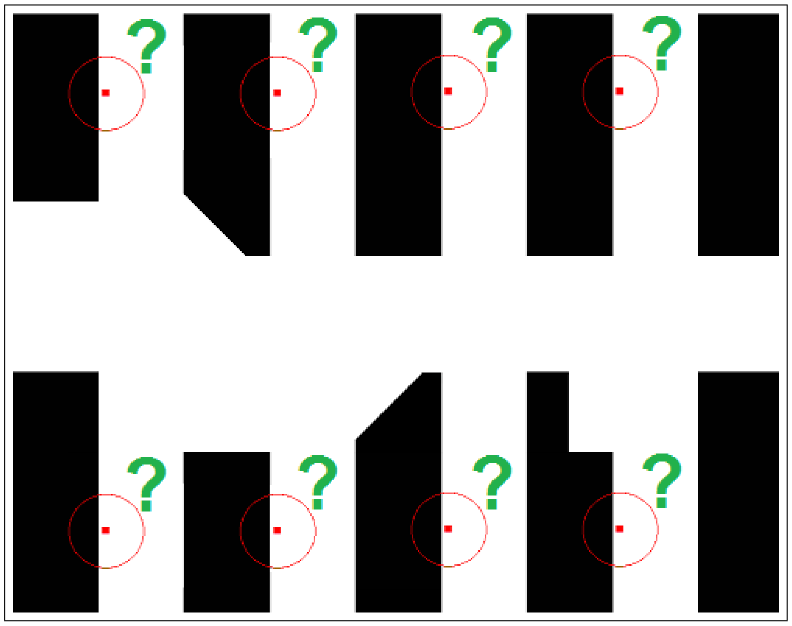


Figure 1.1: A robot with a 360° range finder can not find its exact location only with one observations.

or robot sensor limitations. Figure 1.1 shows an example of this situation. In this kind of situation the robot should perform an action or even multiple actions to localize itself.

Localization is an essential task in mobile robotics. A robot cannot be called autonomous without the ability to localize itself. Furthermore, a robot is often unable to perform other tasks without being aware of its location. For example, robot navigation and search are tasks that are not possible without knowing the position of the robot. A robot's ability to localize itself accurately and efficiently is the subject of numerous research papers in recent years. It has been proven that localizing with minimum cost is an NP-hard problem [6]. Therefore, researchers have sought to find near optimal solutions efficiently. Active strategies are good candidates to fulfill these two criteria: near optimality and efficiency. Some papers covering these strategies have defined heuristics such as taking the robot to places with the maximum number of new features [15] or choosing the actions that lead to minimum expected number of possible states [4] Another active approach is entropy-based that chooses the action that minimizes the expected entropy [7]. In this thesis, we introduce a new active strategy based on the pairwise heuristic for the localization problem, that can be efficiently computed.

Our main contribution, however, is providing an algorithm to deal with the problem of self similarity of environments. The problem is that, due to the self similarity of the environment, the robot gains no further information by performing basic actions. Examples of these situations are corridors, identical offices and big rooms with no objects in them when the robot uses a limited range laser sensor. In the first two cases, the robot merely sees similar walls and in the third, it finds nothing in its laser range. If the robot uses vision instead of a laser, it may find many similar places in environments such as hospitals or schools. Many rooms look the same in hospitals. Also, classrooms and offices may have the same pattern in schools and the robot cannot localize itself until it exits a room. In these cases, the robot should perform a sequence of actions instead of just one. In this thesis, we propose an algorithm to generate optimal cost sequences that can reduce the ambiguity in any situation. Also, this algorithm can be used to determine whether or not a robot can localize itself in an environment.

In the pairwise heuristic, we find an optimal sequence of actions that can localize the robot if the robot is uncertain about being in one of only two states. For each pair of states, this optimal sequence is calculated. These sequences are called macro actions. After finding all macro actions, a simple greedy strategy can find the near optimal path for solving the localization task in the main problem.

1.1.1 Related Work

Proof of NP-hardness of finding the optimal strategy for localization problem was provided by Dudek *et al.* in 1995 [6]. After that, Tovey and Koenig [33] proved that even minimizing the cost with factor $c \log n$ is NP-hard where n is the total number of states. So researchers tried to find near optimal solutions. Koenig *et al.* [17] proposed an $O(\log^3 n)$-factor algorithm on grid-based maps and extended it to polygonal maps. The drawback of this algorithm is that its high computational cost ($\Omega(n^{12})$) allows it to be used only in very small environments [32]. Rao *et al.* [27] also suggested a near optimal randomized algorithm in polygonal maps. Both of these algorithms assumed that actions and sensors are noise-free and hence their application in real environments is still an open issue.

On the other hand, methods proposed for real environments address concerns with noise and computational efficiency. Although researchers in this area tried to minimize the action cost by choosing useful heuristics, they did not discuss the bounds of the cost of the solutions that are found by their active strategies. Jensfelt *et al.* [15] proposed a strategy based on heuristics of going to the places with a maximum number of new features and avoiding getting to previous positions. In the method proposed by Gasparri *et al.* [9], the robot moves to the nearest obstacle in each step. Fox *et al.* [7] proposed an active strategy that chooses the action that minimizes the expected entropy in each step. Porta *et al.* [25] improved the computational cost of their method by using particle filters. Murtra *et al.* [4] proposed another method that minimizes the expected number of hypotheses instead of entropy at each step. The computational cost of their algorithm was less than the two previous algorithms that used entropy.

To solve the problem of self similarity, Fox *et al.* [7] put some relative target points among their actions. The path planning from the current states to the target points is done by the robot, but the target points themselves are given by the domain expert. In the method proposed by Murta *et al.* [4], some random target points were generated in a specific range from possible locations and the selection process was performed among those points. The range should be large enough to give the robot the ability to localize in all circumstances and, in the larger ranges, many target points are needed. This would increase the computational cost. The method we propose generates all macro actions automatically and specifies the whole path instead of just target points. Macro action generation is the subject of some work in related fields as well. Work by He *et al.* [11][13] and Kurniawati *et al.* [18] in planning under uncertainty and the work of McGovern [22] in reinforcement learning are examples of macro action generation. However, their approaches for generating macro actions and the reasons for using them are quite different from ours.

1.2 Planning under Uncertainty

Planning under uncertainty has numerous applications in areas such as target tracking, robot navigation and assistive technology [5][14]. For example, in a target tracking problem, the robot may be uncertain about target's exact position or its next move. As another example, a robot navigation problem can be an extension of localization in that a robot wants to reach a goal without knowing its current location (Figure 1.2). In these situations, the robot should meet a goal, e.g. catching the target or getting to a specific position, without being certain about the whole world where it is doing the task.

Planning under uncertainty is well modeled by the POMDP framework.

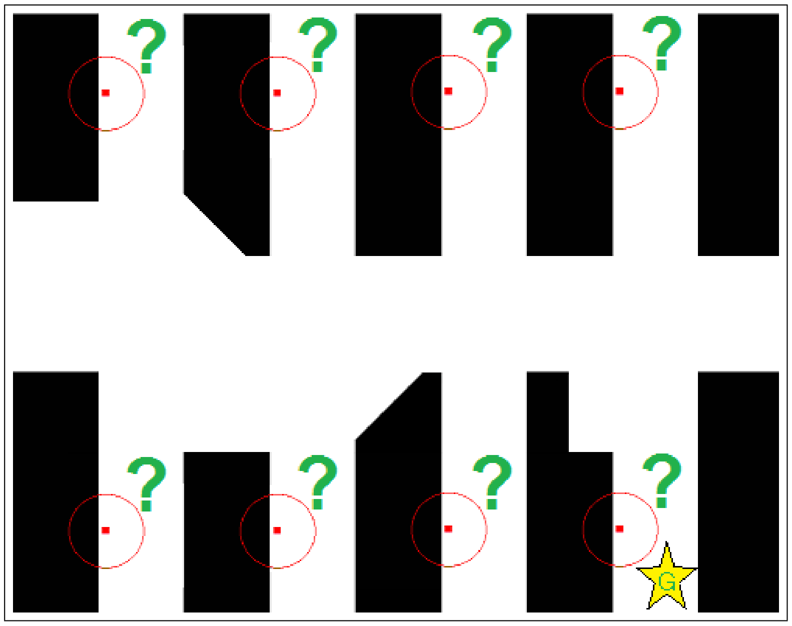


Figure 1.2: Our robot in the localization example wants to reach the star without knowing its current location. The location of the star, i.e. southeast of the map, may help the robot choosing its first actions.

There has been good progress in solving POMDP problems more efficiently in recent years. However, even the state of the art POMDP solvers are still too computationally expensive for large problems. In this thesis we propose a fast online approach for solving POMDPs. Our method achieves total reward close to or better than the total reward gained by state of the art POMDP solvers in much less time. This approach is a one step greedy strategy that uses the pairwise heuristic.

The pairwise heuristic is an approximation of the optimal plan for pairs of states. For each pair of states, we calculate the sequence of actions that resolve the uncertainty and gain the maximum reward, if the agent is uncertain about which one of the two states it is in. This calculation is done by running the value iteration method on an MDP (Markov Decision Process) whose states are pairs of states of the original problem. The whole process is independent of the initial belief state and so can be done only once for each domain. After obtaining the pairwise heuristic values, we use an online greedy strategy using those values to choose the optimal action at each step.

We have tested our method on large classical POMDP problems in the domains of robot navigation, target tracking and scientific exploration. The results are very promising. The resultant solution is near optimal while the time required is low. This is the first time that an online algorithm gains near optimal reward with only one step look ahead search. Other online approaches need to go deep in the search tree to get a good total reward and that increases their computational cost [28]. In addition to computational efficiency, unlike most of the POMDP solvers, the pairwise heuristic is simple to understand and implement.

1.2.1 Related Work

There have been many publications on planning under uncertainty in the last two decades. In one of the early works in 1994, Littman *et al.* [21] introduced the QMDP method. QMDP tries to find the optimal strategy assuming that the uncertainty is eliminated in the next step. This assumption, however, is not realistic especially in large problems. Therefore, QMDP does not usually work well. In 1996, Cassandra *et al.* [3] studied planning under uncertainty in the domain of robot navigation. They proposed a strategy that carries out localization and reward maximization simultaneously. The problem with this strategy is that it only considers single step actions. Localization, however, is sometimes not possible with only one action. One of the successful early methods is the grid based approach. Examples of this method are the work of Brafman in 1997 [1] and Poon in 2001 [24].

The point-based value iteration method was introduced by Pineau *et al.* [23]. Smith and Simmons [30], Kurniawati *et al.* [20] and Poupart *et al.* [26] also proposed algorithms that are based on point-based value iteration. These methods produce significantly better results in comparison with previous methods. However, they are computationally expensive and not suitable for large problems. This problem of scalability led some researchers to consider reducing the size of the problem by abstracting away some details and solving the reduced problem with a point-based solver. In 2005, Roy *et al.* [29] used PCA to reduce the size of the state space and then solve the resulted POMDP . Kaplow *et al.* [16] did this reduction in state space with variable resolution decomposition. Furthermore, He *et al.* [19] and Kurniawati *et al.* [12] proposed methods that ignored some observations and solved the POMDP by macro actions. We should note that the reduced POMDP in the work of He *et al.* is solved by forward search and not by a point-based method.

Point-based approaches are all offline, in that all the planning is done before execution of the strategy. Beside offline strategies, some researchers have worked on online approaches where planning and execution are interleaved. As online methods focus only on the current belief state, they scale better than offline approaches. However, the search tree in these methods has to be expanded enough to produce a good solution. This is problematic in domains with large action and observation spaces. A comprehensive survey on online POMDP solvers was provided by Ross *et al.* in 2008 [28].

1.3 Organization of This Thesis

This thesis is about using the pairwise heuristic on the two problems of localization and planning under uncertainty. Although the core concept of the heuristic, i.e., solving the problem for pairs of states, is the same in both problems, there are some differences between the heuristics of the two problems due to the problems' inherent differences. As a result, the method and the experiments are set forth separately for each problem. The thesis starts with the description of the problems we are focusing on in chapter 2. We explain our method for performing localization in chapter 3. Our algorithm for solving POMDPs is explained in chapter 4. Chapter 5 covers the experiments and the results. Finally, chapter 6 is dedicated to discussion and conclusions.

1.4 Contributions of This Thesis

As explained above, we have invented methods based on the pairwise heuristic for localization problem and planning under uncertainty. Our main contribution to localization is proposing an efficient and entirely autonomous method that can solve the localization problem in any environment. Also, our method can be used to determine whether or not a robot can localize itself in an environment. In the field of planning under uncertainty, our contribution is finding a method that solves POMDPs quite efficiently. We tested our method on large POMDP problems in different domains. The resulting total reward is close to, if not better than, the total reward obtained by other state-of-the the-art POMDP solvers while the time required to find the solution is much less.

Chapter 2

Problem Definition

As with any other concept in computer science, localization and planning under uncertainty should be studied in a mathematical framework. The framework must be both complete enough to depict the phenomena accurately and simple enough to predict the unknowns efficiently. However, these two criteria usually compete. The Markovian framework balances these two criteria quite well and as a result is a popular model in the field of Artificial Intelligence. The methods in this thesis are all based on this framework.

2.1 Localization

Localization is the problem of finding a robot's location in a given map. The main concept in localization is the *Belief* which means the robot's belief about its current location. There are different classes of localization methods based on their model for representing and updating the belief. The most important types of localization are: Markov localization, Monte Carlo localization, and extended Kalman filter localization, each with its own drawbacks and advantages [31]. Our method is based on Markov localization.

Localization consists of three steps: (i) action selection; (ii) performing the action and updating the belief based on that action; and (iii) observing and updating the belief based on that observation. First, we put aside step (i) and explain the other two steps assuming that the action has been selected. Next, we will go over step (i), i.e., the action selection.

In Markov localization [8], a map is a discrete set of states, S. The state of the robot at step k is x_k. The system is Markovian, in that the new state of the robot only depends on the previous state and the action it took in that state. In each state, the robot has an observation in a set of all possible observations, O. The observation at step k is z_k and the sequence of all observations up to step k is Z_k. A is the set of all possible actions which the robot can perform. Each action $a \in A$ in state s has a positive cost, $C(s, a)$. In addition, the action performed by the robot at step k is u_k and the sequence of all actions performed up to step k is U_k . A Graphical

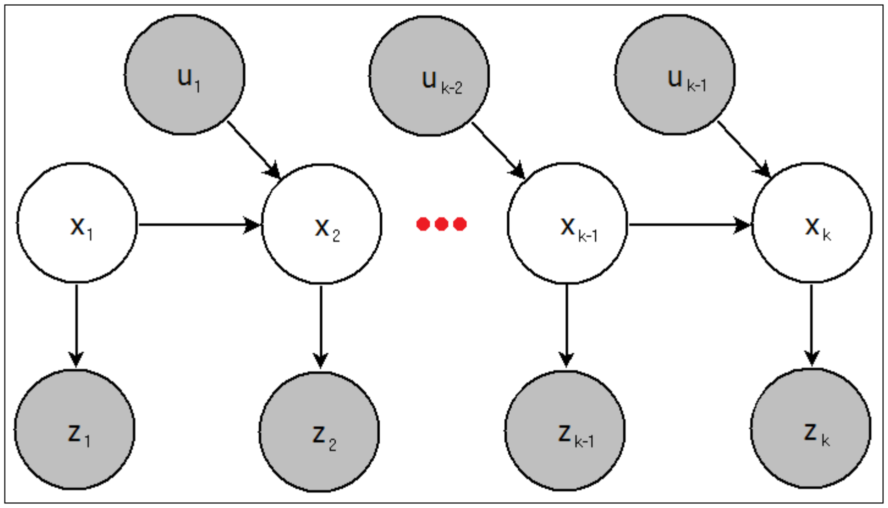


Figure 2.1: Graphical model of Markov localization.

Model of Markov localization is illustrated in Figure 2.1. The shaded nodes are known.

During localization, the robot's belief about being in state s is $Bel(x_k = s)$.

$$Bel(x_k = s) = p(x_k = s|U_{k-1}, Z_k) \tag{2.1}$$

Since the system is Markovian, this probability is:

$$Bel(x_k = s) = p(x_k = s|u_{k-1}, x_{k-1}, z_k) \tag{2.2}$$

Suppose the robot is in x_{k-1} when it performs action u_{k-1}. The belief states will be updated as follows. In this step, belief is shown by B^- because only u_{k-1} and x_{k-1} are considered. B^- does not include z_k.

$$Bel^-(x_k = s) = \sum_{s'} p(x_k = s|x_{k-1} = s', u_{k-1})Bel(x_{k-1} = s') \tag{2.3}$$

Since $Bel^-(x_k)$ depends on $Bel(x_{k-1})$, $Bel(x_1)$ should be given. If the robot has some prior knowledge about its location, $Bel(x_1)$ is defined. On the other hand, in the case that the robot knows nothing, the probability density of $Bel(x_1)$ is uniform across all states. That case is called global localization.

After performing the action, when the robot observes z_k, the belief states will be updated, using Bayes' theorem:

$$Bel(x_k = s) = p(z_k|x_k = s) \times Bel^{-}(x_k = s)/p(z_k) \tag{2.4}$$

These updates occur in steps (ii) and (iii) defined above. Even with a random strategy in the action selection step, the robot may localize itself after some actions. However, this accomplishment is not guaranteed, especially in self similar environments. In self-similar environments, the robot may need to perform a sequence of actions instead of just one to observe a change in the belief state. Even if the robot succeeds in the localization task, it may take many actions with far higher cost than the optimal strategy. Finding the optimal strategy is NP-hard [6], so the best alternative is to find a near-optimal strategy for action selection to reduce the cost as much as possible.

2.2 Planning under Uncertainty

In the planning under uncertainty problem, an agent wants to get the maximum discounted reward by performing a sequence of actions, while being uncertain about its state. This problem is modeled by the POMDP framework. There is a simpler problem in planning where the agent knows its exact state. This problem is modeled by MDP (Markov Decision Process). Although our main problem is POMDP, we need to explain MDPs as well because we solve an MDP in the process of finding our heuristic. Solving an MDP is computationally much cheaper than solving a POMDP with the same set of states and actions.

2.2.1 MDP

Formally, an MDP is represented as a tuple (S, A, T, R, γ) where S is the finite set of states, A is the finite set of possible actions, and $T : S \times A \times S \to [0, 1]$ is the transition function, defining $p(s'|s, a)$ for all $s, s' \in S$ and $a \in A$. The system is Markovian, meaning the state in each step depends only on the previous state and the most recent action.

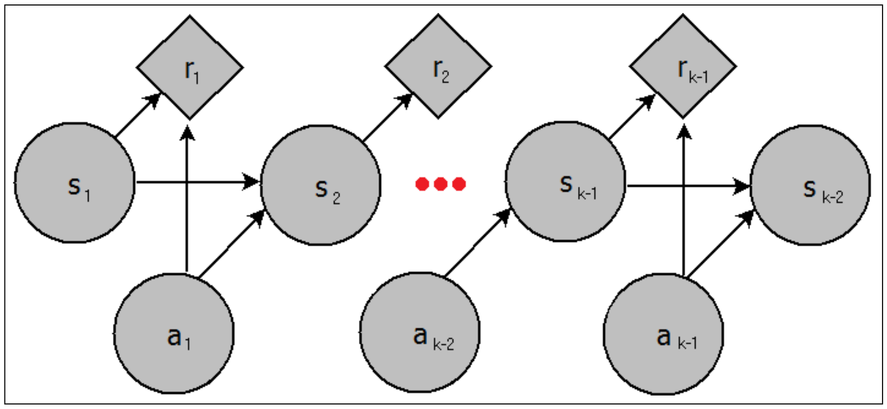


Figure 2.2: Graphical model of an MDP.

$$T(s, a, s') = p(s_{k+1} = s' | s_k = s, a_k = a) \tag{2.5}$$

$R : S \times A \rightarrow R$ specifies the reward for performing each action in each state. Finally, γ is the discount factor, a positive value smaller than 1. The goal is to choose a sequence of actions to maximize the expected total reward, $E[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$. Graphical model of an MDP is shown in Figure 2.2.

2.2.2 Value Iteration

Value iteration is a method to solve an MDP. First, we need to define two terms, policy and value function. The policy function is simply a map from states to actions, $\pi : S \rightarrow A$, defining what action the agent should perform in each state. The total reward of a policy is called the value function. An optimal policy is a policy which gains the maximum reward and its reward is the optimal value function. In the planning with infinite horizon, which is our case, the optimal value function converges to equilibrium:

$$V(s) = \gamma max_a[r(s, a) + \sum_{s' \in S} V(s')p(s'|s, a)] \tag{2.6}$$

This equation is called the *Bellman equation* [31]. The optimal value functions can be estimated by the value iteration method. In this method all value functions are set to the minimum reward, $r_{min} = min_{s,a} r(s, a)$ and get updated by the Bellman equation. Updating continues until the values converge. The value iteration algorithm is shown as Algorithm 1.

Algorithm 1: Value Iteration Algorithm

```
   Data: $(S, A, T, R, \gamma)$
   Result: $V(s)$ for all $s \in S$
1  foreach $s \in S$ do
2      $V(s) = r_{min}$
3  repeat
4      foreach $s \in S$ do
5          $V(s) = \gamma max_a[r(s,a) + \sum_{s' \in S} V(s')p(s'|s,a)]$
6  until convergence
```

2.2.3 POMDP

A POMDP contains all the sets and functions of an MDP with the same definition and also some additional concepts. A POMDP is represented as a tuple $(S, A, O, T, Z, b_0, R, \gamma)$ where S, A, T, R and γ are exactly defined as in MDP. O is the finite set of possible observations and $Z : S \times A \times O \to [0,1]$ specifies the probability of each observation after entering into a state by an action, $p(o|s,a)$ for all $o \in O$, $s \in S$ and $a \in A$ (note that s is the posterior state). b_0, $S \to [0,1]$, is the initial belief state, the probability distribution over possible initial states.

$$Z(s,a,o) = p(o_k = o | s_{k+1} = s, a_k = a) \tag{2.7}$$

$$b_0(s) = p(s_0 = s) \tag{2.8}$$

As the system is Markovian, the belief state in each step depends only on the previous belief state and the most recent action and observation.

$$b_{k+1}(s) \propto Z(s, a_k, o_k) \sum_{s'} T(s', a_k, s) b_k(s') \tag{2.9}$$

As with an MDP, the goal is to choose a sequence of actions to maximize the expected total reward, $E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)]$. Figure 2.3 shows graphical model of POMDP.

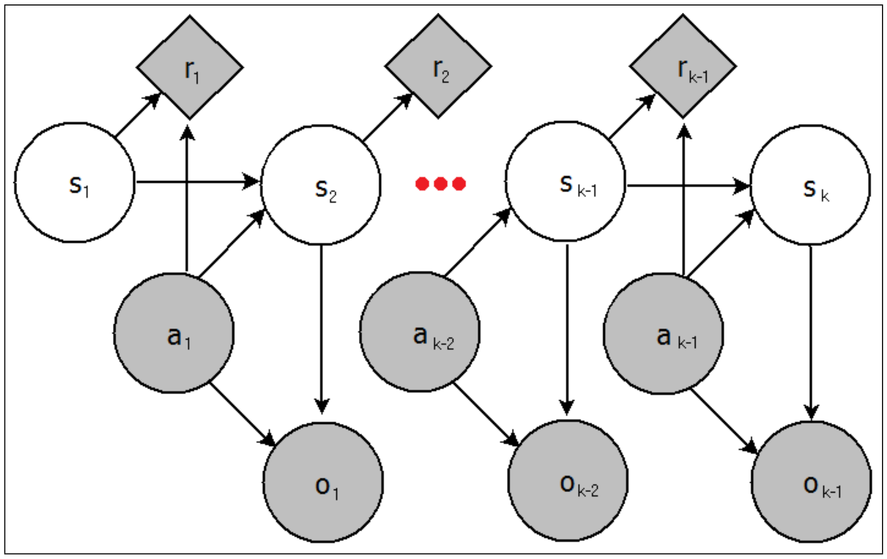


Figure 2.3: Graphical model of POMDP.

2.2.4 Solving POMDPs

It is possible to apply value iteration on a POMDP. However, as the states are not observable, instead of a value function of the states, a value function of the belief states should be calculated:

$$V(b) = \gamma max_a[r(b,a) + \int V(b')p(b'|b,a)] \tag{2.10}$$

This time, we are dealing with belief state space instead of original state space. The number of different cases in belief state space grows exponentially from the very first iterations. This makes the algorithm computationally very expensive. In fact, it is impossible to use value iteration for a POMDP with more than a few states. So, we need to find another alternative to estimate the optimal policy. Point-based value iteration approaches [23][30][20][26], grid based methods [24][1] , and heuristics [3] are some of the alternatives to solve POMDPs. We discussed these approaches Chapter 1.

Chapter 3

The Pairwise Heuristic for Localization

We propose an active strategy based on the pairwise heuristic for action selection in localization. In addition to the variables that should be defined in Markov localization, we need to define additional variables based on those ones for our action selection strategy. In this chapter, we will first define the required variables for action selection. Then, we will explain our strategy and the intuition behind using the pairwise heuristic for localization. Finally, a method to deal with self similar environments is introduced and binded to our active strategy.

3.1 Additional Variables

Considering the previous variables introduced in Chapter 2 for Markov localization, the new variables are: the observation difference between two states, the transition function between states assuming the actions are deterministic, and the cost of an action when we deal with two states instead of one.

$d(s, s')$ is the *observation difference* between two states s' and s' calculated as:

$$d(s, s') = \frac{1}{2} \sum_{o \in O} [p(o|s)(1 - p(o|s')) + p(o|s')(1 - p(o|s))] \tag{3.1}$$

Two states s and s' are *distinguishable* if and only if $d(s, s')$ is greater than a threshold γ. We should set a suitable value for γ based on the noise in the observations.

Observation difference is a number in $[0, 1]$. When observation o has a high probability in s and a low probability in s', $p(o|s).(1 - p(o|s'))$ would be close to 1 and when o has high probability in both states, $(1 - p(o|s'))$ is very small and as a result $p(o|s).(1-p(o|s'))$ is close to 0. So for the pairs with similar high probability observations $d(s,s')$ is close to 0 but for the states with different observations, it would be close to 1. Thus, when $d(s,s')$ is above a threshold, for most of the expected observations s and s' can be distinguished from each other with high probability.

Let $f^*(s,a)$ be the state that the robot goes to from state s with action a in the noiseless case. We can assume that this state is the most probable state that the robot goes to:

$$f^*(s,a) = argmax_{s'} p(s'|s,a) \tag{3.2}$$

If there is more than one s' with maximum probability, we choose one of them arbitrarily.

Last, we define the cost of an action for a pair of states:

$$C(s,s',a) = max(C(s,a), C(s',a)) \tag{3.3}$$

3.2 Explanation of Algorithm for Action Selection

The original idea for the strategy we are using comes from work in the field of active learning by Golvin *et al.* [10]. They called their algorithm EC^2 (Equivalence Class Edge Cutting). To explain our strategy, we should define a graph. The vertices of this graph are the states and the edges are the actions. Two states s and s' are connected to each other by action a if and only if $f^*(s,a)$ and $f^*(s',a)$ are distinguishable. The weight of the edge is:

$$\begin{aligned} w(s,s',a) = &Bel(s) \times Bel(s') \times \\ &min(p(f^*(s,a)|s,a), p(f^*(s',a)|s,a))/C(s,s',a) \end{aligned} \tag{3.4}$$

If the robot assigns zero belief to either of the two states, the weight of the edge would be 0, so there is no need to calculate the weight. We only consider the edges between states with positive belief functions.

For each action a, $w(a)$ is:

Algorithm 2: Action selection with basic actions

```
input  : Belief states $Bel(.)$, actions $A$ and their costs $C$, observation
         differences of all pairs $d$, and transition function $f^*$
output: $a^*$, the action to be performed by the robot
1 foreach $a \in A$ do $w(a) = 0$
2 foreach $s, s', a$ do
3     if $d(f^*(s,a), f^*(s',a)) > \gamma$ then $w(s,s',a) =$
      $Bel(s) \times Bel(s') \times min(p(f^*(s,a)|s,a), p(f^*(s',a)|s',a))$
      $/C(s,s',a)$
4     else $w(s,s',a) = 0$
5 foreach $s, s', a$ do $w(a) = w(a) + w(s,s',a)$
6 Select $a^* = argmax_a w(a)$
```

$$w(a) = \sum_{s,s'} w(s,s',a) \tag{3.5}$$

In each step of action selection, we choose the action with the highest assigned weight:

$$a^* = argmax_a w(a) \tag{3.6}$$

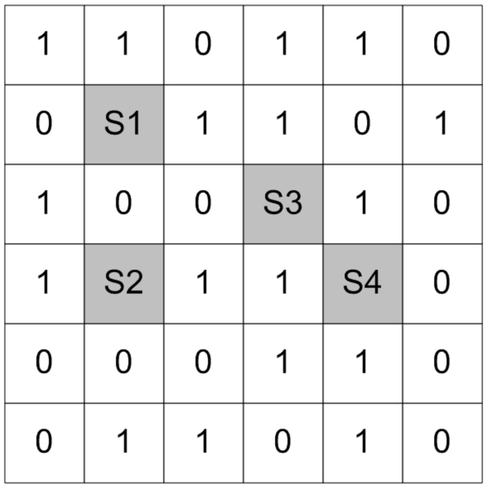
The algorithm is shown as Algorithm 2.

3.3 Intuition

The intuitions behind the concept of weight and the action selection criteria are explained most easily by an example. Each edge in the graph means the action assigned to the edge can remove at least one of the two vertices from belief states (not the states themselves but their subsequent states, f^*). So, the action with highest weight has most power to distinguish between states, especially the ones with higher probability.

Consider the map in Figure 3.1. The robot is initially in one of the shaded squares, $s1$ to $s4$, and the beliefs of the robot are:

$$Bel(s1) = 0.2$$



1	1	0	1	1	0
0	S1	1	1	0	1
1	0	0	S3	1	0
1	S2	1	1	S4	0
0	0	0	1	1	0
0	1	1	0	1	0

Figure 3.1: Example of localization for intuition, the map and the observations

$$Bel(s2) = 0.2$$
$$Bel(s3) = 0.1$$
$$Bel(s4) = 0.5$$

We have an observation in each square, 0 or 1, as shown in the map. In addition, the observation in s1 to s4 is 0. We have four actions: up, down, right and left, all with unit costs. The actions and observations are noise free. With these assumptions we would like to localize the robot with this active strategy myopically. Which action should be chosen next? To answer this question we build the graph (Figure 3.2). For example, if the robot performs *up*, it will observe 1 if it is in $s1$ and it will observe 0 if it is in $s2$. So after performing *up* at least one of these states would be removed. The weights of the actions are:
$w(up) = Bel(s1) * Bel(s2) + Bel(s2) * Bel(s3) + Bel(s2) * Bel(s4) = 0.16$
$w(down) = Bel(s1) * Bel(s3) + Bel(s1) * Bel(s4) + Bel(s2) * Bel(s3) + Bel(s2) * Bel(s4) = 0.24$
$w(right) = Bel(s1) * Bel(s4) + Bel(s2) * Bel(s4) + Bel(s3) * Bel(s4) = 0.25$
$w(left) = Bel(s1) * Bel(s2) + Bel(s1) * Bel(s4) + Bel(s2) * Bel(s3) + Bel(s3) * Bel(s4) = 0.21$
So *right* would be chosen. After performing this action either the robot would know its exact position by observing 0 or a state with high probability would be removed from the belief states. In the latter case, as the state was not a correct hypothesis, this elimination is a significant achievement. On

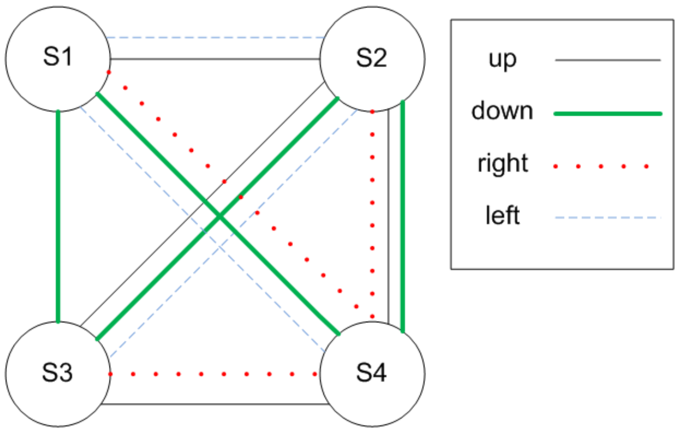


Figure 3.2: Example of localization for intuition, the graph that is built in our strategy

the other hand, if the robot performs *up*, with the chance of 0.8 it would only eliminates a state with probability of 0.2. No matter what the robot observes, actions *down* and $left$ reduce the number of states with positive probability to 2. This reduction is a good improvement, but the high prior probability of $s4$ leads the algorithm to choose *right*.

The presence of $min(p(f^*(s,a)|s,a), p(f^*(s',a)|s',\ a))$ in the formula, is a way to deal with noise. An action with greater noise has less chance of being selected because the states would be removed from belief set only if the action is performed accurately. We also consider the cost of the action in our formula, because our goal is to perform the localization task with the least possible cost.

In the active learning field, Golovin *et al.* proved that the cost of the policy that is generated by EC^2 is near optimal and its bound is:

$$C(\pi_{EC^2}) \leq (2\ln(1/p_{min}) + 1)C(\pi^*) \tag{3.7}$$

In their problem, the algorithm tries to select the correct hypothesis among many by choosing a subset of tests with minimum cost from a set of all possible tests. π^* is the optimal policy and p_{min} is the probability of the least probable hypothesis. In that problem tests are performed without noise and the weights only contain probabilities of two hypotheses and their cost [10].

3.4 Macro Actions

There are many situations where, because of the symmetry in the environment, none of the actions are useful for localization. In these cases, we need a sequence of actions instead of a single basic action. In this section we propose an algorithm that generates minimum cost sequences of actions that solve the problem of localization no matter where the robot is. We call these sequences macro actions.

Our strategy of action selection is based on the actions that distinguish between each of a pair of states. So if there is a situation where no action can be selected, it means that there is no action for any pair of states. Therefore, we build the macro actions based on these pairs. In other words, for any pair of states, we find a sequence of actions that can distinguish between them. In generating these macro actions the actions are assumed to be deterministic, so only f^* is used.

3.4.1 The Macro Action Generation Algorithm

Algorithm 3 generates macro actions. For each pair of states, we should find a macro action that can resolve the localization problem. This macro action is called $macro\_action(s, s')$ and its cost is $C(macro\_action(s, s'))$. First we set $macro\_action$ empty for all pairs. Then, for each pair of s and s' that are distinguishable, $macro\_action(s, s')$ is set to zero with zero cost, which means that without any action we can distinguish between them. In the next step, we look at each pair with an undefined macro action (Line 11) and for each basic action a we check whether $macro\_action(f^*(s, a), f^*(s', a))$ is defined or not (Line 12). If it is, we can build a path for s and s' by adding a to the head of $macro\_action(f^*(s, a), f^*(s', a))$ (Line 24). The cost of the new path would be $max(C(s, a), C(s', a)) + C(macro\_action(f^*(s, a), f^*(s', a)))$ (Line 25). However, we do not set this macro action at the first step. We merely calculate the cost for all pairs and only set the macro actions of those with minimum costs (Line 23). After the first step of calculating new macro actions for all pairs and setting the minimum sequences, we repeat this step and recalculate the paths, find minimum ones (Line 14 and Line 18) and update them (Lines 23-26). We repeat this step until no macro action can be built. Most of the time this would happen because we have already built the macro actions for all of the pairs, but there may be situations when there exist two states for which we cannot find a sequence. In this situation, we can say that if the robot is in one of these states, it can never localize itself. In fact, our algorithm can be used as a test to see whether a robot can always localize itself in the map.

Algorithm 3 generates exactly one action sequence for each pair of states. This sequence is optimal in deterministic environments and near-optimal in noisy environments. The proof of optimality of these sequences in deterministic environments is given in the Appendix.

3.5 Performing Active Localization with Macro Actions

The localization algorithm with macro actions, Algorithm 4, is quite similar to Algorithm 2. This time, we consider not only the basic actions of the robot, but also macro actions that were generated previously. To consider the macro actions in the graph we should define f^* and the cost for them. If we have a macro action a' that is a sequence of basic actions $\langle a_1, a_2, ..., a_l \rangle$, then $f^*(s, a')$ is calculated by this recursive formula:

$$f^*(s, \langle a_1, ..., a_l \rangle) = f^*(f^*(s, \langle a_1, ...a_{l-1} \rangle), a_l) \tag{3.8}$$

The cost of this macro action for states s and s' is calculated the same way we calculate costs in generating macro actions:

$$\begin{aligned} C(s, s', \langle a_1, ...a_l \rangle) =& max(C(s, a_1), C(s', a_1)) + \\ & C(f^*(s, a_1), f^*(s', a_1), \langle a_2, ..., a_l \rangle) \end{aligned} \tag{3.9}$$

Also, $p(f^*(s, a')|s, a')$ is obtained by the following formula:

$$\begin{aligned} & p(f^*(s, \langle a_1, ..., a_l \rangle)|s, \langle a_1, ..., a_l \rangle) = p(f^*(s, a_1)|s, a_1) \times \\ & p(f^*(f^*(s, a_1), \langle a_2, ..., a_l \rangle)|f^*(s, a_1), \langle a_2, ..., a_l \rangle) \end{aligned} \tag{3.10}$$

The other parts of the algorithm are exactly the same. The algorithm is shown as Algorithm 4.

The whole localization task consists of two main parts, i.e. offline computation and online computation. Offline part is performed only once for each environment but the online computation should be executed each time the robot wants to localize itself. The main algorithm in offline part is Algorithm 3. The online computation is performing Algorithm 4 iteratively until the robot finds its actual location. Offline and Online computations are discussed in detail in Chapter 6.

Algorithm 3: Macro actions Generation

```
input : Set of states $S$, actions $A$ and their costs $C$, observation
        differences of all pairs $d$, and transition function $f^*$
output: Macro actions for all pairs of states
1  $MAX$ = very large value
2  foreach $s, s'$ do $macro\_action(s, s') = \emptyset$
3  foreach $s, s'$ that $d(s, s') > \gamma$ do
4      $macro\_action(s, s') = 0$
5      $C(macro\_action(s, s')) = 0$
6  $min\_set = \{(s_1, s_2, a_1)\}$ //set this value just to enter the while loop
   at first
7  while $min\_set \neq \emptyset$ do
8      $min\_set = \emptyset$
9      $min\_cost = MAX$
10     foreach $s, s', a$ that $macro\_action(s, s') = \emptyset$ do
11         if $macro\_action(f^*(s, a), f^*(s', a)) \neq \emptyset$ then
12             if $C(s, s', a) + C(macro\_action(f^*(s, a), f^*(s', a))) =$
               $min\_cost$ then
13                 add $(s, s', a)$ to $min\_set$
14             if $C(s, s', a) + C(macro\_action(f^*(s, a), f^*(s', a)))$
               $< min\_cost$ then
15                 Set $min\_set$ empty
16                 add $(s, s', a)$ to $min\_set$
17                 $min\_cost =$
                   $C(s, s', a) + C(macro\_action(f^*(s, a), f^*(s', a)))$
18     foreach $(s, s', a)$ in $min\_set$ do
19         $macro\_action(s, s') = \langle a, macro\_action(f^*(s, a), f^*(s', a)\rangle$
20         $C(macro\_action(s, s')) = min\_cost$
```

Algorithm 4: Action selection with macro actions

```
input  : Belief states $Bel(.)$, actions $A$ and their costs $C$, observation
         differences of all pairs $d$, and transition function $f^*$, and
         macro actions
output: $a^*$, the action or the sequence of actions that should be
         performed by the robot
1 $A' = \emptyset$
2 foreach $s, s'$ that $Bel(s) \times Bel(s') > 0$ do  add $macro\_action(s, s')$ to
  $A'$
3 foreach $a' \in A'$ do $w(a') = 0$
4 foreach $s, s', a'$ do
5     if $d(f^*(s, a;), f^*(s', a')) > \gamma$ then $w(s, s', a') =$
      $Bel(s) \times Bel(s') \times min(p(f^*(s, a')|s, a'), p(f^*(s', a')|s', a'))$
      $/C(s, s', a')$
6     else $w(s, s', a') = 0$
7 foreach $s, s', a'$ do $w(a') = w(a') + w(s, s', a')$
8 Select $a^*$ which $a^* = argmax_{a'} w(a')$
```

Chapter 4

The Pairwise Heuristic for Planning Under Uncertainty

Our approach to solving a POMDP is a one step greedy search using the pairwise heuristic. In this chapter, we first explain the intuition behind using the pairwise heuristic for planning under uncertainty. Then, the pairwise heuristic is explained in detail. Finally, we explain the greedy strategy that uses the heuristic.

4.1 Intuition

In an MDP, the only goal is maximizing the reward. But when we deal with uncertainty, information gathering and ambiguity resolution must be considered as well. Actually, a POMDP solver implicitly does information gathering and reward maximization simultaneously. Cassandra *et al.* [3] and He *et al.* [12] used this fact in developing their approaches and we use it as well. We use the term *localization* with a little change to its classical meaning to explain our approach in gathering information. As mentioned before, localization means finding the robot's current state [31], where the state is the robot's position. Similarly, in our approach localization means finding the actual current state in the problem. But this time, the state is more general than the agent's position. For example, in the target tracking problem, the position of the agent and the target are both part of the state.

Let us explain the intuition with a simple example. Consider the map shown in Figure 4.1. A robot wants to go to the goal state, cell G, knowing that it is in cell A or cell B with equal probability at first. The actions are deterministic. The observation in all of the states except cells C, D, E and F is o_0. In cells C and E the observation is o_1 and in cells D and F the robot observes o_2. These observations are completely different, that is $o_0 \neq o_1, o_1 \neq o_2, o_0 \neq o_2$. How can the robot reach the goal state in the minimum number of steps? If there were no uncertainty and the robot knew it was in A or knew it was in B, it could reach the goal in only 4 steps.

But with uncertainty, going left or right doesn't help the robot. It should determine its exact position while traveling to the goal state. If it goes down for 3 steps it would then know its exact position and reach the goal in 3 steps after that. So it would reach the goal in 6 steps. Alternatively, it could go up for two steps and find itself. After those 2 steps, the robot could reach the goal in 6 steps. So this policy takes 8 steps and is worse than the previous one, even though it removes the uncertainty sooner. In this robot navigation problem the solution contains two parts, a path before localization and a path after that. In the optimal strategy, the aggregate of those two paths should be minimal. Actually, this structure is nearly the same in the general robot navigation problem with uncertainty. In nearly all of the robot navigation problems that deal with uncertainty, the robot should remove the uncertainty during its path toward the goal and the path should be optimal. Finding the solution for this problem is difficult if the robot is uncertain about being in many states. But if the uncertainty is limited to only two states, the problem is not that hard. We can solve the problem for every pair of states and then use these solutions to solve the main problem. In addition we can generalize this solution to cover all problems in planning with uncertainty. Instead of minimizing the number of steps for the paths to achieve localization and after localization, we find the reward maximizing path for localization and after localization for each pair of states and then use these paths as a heuristic for the main problem.

4.2 The Pairwise Heuristic

As explained above, the heuristic needs an optimal sequence of actions for each pair. We call the reward of this optimal sequence, the value function of the pair. To explain how to find these sequences, we assume that we only want to do the localization task for each pair at first. A similar method to our approach for localization in previous chapter is used for doing this task. We use *distinguishable* pairs again, i.e. the pairs that do not need localization. In fact, there is a little difference in defining distinguishable states in the new approach in comparison to previous one as the world model is more complex in a POMDP. For other states, we can carry out the localization by going to distinguishable pairs. After the localization, the current state is determined and to fulfill the reward maximization goal we just need to solve the problem in the MDP framework. One could argue that as the states are partially observable and the actions are not deterministic, the uncertainty about the state may arise again. In this situation we do the localization task

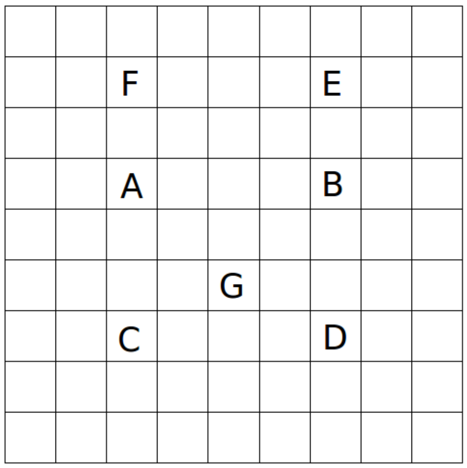


Figure 4.1: The map of the example for intuition. The robot wants to go to cell G while being uncertain about its initial position: cell A or cell B. The localization information is in cells C, D, E and F.

again. This task is further explained in the next sections of the chapter.

As the observation depends on both the state and the action that leads to that state, instead of finding distinguishable states, we find states distinguishable by an action. Two states are distinguishable by an action if, after performing that action, there is a high probability that different observations are recorded in the two states. Formally, s and s' are distinguishable by action a if and only if:

$$o^* = argmax_o p(o|s'', a)$$
$$o'^* = argmax_o p(o|s''', a)$$

$$\sum_{s'',s'''} p(s''|s,a)p(s'''|s',a)[p(o^*|s'',a)(1-p(o^*|s''',a))+ \\ p(o'^*|s''',a)(1-p(o'^*|s'',a)] \geq 2\lambda \qquad (4.1)$$

λ is a constant that is specified by a domain expert. If it is 1, the observations should be completely different. But, as in the localization problem where a robot is usually considered localized if the probability of an state is more than a threshold like 0.95, we can set this threshold to a value less than 1 in noisy environments. As shown in the formula the observations that are considered are the most probable observations of the posterior states. If there is more than one observation with maximum probability, one would be chosen arbitrarily.

For states s and s' distinguishable by action a the value is set to:

$$V(s,s') = 0.5[R(s,a) + R(s',a) + \gamma \times (V(s) + V(s'))] \tag{4.2}$$

$V(s)$ and $V(s')$ are the value functions of s and s' in the underlying MDP. Also, $u(s,s')$, the optimal action of s and s' is set to a. In the case that there are more than one action that can distinguish between two states, the one with maximum average reward $(0.5[R(s,a) + R(s',a) + \gamma \times (V(s) + V(s'))])$ would be chosen.

To find the value function and optimal action for indistinguishable pairs, we use a value iteration algorithm in an MDP where the states are pairs of states of our original problem. The transition function is determined as follows:

$$s^* = argmax_{s''} p(s''|s,a)$$
$$s'^* = argmax_{s''} p(s''|s',a)$$

$$p((s^*, s'^*)|(s,s'),a) = 1 \tag{4.3}$$

$$\forall s'', s''' : s'' \neq s^* \vee s''' \neq s'^* : p((s'', s''')|(s,s'),a) = 0 \tag{4.4}$$

The equations above show that we ignore the noise of actions in the new MDP and consider only the most probable posterior states. Again, if there is more than one most probable state, one would be chosen arbitrarily. Also the reward, $R((s,s'),a)$ is equal to $0.5[R(s,a) + R(s',a)]$ where $R(s,a)$ and $R(s',a)$ belong to the original problem. We run the value iteration only for indistinguishable pairs. The initial value function for these pairs is set to the minimum reward in the main problem. Actions are the same as actions in the original problem. In addition, the discount factor of the new MDP is the same as the discount factor of the original problem. This algorithm is shown as Algorithm 5.

As shown above, we use the value of 0.5 in all of the equations for value functions which gives the unweighted average. This means we assume equal probability for the two states in computing their optimal action and value function. The states may not have equal probability in the original problem, but the pairwise value function is used as a heuristic and does not need to be exact. By using this simplification, obtaining value functions and optimal actions does not depend on the initial belief state and would be an offline calculation. So for each domain, we only need to run this algorithm once.

Algorithm 5: Finding the value functions and optimal actions for the pairs

```
   Data: $(S, A, O, T, Z, R, \gamma)$
   Result: $V(s, s')$ and $u(s, s')$ for all pairs
1  Calculate value functions, $V(S)$ of $MDP(S, A, T, R, \gamma)$
2  foreach pair $(s, s')$ do $V(s, s') = R_{min}$
3  foreach pair $(s, s')$ and action $a$ do
4      $R((s, s'), a) = 0.5[R(s, a) + R(s', a)]$
5  foreach pair $(s, s')$ and $(s'', s''')$ and action $a$ do
6      $p((s'', s''')|(s, s'), a) = 0$
7      $s^* = argmax_{\hat{s}} p(\hat{s}|s, a)$
8      $s'^* = argmax_{\hat{s}} p(\hat{s}|s', a)$
9      $p((s^*, s'^*)|(s, s'), a) = 1$
10 foreach pair $(s, s')$ distinguishable by action $a$ do
11     $V(s, s') = 0.5[R(s, a) + R(s', a) + \gamma(V(s) + V(s'))]$
12     $u(s, s') = a$
13 repeat
14     foreach indistinguishable pair $(s, s')$ do
15         $V_k(s, s') =$
           $max_a[R((s, s'), a) + \gamma \sum_{s'',s'''} V(s'', s''')p((s, s')|(s'', s'''))$
16         $u(s, s') =$
           $argmax_a[R((s, s'), a) + \gamma \sum_{s'',s'''} V(s'', s''')p((s, s')|(s'', s'''))$
17 until convergence
```

4.3 The Greedy Strategy

To solve the POMDP, we only need a one step greedy strategy that uses the value functions of the pairs. In each step, the selected action should maximize the expected total value function of the pairs. However, the expected instant reward of the actions should be considered as well. We ignore the noise of actions in the greedy strategy. As a result the selected action is:

$$s^* = argmax_{s''} p(s''|s, a)$$
$$s'^* = argmax_{s''} p(s''|s', a)$$

$$a_k^* = argmax_a \sum_{s,s'} [(0.5(R(s, a) + R(s', a)) + \gamma V(s^*, s'^*))b_k(s)b_k(s')] \quad (4.5)$$

Algorithm 6: Choosing the optimal action in step k

```
    Data: $(S, A, O, T, Z, b_k, R, \gamma)$, compare ratio
    Result: Optimal action, $a_k^*$
1   maxBel = $max_s b_k(s)$
2   $S' = \{s | b_k(s) \geq$ maxBel/compare ratio$\}$
3   $A' = \{a | a = u(s, s')\ s, s' \in S'\}$
4   if $|S'| = 1$ then
5       $a_k^*$ = optimal action of $S'$ in the MDP
6   else
7       foreach $a \in A'$ do
8           foreach $s \in S'$ do $s^* = argmax_{s'} p(s'|s, a)$
            $H(a) = \sum_{s,s'}[(0.5(R(s, a) + R(s', a)) + \gamma V(s^*, s'^*))b_k(s)b_k(s')]$
9       $a_k^* = argmax_{a \in A'} H(a)$
```

We should note that maximization is not done over all possible actions and the selected action should be the optimal action for at least one pair of states. Also, one may argue that using value functions of the pairs is a kind of localization strategy (to be precise both localization and reward maximization) and the algorithm may get stuck in localization and never collect rewards. This is, in fact, true; so to resolve this issue only the states with probability greater than a specified threshold are considered in the heuristic. This threshold is relative and is equal to the probability of the most likely state divided by a constant greater or equal to 1 which is specified by the domain expert. This constant is called the *compare ratio.* If in one of the steps of the planning the probabilities of all states, except the most likely one, become less than the threshold, the selected action would be the optimal action of the underlying MDP for that most likely state in that step. Obviously, as the compare ratio is greater than or equal to 1, the probability of the most likely state is always above threshold. The whole strategy is shown as Algorithm 6.

The whole algorithm for solving the POMDP contains two main computations, i.e. offline and online. The offline part, Algorithm 5, is executed only once for each problem no matter what the belief state is. On the other hand, the online part, which is performing Algorithm 6 iteratively, should be performed every time we want to solve the problem. We shall discuss these two computations in detail in Chapter 6.

Chapter 5

Experiments and Results

For our experiments, we use test benchmarks and methods that test our main contributions to the field. In localization our main contribution is accomplishing the task in self-similar environments. The most important part of our algorithm that helps us in these environments is the part where we use macro actions. As a result, our test benchmark in localization is a self-similar environment. We tested the pairwise heuristic both with and without using macro actions to show the advantage of using macro actions. In planning under uncertainty, our main contribution is the computational efficiency. Therefore, our test benchmarks are large problems. Also, we include the results of the state of the art POMDP solvers in our reporting to show the efficiency of the pairwise heuristic.

5.1 Experiments on Localization

We simulated a robot with a noisy laser range finder in a $30m \times 18m$ self similar environment, shown in Fig. 5.1. It has long corridors ($14m$) that are usually the main cause of similarity in real environments for laser range finder robots. The angular resolution of the range finder is $1°$ and its maximum angular range is $240°$. The maximum laser range is $10m$. The basic actions are going $20cm$ forward, backward, left and right and also turning $5°$ clockwise or counterclockwise, all with unit costs. All actions are performed accurately only 85% of the time. The grid size for the state space is $(20cm, 20cm, 5°)$ and as a result, there exist 2,881,196 pairs of indistinguishable states.

Our macro action generation method generated 6,073 macro actions with lengths of 1 to 31 for this environment. To show the effect of macro actions, we tested our method with and without macro actions. In the case where we only used basic actions, when there was no action able to reduce uncertainty, a random basic action was generated. In both tests the robot is considered localized, if and only if its certainty about its position is greater than 0.95 or the number of basic actions performed exceeds 500. We ran 1000 trials of global localization and tested both strategies in each trial. In each trial,

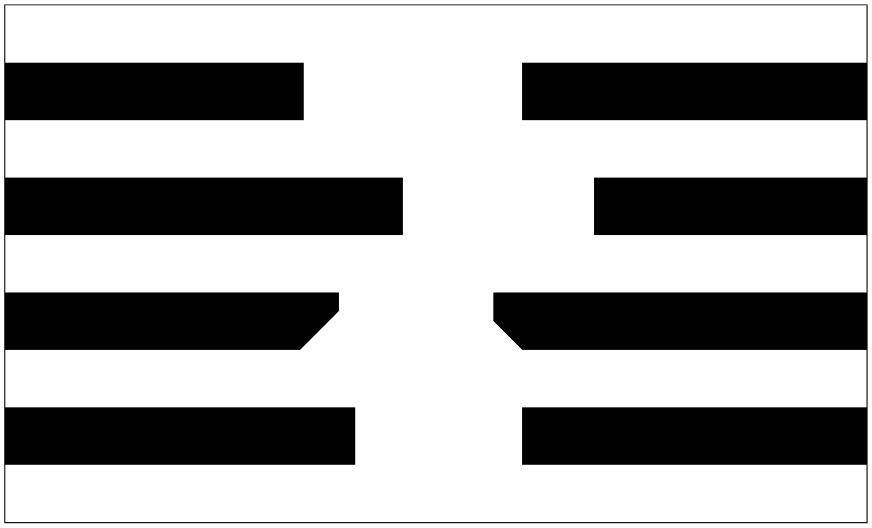


Figure 5.1: The map of the self similar environment used in the experiment. The environment is $30m \times 18m$. The width of the corridors is $2m$ and their maximum length is $14m$.

Table 5.1: Fraction accomplished in the localization task with fewer than 500 basic actions with and without macro actions

Method	Fraction accomplished
Macro actions	100%
Basic actions and random walk	86.2%

the initial location was selected randomly from the locations which have the same observation in the noiseless case with at least one other location, meaning that the robot has to perform at least one action to accomplish the localization task. The strategy of using macro actions, succeeded in all the trials in less than 500 actions. The other strategy failed to accomplish the task in less than 500 actions in 138 trials (Table 5.1). Moreover, in the trials where both strategies accomplished the task, the macro action strategy performed the localization task significantly better than the other one on average, using fewer than one third the basic actions. The average number of basic actions that are needed for localization in successful trials is shown in Table 5.2.

Table 5.2: Number of average basic actions for localization in successful with and without macro actions

Method	Average number of basic actions
Macro actions	25.61
Basic actions and random walk	79.13

5.2 Experiments on Planning under Uncertainty

We tested our algorithm on three large problems in the POMDP literature in three different domains: robot navigation, target tracking and scientific sampling. We also tested two state of the art POMDP solvers, HSVI2 and SARSOP, and compared the total reward and the time required for these approaches with our method. For both solvers we used the implementations available from the inventors of those methods, ZMDP 1.1.7 for HSVI2 [1] and APPL 0.94 for SARSOP [2]. The problems are as follows:

Fourth: This is a robot navigation task in a topological map constructed from the fourth floor of the CIT building, Brown University. Fourth was introduced by Anthony Cassandra in his Ph.D. thesis [2]. The description of the problem including all states, actions, observations, the discount factor is available in an standard format[3]. There are 1052 states, 4 actions of {*Rotate left, Rotate right, Go forward, Catch Goal*} and 28 different observations in this problem. Action *Catch Goal* has a reward of $+1$ in the goal states and -1 in other states. All other actions have zero reward everywhere. The robot is uncertain about where it starts. Its initial position may be anywhere except goal states. The discount factor is 0.99. The size of this problem does not seem great, but the noise in actions and observations make this problem very challenging. Another feature of this problem is that it models a real environment and it is a good example of a practical problem in planning under uncertainty.

Homecare: This is a target tracking problem for care-taking, introduced by Kurniawati *et al.* in their paper on SARSOP [20]. In this problem, a robot should follow an elderly person who moves in a fixed path. However, the motion of the person is non-deterministic. In each time step, the person may stop or continue his path with equal probability. Also, the visual field

[1]http://www.cs.cmu.edu/ trey/zmdp/

[2]http://bigbird.comp.nus.edu.sg/pmwiki/farm/appl/

[3]http://www.cs.brown.edu/research/ai/pomdp/examples/

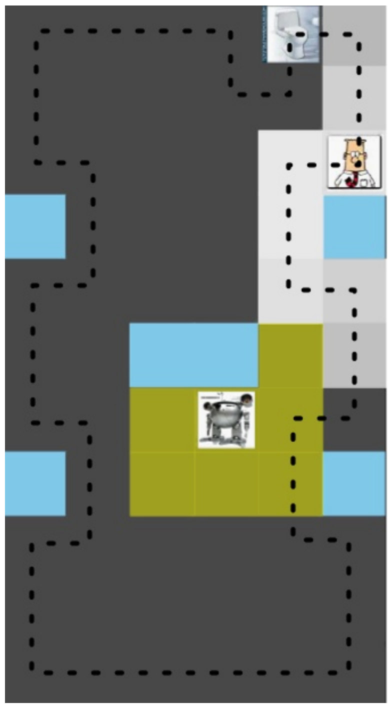


Figure 5.2: The map for Homecare. The figure is from [20].

of the robot is limited and the robot can observe the person only if he is close enough. There is a bathroom along the path where the person may stay for some time. A call button is at the bathroom which calls the robot to come for help. This button stays on for a non-deterministic duration. The robot gets reward only if it arrives when the button is on. The robot should stay close to the person to know its current position and more importantly reach the bathroom in time. On the other hand, it should minimizes its movements for power consumption. The environment of this problem including the path, the bathroom, and the visual field of the robot is illustrated in figure 5.2. This problem has $5,408$ states and 30 observations. The actions are *North, South, East, West, North East, North West, South East, South West, Stay*. The robot gets $+100$ reward if it reaches the bathroom in time. Action *Stay* has 0 value reward and the reward of moving in the main directions is -1. Other directions lead to -1.44 reward. The discount factor is 0.95.

Rock Sample: This problem was introduced by Smith and Simmons in their paper on the HSVI approach [30]. Rock sample is a scientific exploration problem where a rover wants to collect rocks in an $n \times n$ map. Some

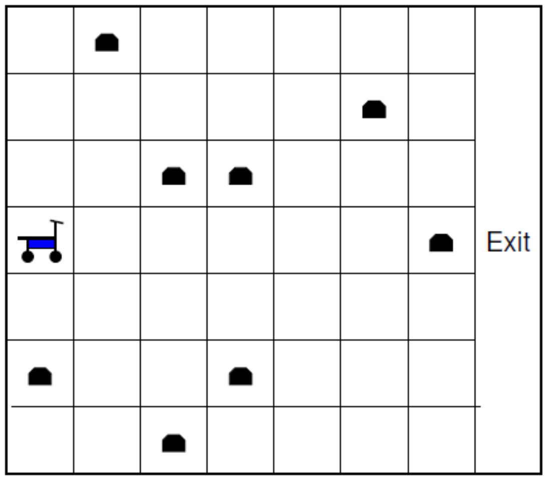


Figure 5.3: The map of RockSample[7,8], showing the initial location of the rover and all rocks. The figure is from [30].

of the rocks are good and some of them are bad. The robot gains +10 reward by sampling good rocks and gains −10 by sampling the bad ones. Also exiting the map has the reward of +10. The rover's position and the locations of the rocks are fully observable, but the robot is uncertain about the value of each rock (good or bad). The action set is {*North, South, East, West, Sample, Check*$_1$,…, *Check*$_k$} where k is the number of the rocks. All actions except *Sample* have zero reward. The action $Check_i$ gives the rover information about the value of $rock_i$. The accuracy of this information depends on the distance of the rover to $rock_i$. The information is 100 percent accurate if the robot is in the location of $rock_i$ and the accuracy decreases when the robot goes further form the rock. The problem of rock sampling in an $n \times n$ map with k rocks is specified as *RockSample[n,k]*. The map of *RockSample[7,8]* which we use in this paper is shown in Figure 5.3. In this problem, the discount factor is .95 as in Homecare.

The general parameters of the problems are shown in Table 5.3.

All approaches were tested on a personal computer with Intel core i7-2600K 3.40 GHz CPU and 16GB DDR3 RAM. The operating system was Ubuntu 10.4 and the programming language for all methods was C++. All methods were compiled with g++ 4.4.3. We tested HSVI2 and SARSOP on the problems many times to find the optimum reward that they could gain and the minimum time needed for that reward. In the cases that these

Table 5.3: General parameters of the POMDP problems

Problem	$\|S\|$	$\|A\|$	$\|O\|$	γ
Fourth	1,052	4	28	.99
RockSample[7,8]	12,545	13	2	.95
Homecare	5,408	9	30	.95

Table 5.4: The parameters of the pairwise heuristic in the POMDP problems

Problem	λ	compare ratio	maximum iterations
Fourth	.56	6	551
RockSample[7,8]	.85	3	151
Homecare	1	6	201

strategies could not reach optimal reward the reported reward is the reward of running theses algorithms for an hour. For all three methods we found the average reward for performing the test on a run of 1000 trials. We then determined the range of average rewards achieved over a set of 10 runs of the methods, and report the range and its midpoint. The time needed is not the same in all trials for our approach as it is online and in one trial the agent may reach the goal in fewer steps. Because of this, we reported the maximum total time (including all steps) among all trials. We should add that in performing a test, the code would continue until the agent reaches the goal states or the possible instant reward gets lower than 0.005. It means that the loop terminates when for the loop variable, t, $\gamma^t max_{s,a}|R(s,a)|$ is less than 0.005. The maximum number of iterations, compare ratio and λ in all four problems are shown in Table 5.4. The reward and the time needed for all problems and methods are in Table 5.5. We could not find the reward of HSVI2 for the Fourth problem as we got a "bad alloc" error after a few seconds of running the method.

As shown in Table 5.5, the pairwise heuristic gains a higher reward than HSVI2 and SARSOP in the Fourth problem and its reward in the other two problems is close to the reward of the other methods with only one step search and in much less time. The offline time column in Table 5.5 is a little confusing especially because we put zero for our method. Actually, the offline computation in HSVI2 and SARSOP is different from our offline

Table 5.5: The average reward and the time required for the methods on the POMDP problems

Method	Average reward	Offline time(s)	Online time(s)
Fourth			
Pairwise Heuristic	$.56 \pm .02$	0	0.06
SARSOP	$.53 \pm .01$	3651.56	0
RockSample[7,8]			
Pairwise Heuristic	$18.76 \pm .23$	0	0.03
HSVI2	$21.20 \pm .15$	80.00	0
SARSOP	$21.22 \pm .12$	31.33	0
Homecare			
Pairwise Heuristic	15.79 ± 0.81	0	0.45
HSVI2	$16.44 \pm .85$	1108.00	0
SARSOP	$16.85 \pm .63$	302.19	0

Table 5.6: The time required for the offline stage of the pairwise heuristic

Problem	Time (s)
Fourth	4.68
RockSample[7,8]	231.69
Homecare	77.96

computation. Unlike our method, the offline computation in HSVI2 and SARSOP is dependent on the initial belief state. As a result, they should be performed again each time the initial belief state changes. But, our offline part should be performed only once for each problem, no matter what the initial belief state is. Table 5.6 shows the time required for the offline computation in our pairwise heuristic method. Also, note that the total online time is the time needed to find and execute the entire plan; it is not the time required for just one step.

Table 5.5 shows that the pairwise heuristic is definitely a better approach for solving the Fourth problem and its time needed for solving RockSample[7,8] and Homecare is much less than the time needed for other methods. In regards to RockSample[7,8] and Homecare, one may argue that even

Table 5.7: The rewards gained by SARSOP and HSVI2 for different times on RockSample[7,8] compared to the pairwise heuristic

Method	Average reward	Total time(s)
Pairwise Heuristic	$18.76 \pm .23$	$0 + 0.03$
SARSOP	$7.35 \pm .00$	$0.22 + 0$
SARSOP	$17.57 \pm .30$	$0.37 + 0$
HSVI2	$10.43 \pm .00$	$2.00 + 0$
HSVI2	13.87 ± 0.12	$4.00 + 0$

Table 5.8: The rewards gained by SARSOP and HSVI2 in different times on Homecare compared to the pairwise heuristic

Method	Average reward	Total time(s)
Pairwise Heuristic	$15.79 \pm .81$	$0 + 0.45$
SARSOP	$14.44 \pm .67$	$230.12 + 0$
HSVI2	$14.36 \pm .52$	$250.00 + 0$

though SARSOP and HSVI2 need more time to gain the optimum reward, they might gain the same or even higher reward than our method in its required time ($0.03s$ for Rocksample and 0.6 for Homecare). For Rocksample problem, we tested these methods in $0.22s$, 7 times our time needed and $0.37s$, 10 times our time needed for SARSOP and $2.00s$ and $4.00s$ for HSVI2 and show the results in Table 5.7. As shown, they achieve less reward in 10 times the time required by the pairwise heuristic.

For Homecare, SARSOP needs $208s$ only for initialization, meaning that it can not generate any policy in less than 460 times the time required by the pairwise heuristic. Initialization time is $244s$ for HSVI2. We report the reward of SARSOP in $230.12s$, and HSVI in $250s$ in Table 5.8 showing that these methods gain less reward in more than 500 times the time required by our method.

Our experiments on localization show the necessity of macro actions for performing the task successfully in self-similar environments. Also, the experiments on different POMDP problems illustrate the computational efficiency of the pairwise heuristic.

Chapter 6

Discussion and Conclusions

In this chapter, we will first go over the computational and memory costs of our algorithms, and the methods that can be used to reduce this cost. Also advantages and drawbacks of pairwise heuristics are discussed. Finally, the conclusion and possible future work are represented.

6.1 The Pairwise Heuristic in Localization

Our method puts most of the computational work into the offline computation that is done only once per map and can be used many times. The most time consuming algorithm in this computation is the algorithm for macro action generation. On the other hand, we try to make the online part of our method as fast as possible and use a common technique to reduce its time complexity.

6.1.1 Offline Computation

The first precomputation is calculating $p(o|s)$ for all observations and states and putting them in a lookup table as in [7]. This lookup table reduces the time needed for updating beliefs after an observation and also finding distinguishable pairs. The second precomputation is generating macro actions for all pairs of states. This algorithm is quite expensive and in the case where all actions have unit cost, it takes $O(|S|^2 \times k)$ where k is the maximum length optimal path required to distinguish two states. In the worst case k would be $O(|S|)$ and the total time needed is $O(|S|^3)$. But usually k is far less than $|S|$. For actions with different costs, if the cost of the maximum cost action is C_{max} and the cost of minimum cost action is C_{min}, in the worst case, the computational cost would be $O(|S|^3 \times C_{max}/C_{min})$. C_{max}/C_{min} is the maximum length of a sequence of actions that could be replaced by one single action. For each pair the macro action should be stored and this requires $|S|(|S|-1)/2$ memory. For memory usage efficiency, we need only store the macro actions for the states that are not distinguishable, not storing empty sets.

6.1.2 Online Computation

In the online stage of the method, if there are n states with positive probability, action selection takes $O(n^2 \times |A'|)$ where A' is the set of candidate actions and macro actions. A common method for reducing computational cost is to assume the probability density is a mixture of Gaussians [7] [4]. To do this, states with a probability higher than a threshold would be selected and assumed to be the mean of Gaussian densities that model the probability of all states. If the number of these means is m, the complexity of the action selection is $O(m^2 \times |A'|)$. This computational cost is $O(n \times m \times |A|)$ for entropy based localization [7] and as $m \ll n$ our approach is much more efficient.

In the case where the robot has no prior knowledge about its position (global localization), to avoid high computational cost the robot selects the first few (3 to 10) actions randomly because the number of possible states is very large, but it would drastically reduce in the first few steps.

6.1.3 Advantages and Drawbacks

The two main advantages of pairwise heuristic in localization are computational efficiency and success in self-similar environments. However, this heuristic has one major drawback. Our method is based on Markov localization which means that we need to represent states explicitly. A large number of states is needed for large environments especially for 3D navigation. As a result, our method is more useful for 2D navigation in small to medium-sized self-similar environments where single action-based methods do not work.

6.2 The Pairwise Heuristic in Planning under Uncertainty

As in localization, most of the computational work is in the offline part of the method. This part is run only once per environment and is totally independent of the initial belief state. In fact, the computational cost of our method, even in the offline part, is not very high and does not create any problems. The major drawback of the pairwise heuristic is its memory requirement.

6.2.1 Offline Computation

The first stage in our method is solving the underlying MDP of the problem. The value iteration method is used for this part. Each iteration of this method takes $O(|S|^2 \times |A|)$ in the worst case. In fact, the actual time is a lot less, usually. In the Bellman equation, we need to consider only the states where the agent can go from the source. We borrow the terms neighbor and degree from graph theory. State s' is considered as a neighbor of state s if and only if there exists an action $a \in A$ which $p(s'|s, a)$ is larger than zero. Each iteration takes $O(D)$ where D is the total degree of the states. Usually, $D = \Theta(|S|)$ so each iteration takes $O(|S|)$. The value functions converge very quickly. We ran only 1000 iterations to obtain them in our experiments.

The most important component of the offline computation, and also the whole method is the algorithm for finding value functions of the pairs. In fact, this algorithm is a value iteration of an MDP with the set of all pairs as the state space. To be more precise, it is a value iteration only on indistinguishable pairs. But the number of indistinguishable pairs could be very close to the total number of pairs. As a result, each iteration should take $O(|S|^4 \times |A|)$ time in the worst case. However, as we only consider most probable states in the transition function, each iteration takes $O(|S|^2 \times |A|)$. Again, we ran 1000 iterations for our problems.

6.2.2 Online Computation

In each step, we only need to calculate the heuristic for each action which needs the value functions of the pairs. As the total number of the pairs is $|S|^2$, the worst time for each step is $O(|S|^2 \times |A|)$. However, only the states with the probability of larger than a certain threshold are selected and the number of these states is much less than the total number of states, $|S'| << |S|$. As a result, the time needed for each step, $O(|S'|^2)$, is usually much less than the time of the worst case scenario. Also, updating the belief state, which is common to all methods, takes $O(|S|^2)$ in the worst case. Similar to the analysis in the offline part, the time it takes to update the belief state can be presented as $O(D)$ and as a result is usually $O(|S|)$.

6.2.3 Memory usage

In addition to the memory that is needed to represent the POMDP, we need to save $|S|^2$ value functions of the pairs, $|S|^2$ optimum actions of the pairs, $|S|$ value functions of the underlying MDP and $|S|$ optimum actions of the underlying MDP. As a result, additional memory usage is $O(|S|^2)$ which is not a problem for solving a POMDP in general since representing the transition function alone needs $O(|S|^2 \times |A|)$.

6.2.4 Advantages and Drawbacks

The most important advantage of the pairwise heuristic in solving POMDPs is computational efficiency. Large problems with thousands of states can be solved in less than a second, turning the algorithm into a real-time approach. The resulting reward of the pairwise heuristic is close to or even better than the reward gained by the state of the art POMDP solvers while its execution time is much less than the time required by those solvers. Despite its computational efficiency, there is one bottleneck that makes the pairwise heuristic useless for problems with very large state spaces, such as those with more than 1 million states. That is memory usage. Although we mentioned that a POMDP needs $O(|S|^2 \times |A|)$ memory to be represented, usually a transition function can be represented in $O(|S| \times |A|)$. Actually, this fact helps us in the offline and online computation as explained before. However, it also makes some problems representable yet unsolvable by the pairwise heuristic. We should add that this drawback is not major in comparison to the other solvers as in most cases, other solvers could not even solve these problems because of their high computational cost. Another advantage of the pairwise heuristic is its simplicity. Unlike most of the POMDP solvers, the pairwise heuristic is simple to understand and implement.

6.3 Conclusion

Uncertainty often makes a problem so complex that finding an optimal solution for it in an efficient time is almost impossible. The solution time is an important factor and in many practical application only real-time approaches are acceptable. For example, in a target tracking problem, if the robot does not respond quickly enough, the target would soon go out of its visual range. On the other hand, in most cases, a near-optimal solution is good enough. In the target tracking problem, catching the target quickly and minimizing power consumption are the two most important criteria. However, as long as the robot can catch the target, the exact optimality of the time required and power usage is not crucial. As another example, in a robot navigation problem, as long as the robot does not hurt itself or others or wander aimlessly around the room, its path may be acceptable. In general, in practical applications, the time needed for decision making is more important than the exact optimality of the solution. In these situations, an algorithm that can find the solution very quickly is a perfect candidate even if the solution is not perfectly optimal. An active strategy with a good heuristic has both the parameters of efficiency and near optimality. In this thesis, we introduced a heuristic to deal with uncertainty, the pairwise heuristic. This heuristic solves the problem assuming that uncertainty exists only between two states. Then, an online strategy uses the solutions of all pairs of states for decision making in the main problem. We tested our approach on two kinds of problems where uncertainty plays an important role, that is, localization and planning under uncertainty. Experiments on complicated benchmarks show that this heuristic gives us the near-optimal solution in an efficient time and can be used as a real-time approach on practical applications.

6.4 Future Work

It is possible that the pairwise heuristic could also be used in other kinds of problems, especially in problems where constraints are given. An example is the scheduling problem. It may be useful to solve the scheduling problem for each pair of the tasks first and then use these solutions in the main problem. However, that is just a hypothesis and should be examined as a new research topic.

Another future topic could be modeling large practical applications as POMDPs and solving them by the pairwise heuristic. Currently, most practical problems are very small because the previous solvers are computationally expensive. With the pairwise heuristic, large problems may be solved as well. For example, we can solve large target tracking or navigation problems in the real world. Modeling the problem, however, is not always easy and may itself be the topic of further research.

In the end, modifying the pairwise heuristic or the online approach may lead to better results. Finding ways to improve the pairwise heuristic would be an interesting and challenging research topic.

Bibliography

[1] Ronen I. Brafman. A heuristic variable grid solution method for POMDPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 727–733, 1997.

[2] Anthony R. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* PhD thesis, Brown University, Department of Computer Science, Providence, RI, 1998.

[3] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 963–972, 1996.

[4] Andreu Corominas Murtra, Josep M. Mirats Tur, and Alberto Sanfeliu. Efficient active global localization for mobile robots operating in large and cooperative environments. In *IEEE International Conference on Robotics and Automation*, pages 2758–2763, May 2008.

[5] Yanzhu Du, David Hsu, Hanna Kurniawati, Wee Sun Lee, Sylvie C.W. Ong, and Shao Wei Png. A POMDP approach to robot motion planning under uncertainty. In *International Conference on Automated Planning & Scheduling, Workshop on Solving Real-World POMDP Problems*, 2010.

[6] Gregory Dudek, Kathleen Romanik, and Sue Whitesides. Localizing a robot with minimum travel. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, SODA, pages 437–446, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[7] D. Fox, D. Burgard, and S. Thrun. Active Markov localization for mobile robots. *Robotics and Autonomous Systems*, 25:195–207, 1998.

[8] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11, 1999.

[9] Andrea Gasparri, Stefano Panzieri, Federica Pascucci, and Giovanni Ulivi. A hybrid active global localisation algorithm for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 3148 –3153, april 2007.

[10] D. Golovin, A. Krause, and D. Ray. Near-optimal Bayesian active learning with noisy observations. *CoRR*, abs/1010.3091, 2010.

[11] R. He, E. Brunskill, and N. Roy. Efficient planning under uncertainty with macro-actions. *Journal of Artificial Intelligence Research*, 40:523–570, 2011.

[12] Rojie He, Emma Brunskill, and Nicholas Roy. PUMA: Planning under uncertainty with macro-actions. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, Atlanta, GA, 2010.

[13] Ruijie He, Emma Brunskill, and Nicholas Roy. Puma: Planning under uncertainty with macro-actions. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI)*, Atlanta, GA, 2010.

[14] Jesse Hoey, Pascal Poupart, Axel von Bertoldi, Tammy Craig, Craig Boutilier, and Alex Mihailidis. Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding*, 114(5):503 – 519, 2010.

[15] P. Jensfelt and S. Kristensen. Active global localization for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation*, 17(5):748–760, October 2001.

[16] Robert Kaplow, Amin Atrash, and Joelle Pineau. Variable resolution decomposition for robotic navigation under a POMDP framework. In *Proceedings of the International Conference on Robotics and Automation*, 2010.

[17] S. Koenig, M. S. B Mitchell, A. Mudgal, and C. Tovey. A near-tight approximation algorithm for the robot localization problem. *SIAM*, 39(2):461–490, 2009.

[18] H. Kurniawati, D. Yanzhu, D. Hsu, and W. S. Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *International Journal of Robotics Research*, 30:308–323, March 2011.

[19] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. Motion planning under uncertainty for robotic tasks with long time horizons. *International Journal of Robotics Research*, 30:308–323, March 2011.

[20] Hanna Kurniawati, David Hsu, and Wee Sun Lee. SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Proceedings of Robotics: Science and Systems*, 2008.

[21] Michael L. Littman, Anthony R. Cassandra, and Leslie Pack Kaelbling. Learning policies for partially observable environments: Scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[22] Amy Mcgovern. acQuire-macros: An algorithm for automatically learning macro-actions. In *NIPS Workshop on Abstraction and Hierarchy in Reinforcement Learning*, 1998.

[23] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27, 2006.

[24] Kin Man Poon. A fast heuristic algorithm for decision-theoretic planning. Master's thesis, The Hong Kong University of Science and Technology, 2001.

[25] J. M. Porta, J. J. Verbeek, and B. J. A. Kröse. Active appearance-based robot localization using stereo vision. *Autonomous Robots*, 18:59–80, January 2005.

[26] Pascal Poupart, Kee-Eung Kim, and Dongho Kim. Closing the gap: Improved bounds on optimal POMDP solutions. In *International Conference on Automated Planning & Scheduling*, 2011.

[27] M. Rao, G. Dudek, and S. Whitesides. Randomized algorithms for minimum distance localization. *International Journal of Robotics Research*, 26:917–933, September 2007.

[28] Stephane Ross, Joelle Pineau, Sebastien Paquet, and Brahim Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 2008.

[29] Nicholas Roy and Geoffrey Gordon. Exponential family PCA for belief compression in POMDPs. In *Neural Information Processing Systems (NIPS)*, pages 1043–1049. MIT Press, 2003.

[30] Trey Smith and Reid G. Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[31] Wolfram Thrun, Sebastian Burgard and Dieter Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA,, 2005.

[32] C. Tovey and S. Koenig. Localization: Approximation and performance bounds to minimize travel distance. *IEEE Transactions on Robotics*, 26(2):320–330, April 2010.

[33] Craig Tovey and Sven Koenig. Gridworlds as testbeds for planning with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*, pages 819–824, 2000.

Appendix A

Optimality of Macro Actions Generation Algorithm

Theorem: In a deterministic environment, for any pair of states, Algorithm 3 finds the minimum length macro action that can distinguish between them.

Proof: We do this by induction:

Basis: The algorithm finds the macro action for the states that are distinguishable by zero cost.

As costs of all actions in any state are positive, zero cost means that the states are distinguishable and there is no need for macro action. We set these macro actions to zero in the line 3 of Algorithm 2 which means there is no need for a macro action. Therefore, costs and macro actions of all of those pairs are defined initially in our algorithm.

The inductive step: Assuming that macro actions of all pairs with minimum cost macro action of less than cost P are defined and no other macro action is set, the macro action of pairs with minimum cost macro action of cost P will be defined in the next step.

If the minimum cost macro action of s_i and s_j costs P, first of all it is not defined yet because only macro actions with cost of less than P have been defined. If this macro action is a single action a with cost P, it will be defined in this step because $f^*(s_i, a)$ and $f^*(s_j, a)$ are distinguishable and macro actions of all distinguishable pairs are proven to be set in basis. Macro action of (s_i, s_j) will be set to a attached to $macro\_action(f^*(s_i, a), f^*(s_j, a))$ which is zero, meaning no action. And if this macro action is not a single action we show it by $\langle b_1, b_2, ..., b_{k-1}, b_k \rangle$ that each b_i is one basic action. As all actions have positive cost, $C(s_i, s_j, b_1)$ is positive and $(C(f^*(s_i, b_1), f^*(s_j, b_1), \langle b_2, ..., b_k \rangle))$ is less than P. Thus $macro\_action(f^*(s_i, b_1), f^*(s_j, b_1))$ has been already defined and as both $\langle b_2, ..., b_k \rangle$ and $macro\_action(f^*(s_i, b_1), f^*(s_j, b_1))$ are minimal, the costs of them are exactly equal. $macro\_action$ $(f^*(s_i, b_1), f^*(s_j, b_1))$ is minimal by the induction assumption and $\langle b_2, ..., b_k \rangle$ is minimal because if it is not, we can put the minimal path after b_1 and find another path with less cost for distinguishing between s_i and s_j. This contradicts the assumption that the minimum cost macro action of s_i and s_j costs P. So macro action of s_i and s_j will be defined by attaching b_1 to $\mathit{macro\_action}(f^*(s_i, b_1), f^*(s_j, b_1))$ with cost P.

As a macro action with cost P is defined, all macro actions with cost of greater than P are not minimum and would not be defined in this step.