Automated Planning for Hydrothermal Vent Prospecting Using AUVs

Zeyn A Saigol

A thesis submitted to the
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
April 2011

Abstract

This thesis presents two families of novel algorithms for automated planning under uncertainty. It focuses on the domain of searching the ocean floor for hydrothermal vents, using autonomous underwater vehicles (AUVs). This is a hard problem because the AUV's sensors cannot directly measure the range or bearing to vents, but instead detecting the plume from a vent indicates the source vent lies somewhere up-current, within a relatively large swathe of the search area. An unknown number of vents may be located anywhere in the search area, giving rise to a problem that is naturally formulated as a partially-observable Markov decision process (POMDP), but with a very large state space (of the order of 10^{123} states). This size of problem is intractable for current POMDP solvers, so instead heuristic solutions were sought.

The problem is one of chemical plume tracing, which can be solved using simple reactive algorithms for a single chemical source, but the potential for multiple sources makes a more principled approach desirable for this domain. This thesis presents several novel planning methods, which all rely on an existing occupancy grid mapping algorithm to infer vent location probabilities from observations. The novel algorithms are information lookahead and expected-entropy-change planners, together with an orienteering problem (OP) correction that can be used with either planner. Information lookahead applies online POMDP methods to the problem, and was found to be effective in locating vents even with small lookahead values. The best of the entropy-based algorithms was one that attempts to maximise the expected change in entropy for all cells along a path, where the path is found using an OP solver. This expected-entropy-change algorithm was at least as effective as the information-lookahead approach, and with slightly better computational efficiency.

Acknowledgements

First I would like to thank my supervisors, Richard Dearden and Jeremy Wyatt, for all their insight and encouragement. They were amazingly supportive and helpful even when I had to do tasks that were both tedious and taxing, such as writing 170-page documents.

Second, I would like to acknowledge Mike Jakuba, who was really helpful and very generous in allowing me to use his code during my PhD.

I would like to thank my mum for her enthusiasm towards my academic career, and for proof-reading most of this document.

I would like to say a big thanks to everyone in the Wayfarers, and especially Charlie and the rest of the Tuesday Night Climbing Club, for making living in Birmingham (and managing to escape it frequently) fun.

Thanks also to all the 2008 MBARI interns, George, my MBARI mentors, and again Jeremy and Richard, for providing me with an experience that certainly formed the highlight of my PhD. I had an amazing time in California with Amanda, Erin, Jon, Zach, and the rest of the interns.

I have enjoyed sharing an office with many interesting people during my time in the School of Computer Science, and I would like to acknowledge fruitful (or occasionally just procrastinatory) conversations with the IRLab crew (Damien, Mike, and Noel) in particular.

Xin Yao and Hamid Dehghani (who were on my thesis committee) provided criticism in a very constructive way, and helped me understand what research is all about.

The IRLab provided an invigorating research environment, and I have enjoyed being part of it. I would also particularly like to thank everyone who had a go at the human-controlled CPT simulation.

Finally, thanks to Bramley Murton for motivating this whole domain and being happy to help whenever I had any questions about the oceanographic side, and to the School of Computer Science for funding my PhD.

Contents

Chapter 1

Introduction

1.1 Motivation

One of the fundamental challenges in mobile robotics is how to make good decisions given incomplete and uncertain information. In the real world sensors never return perfect information, and it is rarely possible for a robot to observe all aspects of the environment that are of interest to it. If a robot does not know the true state of the world, it will usually not know what effect its actions will have, which makes it very hard to form a plan to achieve its goals. For example, a robot exploring the surface of Mars may be unable to judge the exact distance to a nearby rock, so may not know if it has to move closer before it can pick up the rock.

This thesis considers the problem of an autonomous robot 'scientist' that should act to gather as much useful scientific data as possible given constraints on its mission duration and range. Specifically it considers how an autonomous underwater vehicle (AUV) should act to efficiently locate and visit hydrothermal vents, given the noisy environmental picture provided by its sensors. Hydrothermal vents are found on the ocean floor, several kilometres below the surface, and they discharge superheated water into the ocean. For marine scientists, hydrothermal vents are among the most exciting finds the oceans have to offer; for geologists they provide evidence of the geochemical processes occurring below the Earth's crust, and for biologists they mark the location of many strange life forms that, uniquely amongst life on Earth, derive their energy from the Earth's core (via the vent) rather than from the Sun.

Hydrothermal vents are formed where tectonic plates diverge and magma is pushed up to form new ocean floor. Cracks in the seabed allow water to seep down and come into contact with the magma, heating it up and dissolving minerals into it. The heated water then rises and erupts back into the ocean at temperatures of 300-400°C, forming a hydrothermal vent. This water quickly cools to very close to ambient temperature, but still contains chemicals that give it a detectable signature as it is spread out over an area dozens of kilometres wide as a *hydrothermal plume*.

Sensors on board an AUV can detect this plume signature, but this gives only limited information about the location of the source vent, for several reasons:

- Turbulent currents together with the large dilution of the hydrothermal chemicals mean there is no steady gradient of increasing chemical concentration that can be followed to the source. Instead, hydrothermal tracers are found in disjoint patches.
- The output from different vents differs in both volumetric flow rate and in the relative concentrations of different chemicals, which means chemical intensity is not a reliable indicator of distance to the source.
- The only information available about the direction to the source comes from measuring the current, as plumes are largely dispersed by ocean currents. However, the current varies in

both magnitude and direction over the lifetime of an AUV mission, and additionally there is a diffusion component to the spread of plumes.

- Vents are often found in clusters known as *vent fields*, and the plumes from several vents can combine together, further complicating any analysis.

Contemporary methods for vent prospecting rely on firstly using a research ship to take measurements of the water column over a large area, perhaps several hundred kilometres wide, to detect evidence of a hydrothermal plume. Once a plume has been detected, a submersible vehicle is used to search a smaller area in more detail. The submersible may be a remotely operated vehicle (ROV), an AUV, or even a crewed vehicle. With an AUV an exhaustive search pattern is generally used, and several successive deployments are needed; between each deployment the vehicle must return to the surface for scientists to identify promising areas to explore in more detail.

The aim of this work is to present novel algorithms that allow an AUV to plan its own path based on the sensor data it records, and to find vents more efficiently than an exhaustive search pattern would. These algorithms enable vents to be located precisely during a single AUV deployment, freeing the AUV to either find further vents in a different area or to carry out other tasks on subsequent missions. Additionally, although the AUV will still have to be recovered to the ship in order to recharge, scientists will not have to analyse data and plan follow-up missions, allowing them to focus on other tasks. As well as these novel algorithms, this thesis describes experiments (conducted in simulation) that demonstrate the effectiveness of the algorithms in finding vents when compared to conventional means. While the focus is on the vent prospecting domain, aspects of this work may have broader implications, specifically the comparison of two different planning regimes: reward-driven and entropy-driven planning.

1.2 Problem Overview

From an AI planning point of view, the state space for the vent prospecting problem consists of the 3D location of the vehicle, the ocean current vector and its history, the time and battery power remaining to the AUV, and the location(s) of vents in the search area. The agent does not know the vent locations, but instead has to act on the basis of noisy sensor readings that provide clues to the vent locations. The vehicle's sensors provide continuous measurements of various physical and chemical quantities that are useful in determining the presence of a hydrothermal plume from a vent. The only actions available to the vehicle are moving in the world, and the objective of the agent is to locate as many vents as possible within a fixed search area during a fixed-duration mission. While existing reactive algorithms are capable of efficiently locating a single chemical source (see Section 2.3), the ability for an AUV to locate multiple vents during a single mission would be especially valuable to the oceanographic community.

The natures of the partial observability, reward function, and observations make this problem particularly hard. First, not only are vent locations unknown (initially) to the agent, the number of vents is also unknown, which is roughly equivalent to having a partially-observable state space of unknown dimensionality. Second, the agent is rewarded for finding specific targets, rather than simply for mapping an area. It must therefore balance exploration (improving its map) with exploitation (visiting areas where the current map indicates there are likely to be vents). This is similar to the exploration/exploitation trade-off encountered in the reinforcement learning problem [Sutton and Barto, 1998], and means that to act optimally the agent will have to reason about the informational effects of actions. Finally, observations the agent receives are not simply noisy estimates of the location of objects, as is often the case with mobile robots using sonar or laser sensors. Instead, observations are noisy indicators of the strength of a hydrothermal plume, and plume strength indicates a large range of possible vent locations up-current.

This vent prospecting problem is naturally formulated as a partially-observable Markov decision process (POMDP), but the continuous state space and the type of uncertainty in the problem make the domain intractable for current POMDP solvers (see Section 3.2 for an overview of POMDPs, Section 4.3 for the full POMDP model of this problem and Section 6.1 for a discussion of relevant POMDP work in the literature). Classical planning techniques can not be used either, as these cannot handle the type of observational uncertainty present in this problem [Ghallab *et al.*, 2004; Kushmerick *et al.*, 1995]. Given this, the aim of this work was to develop satisficing solutions, in other words solutions that achieve the desired goal but are not necessarily optimal.

1.2.1 Evaluation

My algorithms are evaluated using two key performance metrics: their effectiveness in finding vents, and their computational efficiency. Algorithm run-time is an important consideration because the ultimate goal is to implement the algorithms on an AUV, where processor speed is limited, and furthermore any savings in computation will lead to savings in battery power. I will examine both the computational complexity of the algorithms, and run-time for a simulated mission. However given that the current work is all conducted in simulation, computational complexity is less important than vent-hunting performance. Note that in the remainder of this thesis, "algorithm performance" is used to refer to effectiveness in finding vents rather than computational efficiency.

The measure used for evaluating performance is the average proportion of vents found, where the average is taken over several missions with different vent locations and plume dispersion. The proportion of vents found is used rather than the absolute number found to allow comparison between trials with different numbers of vents in the environment.

To provide a benchmark for the performance of my novel algorithms, two methods commonly used in chemical plume hunting are used: mowing-the-lawn (MTL) and chemotaxis. MTL prescribes an exhaustive search pattern, and chemotaxis is a reactive search inspired by the behaviour of moths, using up-current surges followed by a spiral search pattern. An interactive simulation allowing human performance on the problem to be measured was also created, and this simulation as well as both comparison algorithms are described in Section 2.4.

Finally, the obvious evaluation method for vent prospecting algorithms is to apply them to finding vents using a real AUV. This was outside the scope of this work. However, there are also two drawbacks to this approach for evaluating novel algorithms: firstly such scientific research cruises are very costly, so only a small sample size can be obtained for evaluation (how would we know the algorithm did not perform well purely by chance?), and secondly it is difficult and costly to establish the ground truth of how many vents there actually were in the AUV's search area.

1.3 Solution Overview

The problem was divided into two components: creating a map of the vent locations, and planning a path given this map. Given that the AUV only receives imperfect and partial information from its sensors, the map will have to accommodate uncertainty in vent locations. A directly applicable mapping algorithm was developed by [Jakuba, 2007]; this algorithm takes sensor data from an AUV and builds an *occupancy grid* (OG) map showing the likelihood of finding a vent in each location. The algorithm uses plume measurements together with the sea-floor current to assign probability to vent locations, and is described in more detail in Section 3.1.2. This thesis builds on Jakuba's mapping algorithm, which allows it to focus on the planning aspects of the vent prospecting problem; however, using an OG mapping method dictated the state space of the problem, and this state space placed restrictions on the planning approaches that were possible (as discussed further in Section 8.2).

The planning algorithms are sub-optimal strategies based on the OG map (which is a Bayesian estimate of the state), as the optimal POMDP solution is intractable. The planning strategies fall into two distinct camps: entropy-based and information-lookahead–based. Entropy-based strategies try to locate vents by minimising the entropy (or uncertainty) of the map, whereas information-lookahead–based strategies search ahead several steps in the state-action space, and then use these potential future states to help estimate the optimal action. Finally, I developed a correction term that can be applied to both the entropy and information-lookahead cases, which addresses a problem with the agent gaining reward by 're-discovering' vents. The formal problem model and these three technical approaches are discussed in the following subsections.

1.3.1 Problem Model

I have abstracted the vent prospecting domain into a POMDP model that is computationally tractable for the heuristic and approximate algorithms presented in this thesis. The model still keeps enough fidelity to make it plausible that such implementations could be transferred to the real domain, and it is outlined below, and described in full in Chapter 4.

The search area is divided into a regular 2D grid of cells, each of which may either contain a vent or not, which allows the use of an occupancy grid to model the agent's knowledge of vent locations. The agent is restricted to moving between adjacent grid cells in the north/east/south/west directions, one cell per timestep, and always knows exactly which grid cell it is in. The aim of the agent is to find as many vents as possible during a fixed-duration mission.

The model of hydrothermal plumes is based closely on that used by Jakuba, and this is described in Section 4.2. It results in a binary observation model for the agent, to either detect a hydrothermal plume, or detect nothing, on each timestep. In addition to this I allow the agent to know when it has visited a vent, and I also derived an observation function giving the probability of each observation for a known AUV location and OG. This function is based on the same simple plume propagation model that underpins Jakuba's OG update algorithm, and the derivation is given in Section 4.6.

1.3.2 Entropy-inspired Planning Methods

I have created a family of novel algorithms inspired by the idea of acting to reduce map entropy in order to localise vents. Firstly, the *infotaxis* algorithm described in [Vergassola *et al.*, 2007] was adapted to work with the model described in Section 1.3.1, which differs in the observation model and map representation. Infotaxis performs a one-step lookahead search over actions, and selects the action that results in the greatest reduction in expected map entropy, where the expectation is taken over possible observations.

I build on this with the $\Sigma\Delta H$-algorithm, which copes better with a peculiarity of this domain: due to the low prior probability of grid cells containing a vent, the entropy of cells will often rise rather than fall when a useful observation is made (which means the entropy of the whole map will also often rise). $\Sigma\Delta H$ addresses this by maximising the *change* in entropy, even if this is an increase in entropy, because such changes correspond to the agent making useful observations. Finally, $\Sigma\Delta H$-MDP is introduced, which is a non-myopic version of the algorithm that allows future actions to be taken into account by essentially ignoring their informational value. $\Sigma\Delta H$-MDP first calculates the expected entropy change for making an observation from every cell, as if the agent could 'teleport' instantly to that cell. It then solves a Markov decision process (MDP) where these entropy changes represent the reward the agent gets for visiting cells (see Section 3.2 for an introduction to MDPs). The policy defined by the MDP solution is optimal for this reward model, in that it maximises the agent's expected long-term reward.

1.3.3 Information-lookahead Planning Methods

A completely separate approach using information lookahead in the belief-space of the POMDP was also developed. The information-lookahead (IL) algorithm uses forward search in the action/observation space of the POMDP for a small number of steps to assign a value to each possible action. In basic IL, the value used for the leaf nodes of the forward search is simply the OG occupancy probability for the terminal cell. In the IL-CE (certainty equivalent) variant, leaf node values are found by solving an MDP using OG probabilities as the rewards. This is similar to the $\Sigma\Delta H$-MDP algorithm, as the MDP correction ignores the informational effects of future actions.

1.3.4 Orienteering Problem Correction

The final contribution I developed was a fix to the MDP-based algorithms that uses the *orienteering problem* (OP) [Tsiligirides, 1984; Golden *et al.*, 1987], a variant of the travelling salesman problem. When the informational effects of future actions are disregarded, treating the planning problem as an OP instead of an MDP means that a reward can only be claimed the first time a cell is visited, rather than every time. This is a more accurate model of the underlying problem, and also prevents the MDP policy from becoming stuck in solutions where high-value cells are repeatedly re-visited. The OP solver is applied to the problem in three ways:

- As a standalone planner where informational effects of actions are completely ignored.
- Using the $\Sigma\Delta H$ algorithm to encode informational effects, and the OP solver to project these effects to future actions.
- As a correction to the information lookahead calculation, where the OP is used to calculate the value of leaf nodes, as a replacement for the MDP in IL-CE.

1.4 Contributions and Results

The key contribution of this thesis is a solution to the vent-prospecting problem, which will be of interest to the AUV community, and illustrates a novel application and extension of AI planning techniques, which is effective given the particular characteristics of this domain.

In particular, I show that:

- Planning using an occupancy grid representation enables an agent to find more vents than either chemotaxis or exhaustive search (neither of which uses a representation of the environment).
- Entropy-based planning is able to find vents as efficiently as forward search in the action/observation space, but with better computational scalability.

The contributions are:

- A formal model of the planning problem as a POMDP, as outlined in Section 1.3.1 (details in Chapter 4). This model will enable other researchers to tackle the problem more easily in the future. It is an interesting problem for POMDP research because it is a real-world application where classical planning techniques will not work, and where the size of the state space makes it a challenge for existing POMDP methods.
- A set of entropy-based planning algorithms, as outlined in Section 1.3.2 (details in Chapter 5). I show that $\Sigma\Delta H$-MDP performs better than $\Sigma\Delta H$ and infotaxis, and all of these algorithms are significant improvements on MTL and chemotaxis.

- A set of information-lookahead planning algorithms, as outlined in Section 1.3.3 (details in Chapter 6). The key contribution here is to apply online POMDP methods [Ross *et al.*, 2008] to the model described in Chapter 4, and show that these methods perform at least as well as the entropy-based methods of Chapter 5. I also show that, in this case, the IL-CE approximation where informational effects of future actions are ignored results in significantly worse performance.

- A correction using an orienteering problem solver, as outlined in Section 1.3.4 (details in Chapter 7). When applied to the $\Sigma\Delta H$ algorithm, the OP correction produces significantly better performance than the $\Sigma\Delta H$-MDP approach, and at least equivalent performance to the IL algorithm. Further, $\Sigma\Delta H$-OP runs significantly faster than IL. When applied to the IL algorithm, the OP correction does not significantly improve algorithm performance, which is surprising as it creates plans based on simulating actions further into the future. However this is explained by the high quality of the map contained in the OG, which arises largely from the plume model.

While all work for this thesis was conducted in simulation, care was taken to avoid algorithms that would not be practical on a real vehicle. The issues that would need to be addressed to implement my solutions on a real AUV are discussed in Section 8.3.5.

In addition to the problem of finding hydrothermal vents, the algorithms I have developed could be applied to several other domains which share the key goal of tracing chemical plumes in a turbulent water flow. Examples of such domains are searching for unexploded naval mines or sunken nuclear submarines, tracing leaks in underwater pipelines, tracking and measuring plumes carrying polluted water into the seas from inland waterways, and potentially even plume tracing in air as opposed to water.

This thesis has resulted in three publications:

- Richard W. Dearden, Zeyn A. Saigol, Jeremy L. Wyatt, and Bramley J. Murton. Planning for AUVs: Dealing with a continuous partially-observable environment. In *ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems*. [Dearden *et al.*, 2007].

- Zeyn A. Saigol, Richard W. Dearden, Jeremy L. Wyatt, and Bramley J. Murton. Information-lookahead planning for AUV mapping. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*. [Saigol *et al.*, 2009a]. In addition an extended technical report based on this paper was authored [Saigol *et al.*, 2009b].

- Zeyn A. Saigol, Richard W. Dearden, Jeremy L. Wyatt, and Bramley J. Murton. Belief change maximisation for hydrothermal vent hunting using occupancy grids. In *Proceedings of the Eleventh Conference Towards Autonomous Robotic Systems (TAROS-10)*. [Saigol *et al.*, 2010a].

1.4.1 Note on Code Re-use

My implementation makes extensive use of code that Michael Jakuba generously supplied me with for my personal use on this project. This code implements the Independence of Posteriors (IP) semi-recursive occupancy grid algorithm from [Jakuba, 2007] §3.2.1, and a basic 2D version of the plume model described in [Jakuba, 2007] §4.5 and §5.2.1 (and in Section 4.2 of this work), both of which I used without modification (apart from adding the deterministic vent detector). The IP OG implementation contained code to group detections into approximately independent groups, but this facility was not used. The source also contained some framework code and tools for plotting the OG and vehicle path, which I used with major modifications and extensions.

The code supplied to me did not include any mechanism for dynamically changing the agent's path, any of the comparison algorithms described in Section 2.4, or any of my novel algorithms described in Chapters 5, 6 and 7.

1.5 Background

This work was motivated by earlier work on autonomous 'robot scientists', particularly in the context of planetary rovers. The challenge of the rover domain for automated planning is summarised in [Bresina *et al.*, 2002]: the problem has a continuous state and action space, partially observable state, and uncertain action effects. Actions can be executed concurrently, take a finite amount of time, and often have to be executed within a specific time window. Finally, the plan must be constructed taking into account limited resources, most notably limited battery power.

Progress on this problem has addressed the uncertain action outcomes [Dearden *et al.*, 2003] and continuous state variables [Feng *et al.*, 2004; Mausam *et al.*, 2005; Kveton *et al.*, 2006], but not the partial observability. For the vent prospecting domain, the partial observability (unknown vent locations) turns out to be what makes the problem hard to solve, so most of the relevant previous work comes from other areas within robotics. In this thesis, prior work is covered at the most appropriate place: conventional methods for vent prospecting and methods for robotic chemical plume tracing in Chapter 2; mapping methods and POMDP theory in Chapter 3; POMDP domains similar to the vent prospecting domain in Chapter 4; general adaptive robotic exploration methods in Chapter 5; current methods for solving POMDPs in Chapter 6; and orienteering problem solvers in Chapter 7.

1.6 Document Structure

The remainder of this thesis is structured as follows: in Chapter 2, the oceanography background and current methods for finding hydrothermal vents are described in more detail, and methods used for benchmarking the performance of the novel approaches developed in this thesis are presented. Chapter 3 contains some necessary background material, in particular occupancy grid methods, Jakuba's occupancy grid algorithm, and an overview of POMDPs. Then Chapter 4 lays out the exact model of the problem, including the details of the simulation environment used and the POMDP formulation of the problem.

Chapters 5, 6 and 7 describe the contributions outlined in Section 1.4 above. Each chapter is self-contained, and includes a summary of relevant previous work, the novel algorithms themselves, results for that family of algorithms, and a discussion. Chapter 5 discusses entropy-based algorithms and results, Chapter 6 covers information-lookahead (online POMDP) work, and Chapter 7 describes how an orienteering problem approximation can be used to improve the previous solution methods.

Finally Chapter 8 presents a summary and overall evaluation of this work, and describes areas for future research.

Chapter 2

Oceanography Background

2.1 Hydrothermal Vents

This section provides an overview of the geophysical processes behind hydrothermal vents, and the characteristics of vents. For a longer introduction, see for example [WHOI, 2005] or [Garrison, 2002], and for a more detailed treatment see for example [Parson *et al.*, 1995; German and von Damm, 2003; Pirajno, 2009].

Hydrothermal vents form because the tectonic plates that make up the crust of the Earth are being pulled in different directions by convection currents in the liquid mantle beneath the crust. The movement of plates leads to three different types of plate boundary: spreading centres where plates are moving apart, subduction zones where they are moving together such that one plate slides under another, and transform boundaries where plates are sliding past one another. Vents form on spreading centres, where magma flows up from the mantle to form new ocean floor. Spreading centres create oceanic ridges (such as the Mid-Atlantic Ridge), which are submerged mountain ranges commonly found running along the centre of an ocean, and often rising 2 km above the normal level of the ocean floor.

A hydrothermal vent forms when cold seawater seeps down through cracks in the rocks on an oceanic ridge and comes into contact with the hot magma below. This heats the water up, and convection causes it to rise back up to the sea, dissolving minerals and metals from the surrounding rocks on the way. Water then emerges through a vent with a very different chemical composition to the surrounding seawater – it is more acidic, and contains dissolved metals and minerals including iron, copper, manganese, zinc, methane, sulphides and sulphates. Water from the hottest vents reaches temperatures of 350–400°C, and the chemicals it contains react with oxygen-rich seawater to form clouds of black particles. These most spectacular vents are nicknamed 'black smokers' (an example is shown in Figure 2.1), and often exit from tall chimneys that have been gradually built up by particles they deposit.

The size, temperature, and output volumetric flow rate vary between individual vents depending on a variety of factors, such as the thickness of the oceanic crust on the ridge, the amount of magma activity below the crust, the size and depth of the rock fissures the water flows through, and how much the hot hydrothermal fluid mixes with cold seawater before exiting the crust. Hydrothermal activity also varies considerably along a ridge axis, with magma "hot spots" and other considerations causing vents to be found in "vent fields" clustered together at intervals along the ridge.

Hydrothermal vents are thought to exist along all mid-ocean ridge segments, but only about 20% of the ~67000 km of the global ridge system has been surveyed at all for vent activity [Baker and German, 2004]. Vents are very useful for understanding the heat flow between the Earth's mantle and the oceans, which is key to much of the planet's climate. They also support communities of unique life forms, including tube worms several feet long, and giant white crabs, which all obtain their energy

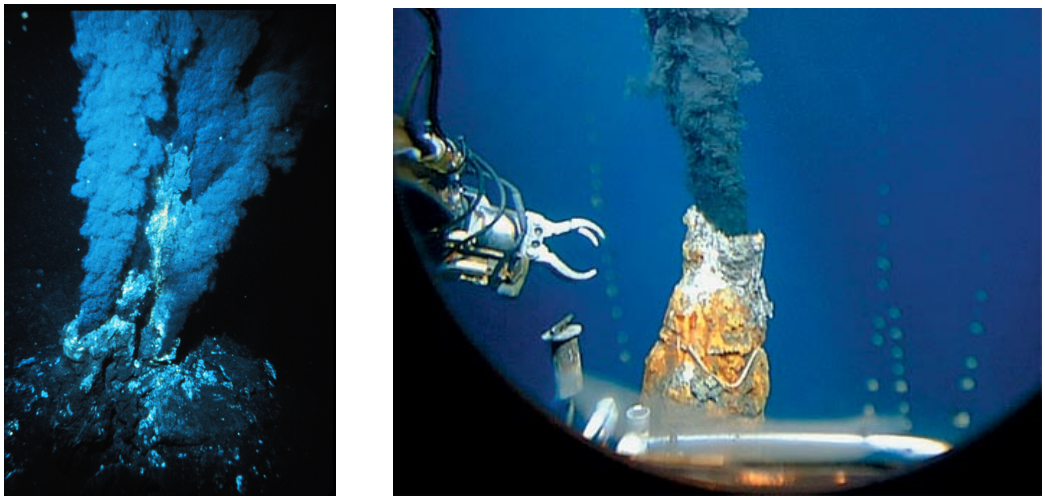


Figure 2.1: Examples of black smokers, a particularly dramatic type of hydrothermal vent. Photo credits: (left photo) NOAA photo library, (right photo) Meg Tivey, WHOI

directly or indirectly from the hot, chemically enriched hydrothermal fluids [Gold, 1992]. However, locating vents is a complex and costly process, partly because of the scale of the oceanic ridge system that must be surveyed. Further, vents are usually found at depths of 2 km or greater, where the ocean is completely black, and the water pressure means it is very expensive to use a human-occupied vehicle (HOV).

2.2 Vent Prospecting

2.2.1 Research Cruise Overview

This section describes the process by which hydrothermal vents are found, and describes some of the key sensor systems used in locating them [German *et al.*, 2008; Baker *et al.*, 2007; Baker *et al.*, 1995].

The hydrothermal fluid emanating from vents contains chemical and physical "tracers" which identify it as coming from a vent, for example the presence of dissolved methane. Fluid rising from a vent will mix with cold sea water, and continue rising and cooling until it reaches a height where it has the same buoyancy as the surrounding water (generally at around 150–400 m). The fluid, still containing tracers from the vent, then disperses in a plume driven by the current as shown in Figure 2.2. Several issues make it hard to detect this plume: firstly, the vent water is mixed with sea water and is not significantly different in temperature from the surrounding water (however, the interaction between salinity, temperature and density in seawater does mean hydrothermal fluid contains a "temperature anomaly"). Secondly, the current flow is not constant, but will change direction every few hours. Thirdly, sea water often separates into vertical layers with different densities, and hydrothermal fluid will not easily cross these layer boundaries, which means it can sometimes become trapped at a higher or lower altitude than normal. Finally, the water column movement is somewhat turbulent, and tracers will only appear in the plume water intermittently at best. In fact, it is possible for vent tracers to become trapped in a vortex, and there to be no trail leading back to the vent.

A typical research cruise with the aim of finding vents will first try to make contact with a non-buoyant hydrothermal plume, and then home in on the source of the plume, making use of a variety of methods:

1. Gathering low-resolution bathymetry data (XYZ data describing the relief of the ocean bottom) over several hundred kilometres of the ridge axis, using the ship's multibeam sonar sensors.

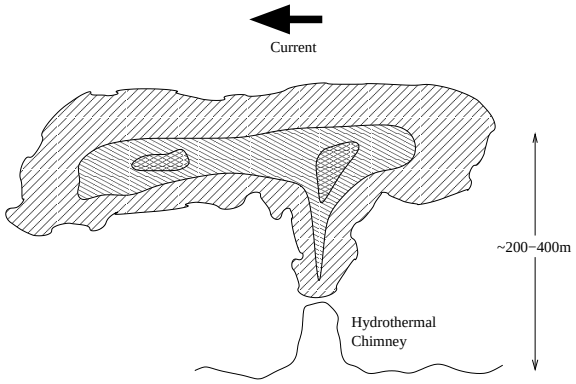


Figure 2.2: Diagram showing the typical dispersion of a hydrothermal plume (not to scale).

2. CTD casts. CTDs are Conductivity/Temperature/Depth sensors, which are attached to a cage containing sampling bottles which is slowly lowered to the ocean floor while keeping the ship on station in a fixed location.

3. Sensor platforms towed behind the ship, at a depth of a few hundred metres above the sea floor, comprising CTD, bathymetry, and often other sensors. To obtain a picture of vertical variation in the water column, either a CTD tow-yo is used where a CTD instrument is raised and lowered while being towed, or additional sensors are attached at intervals along the tow cable. These methods are shown pictorially in Figure 2.3.

4. A submersible, whether ROV, AUV or HOV, is needed for the exact localisation and surveying of vents.

Bathymetry data is captured by the research ship's on-board swathe bathymetry sensors, which can map a region several kilometres wide in a single pass. Bathymetry can be used by experts on vent geology to identify likely locations for vents based on the relief of the area, and can guide deployments of the CTD and deep-tow platforms. CTD casts and deep-tow data will hopefully identify vent plume(s), which may be detected tens of kilometres from the source vents. Despite this, variations in plume intensity together with bathymetric data often enable marine scientists to determine the location of a vent field to within a few kilometres. CTD tow-yos can narrow this down to a few hundred metres, and then low-altitude surveys in a submersible can find and photograph the vents.

2.2.2 Chemical and Physical Tracers

CTDs, deep-towed platforms and ROVs/AUVs can all make use of similar biogeochemical sensors to track down vents, which I describe in this section. Some of these track *conservative* chemicals, which do not react or degrade (but may be diluted), and some track *non-conservative* quantities. The important sensors are [Baker *et al.*, 1995]:

Optical backscattering When the hydrothermal fluid meets cold sea water, many of the minerals precipitate out into particle form, making the vent water very cloudy. The particle concentration in the water can be estimated by measuring the amount of light reflected back using a *light scattering sensor* (LSS, or *nephelometer*). This is a non-conservative quantity, as the particles gradually settle out of the water. The background level is nearly constant below 1000 m, which makes the readings easy to interpret.

Methane Dissolved methane is commonly used, as it is an unambiguous indicator of vent activity and is easily measured. It is non-conservative.

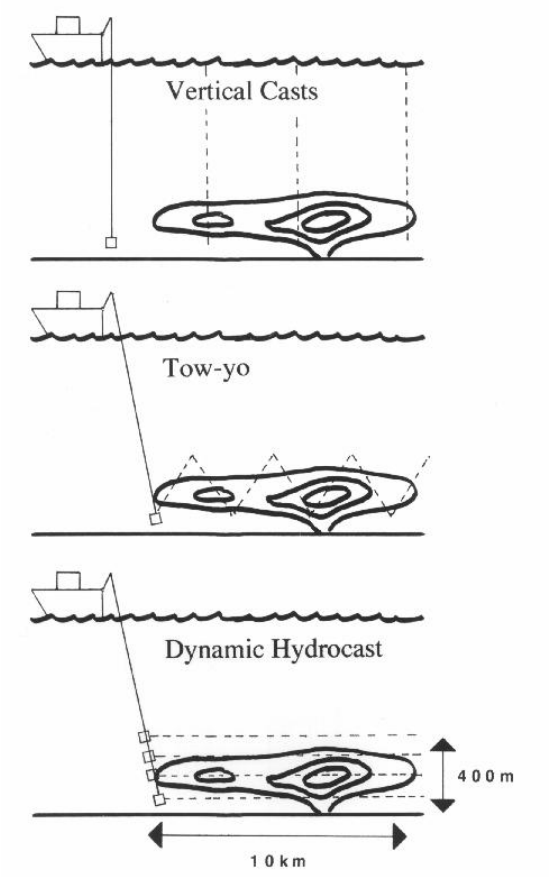


Figure 2.3: Diagrammatic representation of basic methods for hydrothermal plume detection. (Figure taken from [Baker *et al.*, 1995].)

Reduction potential (often referred to as Eh). Redox potential measures the chemical reactivity of the water; hydrothermal water has low redox potential, as it is low in oxygen. Eh is useful as strong changes in it are only observed close to a vent, within a few hundred metres.

Manganese, Iron, and other metals Manganese in particular is often used as it is enriched by several orders of magnitude in hydrothermal fluid compared to normal seawater. It is nearly conservative.

Potential temperature versus salinity anomalies While the temperature of hydrothermal fluid far from the vent is indistinguishable from that of the surrounding water, it will be identifiable by a change in the normally linear relationship between potential temperature and salinity in a given region of ocean. Such anomalies can be found by examining a series of CTD data, as salinity and potential temperature are just functions of conductivity, temperature and density.

Note that in Chapter 3 I describe the mapping method used in this thesis, the occupancy grid algorithm of [Jakuba, 2007]. This algorithm compresses measurements from these tracers into a single binary detection, and therefore abstracts away the specific qualities of tracers detailed above. Jakuba's algorithm is described in Section 3.1.2, and the reasons for adopting it are covered in Section 3.1.3. Drawbacks to compressing tracer measurements into a single binary observation are discussed in Section 8.1.4.

2.2.3 Submersible Platforms

Early vent prospecting was performed using HOVs with a human crew to confirm the presence and location of vents, for example the initial discovery of hydrothermal vents in 1977 used the Alvin submersible [Corliss *et al.*, 1979]. However vents are generally found at depths of two kilometres or more, where the water pressure is over 200 times atmospheric pressure. HOVs have to be engineered to keep humans safe at these pressures, and to provide them with breathable oxygen, which makes them expensive to build and expensive and risky to operate.

ROVs were the first alternative developed, robotic platforms that are tethered to a mothership by a cable providing a data and power connection. ROVs are piloted by human operators from a control room on the ship, who have access to a camera feed from the ROV, and can operate any robotic arms, sampling bottles, and other equipment fitted to the vehicle. The tether means that the ROV must stay relatively close to the mothership, so ROVs have limited speed and range, plus they require the research ship to remain in a fixed location instead of conducting further exploration itself. However ROVs are very useful for collecting biological and geological samples, and for visual exploration of an area.

AUVs have traditionally been used for pre-planned automated surveys of large areas, for example gathering bathymetry data from a low altitude above the sea floor. AUVs generally require some onboard processing to enable them to descend to and maintain a constant depth, follow a specific path (including calculating their own position by some means), and in some cases to follow the terrain of the sea floor without crashing. Path planning in a continuous 3D space with variable ocean currents is a challenging problem, and several interesting algorithms have been developed [Pêtrès *et al.*, 2007; Hert *et al.*, 1996; Kruger *et al.*, 2007], but this problem is outside the scope of this work. Recent work has focused on giving more autonomy to AUVs to optimise scientific returns (for example [McGann *et al.*, 2008b; McGann *et al.*, 2008a; Jakuba and Yoerger, 2008; German *et al.*, 2008]), but progress has been relatively slow, due to factors including the high cost of both vehicles and deployments and consequent requirement for highly reliable control systems [Saigol *et al.*, 2010b; McGann *et al.*, 2008c]. Figure 2.4 shows a photograph of the Autosub6000 AUV, which is operated by the National Oceanographic Centre, Southampton.

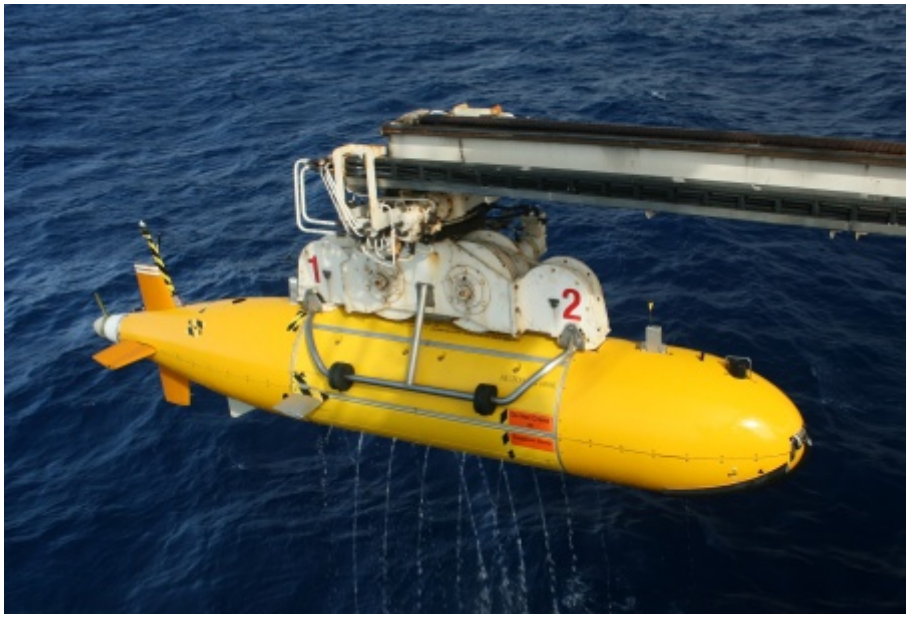


Figure 2.4: The Autosub6000 AUV operated by NOC Southampton. Photo credit: Richard Dearden.

2.2.4 Work with the AUV ABE

ABE, an oceanographic research AUV operated by Woods Hole Oceanographic Institution, is frequently used to search for hydrothermal vents. Dana Yoerger, Michael Jakuba and other members of the ABE team, working in partnership with oceanographers including Christopher German, have recently done much work to improve the methods used for hydrothermal vent prospecting [Yoerger *et al.*, 2007b; Yoerger *et al.*, 2007a; German *et al.*, 2008]. Before about 2000, finding a vent would require at least two research cruises: an initial cruise to identify regions with hydrothermal activity by detecting plumes (for example [German *et al.*, 1998]), and a subsequent cruise using a submersible to track the source of the plumes. Yoerger's team developed a method to allow vents to be found in previously unexplored sections of oceanic ridge on a single cruise. This method uses conventional ship-based means (as described in Section 2.2.1) to find the strongest plume signal (to within about 5 km), and then a programme of three ABE dives to locate a vent within 5 m and obtain photographs of the surrounding sea floor.

On each ABE dive the vehicle follows a mowing-the-lawn path, where the search area boundaries and trackline spacing are set manually based on an examination of the data obtained so far. The three phases are:

1. A survey of about 3 km x 3 km at the altitude of the non-buoyant plume (100-400 m), using OBS and Eh sensors.

2. A survey of about 1 km x 1 km at an altitude of 50 m, using multi-beam sonar as well as OBS and Eh sensors.

3. A survey at ~5 m altitude, covering 200 m x 200 m of seafloor, taking photographs as well as using the OBS, Eh and sonar sensors.

This three-phase methodology still requires the vehicle to be recovered back to the ship between dives, and scientists to examine the data recorded and plan the location and parameters of the next dive. The recovery, planning, and re-launching occupies approximately twice as much time as the vehicle spends gathering data ([Jakuba, 2007], §4.2). While surveys are designed to make use of

the maximum range of the AUV for each phase, the ABE team recognised that further efficiencies could be made by performing on-board analysis of sensor data and subsequent path planning. [Yoerger *et al.*, 2007b] describe how phase 3 ABE dives were enhanced by such on-board processing to determine the area surveyed during the last 5% of the mission. Their algorithm creates clusters of plume detections, and then chooses the highest-intensity cluster as the centre point of the survey filling the remainder of the mission time. As part of his PhD work, Michael Jakuba developed an occupancy-grid–based mapping algorithm that could identify likely vent locations [Jakuba, 2007; Jakuba and Yoerger, 2008], and this algorithm is described in Section 3.1.2. However it has yet to be run on-board the AUV, and Jakuba does not suggest a planning framework to use with his mapping algorithm.

2.3 Reactive Chemical Plume Tracing

In this section I will examine work on the basic problem of locating a chemical source with a mobile agent, which is superficially very similar to the problem addressed in this thesis. However, the vent prospecting problem has multiple sensors, multiple potential targets, and a finite mission time, all of which mean the work discussed below can not be applied directly.

Chemical plume localisation algorithms are needed because in the turbulent fluid flow environments found in the real world, there is seldom a strong or consistent enough signal for simple gradient following to be used to trace the source of an emission. Four common algorithms for locating a chemical plume using a mobile agent (known as *chemotaxis*) are reviewed in [Russell *et al.*, 2003]. Three of these algorithms are biologically inspired, copying the behaviour of the *E. coli* bacterium, the silkworm moth, and the dung beetle, and the fourth algorithm is simple gradient following.

Russell *et al.* identify two stages in chemotaxis: firstly making contact with the plume, and secondly tracking the chemical to the source. For making contact with the plume, the biological strategies were:

E. coli	Random walk
Moth	Passive monitoring from a fixed location
Dung beetle	Linear upwind path at some angle to the wind

The best strategy was found to be a straight path at 90° to the wind.

Once a chemical signal was detected, the strategies for following the plume to its source were:

E. coli	If signal increasing, rotate a small random angle and move a fixed distance; Else, choose new random direction and move a fixed distance.
Moth	First, a rapid upwind surge for about 0.4 s, then 'cast' side-to-side with increasing amplitude for 4 s, next turn in circles for approximately 10 s, and finally turn with random changes in direction for 20 s. If another chemical signal is detected at any point, the procedure is restarted.
Dung beetle	Zig-zag diagonally upwind, changing direction each time leave the edge of the plume. When contact with the plume is lost, the agent assumes it has just passed the source.
Gradient-based	Stereo sensors orient robot by turning toward the sensor with the stronger signal. Preference is given to moving upwind.

Lobsters-like behaviour was not considered by Russell *et al.*, but [Vickers, 2000] notes that plume tracing paths produced by lobsters are very similar to those of moths. Lobsters do differ in that they can use 'stereo' odor detectors to turn toward the strongest odor concentration [Weissburg, 2000], which means their behaviour is likely to be a combination of the moth and gradient-based behaviours.

Results were obtained for two paradigms, where the air flow was viscosity dominated (producing a smooth chemical gradient), and where it was turbulence dominated. For robots of more than a few centimetres in size, the authors suggest that fluid flow will almost invariably appear turbulent.

Russell *et al.* find that the *E. coli* algorithm took significantly longer than the others to find the source in the smooth gradient environment, but in the turbulent environment it failed completely. With the macroscopic airflow conditions, the gradient based algorithm tended to find the shortest path when it worked, but failed completely more often than the moth or beetle approaches, as it just carries on in a straight line if it loses the signal. The moth algorithm performed slightly better than the beetle, but the beetle approach may be superior for stronger, more consistent plumes.

Also, the authors suggest pointers for how to work out if the source has actually been found:

- Analysis of eddys in the plume, given a sensor with a fast enough response.
- Comparison to known chemical intensity, if the characteristics of the source are very well known.
- If using the dung beetle algorithm, which keeps track of the plume width as the source is approached, the point at which the width will be zero can be extrapolated.

In [Farrell *et al.*, 2003], an AUV is used to find the source of an artificially introduced chemical plume. They use a near-shore environment where the dispersion of the plume is dominated by turbulent effects, and therefore gradient-based searching cannot be used. They also implement a behaviour-based planning system to control the actions of the AUV, using six distinct behaviours inspired by the odour-localisation behaviour of moths. Switching between the behaviours is triggered by either detecting an odour signal, or failing to detect a signal for a specified period of time. To predict the exact location of the source, they use a simple test based on encountering three separate detection events within a specified distance of each other. Following seven runs used to fine-tune the parameters of the algorithm, the AUV was able to find the chemical source in seven out of eight runs.

Finally, [Ferri *et al.*, 2008] describe a vent prospecting mission where a spiral search pattern was used when a vent plume was detected, and [Jakuba and Yoerger, 2008] describe the application of Jakuba's OG algorithm to the data collected by an AUV mission.

2.4 Comparison Algorithms

Given the complexity and size of the vent prospecting domain (as described fully in Chapter 4), it is not possible to compute the optimal behaviour for a given set of observations. However, it is important to have a benchmark against which novel algorithms for this domain can be evaluated, and for this purpose I have developed three alternative behaviours. These behaviours are described more fully below, and consist of:

- An exhaustive search method.
- A reactive algorithm inspired by chemical plume tracing in nature.
- An interactive tool where a human operator guides the agent in the same simulation environment used for evaluating the novel algorithms.

All the algorithms discussed work in 2D, with the altitude of the AUV fixed. Results for the comparison algorithms are presented alongside results for the novel algorithms in Chapters 5, 6, and 7, and in particular some of the human-driven results are given in Sections 5.7.1 and 6.5.3.

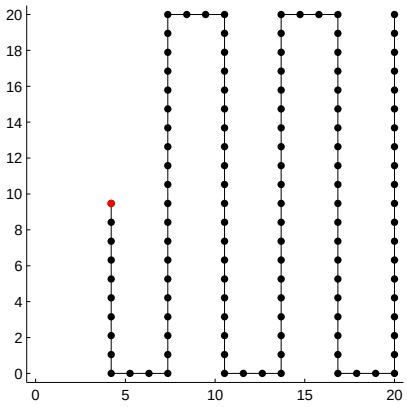


Figure 2.5: Example of the agent's path using the mowing-the-lawn (MTL) algorithm. This is an exhaustive search. The agent starts in the north-east (top-right) corner of the search area.

2.4.1 MTL and Chemotaxis

The first comparison algorithm is mowing-the-lawn (also known as ploughing-the-field), which performs an exhaustive search of a rectangular area. The vehicle moves along parallel tracklines linked by short perpendicular segments at either end of the tracklines, as shown in Figure 2.5. This approach is commonly used in marine exploration to survey an area, and the trackline spacing can be varied to obtain different survey resolutions.

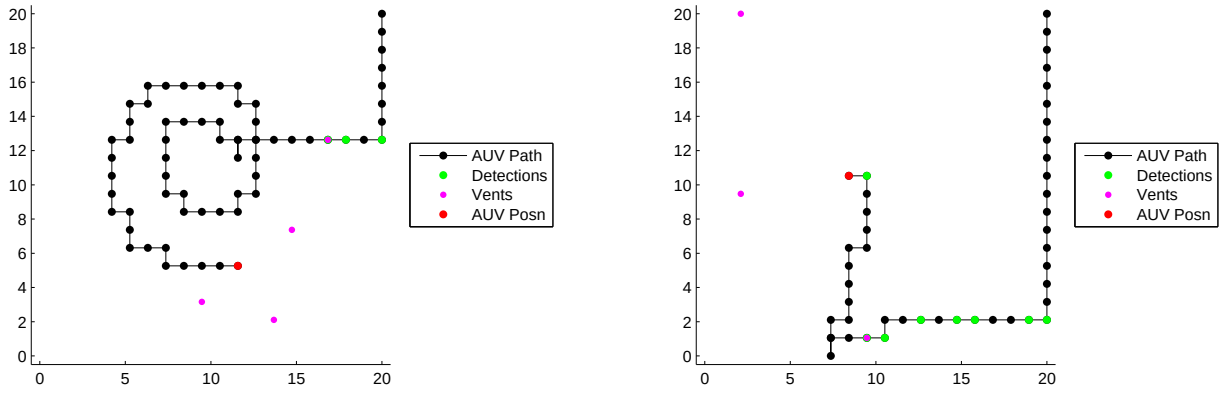
The second comparison planner was a chemotaxis method loosely based on the behaviour of the male silkworm moth, as described in [Russell *et al.*, 2003]: on detecting a plume it firstly surges directly up-current for six timesteps, and then searches in an increasing-radius spiral. If it detects a plume again at any point, the behaviour is re-started from the beginning. Prior to the first detection, the chemotaxis algorithm follows an MTL pattern, and if it tries to exit the search area at any point, it is redirected on a linear path approximately toward the centre of the area at a randomly-chosen angle. These behaviours are illustrated in Figure 2.6, and differ from the moth described in [Russell *et al.*, 2003] in that, first, the agent does not remain still while it waits for a plume detection, and second it goes straight into a spiral after completing a surge, instead of first performing a zig-zag search.

The chemotaxis algorithm is described in more detail in Appendix A.2.

2.4.2 Human-Controlled Vent Prospecting

As it is not computationally feasible to find optimal solutions to the planning problem I am addressing, a useful alternative form of evaluation is to compare the performance of my algorithms with that of humans on the same problem. For small-to-medium problem sizes, there is evidence that humans have near-optimal performance on certain types of optimisation problem. An example is the *travelling salesman problem* (TSP) which is closely related to the vent prospecting problem, as described in Section 4.8 and Chapter 7. The TSP is a path-length minimisation problem, and research has shown that humans can generate paths within a few percent of the optimal length [MacGregor and Ormerod, 1996; Vickers *et al.*, 2001]. For example, for different variations of a 20-city problem, MacGregor and Ormerod found the average path length for human subjects was between 3-10% greater than optimal, despite there being approximately 6×10^{16} different paths available[1]. In the POMDP world, [Littman *et al.*, 1995] created an interactive simulator of their problem, and found that for a 57-state version,

[1] An online interactive simulator of the TSP is available at `http://www.tsp.gatech.edu/games/tspOnePlayer.html`, and it is relatively easy to find the optimal tour from the 43 billion possible tours in a 15-city problem.



(a) The increasing-radius spiral search path initiated when the agent has not contacted a plume for six timesteps. Note that the path appears to move down one cell at the start of the spiral, but this movement is part of the linear up-current surge, and is due to the rasterising of the up-current line.

(b) The redirection towards the centre of the grid initiated when the agent attempts to leave the search area. At coordinates (7,0) the agent is directed back on a random linear path approximately towards the centre of the grid, and continues on this path until it gets another plume detection at about the middle of the search area.

Figure 2.6: Examples of the agent's path using the chemotaxis algorithm, with Subfigure a showing the spiral behaviour, and Subfigure b showing the redirect if the agent attempts to go out-of-bounds. The agent starts in the north-east (top-right) corner of the search area. Note that the vent locations (marked by pink circles) are not known to the agent.

humans were able to achieve the goal 100% of the time, which was matched by only one automated algorithm. However, for an 89-state problem, only humans managed a 100% success rate, with the best algorithm attaining just 59%.

To enable human performance on the vent prospecting problem to be evaluated, I developed an interactive version of the environment simulation, where the OG updates are done as before, but the AUV motion is controlled by a human (using the arrow keys on a computer keyboard). This simulation hides the vents (until they are found) and plumes, but shows the OG, the direction of the current and the number of timesteps remaining. A screenshot is shown in Figure 2.7.

Two different sets of experiments were performed: one set using a 40×40 grid with 10 human subjects (members of the Intelligent Robotics Lab at the University of Birmingham), and another set with a 20×20 grid and just myself as a subject. The 40×40 setup was used despite the automated algorithms being tested on a 20×20 grid because the larger grid gave the humans more scope to plan a strategy, whereas the smaller grid required an almost reactive approach. For the 40×40 configuration, 560 timesteps were allowed, and vent locations were set randomly by first sampling the number of vents from a Normal distribution $\sim \mathcal{N}(5, 2.25)$, and then distributing the vents randomly and uniformly around the grid. However, it was found that the 40×40 results were not ideal for comparing to my novel algorithms, because of the different grid size and the limited number of trials available. Therefore I decided to perform a large number of trials myself using identical conditions to those used for the automated experiments. These experimental conditions are described in Section 5.2, and include a 20×20 grid, a horizon of 133 timesteps, and trials with each of 3, 4, 5 and 6 vents.

A summary of results for the 40×40 setup is presented in Section 5.7.1, and results for the 20×20 setup are presented alongside the novel algorithm results in each of Chapters 5, 6 and 7. While it was not practical to evaluate all the novel algorithms on a 40×40 grid, given that the run-

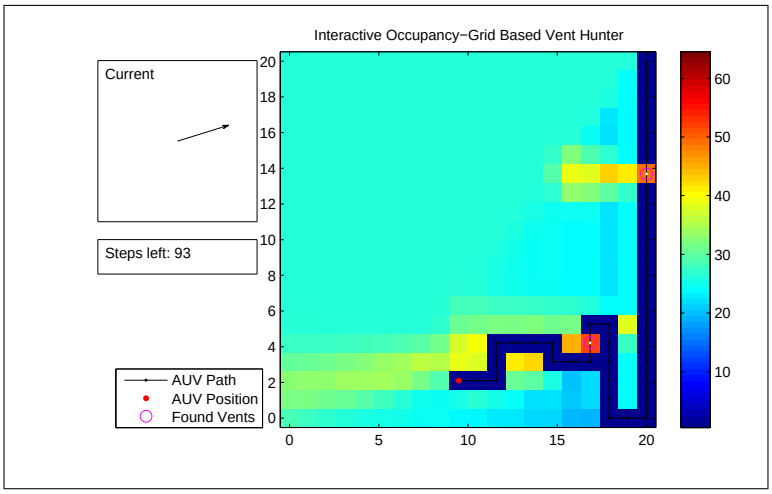


Figure 2.7: Screenshot of the human-controlled vent prospecting simulation.

times are approximately 16 times longer[2], 40×40 experiments were performed for the IL4 algorithm introduced in Chapter 6. Section 6.5.3 presents the results comparing IL4 with human-driven results using both the 20×20 and 40×40 setups.

[2]Because most algorithms are linear in the number of grid cells, and the 40×40 grid has four times as many cells, plus the number of timesteps has to be adjusted proportionately to also be four times more.

Chapter 3

Mathematical Background

This chapter covers some basic theory and algorithms that are used extensively in the rest of the thesis. Section 3.1 describes occupancy grids and in particular Michael Jakuba's OG algorithm which is used to convert sensor measurements into a map of likely vent locations. Section 3.2 presents Markov decision processes (MDPs) and partially-observable Markov decision processes (POMDPs), the formal modelling systems that are used to define the problem addressed in this thesis[1]. Section 3.2 also introduces the early, canonical algorithms for solving MDPs and POMDPs.

3.1 Mapping

The mapping component of a vent-hunting agent should create a map of vent locations, given the following problem characteristics:

- A single, mobile sensor platform.
- Several biogeochemical variables of interest, where the relevance of each variable depends on some subset of the other variables.
- Tracer dispersion that depends on water currents.
- The potential for multiple sources.

Given the uncertainties inherent in the mapping problem, probabilistic techniques are appropriate, and I chose to adopt the occupancy grid method. Section 3.1.1 covers occupancy grids in general, Section 3.1.2 describes the specific algorithm I used, and Section 3.1.3 briefly mentions other options and the reason for choosing the mapping approach I did.

3.1.1 Occupancy Grids

Occupancy grids (OGs) were developed by Hans Moravec's lab at Carnegie Mellon University in the 1980s to improve the robustness of mobile robot navigation [Martin and Moravec, 1996; Elfes, 1989]. The lab's adoption of cheap sonar range-finding sensors, which are noisy and detect objects over a broad cone, meant that their previous mapping method no longer worked. This previous method relied on finding and tracking features of the environment, but these features could not be distinguished by the sonar sensors. Instead they developed a grid mapping technique where the area to be mapped is divided into rectangular cells, each of which is designated as either empty or occupied by some obstacle, for example a wall or a filing cabinet in an indoor office environment. The occupancy grid

[1] The formal definition of the problem is given in Chapter 4

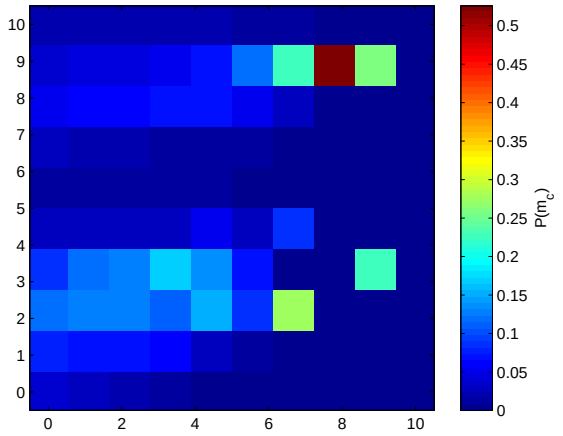


Figure 3.1: Example of an OG, where the colour of each cell indicates its probability of occupancy, $P(m_c)$.

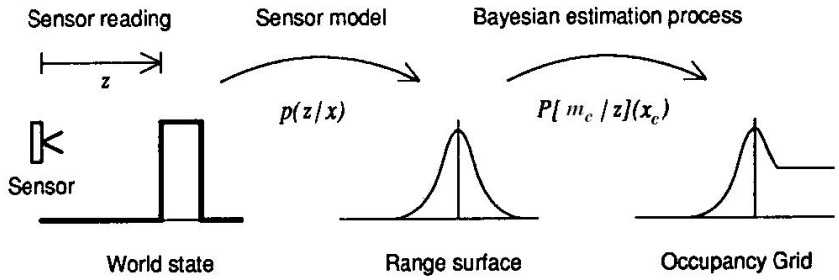


Figure 3.2: Derivation of the inverse sensor model used by the standard occupancy grid algorithm. (Figure taken from *[Elfes, 1989].*)

stores the probability of each cell being occupied, and each new sensor reading is incorporated into the OG using a Bayesian update rule, giving a very robust map and allowing much more effective robot navigation than the previous approaches [Martin and Moravec, 1996]. An example of an OG map is shown in Figure 3.1.

Formally, the true map $\mathbf{m}$ consists of C boolean values indicating whether each of C cells is occupied, $\mathbf{m} = (m_1, m_2, \ldots, m_C)$, and the occupancy grid $\mathbf{O}$ consists of C real values $P(m_c)$, the probability that cell c is occupied, where $0 \leq P(m_c) \leq 1$. OG algorithms require a known prior probability of occupancy for cells, P_c^p, and then evidence from sensor readings is incorporated incrementally by leveraging a *sensor model*. Classic OG methods need an inverse sensor model of the form $P(m_c|z)$, the probability of a cell being occupied given a sensor reading z (and an implicitly known sensor/robot pose). Often the specification for a range-finding sensor will be of the form $P(z|x)$, where x is the true distance from the sensor to the nearest obstacle in the sensor cone, and this will have to be converted to the correct form. Bayes' rule can be used to get $P(x|z)$ from $P(z|x)$, and then a model for $P(m_c|z)$ is obtained using $P(x_c|z)$ (where x_c is the distance from sensor to cell c). The probabilities for cells located at $z+\delta$ and further away are set to 0.5, as the model cannot predict their occupancy status (where δ is set to some small value, for example one standard deviation of the distribution $P(x|z)$). This procedure for deriving the inverse sensor model is shown graphically in Figure 3.2.

A key assumption underlying OG methods is that *the probability of each cell being occupied can be calculated independently of the probabilities for all the other cells*. If sensor measurements from time 1 to time t are denoted by $\mathbf{z}_{1:t}$, we are interested in $P(m_c|\mathbf{z}_{1:t})$. Separating the measurements into z_t and $\mathbf{z}_{1:(t-1)}$ and applying Bayes' rule gives

$$P(m_c|\mathbf{z}_{1:t}) = \frac{P(z_t|m_c,\mathbf{z}_{1:(t-1)})P(m_c|\mathbf{z}_{1:(t-1)})}{P(z_t|\mathbf{z}_{1:(t-1)})}$$

for the occupancy probability of cell c. We then make another important assumption: if we know the true occupancy status of cell c, then the probability of observing z_t is independent of all previous observations, so $P(z_t|m_c,\mathbf{z}_{1:(t-1)}) = P(z_t|m_c)$. We call this the *conditional independence of measurements* (CIM) assumption, after [Jakuba, 2007], and it gives

$$\begin{aligned} P(m_c|\mathbf{z}_{1:t}) &= \frac{P(z_t|m_c)P(m_c|\mathbf{z}_{1:(t-1)})}{P(z_t|\mathbf{z}_{1:(t-1)})} \\ &= \frac{P(m_c|z_t)P(z_t)P(m_c|\mathbf{z}_{1:(t-1)})}{P(m_c)P(z_t|\mathbf{z}_{1:(t-1)})} \\ &= K\frac{P(m_c|z_t)P(m_c|\mathbf{z}_{1:(t-1)})}{P(m_c)} \end{aligned}$$

where the second step follows from a second application of Bayes' rule, and the third step has introduced $K = {}^{P(z_t)}/_{P(z_t|\mathbf{z}_{1:(t-1)})}$. We now have the new occupancy probability in terms of the sensor model $P(m_c|z_t)$, the occupancy estimate from the previous timestep $P(m_c|\mathbf{z}_{1:(t-1)})$, and the prior probability $P(m_c) \triangleq P_c^p$. The probability of a cell being empty, $P(\neg m_c|\mathbf{z}_{1:t}) \equiv 1 - P(m_c|\mathbf{z}_{1:t})$, can be found by an identical derivation to be

$$P(\neg m_c|\mathbf{z}_{1:t}) = K\frac{P(\neg m_c|z_t)P(\neg m_c|\mathbf{z}_{1:(t-1)})}{P(\neg m_c)}$$

Using the ratio of the occupied to empty probabilities means that the K terms cancel out, leading to a simple update rule:

$$r_{c,t} \triangleq \frac{P(m_c|\mathbf{z}_{1:t})}{P(\neg m_c|\mathbf{z}_{1:t})} = \frac{P(m_c|z_t)}{1-P(m_c|z_t)}\frac{1-P_c^p}{P_c^p}r_{c,t-1}$$

Finally, most implementations calculate and store $\log r_{c,t}$ so that the update on each timestep is an efficient addition rather than a multiplication. The occupancy probability can easily be recovered using

$$P(m_c|\mathbf{z}_{1:t}) = 1 - \frac{1}{1+\exp(\log r_{c,t})}$$

3.1.2 Occupancy Grid Algorithm for Vent Finding

3.1.2.1 Introduction

The OG algorithm used in this thesis is one that was developed specifically for mapping hydrothermal vents, and is due to Michael Jakuba. This section provides an overview of the work contained in Jakuba's PhD thesis [Jakuba, 2007], which was completed under a joint program between the Woods Hole Oceanographic Institution (WHOI) and the Massachusetts Institute Of Technology (MIT).

Jakuba's thesis describes an OG algorithm for vents, where cell occupancy is defined by whether or not a cell contains a vent. Classic OG algorithms do not work well for vent mapping because they use the CIM assumption, that a sensor reading is determined by a single test cell and is independent of previous sensor readings. However, in vent mapping, the sensor can register a plume emitted from

any of a large number of clustered cells, and previous sensor readings are vital to help narrow down which cells are likely to have been responsible. As Jakuba points out, the errors induced by the CIM assumption are exacerbated when the prior probability of occupancy is very low (as it will be for vents, because only a handful of vents will be found over a search area of several square kilometres).

3.1.2.2 Binary Forward Sensor Model

To develop a more accurate OG algorithm, Jakuba firstly assumes a sensor system which outputs a binary detection (denoted by d) or non-detection of a hydrothermal plume. Given this, he builds a specific kind of forward model for the probability of a detection given a map showing which cells are occupied, which incorporates the possibility of a false positive detection (d^F) with probability $P(d^F) \triangleq P^F$. The basic building block for this model is a distribution for $P(d_c|m_c) \triangleq P_c^d$, where d_c indicates a plume detection due to a vent in cell c. The specific distribution P_c^d used in this thesis is described in Section 4.6.

Sensor models must then have the form

$$P(d|\mathbf{m}) = 1 - \left[(1 - P^F) \prod_{c \in \mathscr{M}} (1 - P_c^d) \right] \tag{3.1}$$

to be used with Jakuba's algorithm, where $\mathscr{M}$ is the set of all cells containing a vent. Equation 3.1 basically states that the probability of detecting a plume is one minus the probability of not getting a false positive and not getting a detection from any of the vents in the map. Note that this is a *forward* model giving the probability of an observation given a map, which is a more natural way of modelling a system than the inverse model required by the standard OG algorithm. Also the form of Equation 3.1 does not restrict the algorithm to hydrothermal vent mapping; it is easy to define the model for other sensors, such as sonar range-finders, in that form.

3.1.2.3 Inverse Sensor Model

As was seen in Section 3.1.1, the core map update in OG algorithms requires an inverse sensor model of the form $P(m_c|z)$, the probability of a cell being occupied given an observation. Starting from a forward model of the form $P(z|\mathbf{m})$ (the probability of an observation given a map), the inverse model $P(m_c|z)$ can be found by applying Bayes' rule and then marginalising over all possible maps:

$$P(m_c|z) = \frac{\sum_{\mathbf{m}:m_c} P(z|\mathbf{m})P(\mathbf{m})}{\sum_{\mathbf{m}} P(z|\mathbf{m})P(\mathbf{m})} \tag{3.2}$$

where the summation in the numerator is over all maps with cell c occupied. This summation is generally intractable, given that there are 2^C possible maps for an OG with C cells.

However, the form of Equation 3.1 allows these sums over maps to be converted to products over cells, which makes them tractable. Specifically, substituting Equation 3.1 into Equation 3.2, with $z \in \{d, \neg d\}$ where d represents a plume detection and $\neg d$ a non-detection, leads to the following formulae for cell occupancy:

$$P(m_c|d) = \frac{1 - \left(1 - P^F\right)\left(1 - P_c^d\right)\prod_{i \neq c}\left(1 - P_i^d P(m_i)\right)}{1 - (1 - P^F)\prod_{i=1}^{C}\left(1 - P_i^d P(m_i)\right)} P(m_c) \tag{3.3}$$

$$P(m_c|\neg d) = \frac{\left(1 - P_c^d\right)}{1 - P_c^d P(m_c)} P(m_c) \tag{3.4}$$

where $P(m_c)$ represents the prior P_c^p. This result can be obtained using the assumption that the prior occupancy probability of all cells is independent, an iterative method for replacing the sums over maps by products over cells, and basic simplification (see [Jakuba, 2007] App. B for full details).

3.1.2.4 Exact Algorithm

Using the forward model of Equation 3.1, Jakuba was able to develop an exact algorithm to calculate $P(m_c|\mathbf{z}_{1:t})$, the cell occupancy probabilities given a sequence of observations. This algorithm avoids the CIM assumption, and has a simple recursive form for non-detections, but requires batch processing of all previous detections to incorporate a new detection. This batch processing is exponential in the total number of detections, which makes the algorithm impractical for real map building. The exact algorithm is useful in that first it proves that the posterior probabilities given a sequence of non-detections remain independent (as suggested by the fact that cells other than c do not appear in Equation 3.4), and second in that it formed a starting point for Jakuba to develop tractable approximate OG algorithms.

3.1.2.5 Independence of Posteriors Algorithm

Jakuba developed two families of approximate algorithms that require weaker independence assumptions than the CIM assumption, but are tractable and produce significantly better results in low-prior environments than the standard OG algorithm. These are termed the *conditional independence of detections* (CID) and *revised prior* (or *independence of posteriors*, IP) algorithms. The CID algorithm is based on a relaxed version of the CIM assumption, where independence holds only between detections, so that given only non-detections, the algorithm computes the exact posterior probabilities. However, the revised prior (IP) algorithm was found to perform better overall, and was the algorithm used in this thesis, so I will describe only the IP algorithm in detail here.

The revised prior (IP) algorithm is derived by assuming that the posterior occupancy probabilities for all cells are independent, so that

$$P(\mathbf{m}|\mathbf{z}_{1:t}) = \prod_{c\in\mathscr{M}} P(m_c|\mathbf{z}_{1:t}) \prod_{c\notin\mathscr{M}} P(\neg m_c|\mathbf{z}_{1:t}) \tag{3.5}$$

This independence-of-posteriors assumption is preferable to the CIM assumption because first it is a necessary assumption in the OG framework if we wish to recover the posterior of the full map $\mathbf{m}$. Second, the IP assumption is true for non-detections (unlike the CIM assumption), and finally it is an intuitively better assumption that a sequence of sensor readings can determine the occupancy of each cell independently than the assumption that a single sensor reading can be determined by a single cell.

The IP assumption (Equation 3.5) leads to a recursive update algorithm, which essentially uses the posterior probabilities from the previous timestep as "revised prior" inputs to the next timestep. This means that, while the algorithm produces more accurate results than the standard algorithm, the results depend on the order updates are applied in.

Non-detections and detections are processed differently; for non-detections, the updated odds-ratio r'_c is computed using

$$r'_c = \left(1 - P_c^d\right) r_c \tag{3.6}$$

and for detections r'_c is computed by

$$r'_c = \frac{1 - \left(1 - P^F\right)\left(1 - P_c^d\right)\prod_{i\neq c}\left(1 - P_i^d P(m_i)\right)}{1 - \left(1 - P^F\right)\prod_{i\neq c}\left(1 - P_i^d P(m_i)\right)} r_c \tag{3.7}$$

Note that in Equation 3.7 and in the remainder of this thesis, I will use the notation $P(m_c)$ as shorthand for the current estimate of the occupancy probability $P(m_c|\mathbf{z}_{1:t})$, unlike Jakuba who uses $P(m_c)$ for the prior at $t = 0$ (which I refer to using P_c^p exclusively). The full IP algorithm (Alg. 6 in [Jakuba, 2007]) is given as Algorithm 3.1.

Algorithm 3.1 The semi-recursive occupancy grid (*srog*) IP algorithm from [Jakuba, 2007]. Here $\mathbf{r}$ is the odds-ratio for each cell, $r_c \triangleq \frac{P(m_c)}{P(\neg m_c)}$, $\mathbf{P^d}$ is the vector of detection probabilities given an occupied cell, $P_c^d \triangleq P(d_c|m_c)$, z is the observation at the current timestep, and finally C is the total number of cells in the OG.

```
Procedure srog(r, P^d, z) : r'
  If z = d // plume was detected
    For c = 1 : C
      P(m_c) = r_c / (1 + r_c)
    Endfor
    For c = 1 : C
      r'_c = [1 - (1 - P^F)(1 - P^d_c) ∏_{i≠c} (1 - P^d_i P(m_i))] / [1 - (1 - P^F) ∏_{i≠c} (1 - P^d_i P(m_i))] r_c
    Endfor
  Else // non-detection
    For c = 1 : C
      r'_c = (1 - P^d_c) r_c
    Endfor
  Endif
Endproc
```

Equations 3.6 and 3.7 are simply rearrangements of Equations 3.4 and 3.3 in terms of $r_c \triangleq \frac{P(m_c)}{P(\neg m_c)}$, but where $P(m_c)$ represents the "revised prior" rather than the prior P_c^p at $t = 0$. However, Jakuba's thesis provides a proof that, given the IP assumption, these updates calculate the exact posterior probabilities.

Finally, Jakuba also developed an extended version of the IP algorithm, where the assumption of Equation 3.5 is relaxed slightly by grouping the observations that were plume detections into sets. Then the posterior occupancy probabilities conditioned on detections within a set are allowed to depend on each other, and independence is required only between sets of detections. This relaxation comes at the cost of needing a modified CIM assumption, where observations must be independent between sets, plus additional computational requirements. All work in this thesis used the standard IP algorithm, and not this extended IP algorithm.

3.1.2.6 Jakuba's Other Contributions

Jakuba's thesis also makes two contributions directly concerned with applying his OG algorithms to the problem of mapping vents using an AUV: firstly a method for extracting binary detections from a set of scalar measurements of various plume tracers, and secondly a physical model for hydrothermal plume evolution. This physical model allows for multiple source vents, outputs the probability of a detection, and is of the correct form to use with Jakuba's OG algorithms. Chapter 4 describes this model in more detail.

3.1.3 Choice of Mapping Algorithm

A wide variety of algorithms are used in robotics for generating maps based on noisy observations, of which OG methods are only one algorithmic family. This section briefly mentions some of the alternative approaches and justifies the choice of Jakuba's algorithm for mapping.

The *Kalman filter* [Arulampalam *et al.*, 2002; Thrun, 2002b] is a mainstay of state estimation given noisy data, and is based on observation and state transition functions that are both linear functions of the state, with additive Gaussian noise. As these functions are usually not linear in robotics domains, the *extended Kalman filter* (EKF) is often used instead, where a Taylor expansion is used to create locally-linear approximations of the observation and transition functions. In the *simultaneous localisation and mapping* (SLAM) problem [Dissanayake *et al.*, 2001; Thrun, 2002b], the state consists of both the agent's location, and the map as defined by the coordinates of a set of 'landmarks'. The EKF is often used to solve SLAM problems by estimating a Gaussian joint posterior over the agent location and all landmarks. However the observation model for a plume given a vent location is highly non-linear, so the EKF is not likely to be an effective mapping method for this domain.

Particle filters [Arulampalam *et al.*, 2002; Thrun, 2002a] are an alternative to the EKF that work well for highly non-linear observation and transition functions. The idea is to represent the probability distribution over states by a sample of particles, each representing a particular state, and apply the non-linear transition function directly to each particle. The particles are then weighted according to their likelihood given the most recent observation, and the set of particles is re-sampled using this weighting. While particle filters scale badly to high dimensions and therefore do not work well for mapping, *Rao-Blackwellised* particle filters [Doucet *et al.*, 2000; Murphy and Russell, 2001] have largely overcome this limitation. Rao-Blackwellisation works by separating the state space into two sets of variables, for example the map and the pose of the robot, and tracking one set (the robot's pose) using a particle filter, and the other set (the map) using an EKF associated with each particle, and conditioned on the value of the pose represented by that particle.

Jakuba's occupancy grid algorithm was adopted for several reasons, starting with the fact that it allows any number of potential vents to be represented in the same map. Most alternative approaches would require a separate map for each vent-count, which would make it harder to develop and evaluate planning algorithms. A related point is that using an OG approach avoids the need to specify a maximum number of vents the agent can find. The next reason is that any other choice would have necessitated the development of a novel mapping algorithm to work for hydrothermal vents, as no existing algorithm would cope acceptably with the unique features of the domain. Avoiding the creation of a mapping algorithm allowed me to focus on solving the planning problem, and therefore to produce a complete solution to automated vent prospecting with AUVs. Further, if I had decided to produce an alternative solution to a problem that had already been solved by Jakuba, the only sensible way of evaluating it would be to compare it to Jakuba's algorithm, which would mean I needed an implementation of Jakuba's algorithm anyway. Having opted to use Jakuba's OG method, Mike Jakuba was generous enough to provide me with his implementation of the algorithms to use in this work[2].

The occupancy grid algorithm is a key building block of the work presented in this thesis, as it is used to create the occupancy grid that forms the input to the planner. The OG algorithm acts as a pre-processing stage for raw observations, as it converts oceanographic readings into a probabilistic map and isolates the planning algorithm from any issues with interpreting sensor data.

[2]The code supplied implemented the *extended IP* algorithm, which clusters plume detections into groups and enforces independence only between groups. However, parameters in the code were set to switch this grouping off, in which case the algorithm reverted to the standard IP algorithm described in Section 3.1.2. The codebase was also subject to minor modifications: a bug-fix in the OG update for boundary cases which produced infinite values, changes to the environment model and simulation to account for a slightly different observation model (see Section 4.3.4), and performance improvements by vectorising the MATLAB code for the environment model, which resulted in the algorithm running approximately 10 times faster.

3.2 MDP and POMDP Background

Having described the algorithm used to produce an estimate of the state of the world in Section 3.1 (Jakuba's OG algorithm), I now present the framework used to model the problem of what action to take, given this state. This is a planning problem, which essentially means determining a sequence of actions to be applied to a system to transform it from an initial state to some target state. Therefore, to specify a planning problem, a description of the possible system states, the actions available, and a start state and goal state are needed. In classical planning domains, states are represented by logical variables, and actions are specified by their preconditions and postconditions [Ghallab *et al.*, 2004].

The vent prospecting problem has several properties that make it hard to formalise it in the same way as classical planning problems. The major obstacle is partial observability: the goal would naturally be defined in terms of the location of the vent(s), but these are not known in advance. Further, there may be several vents to be found, meaning there may be multiple "goal states" which is problematic even before time constraints are considered. Time constraints often mean not all goals can be achieved, so they have to be prioritised. These properties mean that the vent prospecting problem cannot be expressed in the format required for classical planning methods, and techniques for planning under uncertainty have to be used.

3.2.1 MDPs

MDPs are a mathematical framework for posing planning and learning problems where the system can be in one of a set of states, and the agent takes actions that cause the system to stochastically transition between states. They are a discrete time model where actions are assumed to be instantaneous and an action must be chosen on each timestep. The agent receives a reward for each action/state pair it passes through (where the action is chosen by the agent but the state is 'chosen' by the environment), and the goal of the agent is to maximise its total reward over some horizon. MDPs are used widely in robotics and are well suited to problems where there is uncertainty in action outcomes and/or there are multiple goal states that the agent should be rewarded for visiting.

Formally, the system is in a state s taken from a set $\mathscr{S}$ of possible states at every timestep, and the agent selects an action a from a set $\mathscr{A}$ of possible actions. The successor state s' on the following timestep is stochastically selected by the environment according to a state transition function $T = P(s'|s,a)$. This form of the state transition function implies that MDPs must have the Markov property: future states are independent of all previous states, given knowledge of the current state. Finally, the agent receives a reward r on each timestep, determined by the reward function $w = R(s,a)$. As well as a positive 'desirability' term for states, the reward function R can include a negative term modelling the cost of actions. An MDP can be specified by the tuple $\langle \mathscr{S}, \mathscr{A}, T, R \rangle$.

The agent selects actions according to some policy $\pi(s)$; an optimal policy is one that maximises the total expected future reward, either over a finite horizon (i.e. a fixed number of timesteps), or over an infinite horizon. For the infinite horizon case either future rewards must be discounted by some factor γ, $0 \leq \gamma < 1$, to ensure the expected reward is finite, or a measure of average reward must be used. The discounted future reward following time t is given by

$$R_t = \sum_{k=0}^{\infty} \gamma^k w_{t+k+1}$$

Figure 3.3 shows the flow of decisions between the agent and environment in the MDP framework.

A classic problem MDPs are applied to is a robot navigating in a maze, and an example of this type of problem is shown in Figure 3.4. In such a problem, the goal will be to get to one of a set of terminal cells, but the agent may also gain a positive or negative reward from visiting intermediate cells (such as the ones marked with battery symbols in the figure). The transition function may be

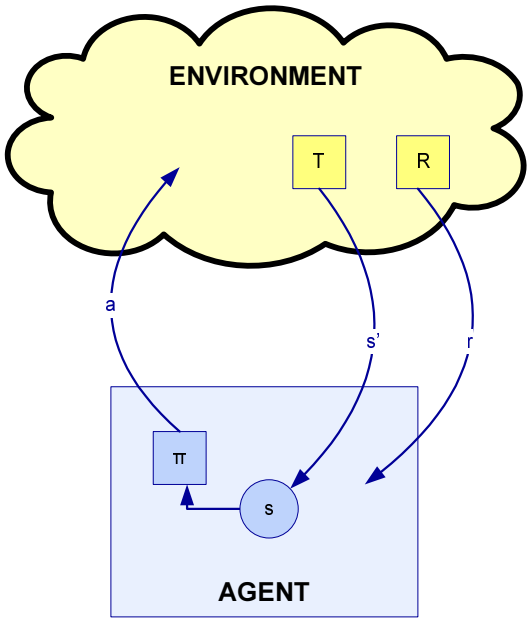


Figure 3.3: Information flow between the planning agent and the environment in an MDP.

probabilistic, for example the robot may have an 80% chance of reaching the desired cell, and a 10% chance of accidentally moving to the cell on either side.

The optimal policy for an MDP can be found by associating a value, $V(s)$, with every state, representing the expected future discounted reward if the agent were to be in that state and follow the optimal policy thereafter. States with higher values are therefore more desirable to the agent. $V(s)$ is given by the Bellman optimality equation

$$V(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right] \tag{3.8}$$

which defines a set of simultaneous equations, one for each state in the system. This formulation is known as *dynamic programming*, and the equations can be solved using the *value iteration* algorithm. Value iteration starts by setting V to an arbitrary constant for all states, then iterates repeatedly over all states, using Equation 3.8 to update the value of each state. The iteration continues until the largest change in V for any state falls below a threshold value δ. Value iteration is guaranteed to converge on the optimal values, given finite rewards and a discount factor $0 \leq \gamma < 1$ [Puterman, 1994].

Note that action-values $Q(s,a)$ are often used instead of, or in conjunction with, state values $V(s)$. Action values are given by

$$Q(s,a) \quad = \quad R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a') \tag{3.9}$$

and Q is related to V by

$$V(s) = \max_a Q(s,a) \tag{3.10}$$

Given a table of Q-values, the optimal action from any state s can simply be read off as the action with the highest $Q(s,a)$ value. Given a table of $V(s)$ instead, a one-step search over possible actions must be performed to find the one with the highest value, where action values are given by $R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s')$ (from substituting Equation 3.10 into Equation 3.9).

As well as being used by the planning community, MDPs are central to formulating reinforcement learning (RL) problems. In RL, the agent has to learn how to act without prior knowledge of the

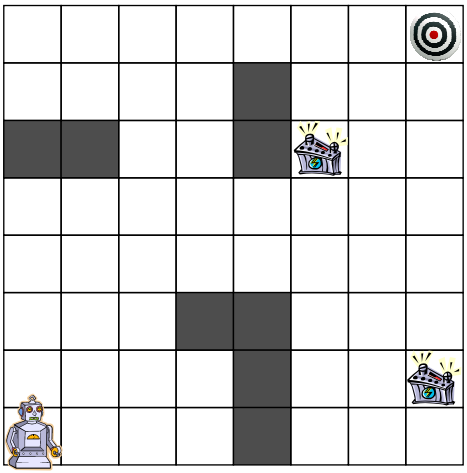


Figure 3.4: Example MDP: the robot's aim is to reach the target cell at the top-right, but it cannot cross black cells, and will collect additional reward for visiting cells with batteries. The MDP formulation will also cope with stochastic movement, for example, the robot has a 10% chance of going up and 10% chance of going down if the command is to go right.

environment model, i.e. the transition function $P(s'|s,a)$ is unknown. There are many RL algorithms that allow the optimal value function to be learnt from experience of acting (either in the world or in simulation), based either on Monte-Carlo statistical methods, or on propagating rewards backward in time. For more details on RL, see [Sutton and Barto, 1998]; for more on MDPs in general, see [Puterman, 1994].

3.2.2 POMDPs

POMDPs [Astrom, 1965; Kaelbling *et al.*, 1998; Smallwood and Sondik, 1973; Sondik, 1978] are used for domains where, in addition to the state transitions being stochastic, the agent cannot directly observe the state that it is in. For example, an indoor mobile robot travelling along a corridor may not be able to determine its location from instantaneous observations, as one corridor often looks like another. However, the agent can still make observations, and these are generated from the underlying state by a stochastic, many-to-one function that is known to the agent.

3.2.2.1 POMDP Structure

A POMDP is defined by the tuple $\langle \mathscr{S}, \mathscr{A}, T, R, \mathscr{Z}, Z \rangle$, where (as in an MDP) $\mathscr{S}$ is a set of possible states, $\mathscr{A}$ is a set of actions available to the agent, $T = P(s'|s,a)$ is a state transition function, $R(s,a)$ is a reward function, but we also introduce a set of possible observations $\mathscr{Z}$, and an observation function $Z = P(z|s,a)$. The observation function Z gives the probability of making an observation z, given that the agent took action a and ended up in state s. Again the aim for the agent is to maximise its expected future discounted reward, given by

$$R_t = \sum_{k=0}^{\infty} \gamma^k w_{t+k+1} \tag{3.11}$$

where R_t represents the total reward after time t, $w_t = R(s_t, a_t)$ is the reward on timestep t, and a discount factor γ has been used, $0 \leq \gamma < 1$. For a standard POMDP, it is assumed that the state, action,

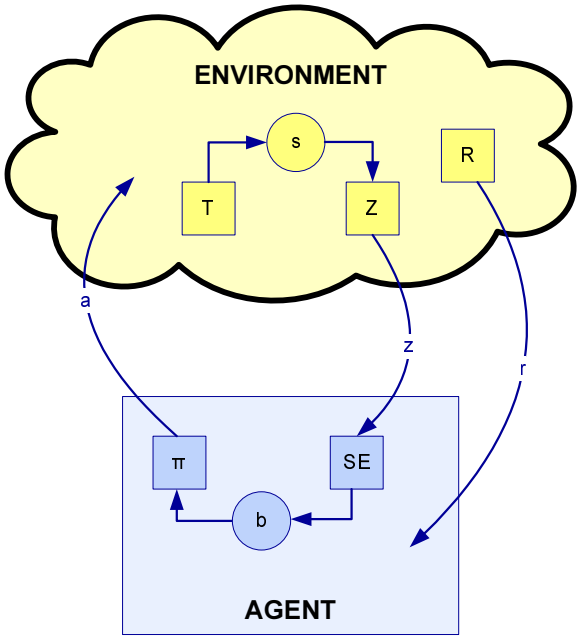


Figure 3.5: Information flow between the planning agent and the environment in a POMDP. Here *b* is a belief state maintained by the agent, giving a probability distribution over the agent's true state, and 'SE' is a state estimator that uses observations to update the belief state.

and observation spaces are discrete and finite. Figure 3.5 shows the flow of decisions between the agent and environment in the POMDP framework.

In the real world, observations made by robots are always noisy to some extent, so it is natural to model many robotics problems as POMDPs. The major drawback of POMDPs is computational tractability: solving them optimally is PSPACE-complete (a class of problem that is harder than NP-complete) [Littman, 1996], and therefore intractable for most real-world problems. Despite this, for domains where the uncertainty is discrete (such as uncertainty over which floor of an office building a robot is on), or where there are actions that have only information-gathering effects, defining the problem as a POMDP can be beneficial. An example POMDP problem is the tiger-food problem from [Kaelbling *et al.*, 1998], which incorporates an informational action. The problem facing the agent is to choose between two closed doors, left and right, one of which results in a moderate positive reward (food) and the other a large negative reward (a tiger). Instead of opening a door, the agent can also choose to listen, but this action has some cost (a small negative reward) and only reports the correct location of the tiger 85% of the time. After the agent chooses a door, the problem is re-started by closing the door and randomly assigning the tiger to one of the doors. If this problem was treated as an MDP, the optimal solution would never choose the *listen* action, as this action does not have any effect on the state of the system. However the optimal POMDP policy chooses the *listen* action over *left* or *right* unless the agent is very sure of the location of the tiger.

Another example POMDP problem is shown in Figure 3.6. This domain represents a partially-observable version of the one shown in Figure 3.4; the idea is to model limited-range sensors, so the robot can observe the cells adjacent to it perfectly, but can only see the contents of cells two steps away with 50% reliability. It cannot see the contents of cells further than two steps away at all, but does know its own location in the grid.

For these kinds of problems, it can easily be seen that the best action depends on the true state the agent is in; in the domain in Figure 3.6 for example, the best action may be to visit a battery, but not if that will lead to a part of the maze from which the direct path to the goal is blocked. Therefore estimating the true state is important; one option would be for the agent to store only the state it is

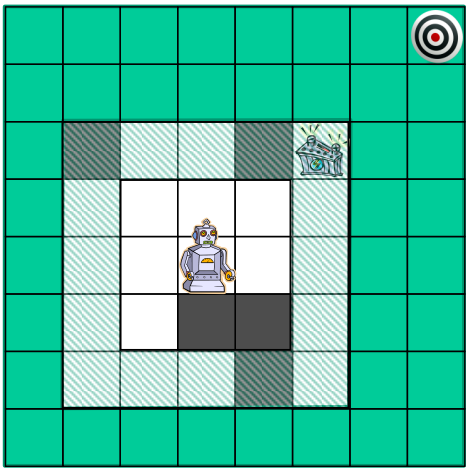


Figure 3.6: Example POMDP: the problem is similar to Figure 3.4, where the robot's aim is to reach the target cell at the top-right, but it cannot cross black cells, and will collect additional reward for visiting cells with batteries in. However, in this case the robot can only reliably observe the nearest eight cells (to determine if they are blocked, or if they contain a battery). The agent can also see the contents of the next-nearest 16 cells, shown shaded in light green, but with only 50% reliability for each observation. No information about further-away cells (in dark green) is available. In this example, the robot's location in the grid and the location of the goal are fully observable, but POMDPs allow for uncertainty about any aspect of the state space.

most likely to be in, but this can lead to bad decision-making. For example, when the goal-seeking robot of Figure 3.6 decides to enter a narrow corridor, the most probable belief could be that it will have a clear path to the goal, but the high cost of entering a dead-end corridor makes the expected reward for such an action low.

In a POMDP, for the agent to have the best estimate possible of the states it is likely to be in, it has to make use of the information from all its past actions and observations. It turns out that the best way to do this is to maintain a *belief state* b as a probability distribution over true states, such that $b(s)$ represents the probability of being in state s. When calculated correctly, the belief state is a sufficient statistic for the complete action/observation history [Kaelbling *et al.*, 1998; Smallwood and Sondik, 1973]; in other words, the belief state encapsulates all relevant information from previous actions and observations.

It can also be seen from the tiger-food example in particular that to act efficiently, an agent may have to take actions purely for the purpose of gathering information. The POMDP framework provides a way to optimally trade-off the informational benefits of actions against their cost, in such a way as to ensure expected long-term reward is maximised, without having to explicitly consider these issues.

3.2.2.2 Belief MDP

To maintain a belief state b, an agent only needs to know the belief state on the previous timestep, and the current action and observation. The belief state is updated using a *state estimator* function 'SE':

$$b' = SE(b,a,z) \tag{3.12}$$

Using Bayes' rule, the Markov property of state transitions ($P(s'|s,a,x) = P(s'|s,a)$ where x is any variable), and similarly the fact that observation probabilities depend only on s and a, we can derive

the state estimator of Equation 3.12 as follows:

$$\begin{aligned} b'(s') &= P(s'|b,a,z) \\ &= \frac{P(z|s',b,a)P(s'|b,a)}{P(z|b,a)} \\ &= \frac{P(z|s',a)\sum_{s\in\mathscr{S}} P(s'|s,b,a)P(s|b,a)}{P(z|b,a)} \\ &= \frac{P(z|s',a)\sum_{s\in\mathscr{S}} P(s'|s,a)b(s)}{P(z|b,a)} \end{aligned} \tag{3.13}$$

where $P(z|s',a)$ is the observation function Z, $P(s'|s,a)$ is the transition function T, and $P(z|b,a)$ can be treated as a normalising constant. Equation 3.12 implies that to keep track of its belief state b_t at time t, the agent must know its initial belief distribution b_0, as well as its action and observation history.

In a POMDP, the optimal action depends only on the belief state b [Astrom, 1965] (which is a direct consequence of the fact the belief state is a sufficient statistic for the agent's history). As the agent knows its belief state, we can transform any POMDP into a belief-MDP or b-MDP, which is an MDP where the states are the belief states from the POMDP. This converts a partially-observable problem with discrete states into a fully-observable problem with continuous states (as the states are probability distributions).

In the belief-MDP, the action space is the same as in the POMDP, $\mathscr{A}$. The state space is the continuous set of belief states, with dimensionality $|\mathscr{S}|$, and the transition function $\tau = P(b'|b,a)$ can be calculated using

$$\tau(b,a,b') = \sum_{z\in\mathscr{Z}} P(b'|b,a,z)P(z|b,a) \tag{3.14}$$

where $P(b'|b,a,z) = \begin{cases} 1 & \text{if } SE(b,a,z) = b' \\ 0 & \text{otherwise} \end{cases}$. The reward function ρ can be found similarly by taking an expectation over states:

$$\rho(b,a) = \sum_{s\in\mathscr{S}} b(s)R(s,a) \tag{3.15}$$

The POMDP can then be solved by finding the value function $V(b)$ for the b-MDP. However, the state space is continuous, so simple dynamic programming will not suffice.

3.2.2.3 Basic POMDP Solution Method

In this section I describe the basic method for solving a POMDP first described by [Smallwood and Sondik, 1973], and summarised in [Kaelbling *et al.*, 1998].

We consider discounted finite-horizon POMDPs, where the agent has a limited number of timesteps available to it. Therefore, the optimal policy will be non-stationary – the best action will probably be different if there is just one step left to when there are many steps left and exploration may be worthwhile. For this case, we can use a policy tree to represent a complete t-step policy for a POMDP, as shown in Figure 3.7. The tree contains an action to take as the root node for when there are t timesteps remaining, and then the action to take for each of the possible subsequent observations, and so on until the final timestep. This is a complete specification of a policy as each action and observation will deterministically specify the resulting belief state (but when acting according to a policy tree, the belief state need not be explicitly calculated). Note that for different starting belief states, a different policy tree will be optimal.

For a fixed policy tree p and a given state s, the POMDP value function is defined by the immediate reward from the first action plus the discounted expected future reward. The immediate reward from

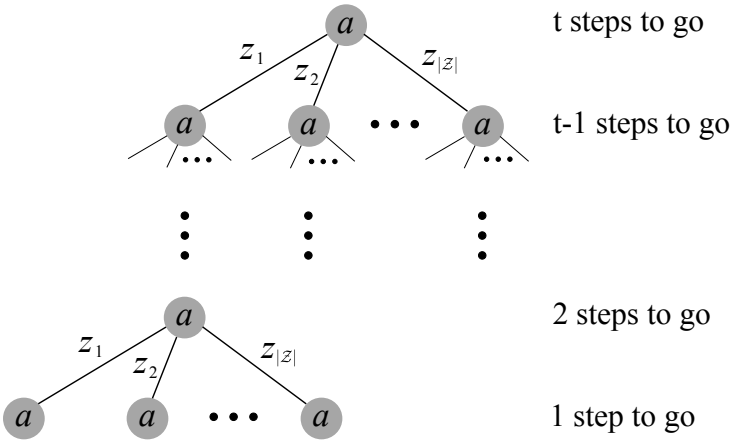


Figure 3.7: A POMDP policy tree. (Figure duplicated from [Kaelbling *et al.*, 1998].)

executing the action $a(p)$ at the root node of p is $R(s,a(p))$. The expected value of the future can be obtained by summing over states, so that

$$\text{value of future} = \sum_{s' \in \mathscr{S}} P(s'|s,a(p)) \cdot (\text{value of subtree with } a(p) \text{ removed, given state } s')$$

If we denote the $(t-1)$-step subtree after observing z_i at the root of p as $z_i(p)$, the overall value of tree p given state s is:

$$V_p(s) = R(s,a(p)) + \gamma \sum_{s' \in \mathscr{S}} T(s,a(p),s') \sum_{z_i \in \mathscr{Z}} Z(s',a(p),z_i) V_{z_i(p)}(s') \tag{3.16}$$

To express the value of a tree in terms of belief states, we can define an α-*vector*, $\alpha_p = \langle V_p(s_1), \ldots, V_p(s_n) \rangle$, so that the value of a belief state is given by

$$V_p(b) = b \cdot \alpha_p \tag{3.17}$$

which is a linear function of the belief state. The optimal value function for a t-step POMDP, $V_t(b)$, is given by

$$V_t(b) = \max_{p \in \mathscr{P}} b \cdot \alpha_p \tag{3.18}$$

where $\mathscr{P}$ is the set of all t-step policy trees. Figure 3.8 shows the optimal value function for a two-state domain (where $b(s_1)$ defines the belief state as $\sum_{s \in \mathscr{S}} b(s) = 1$). This optimal value function is the upper surface of the value functions associated with individual policy trees, and it is therefore piecewise-linear and convex. By projecting the value function onto the belief space, we can find which policy tree should be followed for each region of the belief space.

In practise, we find that most policy trees are dominated by others, as shown in Figure 3.9. This means we can represent the optimal value function V_t by a parsimonious set of policy trees $\mathscr{V}_t$. To work out if a given policy tree p is dominated, we can solve a linear program to find a belief point where the vector α corresponding to p dominates all other vectors, or prove there are no such points (in which case p can be discarded).

This leads to an exhaustive enumeration algorithm for finding the optimal t-step policy for a POMDP:

1. Start with $\mathscr{V}_{t-1}$, the parsimonious set of $(t-1)$-step policy trees.
2. Build $\mathscr{V}_t^+$ with all possible combinations of root action, observation, and subtree from $\mathscr{V}_{t-1}$.

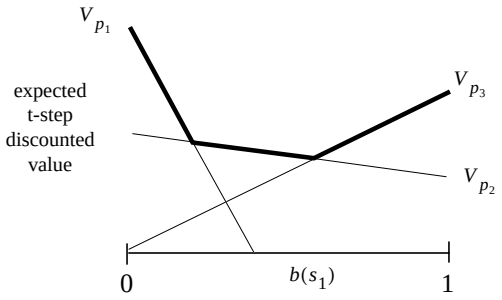


Figure 3.8: POMDP value functions for a domain with just two states. Note the optimal value function (indicted by the bold line) is the maximum of the value functions defined by all the possible policy trees, and it is therefore piecewise-linear and convex. (Figure taken from [Kaelbling *et al.*, 1998].)

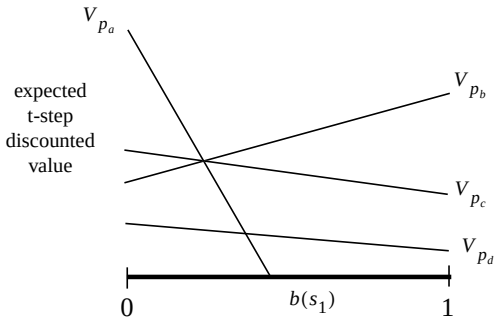


Figure 3.9: Value function example showing how some policy trees can be completely dominated by others. Here, the value function for tree p_d is completely dominated by the one for p_c. (Figure taken from [Kaelbling *et al.*, 1998].)

3. Prune policy trees that are not dominant from $\mathcal{V}_t^+$ to get $\mathcal{V}_t$.

This algorithm can be extended to the discounted infinite-horizon case by finding policies for increasingly large t until the top few levels of the optimal policy tree stop changing significantly as further timesteps are added. At that point the policy approximates the optimal infinite-horizon policy.

This algorithm is exponential in the number of observations, as the size of $\mathcal{V}_t^+$ is $|A||\mathcal{V}_{t-1}|^{|\mathcal{Z}|}$. It is also potentially exponential in the horizon t, as there is the potential for $\mathcal{V}_t$ to grow exponentially on each timestep. This makes it very computationally costly, and in practise it is only tractable for problems with the order of 10^2 states. An extension proposed in [Cassandra *et al.*, 1994; Kaelbling *et al.*, 1998] is the *witness* algorithm, which takes advantage of the important result that the value function for POMDPs is always piecewise-linear and convex [Smallwood and Sondik, 1973], meaning higher-value states are located near the 'edges' of belief-state space. The algorithm attempts to generate the set $\mathcal{V}_t$ directly from $\mathcal{V}_{t-1}$, avoiding the complex step of generating $\mathcal{V}_t^+$. It does this by finding 'witness' points in the belief space that indicate an additional policy tree needs to be added to the $\mathcal{V}_t$ set. Incremental pruning [Zhang and Liu, 1996; Cassandra *et al.*, 1997] improves further on witness in terms of computational efficiency; it achieves this by interleaving the generation of elements of $\mathcal{V}_t$ with pruning of dominated policy trees. This is made possible by splitting the value function computation (Equation 3.16) into three independent components, the overall value, the value of a given action, and the value of a given action and observation. Each of these value function components can be represented by a set of vectors in the same way as α-vectors are used in witness, and most of these vectors are not needed as they are dominated by other vectors. This splitting of the value function allows the set of vectors for the action-specific value function to be built up incrementally, by adding one observation at a time and then purging dominated vectors, leading to a significant performance improvement over witness.

As well as the exact solution methods outlined above, there are many approximate methods for solving POMDPs, some of which are described briefly in Section 6.1. These methods allow solutions to be found for problems with thousands of states or more, although the state space size of the vent prospecting problem is still outside their range of operation. Despite the practical problems with solving POMDPs, it is very useful to have defined the problem as a POMDP for several reasons:

- A POMDP is the most natural and mathematically accurate way of formalising the vent prospecting problem.
- The POMDP formulation and a knowledge of basic POMDP solution methods helps in understanding how hard the problem really is.
- The approximations I introduce later on in this thesis in order to make the problem tractable are more transparent when the problem is specified as a POMDP.
- Some POMDP methods actually can be applied to the problem; Chapter 6 shows how a form of online POMDP solver can be used to produce effective plans in this domain. An understanding of early POMDP methods such as witness is very useful background to the more advanced methods discussed in Chapter 6.

Chapter 4

Problem Model

This chapter presents my formal model of the problem of searching for hydrothermal vents using an AUV. The model is the foundation of work described later in this thesis, and represents a significant contribution itself. The chapter starts with an overview of the model in Section 4.1, followed by a description of the plume dispersion model (Section 4.2). Next the components of the POMDP model are described in Section 4.3: the state space, actions, transition function, observations, observation function and rewards. Section 4.5 shows how this POMDP can be converted into a belief-state MDP (also known as an information-state MDP). The derivation of the observation functions is given separately in Section 4.6 as it is fairly lengthy, and Section 4.7 discusses some of the decisions made in creating the model presented in this chapter. Section 4.8 notes that even if the observation function were to be removed from the model it would still have a large state space, and finally in Section 4.9, the model is compared to other similar POMDP models in the literature.

4.1 Overview of Model

From an AI planning point of view, the vent prospecting problem described in Section 1.2 has the following characteristics:

- Noisy sensors
- Partial observability
- Continuous state and observation spaces
- Continuous action space and uncertain action outcomes
- Resource limits

As my work is based on Jakuba's OG framework, it is natural to define a discrete finite search area (mirroring the OG) and discrete observation space (as the only observations of interest are detecting a plume or finding a vent). To allow the agent location as well as vent locations to be represented in grid form, I assume that the agent can only make discrete movements between adjacent grid cells, and these actions are deterministic. This perfect localisation and navigation is a reasonable approximation for high-end AUVs such as Autosub (operated by the National Oceanography Centre, Southampton), which was the nominal target platform for this work, but may not be a good assumption for mass produced AUVs. Finally, a fixed mission duration is used as a proxy for resource limits.

I assume that at the start of the mission, the agent has no information about where vents are most likely to be found within the search area, so the OG map is initialised with the same prior probability of occupancy for all cells. Given this, the optimum pre-mission plan would be some form

of exhaustive search, whereas the objective of this work is for the AUV to use on-board processing to analyse the data it collects and plan a more efficient path. This means the problem is defined as an *online planning problem*, where the agent re-plans periodically during a mission to take advantage of new observations. In fact, the agent re-plans at every timestep, as some observations will cause the OG to change significantly and the agent should respond to this as quickly as possible. If the sensors make no useful detections, the OG will not change much, but the agent still re-plans as it has the same amount of planning time available (the drawback to this potential waste of CPU cycles is that it will drain the vehicle's battery faster). The quality of some of the assumptions and approximations made above is discussed in Section 4.7, as well as in Chapter 8.

Given these assumptions, the planning problem takes an OG map showing the current probability of each cell containing a vent as input, and has to calculate the path for the agent that maximises the expected number of vents found over the remainder of the mission. The problem is still hard due to the probabilistic nature of the map (resulting from the partial observability and indirect sensors), large state space, and need to optimise for visiting initially unknown targets instead of simply optimising for map quality. Indeed, [Littman, 1996] notes that given partial observability, problems with a deterministic transition function are in the same complexity class as problems with a stochastic transition function.

In summary, the model presented below has:

- Partial observability
- A very large state space (rather than continuous state, observation and action spaces and uncertain action outcomes)
- A fixed, finite horizon (in lieu of resource limits)

4.2 Environment Model

This section describes the simulation of the environment used for all development and experimental work in this thesis, which is based heavily on [Jakuba, 2007], §4.5 and §5.2.1, and on MATLAB code developed by Michael Jakuba.

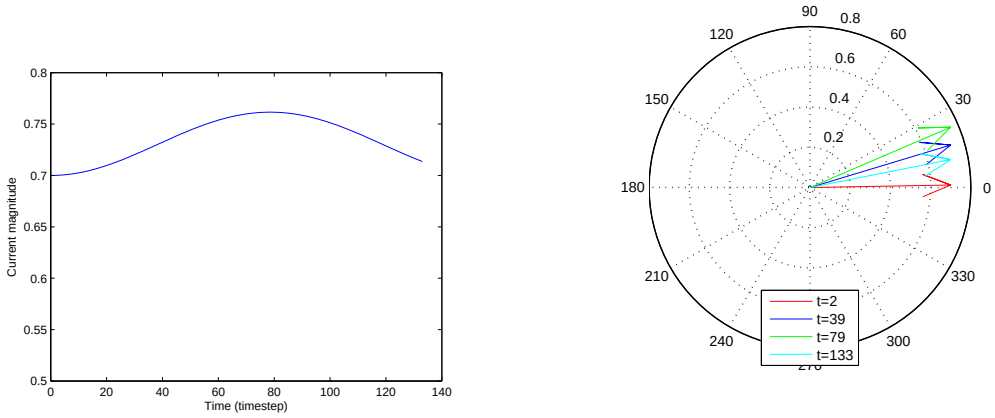
The environment is two-dimensional with all movement in the vertical plane ignored. The simulation does not map directly to a real-world scale, but instead simply works in abstract distance/time units. It encompasses a finite area of side length $\frac{v^2}{v-1}$ units, which is divided up into a $v \times v$ grid of C cells (where $C = v^2$), corresponding to the C cells of the OG. The cells are numbered sequentially to simplify referring to them. Each cell occupies a set of physical coordinates, but vents are only ever placed in the centre of a cell, with a maximum of one vent per cell.

The simulation runs in discrete time, and each simulated vent emits one plume particle on every timestep. All pre-existing plume particles are displaced by the vector for the ocean current plus independent individually generated Gaussian noise on each timestep, and are not restricted to cell-centre coordinates. The current varies on each timestep according to a sine function, and is given by

$$\mathbf{U}(t) = \begin{pmatrix} 0.7 \\ 0.3 \sin 0.02t \end{pmatrix}$$

This defines a periodic current with a magnitude of between 0.7 and 0.76 units/timestep, which varies from heading east at $t = 0$ to east-north-east (at $t = 79$), back through east to east-south-east, and finally back to east at $t = 314$. Figure 4.1 shows how the magnitude and direction of the current varies over time. This current is adopted directly from Jakuba's MATLAB implementation, and should represent a possible real-world current.

The vehicle is assumed to detect a plume if it is within half a unit of a plume particle.



(a) Magnitude

(b) Direction

Figure 4.1: Plots of the periodic current used in the environment simulation. The current is shown between time $t = 1$ and $t = 133$ because most experiments were performed for 133 timesteps.

Note that Jakuba created this simulation to model the buoyant hydrothermal plume, which means the plume as it is still rising and relatively close to the vent, before it attains neutral buoyancy. The model also predicts the location of vent fields rather than individual vents, under the assumption that vents within a vent field will be closely packed together and will fit within a single grid cell.

4.3 POMDP Formulation

This section presents the formal model, which is a POMDP with components $\langle \mathscr{S}, \mathscr{A}, T, R, \mathscr{Z}, Z \rangle$.

4.3.1 State Space

The state $s \in \mathscr{S}$ is composed of

- **m**, the actual vent locations (a length-C vector of booleans indicating whether each cell contains a vent).
- **v**, a similar vector marking all vents that have already been found.
- c_{AUV}, the cell the AUV is in.
- c_{prev}, the cell the AUV was in at the previous timestep.
- $\underline{\mathbf{U}}$, the complete history of the ocean current vector over the agent's lifetime.

Note that all of these with the exception of $\underline{\mathbf{U}}$ are actually discrete variables. The size of the state space is discussed in Section 4.4, where it is noted that in some ways **v** is a convenience variable.

4.3.2 Actions and Transition Function

The actions $a \in \mathscr{A}$ are to move north/east/south/west by one grid cell (meaning the vehicle moves at a speed of 1 unit/timestep), and actions that would leave the search area are disallowed. Actions that would return the vehicle to the cell it was in on the previous timestep are also disallowed, both because we can intuitively see that such actions will not be optimal, and because this provides greater freedom

when setting the grid size for a real-world implementation, as it places less strict requirements on the turning circle of the vehicle.

The transition function $T = P(s'|s,a)$ is deterministic in my model. The vent locations $\mathbf{m}$ do not change, and $\mathbf{v}$, c_{AUV}, and c_{prev} can be calculated trivially assuming actions always succeed. The current will generally change on each timestep, but this change will be tiny and is ignored in the model (mid-ocean ridge currents consist of components that cycle with half-day, one-day and multi-day periods [Berdeal *et al.*, 2006]). Therefore the current at time $t+1$ is assumed to be the same as the current at time t (denoted by $\mathbf{U}_t$), and this is appended to the current-history at time t ($\underline{\mathbf{U}}_{1:t}$) to give the new the current-history:

$$\underline{\mathbf{U}}_{1:t+1} = [\underline{\mathbf{U}}_{1:t}\, \mathbf{U}_t]$$

Note that given c_{AUV} (which is part of the state s), an action a uniquely identifies the cell that the agent will move into, so in the remainder of this thesis a will be used to indicate both an action and the resulting cell.

4.3.3 Reward Function

The agent gains a fixed reward for visiting a *previously unvisited* vent. The reward function for taking an action moving into cell a is given by:

$$R(s,a) = \begin{cases} R_{vent} & \text{if } \mathbf{m}(a) = true \text{ and } \mathbf{v}(a) = false \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where R_{vent} is a positive constant, and the $\mathbf{v}(a) = false$ guard specifies that the vent has not been found previously.

This reward function is a precise reflection of the goal of the vent prospecting problem: to find as many distinct vents as possible. The function does not include any reward for exploring the grid because that is not defined as the goal of the problem; the agent will be able to find vents more effectively if it performs some exploration, but it is the task of the solution algorithm to reason about how best to explore in order to achieve the true goal of finding vents.

4.3.4 Observations and Observation Function

Although the input to the OG algorithm is a single observation, whether or not a hydrothermal plume is detected, our agent also needs to know when it has found a vent. The agent has to know this in order to accommodate the reward model, so it knows when it gets a reward. We therefore introduce a deterministic vent detector, so the possible observations $\mathscr{Z}$ that the agent can receive when moving into cell c are:

$$z = \begin{cases} l & \text{if a vent is located in cell } c \\ p & \text{if a plume is detected in cell } c \\ n & \text{if nothing is detected} \end{cases}$$

The agent is also able to observe its location c_{AUV}, and the current ocean current $\mathbf{U}$.

The observation function $P(z|s,a)$ is derived using the forward model of a plume in [Jakuba, 2007], and this function together with a short explanation of each equation is presented in this section. As the full derivation is quite lengthy, it is deferred to Section 4.6.

The plume model enables us to find $P(d_c|m_c)$, which is the probability of detecting a plume particle from a vent in cell c, given that c contains a vent, and $P(d_c|m_c)$ is also specified in full in Section 4.6. If we denote the probability of a false positive plume detection by P^F, and the number of vents in the search area, $|\{c : \mathbf{m}(c) = true\}|$, by M then the observation function is given by

$$P(z=l|s,a) \quad = \quad \begin{cases} 1 & \text{if } \mathbf{m}(a) = true \\ 0 & \text{otherwise} \end{cases}$$

i.e. if the agent enters an occupied cell, a vent will always be located;

$$P(z=p|s,a) \quad = \quad \begin{cases} 0 & \text{if } \mathbf{m}(a) = true \\ P^F & \text{if } M = 0 \\ 1 - \left[\left(1-P^F\right)\prod_{c:\mathbf{m}(c)=true}\left(1-P(d_c|m_c)\right)\right] & \text{otherwise} \end{cases}$$

i.e. if the agent enters an unoccupied cell and there is at least one vent in the search area, then the probability of detecting a plume is one minus the probability of not detecting a plume (see below);

$$P(z=n|s,a) \quad = \quad \begin{cases} 0 & \text{if } \mathbf{m}(a) = true \\ 1-P^F & \text{if } M = 0 \\ \left(1-P^F\right)\prod_{c:\mathbf{m}(c)=true}\left(1-P(d_c|m_c)\right) & \text{otherwise} \end{cases}$$

i.e. if the agent enters an unoccupied cell and there is at least one vent in the search area, then the probability of not detecting a plume is the probability of not getting a false positive detection *and* not getting a detection from any of the vents in the map.

4.3.5 Mission Length

The duration of missions is fixed, i.e. each mission will have the same fixed number of timesteps, L. For planning purposes, rewards are discounted by a factor γ, $0 \leq \gamma < 1$, on each successive timestep. This models the small probability that the mission may have to be aborted at any time due to a system failure, and encourages planners to collect reward early, providing some robustness to uncertainty over exactly how long the vehicle's batteries will last.

4.4 POMDP State Space Size

Section 4.3 presented the state space, comprising the variables $\mathbf{m}$, $\mathbf{v}$, c_{AUV}, c_{prev}, and $\underline{\mathbf{U}}$. However, for implementational purposes, the reward function in the model makes the vector $\mathbf{v}$ (storing the location of vents that have been visited) unnecessary. Instead the binary vector $\mathbf{m}$ can be used, by removing vents from $\mathbf{m}$ once the agent has visited them. Removing a visited vent from the map will have no effect on the planning algorithms, because in terms of the expected future reward, a cell with a visited vent has exactly the value as a cell that does not contain a vent. Formally, this means changing the transition function so that if an action moves the agent into a cell c containing a vent, the agent gains a reward of R_{vent}, and $\mathbf{m}$ is then updated so that $\mathbf{m}(c) = false$. Of course a real AUV would have to store the locations where it finds vents, but this does not impact on the model from a planning point of view.

The variables $\mathbf{m}$ and $\mathbf{v}$ are nevertheless both included in the state space definition because it is useful to be able to refer to $\mathbf{m}$ as the unchanging true vent map, and having both $\mathbf{m}$ and $\mathbf{v}$ does not affect any of the novel planning algorithms discussed in this thesis. Further, the observation function depends on the true vent map $\mathbf{m}$, so $\mathbf{m}$ must be considered as part of the environment model even if it is not part of the state space implementation.

The size of the state space can then be calculated by firstly noting that given c_{AUV}, the cell the agent is in now, there are only four possible previous cells. Next we assume that the current $\mathbf{U}$, which

must be stored for every timestep, is discretised into $u \times u$ bins. Finally we discount the variable $\mathbf{v}$ and assume the lifetime of the agent is L timesteps, which gives $2^C \cdot C \cdot 4 \cdot L \cdot u^2$ possible states.

4.4.1 Size of State Space in Experiments

With the parameters used for experimental work in this thesis, the problem has approximately 10^{123} possible states, with Table 4.1 showing the breakdown by state variable. This is using a 20×20 grid with $C = 400$ cells, and a fixed current history (as all experiments used the same current history this can be excluded from discussions of the state space size, as states with a different current history will never be visited).

Note that most work on POMDPs has used problems with a far smaller state space, with 10^7 being an unusually large problem for even approximate POMDP solvers to be able to handle [Poupart and Boutilier, 2004; Pineau *et al.*, 2006].

Table 4.1: The number of states for each of the variables comprising the state space of the agent (as described in Section 4.3). These figures are for the experimental setup used in this thesis, based on a 20×20 grid, and a fixed current history (meaning that L and u are not relevant).

Variable	*Possible states*
Unvisited vents, length-C boolean (**m/v**)	$2^{400} \sim 2.6 \times 10^{120}$
c_{AUV}	400
c_{prev}	4
$\underline{\mathbf{U}}$, current history	1
TOTAL	4×10^{123}

4.4.2 Considerations for Real-world State Space

In this section I discuss ideal settings for algorithm parameters for use in a real-world AUV mission, and the size of the resulting state space. The most important parameters are the grid cell size and number of cells. Following [German *et al.*, 2008; Jakuba, 2007; Jakuba and Yoerger, 2008], and discussions with Bramley Murton of the National Oceanography Centre, Southampton (personal communication, 5 July 2007), a resolution of 5 m by 5 m cells and a search area of 1 km by 1 km seem reasonable for an AUV mission where contact with the plume has already been made using ship-based observations. This gives a 200×200 grid with 40000 cells. Note that Jakuba's forward model is based on vent fields, but in this thesis a vent field is considered to be a point source, and is referred to as a 'vent'.

A typical speed for an AUV is 1 m/s, so if each unit distance in the grid represents 5 m, each timestep should represents 5 s. The mission duration is assumed to be 24 hours (although recent AUVs such as Autosub6000 can undertake missions of 2-3 days even with a full payload of scientific sensors [McPhail and Stevenson, 2009]), equating to 17280 timesteps. However the history of the ocean current will not need to be stored over the full lifespan of the mission, or at the temporal resolution of one timestep. Typical ocean currents are about 0.05 m/s over a mid-ocean ridge [Berdeal *et al.*, 2006], and as we only care about particles originating from 1 km or less distant, we only need keep the history for 5.5 hours (3960 timesteps). As the current is composed of several elements, each of which is periodic with a frequency of at least 12 hours, the variation in current is expected to be slow, such that sampling every minute instead of every 5 s will be adequate. If the northerly and easterly components of the current are discretised into 100 bins each, 5.5 hours equate to 330 samples each with 10000 possible values.

Table 4.2 shows the breakdown of state-space-size by variable, and the total number of possible states is 10^{12053}. This strongly suggests that a coarser tiling of the state space than the ideal one described here will be necessary for real missions, which will have some impact on the effectiveness of the algorithms discussed in the remainder of this thesis.

Table 4.2: The number of states for each of the variables comprising the state space of the agent (as described in Section 4.3). These figures are for a potential real-world mission, based on a 200×200 grid.

Variable	*Possible states*
Unvisited vents, length-C boolean (**m**/**v**)	$2^{40000} \sim 10^{12041}$
c_{AUV}	40000
c_{prev}	4
$\underline{\mathbf{U}}$, current history	$330 \times 100^2 = 3.3 \times 10^6$
TOTAL	10^{12053}

4.5 Belief-MDP Formulation

To find a policy in the POMDP defined above we use a standard approach for POMDP solving and translate it to a *belief-state MDP* (b-MDP), where the state space is the probability distribution over the possible states of the POMDP, known as the *belief state*. While the belief state is fully observable by the agent, it is continuous and therefore standard MDP techniques cannot be applied. The components $\langle \mathscr{B}, \mathscr{A}, T, R \rangle$ of the b-MDP are described below.

4.5.1 Belief State Space

In our b-MDP, the belief state $b \in \mathscr{B}$ is comprised of:

- c_{AUV}, the cell the AUV is in; c_{prev}, the AUV's cell at the previous timestep; $\underline{\mathbf{U}}$, the current history; and **v**, the list of previously-found vents (all of which are assumed to be fully observable).
- The occupancy grid **O**. **O** is a length-C vector of values $P(m_c)$, the probability of cell c containing a vent (which will be either zero or one for visited cells). The OG defines a distribution over true vent locations **m**.

Note that *using an OG approach has considerably reduced our belief state space* (the map contributes only C variables to the belief state instead of 2^C), because the occupancy grid is an approximation where we assume the probability of each cell being occupied is independent.

4.5.2 Actions

The actions in the b-MDP are identical to those in the POMDP, deterministically moving N/E/S/W.

4.5.3 Transition Function

Given an initial belief state b, action a and observation z, the resulting belief state can be calculated using Jakuba's OG algorithm in combination with some simple updates. We write

$$b' = \mathrm{SE}(b, a, z)$$

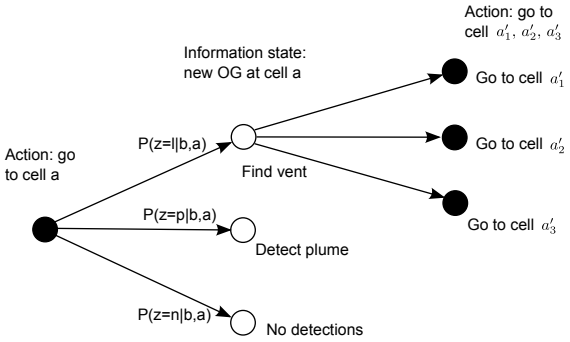


Figure 4.2: Belief-State MDP, where black circles represent actions and white circles are belief states. $P(z|b,a)$ is the probability of observing $z \in \{l,p,n\}$ when moving into cell a, where l represents locating a vent, p detecting a plume, and n observing nothing.

where "SE" stands for *state estimator*, a function that includes the application of Jakuba's OG algorithm to find the new occupancy grid $\mathbf{O}$. As in the POMDP case, the other variables, c_{AUV}, c_{prev}, $\underline{\mathbf{U}}$, and $\mathbf{v}$ are trivial to update; note that the observation z is assumed to include the instantaneous current and whether or not there is a vent at the agent's new location, so $\underline{\mathbf{U}}$ and $\mathbf{v}$ can be updated exactly.

The transition function T is given by

$$P(b'|b,a) = \sum_{z \in \mathcal{Z}} P(b'|b,a,z)P(z|b,a)$$

where $P(b'|b,a,z)$ is zero for all belief states except the one where $b' \equiv \mathrm{SE}(b,a,z)$, for which $P(b'|b,a,z)$ is equal to one. The observation model $P(z|b,a)$ is the probability of observing $z \in \{l,p,n\}$ in cell a (note that the action specifies the cell that is moved into). This observation model is given in Section 4.5.4 below.

Figure 4.2 shows the action/observation transitions for the b-MDP.

4.5.4 Observation Function

The observation model $P(z|b,a)$ is again derived from the plume model in [Jakuba, 2007]. Note that an MDP does not have an 'observation function', and this function is actually part of the transition model given in Section 4.5.3. However it makes sense to consider it separately, as the derivation is closely related to that of the POMDP observation model. This derivation is given in Section 4.6 with the model simply presented here.

Given we assume the vent detector is deterministic, then

$$P(z = l|b,a) = P(m_a)$$

The probability of seeing a plume but not a vent is given by:

$$P(z = p|b,a) = (1 - P(m_a)) \left(1 - \left(1 - P^F\right) \prod_{c \neq a} (1 - P(d_c|m_c)P(m_c)) \right)$$

where P^F is the probability of a false positive plume detection, the product is over all cells c apart from the cell the agent is moving into, and $P(d_c|m_c)$ is the probability of detecting a plume from a source at c, given that c contains a vent. Finally, the probability of seeing nothing is:

$$P(z = n|b,a) = (1 - P(m_a)) \left(1 - P^F\right) \prod_{c \neq a} (1 - P(d_c|m_c)P(m_c))$$

4.5.5 Reward Function

The b-MDP reward function is fully specified by the POMDP reward function (Equation 4.1). For an action a that moves the agent into cell a, the b-MDP reward is given by Equation 3.15 which sums over all possible POMDP states. However, for the vent-prospecting model, Equation 4.1 produces a reward $R(s,a) = 0$ for all states except ones where the destination cell a contains an unvisited vent, in which case the reward is R_{vent}. The summation from Equation 3.15 can therefore be replaced a summation over states where a is both occupied (denoted by m_a) and has not been found previously (denoted by $\neg v_a$):

$$\begin{aligned} \rho(b,a) &= \sum_{s:(m_a \wedge \neg v_a)} b(s) R_{vent} \\ &= \begin{cases} 0 & \text{if } \mathbf{v}(a) = true \\ \sum_{s:m_a} b(s) R_{vent} & \text{otherwise} \end{cases} \end{aligned} \tag{4.2}$$

where the second step follows because if $\mathbf{v}(a) = false$, the probability $b(s)$ for states with $\mathbf{v}(a) = true$ will be zero in the summation for the 'otherwise' condition (and $\mathbf{v}$ is known exactly in the belief state).

Next we use the result $P(X) = \sum_Y P(X,Y) = \sum_Y P(Y)P(X|Y)$ from basic probability theory to find an expression for the probability that cell a contains a vent:

$$\begin{aligned} P(m_a) &= \sum_{s \in \mathscr{S}} P(s) P(m_a|s) \\ &= \sum_{s:m_a} P(s) \\ &= \sum_{s:m_a} b(s) \end{aligned} \tag{4.3}$$

where the second line uses the fact that $P(m_a|s) = 0$ for all states except ones where cell a is occupied (for which $P(m_a|s) = 1$), and the third line replaces $P(s)$ with $b(s)$ which is simply the posterior probability of state s.

Substituting Equation 4.3 into Equation 4.2 gives the full b-MDP reward function:

$$\rho(b,a) = \begin{cases} 0 & \text{if } \mathbf{v}(a) = true \\ P(m_a) R_{vent} & \text{otherwise} \end{cases} \tag{4.4}$$

4.6 Derivation of Observation Function

The observation function is derived in three steps:

- The plume model is used to find the probability distribution for particle locations (Section 4.6.1).
- This distribution is used to calculate the probability of detecting a particle from a single vent in a known location (Section 4.6.2).
- The probability for each of the observations (locate a vent, detect a plume, or detect nothing) is calculated, making use of the map and the single-vent detection probability from Section 4.6.2.

The last step is carried out for the POMDP observation model in Section 4.6.3, and for the belief-MDP observation model in Section 4.6.4.

4.6.1 Plume Model

To find the probability of detecting a plume due to a vent in a known location, we need to use the forward model for a plume (as described in Section 4.2) and the history of the current, $\underline{\mathbf{U}}$. The plume model assumes particles emitted from source location $\mathbf{x}_s$ at time $t-\tau$ are dispersed by the current plus isotropic Gaussian noise, so their 2D location $\mathbf{x}_\omega$ at time t is distributed according to

$$\mathbf{x}_\omega \sim N(\mathbf{x}_s + \sum_{r=t-\tau}^{t-1} \mathbf{U}_r, \tau\sigma^2) \tag{4.5}$$

where σ^2 is the variance of the plume dispersion per timestep. If the current was fixed rather than variable, $\sum_{r=t-\tau}^{t-1} \mathbf{U}_r$ could be replaced by $\tau\mathbf{U}$.

4.6.2 Plume Detection Due to a Single Vent

Given a vehicle location $\mathbf{x}_{AUV}$ and source vent location $\mathbf{x}_s$, we want to find the probability of observing a given particle ω^τ which was emitted at $t-\tau$. Following Jakuba §5.2.1, we assume a particle has a radius b, and will always be detected if $|\mathbf{x}_{\omega^\tau} - \mathbf{x}_{AUV}| < b$. Then we use the approximation ([Jakuba, 2007])

$$\begin{aligned} P(\text{detect}(\omega^\tau)) &= P\left(|\mathbf{x}_{\omega^\tau} - \mathbf{x}_{AUV}| < b\right) \\ &\approx b^2 \frac{1}{\sqrt{2\pi\tau\sigma^2}} \exp\left(-\frac{\left|\mathbf{x}_s + \left(\sum_{r=t-\tau}^{t-1} \mathbf{U}_r\right) - \mathbf{x}_{AUV}\right|^2}{2\tau\sigma^2}\right) \end{aligned} \tag{4.6}$$

Next we need to consider all the particles emitted from a given source, i.e. we want

$$P(d_c|m_c) = P_c^d = P\left(\text{detect}(\omega_c^1) \vee \text{detect}(\omega_c^2) \vee \ldots \vee \text{detect}(\omega_c^n)\right)$$

where d_c represents a plume detection due to a source at c, the source $\mathbf{x}_s$ is assumed to lie in the cell m_c, and ω_c^τ represents the particle emitted from the source at c at time $t-\tau$. Assuming that the detection of each particle is independent, and using $P(d_c|m_c) = 1 - P(\neg d_c|m_c)$, we get

$$P(d_c|m_c) = 1 - \prod_n (1 - P(\text{detect}(\omega_c^n))$$

i.e. the probability of a detection due to a source at c is one minus the probability of *not* detecting any of the particles emitted from the source. Note that $P(d_c|m_c)$ is identical to the quantity P_c^t in [Jakuba, 2007], and the equation above is essentially the same as equation (5.9) in Jakuba §5.2.1, although he derives this equation differently.

In the simple environmental simulation we use (which is based on Jakuba's implementation), each source emits one particle at each timestep, and we further assume that only the particle with the nearest expected location contributes to the probability of a detection, i.e. $\sigma \ll |\mathbf{U}|$ (the variance in particle location is smaller than the magnitude of the current). In this case

$$P(d_c|m_c) = P(\text{detect}(\omega_c^j)) \tag{4.7}$$

where

$$j = \underset{\tau}{\text{argmin}} \left| \mathbf{x}_s + \left(\sum_{r=t-\tau}^{t-1} \mathbf{U}_r \right) - \mathbf{x}_v \right| \tag{4.8}$$

and $P(\text{detect}(\omega_c^n))$ is given by Equation 4.6.

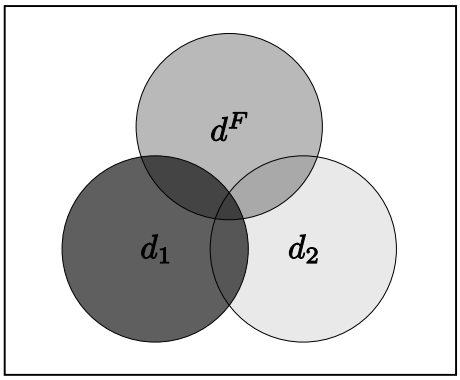


Figure 4.3: Venn diagram showing that a plume detection may be due to a false positive detection (d^F), due to a vent at cell 1 (d_1), due to a vent at cell 2 (d_2), or any combination of these. Note this diagram is only an example as it represents maps that have exactly two vents.

4.6.3 POMDP Observation Model

For the POMDP model, we need to find the observation probabilities in terms of the true state space s.

4.6.3.1 Locating a Vent (z=l)

If the AUV is in the same cell as a vent then it is overwhelmingly likely to detect a plume from that vent, so we ignore plume detections when we find a vent. In the POMDP case, the location of vents is given by the vector $\mathbf{m}$, which is part of the state space, so

$$P(z=l|s,a) = \begin{cases} 1 & \text{if } \mathbf{m}(a) = true \\ 0 & \text{otherwise} \end{cases}$$

4.6.3.2 No Detection (z=n)

First we introduce the possibility of a false positive plume detection, d^F, into our model. This has probability $P(d^F) = P^F$. Figure 4.3 shows that a plume detection d can be due to a false alarm, or due to detecting a particle from any of the vents, or due to a combination of these. This gives

$$P(d) = P\left(d^F \vee d_1 \vee d_2 \vee \ldots \vee d_M\right)$$

where M is the number of vents in the map, $M = |\{c : \mathbf{m}(c) = true\}|$. The probability of no plume detection is therefore

$$P(\neg d) = P\left(\neg d^F \wedge \neg d_1 \wedge \neg d_2 \wedge \ldots \wedge \neg d_V\right) \tag{4.9}$$

so long as there is not a vent in the cell the agent is moving into (because if there was, the observation would be $z = l$).

Next we note that if we detect a particle emitted from a specific cell, it implies that cell must be occupied by a vent, so $P(m_c|d_c) = 1$. Using Bayes' law gives

$$P(m_c|d_c) = \frac{P(d_c|m_c)P(m_c)}{P(d_c)} = 1 \tag{4.10}$$

If we only consider cells we know contain vents, then $P(m_c) = 1$ so

$$P(d_c) = P(d_c|m_c) \tag{4.11}$$

and $P(d_c|m_c)$ can be calculated using Equations 4.7 and 4.8. Finally, we assume plume detections from different vents are independent of each other, and use Equations 4.9 and 4.11 to get

$$P(z=n|s,a) = \begin{cases} 0 & \text{if } \mathbf{m}(a) = true \\ 1-P^F & \text{if } M=0 \\ \left(1-P^F\right)\prod_{c\in\mathscr{M}}\left(1-P(d_c|m_c)\right) & \text{otherwise} \end{cases} \tag{4.12}$$

4.6.3.3 Detecting a Plume (z=p)

The probability of detecting a plume is obtained as the probability of *not* not detecting a plume, where the probability of not detecting a plume is given in Equation 4.12. This gives us

$$P(z=p|s,a) = \begin{cases} 0 & \text{if } \mathbf{m}(a) = true \\ P^F & \text{if } M=0 \\ 1-\left[\left(1-P^F\right)\prod_{c\in\mathscr{M}}\left(1-P(d_c|m_c)\right)\right] & \text{otherwise} \end{cases} \tag{4.13}$$

Note that Equation 4.13 is in the exact form required for a forward sensor model in Jakuba's OG algorithm (Equation 3.1 in Section 3.1.2).

4.6.4 Belief-MDP Observation Model

In the belief-MDP, rather than using the state s, we need to find the observation probabilities in terms of the belief state b.

4.6.4.1 Locating a Vent (z=l)

If we move into a cell containing a vent, the observation will always be $z=l$, so by definition we expect the target cell a to contain a vent with probability

$$P(z=l|b,a) = P(m_a)$$

4.6.4.2 No Detection (z=n)

This derivation proceeds along similar lines to the one in Section 4.6.3.2, but in this case we do not know which cells contain vents. This means we must consider potential detections from all cells, but weight the contribution from each cell by the probability of that cell containing a vent (as given by the OG, which is part of the belief state b).

The probability of a detection is given by

$$P(d) = P\left(d^F \vee d_1 \vee d_2 \vee \ldots \vee d_C\right)$$

where we allow a possible detection from each of the C cells in the OG. The probability of no plume detection is therefore

$$P(\neg d) = P\left(\neg d^F \wedge \neg d_1 \wedge \neg d_2 \wedge \ldots \wedge \neg d_C\right) \tag{4.14}$$

However, we have to take into account the fact that if the cell we move into does contain a vent, the observation will be to locate a vent rather than nothing. So we want

$$\begin{aligned} P(z=n|b,a) &= P(\neg d \wedge \neg m_a) \\ &= P\left(\neg d^F \wedge \neg d_1 \wedge \neg d_2 \wedge \ldots \wedge \neg d_C \wedge \neg m_a\right) \end{aligned}$$

We note that $P(\neg d_a \wedge \neg m_a) = P(\neg d_a|\neg m_a)P(\neg m_a) = P(\neg m_a)$, as we know if a cell does not contain a vent then we cannot detect a particle from it, i.e. $P(\neg d_a|\neg m_a) = 1$. Using

$$P(d_c) = P(d_c|m_c)P(m_c) \tag{4.15}$$

from Equation 4.10, and the fact that $P(\neg m_a) = 1 - P(m_a)$, we get

$$P(z = n|b,a) = (1 - P(m_a))\left(1 - P^F\right)\prod_{c \neq a}(1 - P(d_c|m_c)P(m_c)) \tag{4.16}$$

if we again assume detections from different cells are independent. Note that here the product is over all cells except the target cell, whereas in Equation 4.12 it is only over cells containing vents.

4.6.4.3 Detecting a Plume (z=p)

As in Section 4.6.3.3, the probability of detecting a plume is obtained as the probability of *not* not detecting a plume. Based on Equation 4.16 we find

$$P(z = p|b,a) = (1 - P(m_a))\left(1 - \left(1 - P^F\right)\prod_{c \neq a}(1 - P(d_c|m_c)P(m_c))\right) \tag{4.17}$$

4.7 Discussion of Modelling Issues

This section highlights some alternatives and justifications for aspects of the model presented in Section 4.3.

4.7.1 State Versus Model Uncertainty

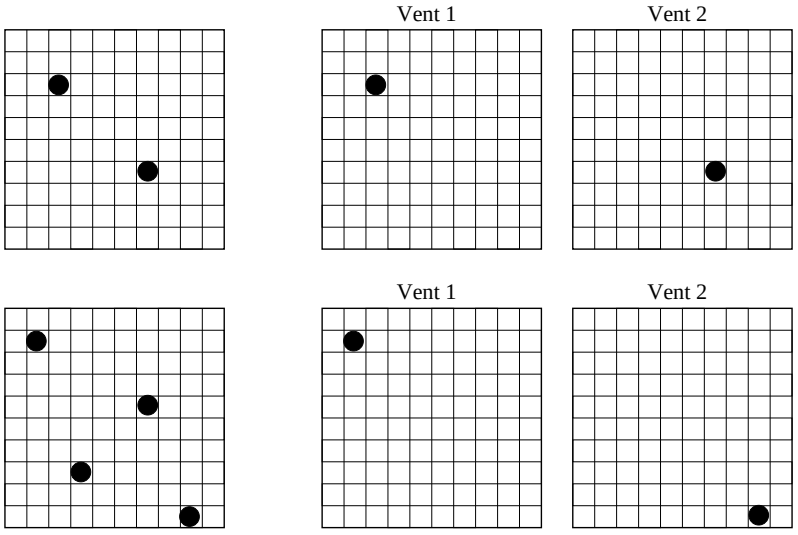
In Section 4.3, the location of the vents **m** has been presented as part of the state of the agent, s. This needs some justifying as **m** is fixed and will never change (except over timescales of dozens of years), so should be thought of as part of the environment model rather than the state of the agent. However **m** is still initially unknown, so we cannot simply include it in the environment model, and observations provide information about **m** so including it in the state of a POMDP provides a good model of the underlying situation. As far as the POMDP formulation is concerned, the main impact this has is that the state transition function is deterministic, rather than stochastic as for most MDPs and POMDPs. This obviously makes it a special case of a POMDP rather than requiring any special treatment, but may allow for some optimisations in solution methods.

4.7.2 Mapping Considerations

The separation of the problem into mapping and planning means that, as far as the planning agent is concerned, physical characteristics of vents such as their peak temperature or output flow rate are irrelevant and so can be excluded from the state.

Using Jakuba's OG methods as a foundation means we have to define a finite, rectangular search area. Care must therefore be taken in specifying the physical size of this search area, as if the area is too small the agent may explore all of it adequately before the end of the mission and have to waste time going over already-mapped areas, and if the area is too large then the agent will spend more computation time than necessary updating it. The ideal would be to have a map that grew automatically as the agent moved, but this is not straightforward to achieve with Jakuba's OG algorithm.

A major advantage of using an occupancy grid is that it intrinsically allows for any number of vents in the search area (given the limit of no more than one vent per grid cell). This is not the case for most mapping techniques, where each vent will be assigned a coordinate so the unknown number of vents means the dimensionality of state space is not known. This could be addressed by



(a) Full occupancy-grid state space model

(b) Factored state space model, with 2 vents

Figure 4.4: Options for representing the vent map. The upper part of the figure shows a vent map as it would be represented by both the full and factored models. The lower part of the figure shows a different map in the full and factored models.

factoring the state space into a variable for the number of vents, and then a separate map for each vent-count. An appropriate algorithm would then build the most likely map for each vent-count, for example build maps with $1, 2, 3, 4$ vents in them, and then use Bayesian model comparison techniques to choose between the maps. The OG and factored approaches are compared in Figure 4.4, where 4.4 (a) shows the full occupancy grid approach, and 4.4 (b) shows the factored approach for a two-vent map.

For a $v \times v$ grid, the OG state space size is 2^{v^2} states, rather than v^{2n} for the factored method for n vents. For example, in Figure 4.4, the full model has about 10^{30} distinct states, compared to 10^4 states for the factored model. Even with several different maps for different vent-counts, the factored approach allows a smaller state space, at the cost of a less clean mapping methodology and the need to handle multiple possible worlds (each of which is still uncertain) when planning.

4.7.3 Map Dimensionality and Vent Detector

The occupancy grid is 2D, and extending it to 3D has no value because the OG stores probabilities of cells containing a vent, and vents can only be found on the ocean floor. However the vehicle moves in a 3D environment, and Jakuba's forward model for plume propagation is also 3D, so it would be desirable to model the problem in 3D instead of the 2D abstraction presented in this chapter. The issue with moving to 3D is that it would significantly increase an already large state space size.

Planning in 2D can be justified by the fact that AUV vent-prospecting surveys are generally conducted at a fixed altitude, as the non-buoyant plume will be found at a fixed altitude. The problem is that this altitude will vary between different oceans and vent sites, so restricting the AUV to moving in one plane means the likely altitude of the non-buoyant plume will have to be estimated pre-mission. An alternative could be to use macro-actions to overcome the 2D restriction. In particular, a pre-programmed action could be for the AUV to dive to near the sea floor, and then determine the altitude at which plume tracer concentration peaks, which would be considered the altitude of the non-buoyant plume.

A similar macro-action could be used to implement the deterministic vent detector (as specified in Section 4.3.4), if plume tracers proved unable to provide sufficient certainty. This vent-detecting action would simply be for the AUV to descend from its survey height to near the ocean floor; if the tracer concentration increased all the way to the bottom, then the AUV is most likely to be in the buoyant plume and therefore directly above the vent (as shown in Figure 2.2), instead of being in the non-buoyant plume.

4.7.4 Actions and Observations

In my model, the only actions available are movement actions. On many AUVs other actions such as taking a photograph or collecting a water sample are available, but these have been excluded in the interest of maintaining a clear reward model.

The dynamics of the vehicle are not considered in the model as it is assumed the agent moves one grid cell per timestep. The only concession is that backtracking to the previous cell is not allowed, which means that the vehicle orientation is implicitly part of the state via c_{prev}, the agent's previous location. Similarly remaining in the same location is disallowed, as few AUVs are capable of hovering. Ignoring dynamics may be a significant approximation as regards vehicle speed, as the agent will cover ground much faster going with the current compared to against the current.

All observations are processed by Jakuba's algorithm with the output being an updated OG. This means that my novel algorithms need not consider any of the complexities of vent dynamics or how to relate sensor readings to possible source vent locations. However I have included an additional observation that Jakuba does not allow: detecting a vent at the agent's current position. This presented a choice for the possible observations to model:

1. {(vent, plume), (vent), (plume), (nothing)}
2. {(vent), (plume), (nothing)}

The second option was chosen, because the simulation setup means the agent will always detect a plume when it locates a vent, as there will always be a new plume particle co-located with the vent. This means that if the agent locates a vent, the fact it also detected a plume provides no further information, and having fewer observations allows more efficient implementations.

4.7.5 Localisation and Navigation

My model makes two related assumptions on localisation and navigation: that the agent always knows exactly which cell it is in, and that navigation actions are guaranteed to move the agent into the desired cell. These assumptions are related because if an AUV can track its own path precisely, then low-level vehicle controllers can provide waypoint-based navigation APIs, so the commanded destination will always be achieved (even if the vehicle has to alter course several times to get there). If grid cell coordinates were converted into waypoints, this would provide a natural and reliable method for migrating from my environment to a real AUV. This accurate navigation is certainly possible with some AUVs, because while GPS does not work underwater, AUVs can use Doppler sonar sensors in combination with inertial navigation systems to track their own movements very accurately (the Autosub team estimate an error of just 0.1% of distance travelled [McPhail and Stevenson, 2009]).

However, if the AUV does not know its location accurately this will invalidate much of the mapping process – finding a hydrothermal vent is of limited value if the vehicle cannot report the location of this newly discovered vent. A particular problem is that, after the vehicle obtains a GPS fix on the surface, it has to rely on dead reckoning to estimate its position while it is descending (until it is within a few hundred metres of the sea floor and can use Doppler sonar). This means its location fix can be subject to considerable drift due to the current [McPhail *et al.*, 2009]. AUV operators make

great efforts to ensure the navigation is accurate, including using, for example, seafloor transponders to communicate acoustically with the AUV [Yoerger *et al.*, 2007b]. Section 8.3.5 discusses further ways localisation could be improved, even when using relatively low-cost AUVs.

4.7.6 Rewards

The reward function simply optimises for finding as many vents as possible during a mission. An alternative goal would be to minimise the "false negative" rate, i.e. minimise the number of vents in the search area that are missed. This was not done because discussions with oceanographers, in particular Bramley Murton of the National Oceanography Centre, Southampton, indicated that finding as many vents as possible would provide the most benefit, rather than ensuring all vents in a particular region are found.

As mentioned in Section 4.3, the reward function does not include any component for exploration, such as visiting previously unvisited areas of the grid. Adding such a component would constitute *reward shaping*, which means modifying the true reward function in order to guide algorithms toward good solutions. If it is not done carefully, reward shaping can lead to worse rather than better solutions [Ng *et al.*, 1999], but in any case such artificial alterations to the reward should be part of the solution method rather than the problem definition (and in fact, the algorithms presented in Chapter 5 make no explicit use of the reward function defined in Section 4.3).

Two other potential contributions to the reward function were not considered: a negative reward for the vehicle hitting the bottom of the ocean, and a negative reward for the vehicle coming *too close* to a hydrothermal vent. Hitting the bottom risks damaging the AUV or even it getting stuck, but existing low-level vehicle control software is already capable of preventing such collisions. Situations relating to vents are potentially more risky, as plumes very near the vent are hot enough to melt parts of the AUV. However the proximity required to find such high temperatures is much smaller than the likely dimensions of a grid cell, making it hard to model in my framework. Additionally at medium-level altitudes (50 m and above), which are likely altitudes for missions of the type considered in this thesis, plumes are unlikely to be hot enough to damage the vehicle.

4.7.7 Resource Limits

Resource limits require special attention in a planning context, especially when resource usage differs between actions and follows a stochastic distribution even once an action is chosen [Bresina *et al.*, 2002; Dearden *et al.*, 2003; Mausam *et al.*, 2005]. The limited resource in vent prospecting is the charge in the AUV's battery, which cannot be recharged during a mission. How quickly the battery is drained depends on a variety of factors, such as the speed of the vehicle, the relative direction of the current, and the depth and temperature.

In a POMDP framework, limited resources could be addressed by simply including the resource as a state variable. The reward function would then include a large negative reward for the AUV running out of power before it has returned safely to the surface, as collecting data from vents has no value if the robot is unable to safely return the data to the surface.

However the formulation in Section 4.3 addresses resource limits by fixing the mission duration, which is common for AUV deployments (for example [McGann *et al.*, 2008c]) and is simple and clear from both a theoretical and implementational point of view. The use of a finite duration mission removes the need to address issues of how different actions will drain the battery faster, and also to make decisions about whether to save the data or do more exploring. In my model it also means the total mission distance is fixed, which is less likely to be the case for real-world AUV deployments.

4.8 Fully-Observable Variant

This section describes the characteristics of the fully-observable version of the problem, which is still a hard problem to solve. We consider two situations: first solving a given instance of the fully-observable problem, assuming an omniscient agent that can see the full state and is capable of on-board planning, and second calculating a policy that will provide the optimal action for all possible states. The latter problem helps us understand the challenges of the full POMDP problem, and is relevant in heuristic methods that are often used to solve POMDPs (see Section 6.1).

A given instance of the fully-observable problem is equivalent to a form of the *travelling salesman problem* (TSP), for which milestone algorithms were presented by [Dantzig *et al.*, 1954; Lin, 1965] (although the problem was known for some time before these works). The standard TSP is to find the shortest possible circuit that visits all of the cities in a set, where the pair-wise distances between all cities are supplied, and the tour must start and finish at a specified city. The fully-observable vent prospecting problem is to find the shortest path that visits all vents (cities), given a non-vent starting location, and no restrictions on the vent the path should end at. This is equivalent to the *path-TSP* [Papadimitriou, 1977], as the starting location can just be treated as an additional city, and the aim is to find the shortest path linking all cities. It is shown in [Papadimitriou, 1977] that the path-TSP is equivalent to the tour-TSP in terms of complexity class, and both problems are NP-complete. Papadimitriou also notes that his results apply to 'manhatten' problems such as our OG domain where cities are laid out in a grid pattern.

While the path-TSP is NP-complete, we expect the number of vents in the search area to be small, of the order of 10 or fewer. Given this, the problem can actually be solved by exhaustive enumeration of possible paths in a very short time (less than a second on a modern computer). Therefore specific instances of the fully-observable problem can easily be solved.

Finding the policy for the fully-observable case is much harder, simply due to the size of the state space. The c_{prev} and $\underline{\mathbf{U}}$ state variables can be discarded given a fully-observable state, but the $\mathbf{m}$ and c_{AUV} variables are required, and result in $C \cdot 2^C$ possible states (as in Section 4.4). For any reasonable grid size, this makes it impossible to even store a complete policy, as the policy consists of a mapping from each state to the action to take in that state. Further, there are $C+1$ state variables, C of which are binary variables, which makes it unlikely that the policy is a smooth surface that could be well represented by a function approximator trained on a relatively small dataset. So while specific instances of the fully-observable problem can generally be solved, finding a full policy for such problems is not possible, and approximating the policy is likely to be an extremely hard problem.

4.9 Relation to RockSample and FieldVisionRockSample

This section compares the model developed in this chapter with two benchmark domains that are well-known in the POMDP community and have many similarities to my model. The first is *RockSample* [Smith and Simmons, 2004], which is an abstracted version of the problem of an autonomous planetary rover choosing which rocks to sample. RockSample is based on a grid-world, and an instance with an $v \times v$ grid containing k rocks is known as RockSample$[v,k]$. Each rock can be either *good* or *bad*, but the agent does not know in advance which are good and which are bad. The agent starts in the middle-left cell in the grid, and has to reach an absorbing state which is accessible from any of the cells along the right-hand edge of the grid. There are $k+5$ actions available to the agent, $\{north, east, south, west, sample, check_1, ..., check_k\}$, where the movement actions are deterministic, *sample* samples the rock at the current location, and $check_i$ runs a sensor action on rock i. The $check_i$ actions are probabilistic, returning the correct classification (good or bad) with probability X where X decreases exponentially with Euclidean distance between the agent and rock i, from 1.0 for adjacent rocks to 0.5 at the maximum possible distance. The *sample* action has reward +10 for sampling a

good rock, in which case the rock is then marked as bad, and -10 for sampling a bad rock, and moving into the terminating state has a reward of +10.

If $k = v^2 = C$, i.e. every cell contains a rock, RockSample is very similar to the POMDP formulation of the vent prospecting problem where a good rock is equivalent to a vent, and a bad rock is equivalent to an empty cell. The state space is almost identical, with $C \cdot 2^C$ states (see Section 4.4), ignoring the current and previous cell in my model and the terminating state in RockSample. The movement actions are identical and deterministic in both cases, but the *sample* and $check_i$ action in RockSample are not present in my model as sensors are assumed to operate continuously. The reward model is very similar; RockSample has a rewarding final state as opposed to a fixed number of timesteps, and also has a negative reward for sampling a bad rock. However the *sample* action would be a good representation of the macro-action of diving to check for a vent (described in Section 4.7.3), which would be an expensive action to take if there turned out to not be a vent there (i.e. a bad rock).

The significant difference between RockSample and the vent prospecting model is the observation model. In RockSample, the model only observes one cell at once, and is fast to compute, whereas the plume detection model observes a large proportion of the search area at once, and is computationally expensive. Another difference is in typical problem size; while RockSample$[v, v^2]$ is equivalent to the vent domain, most problems considered in the literature have $k \approx v$ and both less than 20. For example, [Smith and Simmons, 2005] solve RockSample[10,10] with 10^5 states but consider it too large to be solved by most POMDP algorithms, compared with experimental settings used in this thesis equivalent to RockSample[20,400] with 10^{123} states.

An extension of RockSample, *FieldVisionRockSample*, is introduced by [Ross *et al.*, 2008]. FieldVisionRockSample is in some ways even more similar to the vent prospecting domain, as the $check_i$ actions are removed and instead the agent observes every rock on every timestep (with the same probability model as before). This makes the action space almost identical to my model, and the observation function is more similar in that it runs continuously and observes the whole grid, but less similar in that FieldVisionRockSample observes the state of each rock independently so there are 2^k observations as opposed to just three in my model. Also, the observation model of the vent prospecting model is complex and computationally intensive, which is a key characteristic of the problem and significantly different from FieldVisionRockSample.

Chapter 5

Planning Using Entropy

This chapter presents a set of novel algorithms using entropy to drive vent prospecting. First a review of relevant prior work is presented, then in Section 5.2 the experimental method and settings used to evaluate algorithms in this thesis are described. Sections 5.3, 5.4, 5.5, and 5.6 describe the algorithms, with Sections 5.5 and 5.6 also presenting experimental results. Finally, Section 5.7 presents a summary of all the algorithms and discusses their relative performance.

5.1 Previous Work On Adaptive Exploration

This section examines previous work on the problem of how a robot should efficiently explore an unknown environment. This is a slightly different problem to the one addressed by this thesis, where the aim is to visit sites of interest (vents). Developing a good map is of obvious benefit when planning to visit locations in the map, and optimising for map quality can sometimes directly induce the agent to visit the locations of interest (for example [Vergassola *et al.*, 2007]). Further, some of the methods discussed below are fairly general, in that they include an objective function that could easily be modified to reward visiting certain locations. Having said this, the difference of objectives is the main obstacle to adopting the work described in this section for hydrothermal vent prospecting.

At a high level, the problem can be seen as one of choosing a subset of locations to take samples at, so as to minimise the uncertainty in the posterior map. For the case of robot path planning, these locations are subject to the constraint that they must be contiguous. As the agent will gather more data from its sensors as it moves, it is clear that a better map will result if the agent periodically re-plans its path, taking into account the latest data. Re-planning on every timestep is known as *adaptive planning*, and it allows the agent to concentrate its exploration on the areas it is most uncertain about. Most of the methods discussed in this section are adaptive.

The degree of uncertainty of a map can be characterised by the entropy of the map, so the underlying idea of all entropy-based planning algorithms is to take the action that reduces the map entropy the most, and therefore produces the 'best' map. They build on the theory of *maximum entropy sampling* [Lindley, 1956], a methodology for deciding what the most informative experiment to perform is. The methodology prescribes performing the experiment with the highest entropy in the distribution of predicted observations, and this has recently been applied to adaptive observation planning by [Loredo, 2004].

Much work has also been done in the context of the *SLAM* problem – simultaneous localisation and mapping [Dissanayake *et al.*, 2001; Thrun, 2002b; Montemerlo *et al.*, 2003]. In SLAM, as well as having imperfect mapping sensors, the robot is unable to measure its own position accurately. The objective is to find a good estimate of both the robot's position and the map of the area the robot has explored. SLAM often makes use of an extended Kalman filter (EKF) to collate non-linear observations into a state estimate, and most implementations rely on identifiable landmarks

to form the map rather than using an occupancy grid. Early work on SLAM concentrated on the algorithms used to estimate the map and position, given a fixed path, but recently there has been more interest in path planning to guide data collection. This is known as *active SLAM* (discussed further in Section 5.1.4), and usually involves a trade-off between exploring new areas to enhance the map, and re-visiting places the robot has already been in order to reduce positional uncertainty.

This section is organised as follows: in Section 5.1.1, work on maximising submodular functions is discussed, with applications to optimal sensing problems. Next Section 5.1.2 discusses using entropy as a heuristic to solve POMDPs, then Section 5.1.3 covers some introductory ideas for using entropy to plan in OGs. In Section 5.1.4 I describe some innovative solutions to the active SLAM problem, and finally the infotaxis algorithm of [Vergassola *et al.*, 2007] is discussed separately (Section 5.1.5), as it was the inspiration for much of the work presented in this chapter, and deals with searching for a specific target.

5.1.1 Submodularity

Krause and Guestrin address the problem of choosing where to take measurements to gain the most information by leveraging submodularity [Krause and Guestrin, 2007]. The possible sensing locations are discretised into a set of points V, and the aim is to select a subset A of at most k locations so as to maximise a value function $F(A)$ (where $A \subseteq V$). A set function F is submodular if adding a new sampling location helps more when fewer locations have already been sampled: formally, for any two sets A and B, $A \subseteq B$, and any node $c \notin B$, if F is submodular then

$$F(A \cup \{c\}) - F(A) \geq F(B \cup \{c\}) - F(B)$$

An adaptive greedy approach is often used for such problems, where on each timestep the next sensing location c is chosen to maximise $F(A \cup \{c\})$, given that locations A have already been sampled. [Nemhauser *et al.*, 1978] show that for submodular functions, the greedy algorithm is guaranteed to produce a solution with a value of at least $(1 - 1/e)F_{opt}$ where F_{opt} is the value of the optimal solution, $F_{opt} = \max_{A:|A|<k} F(A)$. In other words, the greedy algorithm performs surprisingly well, attaining approximately 63% of the value of the best possible solution.

In [Krause and Guestrin, 2005], the specific problem of where to position temperature sensors to gain the best possible map of temperatures throughout an office is considered. They show that the information gain for adding a new sensing location, given by the reduction in entropy, is submodular. This means the greedy algorithm is an effective approach for domains where information gain is to be maximised, and they also provide an extension of the greedy algorithm to allow for different measurements having different costs.

If the observations are to be made by a mobile robot, the additional constraint of linking the sensing locations together in a path must be included. Further, the constant acquisition of new measurements implies that an adaptive algorithm will perform best, and both of these issues are handled by an algorithm presented in [Singh *et al.*, 2009]. The authors take advantage of two properties of the problem: submodularity and locality, where locality means that observations from locations that are physically far apart are approximately independent. Given these two properties, they define a nonadaptive algorithm pSPIEL$_{OR}$ that intelligently divides the possible sensing locations into clusters, and then applies a standard modular orienteering algorithm on each cluster. For the adaptive case, they re-run pSPIEL$_{OR}$ on every timestep, and provide a bound relating the performance of this algorithm to the optimal policy.

The key difference between the domain considered by [Singh *et al.*, 2009], that of sampling water in a lake, and the vent prospecting problem is that vent prospecting is not local (because plume detections can originate from vents at the opposite end of the search area).

5.1.2 POMDP Approximation

The exploration tasks being considered in this section are actually instances of POMDPs, and one way of addressing them would be to define the problem as a POMDP, and then leverage entropy reduction as a heuristic to choose actions. In [Cassandra *et al.*, 1996], the goal is for a robot navigating around a known map to reach a specific location, despite uncertainty in action outcomes and its initial location. The authors calculate the correct Bayesian belief state, and investigate various sub-optimal control strategies given this belief state. In the *entropy-weighting* (EW) algorithm, the basic principle is to choose actions that maximise the reduction in the uncertainty of the belief state. However, sometimes uncertainty in the belief state will not affect which action is optimal, and when this is the case entropy reduction is unnecessary. The first step in the EW algorithm is to calculate the solution to the fully-observable (MDP) problem (as in Section 4.8), which is a state-to-action mapping, and use this together with the belief state to calculate the probability that each action is optimal. Then if the entropy of this action-optimality distribution is less than a threshold value, it is used to find the best action. Otherwise, the action that reduces the entropy of the belief state the most is chosen. This approach is compared to the *most likely state* (MLS) algorithm, which chooses the action that would be optimal if the agent were in the true state that is most probable in the belief state. The EW algorithm performed worse overall than MLS, especially when there was more uncertainty in the initial state. However the authors concluded that EW may be more effective in domains that include perceptual actions, whose only effects are informational.

5.1.3 Early Exploration Work with Occupancy Grids

Occupancy grids [Elfes, 1989] are a common approach in mobile robotic planning applications and entropy measures are frequently used to decide how to explore in these representations. [Thrun *et al.*, 2005] (§17.4) provide an overview of standard methods for planning for exploration using OGs. The underlying assumption of these methods is that the information gain expected in each cell can be calculated independently, similarly to the assumption that the occupancy probability of each cell can be found independently. The algorithm proceeds in two steps: first a grid of information gain values, mirroring the OG, is calculated, and second value iteration is used to find a policy based on the information gain values.

The problem is discussed in the context of a mobile indoor robot equipped with range finding sensors such as sonar, which will return an occupied/empty estimate for all cells sensed. Three possible approaches to finding information gain values are discussed:

1. Using the entropy of each cell as a proxy for information gain, where cell entropy is given by $H_c = -P(m_c)\log_2 P(m_c) - (1-P(m_c))\log_2(1-P(m_c))$.

2. Using the full information gain expected when sensing a cell, which is given by difference between the current cell entropy and expected new entropy. To calculate this, a sensor model that outputs the occupancy of a cell with some error probability is assumed. Then if we denote the probability of measuring "occupied" by p^+, and "empty" by p^-, the expected new entropy is $E[H'_c] = p^+H_c^+ + p^-H_c^-$, where H_c^+ and H_c^- are calculated by using the standard occupancy grid updates to find the new occupancy probability for the cell given an occupied/empty measurement respectively. In practise the true information gain values are very similar (apart from a constant factor) to simply using cell entropy H_c, which is faster to calculate.

3. Giving each cell a binary explored/unexplored classification. This is a very simple approximation where a cell that has been updated in the OG is marked as explored, and other cells are marked as unexplored. It is found to work well in indoor environments, where the search area is split into small sub-areas (rooms and corridors) which can be fully explored fairly quickly given sonar-type sensors, but little information is gained about adjacent sub-areas.

The value iteration algorithm outlined in [Thrun *et al.*, 2005] works from an information gain grid calculated using the third option. Unexplored cells at the edges of the current map are assigned a fixed value, and the value of other cells is found by maximising the cost of moving to an adjacent cell plus the value of that cell. The value iteration update is given by

$$V_{t+1}(s) = \begin{cases} \max_a \{r(s,a) + V_t(a)\} & \text{if } s \text{ is explored} \\ I_u & \text{if } s \text{ is unexplored} \end{cases}$$

where $r(s,a)$ gives the cost of moving from cell s to cell a, a indexes over all cells adjacent to s, and I_u is a constant value assigned to unexplored cells.

This leads to an algorithm named *frontier-based exploration*, where following the value function produces *greedy* exploration – it takes the agent to the nearest unexplored area, taking into account obstacles in the OG (via the movement cost function).

5.1.4 Active SLAM

This section deals with three general approaches that are used for the active SLAM problem: greedy entropy minimisation, value function approximation, and entropy minimisation by local path adjustment.

Greedy entropy minimisation [Bourgault *et al.*, 2002; Stachniss *et al.*, 2005] builds on the work discussed in Section 5.1.3 by computing the entropy of the whole occupancy grid, rather than of individual cells. The entropy of the OG is taken to be the sum of the entropies for each individual cell, which is valid given the assumption that the occupancy probabilities of each cell are independent. Actions are again chosen on the basis of maximising the expected reduction in entropy of the belief state.

Both [Bourgault *et al.*, 2002] and [Stachniss *et al.*, 2005] address the SLAM problem where the aim is to minimise uncertainty in the pose of the robot and the map, and both use an OG for the map plus an additional distribution over the robot pose. Bourgault *et al.* maintain a landmark-based EKF to track the pose, and combine the entropy of the OG and the EKF when evaluating actions, but Stachniss *et al.* use a Rao-Blackwellised particle filter where the particle filter tracks the robot's trajectory, and each particle has an associated OG tracking the map. In both cases, the occupancy grid is updated using data from a laser rangefinder. These approaches differ from my domain in that the priors for the grid cells in their mapping problem are 0.5. This means that any observation about a cell reduces the entropy about that cell, whereas in my case an observation of a plume will actually increase the entropy of some grid cells (see Section 5.5 for a fuller explanation).

The approach of learning a function approximator for the optimal value function is taken by [Martinez-Cantin *et al.*, 2007; Bush *et al.*, 2008]. Given a value function it is easy to find the optimal action; the problem is that, given a continuous belief state, it is not possible to use dynamic programming to find the value function. Instead, a parametric function $V_\beta(b)$ can be used to approximate the value function, and the parameters of this can be learnt by simulation combined with targeted Bellman updates.

[Martinez-Cantin *et al.*, 2007] use Monte-Carlo simulation of the mission in a domain where the agent has to plan a path between a start and end point, optimising for SLAM. The objective is to minimise the value function, defined as the average mean square error between the true state (robot pose plus map) and estimated state at the end of the mission.

They parameterise their policy as a path defined by a set of waypoints, and each Monte-Carlo simulation is performed using fixed waypoints. The output from a trial is an estimate of the state, obtained by applying an extended Kalman filter (EKF) algorithm to the observations, and therefore of the cost for that parameter set. Multiple trials are used to learn a Gaussian Process (GP) model of cost for different parameters, and this model is used to select the best parameters to use for the next

trial. Good parameters are ones where either the cost indicated by the GP is low, or the variance of the GP is high, corresponding to optimising for exploitation or exploration.

The key drawback to this method is that it evaluates the cost of a path without dynamically modifying that path. This means it is not correctly evaluating the benefits of information gained during execution of the plan (as this would allow the path to be changed), so it generates an overestimate of the cost of actions.

[Bush *et al.*, 2008] consider an exploration problem where the aim is to maximise coverage of a grid given a noisy remote sensor and a limit on the number of observations. They pose the problem as a POMDP where the reward for each action is the reduction in entropy of an occupancy grid map, and note that given the continuous belief state it is not possible to solve this POMDP using dynamic programming. Instead they use a parametric function $V_\beta(b)$ to approximate the value function, where $V_\beta(b)$ is a linear combination of basis functions. Their algorithm iterates over randomly-sampled states, calculates the expected reduction in entropy for each action, and uses these rewards plus the existing $V_\beta(b)$ in a Bellman update to find an improved estimate of the value of the state. This estimate is in turn used to refine the parameters β of the value function approximator V_β. The calculations are made tractable by using a root-mean-square error function as both the reward (as a close approximation to entropy) and as the basis functions, which allows closed-form solutions.

This problem differs from vent prospecting in that observations do not have to form a path, and again in that the aim is mapping rather than visiting specific targets.

[Low *et al.*, 2009] address the problem of sampling a hotspot field using mobile robots, which is more similar to the vent prospecting domain, from an information-theoretic perspective. They model the map as a log-Gaussian process, and use a variant of real-time dynamic programming to evaluate future action-observation paths according to an entropy minimisation criterion. The approach is interesting as it is independent of the map resolution, and they use an ocean sampling domain for evaluations, but it would not be easy to apply it to an OG map.

Entropy minimisation by local path adjustment is performed by [Kollar and Roy, 2008; Thompson and Wettergreen, 2008]. The underlying idea is to generate an initial path, perform several small perturbations at a point near the start of the path, and evaluate which of the resulting new paths maximises the expected entropy of observations. This is then repeated at locations further along the path.

[Kollar and Roy, 2008] use a feature-based EKF map, which they 'skeletonise' into a discrete set of possible sampling locations, identified by crossings or junctions in the map. A graph search algorithm is then used to find the connected subset of points that maximises the number of features observed, for a fixed length path. The second step is to optimise the order these points are visited in, so as to minimise the expected map entropy after performing the tour. This expected entropy is evaluated by simulating the path and observations, and updating the EKF accordingly. The path is altered by the travelling-salesman-problem inspired heuristic of swapping two edges until no further improvement results. [Thompson and Wettergreen, 2008] describe a similar method using a Gaussian process to predict observations on the basis of location plus ground images captured via remote sensing. They adopt a maximum entropy sampling approach to path planning, by incrementally making small adjustments to a path, and adopting the adjustment if it increases the expected entropy of observations made along that path according to the GP model.

5.1.5 Infotaxis

Entropy reduction has also been applied to chemical plume tracing, where the aim is to locate a chemical source rather than produce a map, by Vergassola, Villermaux, and Shraiman [Vergassola *et al.*, 2007]. Their approach is useful in macroscopic environments where areas of strong chemical signal are separated by regions of limited or no signal, so it is not possible to track the gradient to the source. Their *infotaxis* algorithm treats observations as a sequence of discrete detection events

(ignoring the magnitude of signals); the probability of a detection is higher nearer to the source, so the agent can use detections to build a probability distribution over the source location, $P(\mathbf{r}_0)$. This continuous distribution is built using a model of the source, which specifies the rate of emission of tracers, the lifetime of tracers, a diffusion coefficient and a mean current.

Vergassola *et al.* choose the action of the robot as the one which reduces the entropy (or uncertainty) of the source location by the most, where the entropy of a distribution $P(\mathbf{r}_0)$ is given by $S \equiv -\int P(\mathbf{r}_0) \ln P(\mathbf{r}_0)\, d\mathbf{r}_0$. First a set of points near the agent is selected, then for each point $\mathbf{r}$ in this set the change in entropy expected from moving to that point is evaluated:

$$\Delta H(\mathbf{r}) \quad = \quad P(\mathbf{r})\,[-H] + (1 - P(\mathbf{r}))\,[\rho_0(\mathbf{r})\Delta H_0 + \rho_1(\mathbf{r})\Delta H_1 + \ldots] \tag{5.1}$$

where $P(\mathbf{r})$ is the probability of finding the source at $\mathbf{r}$, and ρ_k is the probability of k chemical signal detections at $\mathbf{r}$ during a small time interval, with ΔH_k the corresponding change in entropy. The probability ρ_k terms are found by assuming a Poisson hit distribution, and integrating the mean rate of hits at a given location, $R(\mathbf{r}|\mathbf{r}_0)$, weighted by the probability of the source location $\mathbf{r}_0$, over all possible source locations.

In Equation 5.1 the first term on the RHS handles the case where the source is located at $\mathbf{r}$, and the entropy drops to zero as the source has been found. The authors note that this term is exploitative, encouraging the agent to go to the most likely source location, but the second term is exploration-biased, as it allows information gain even if the agent does not move. Therefore infotaxis provides a natural balance between exploration and exploitation in search problems.

Experimental results show that, in the absence of hits, the algorithm produces interesting search patterns: without wind, the agent moves in increasing radius spirals, and with wind, it moves in a mixture of cross-wind zigzagging and up-wind casting very similar to the behaviour of moths. Results also confirmed that entropy was an effective measure for searching, showing that residual search time dropped rapidly with decreasing map entropy.

While infotaxis originally worked for a single source only, it has been extended to multiple sources in [Masson *et al.*, 2009]. However this method cannot be applied to the vent prospecting problem considered here, as it requires multiple cooperating robots to produce the map. Each robot maintains only a single-source map, the map updates are conditioned on the (known) number of sources, and the aim of each robot is to visit only one source, all of which make it hard to translate multiple-source infotaxis to my environment.

5.2 Experimental Methods

This section describes the experimental methods and parameters used in this thesis to evaluate the effectiveness of vent-prospecting algorithms.

All experiments used a 20×20 grid with 400 cells, and a fixed mission length of 133 timesteps (corresponding to covering just under $1/3$ of the grid). The agent always started in the north-east of the grid (the top-right cell), and no restriction was placed on where the agent should be at the end of the mission. Vents were distributed randomly and uniformly over the search area. For all algorithms using an MDP a discount factor of $\gamma = 0.9$ was used in solving the MDP. The current simulation defined in Section 4.2 was used for every trial. This current was the same at all points of the grid, and as it flows on average toward the east-north-east, the agent's start location was the most down-current point in the search area.

For each algorithm, 600 trials were performed, 150 trials with each of 3, 4, 5 and 6 vents. Each trial used a different random seed, where the random seed determined both the placement of vents and the dispersion of plume particles from the vents. Random seeds were generated by `http://www.random.org/`, which uses atmospheric noise to produce random numbers, and the same set of 600 random seeds was used for experiments with each algorithm.

The occupancy grid prior was set to $P_c^p = 0.01$, independent of the number of vents. If the prior were to have been changed to reflect the actual number of vents for each trial, this would have provided the algorithm with information about how many vents it should expect, which it would not get in the real world. The value of 0.01 was chosen for the prior as it equates to having four vents in the search area, which is a reasonable expectation. As noted in [Jakuba, 2007], it is important to initialise the OG with a low prior when we expect to find only a few vents, as otherwise it is hard to apportion detection probability correctly to OG cells. Section 5.5 notes further that using $P_c^p = 0.5$ leads to very bad experimental performance.

The evaluation criterion for algorithms was how many vents were found during a 133-step mission. To allow results from trials with different numbers of vents to be combined, the percentage of vents found, f_v, was used. When calculating the mean percent $\bar{f}_v$, trials were weighted equally regardless of the number of vents in that trial, i.e. $\bar{f}_v$ was calculated using

$$\bar{f}_v = \frac{1}{600} \sum_{i=1}^{600} f_v^i$$

where f_v^i is the percent of vents found on trial i.

Mowing-the-lawn (MTL) is treated as a special case, because although the trackline spacing is an algorithm parameter, a fixed subset of cells will always be visited for a given trackline spacing. As the vents are distributed randomly and uniformly around the grid, the expected sample average for f_v using MTL is simply the percent of grid cells visited. Therefore the expected percent of vents found given infinite trials, $\bar{f}_v = 100 \cdot \frac{133}{400}$ (for 133 timesteps and 400 grid cells), is shown in results and no experiments using MTL were actually performed.

Graphs presented also show 95% confidence intervals, which are calculated using

$$I_{conf95} = 1.96\sqrt{\frac{\sigma^2}{600}}$$

where 600 is the number of trials, and σ^2 is the sample variance, $\sigma^2 = \frac{1}{600-1}\sum_{i=1}^{600}(f_i - \bar{f})^2$.

To determine if one algorithm is better than another to a statistically-significant extent, a one-tailed test was used. Each $\bar{f}$ result was taken to be the mean of a large population, and the difference-of-means formula was used to calculate the standardised variable, $Z = \frac{\bar{f}_b - \bar{f}_w}{\sigma_{\bar{f}_b - \bar{f}_w}}$, where the b subscript denotes the result for the better algorithm, and w for the worse algorithm. The standard deviation of the difference-of-means is given by $\sigma_{\bar{f}_b - \bar{f}_w} = \sqrt{\frac{\sigma_b^2}{600} + \frac{\sigma_w^2}{600}}$ assuming a sample size of 600 for each algorithm. For one algorithm to be better than another at a confidence level of 0.05, we require $Z \geq 1.645$.

An overview of the hardware and software used is given in Appendix A.

5.3 Basic H-MDP Algorithm

The first novel algorithm developed was inspired by the methods outlined in [Thrun *et al.*, 2005] (as summarised in Section 5.1.3). In the vent prospecting domain, frontier-based exploration is unlikely to be effective, because a single plume detection will update the occupancy probabilities for many cells, usually including cells at the limit of the search area. Therefore the unexplored frontier quickly becomes small and often far away from vents. This is shown in Figure 5.1; concentrating the search on cells that have not been updated would lead the agent *away* from likely vent locations.

Instead, either entropy or information gain must be used to assign value to cells, and in the interests of simplicity entropy was chosen. The value iteration algorithm described in [Thrun *et al.*, 2005] works by fixing the value of unexplored cells, and propagating value from those cells in toward

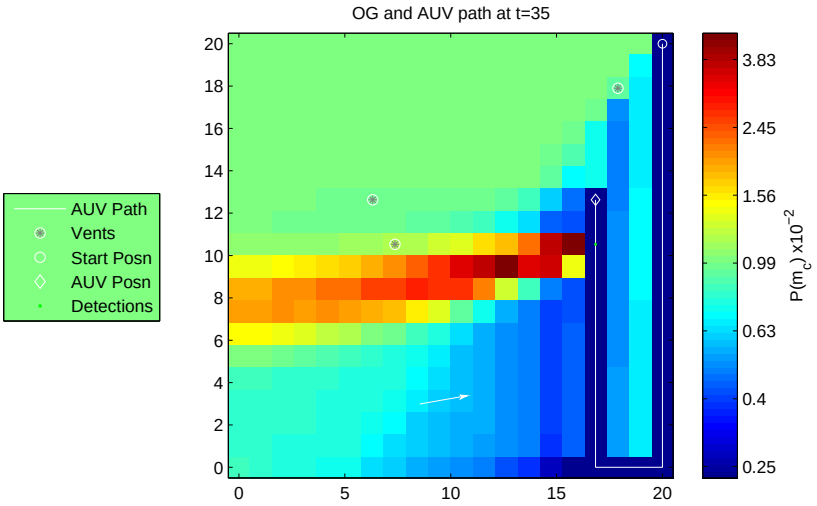


Figure 5.1: Example of the effect of a single plume detection (two timesteps previous to this image) on OG probabilities. The colour of grid cells represents the probability that they contain a vent, with the scale shown on the right of the map. The true locations of vents (unknown to the agent) are shown by grey stars in white circles. The path the agent followed to obtain this OG map is shown in white, starting at the white circle in the top-right corner, and terminating at the white diamond. The single plume detection is marked by a green dot just before the end of the path. The white arrow towards the bottom of the map represents the direction of the current on this timestep.

the centre of the explored regions. As all (non-visited) cells will have a non-zero entropy value, this kind of value iteration cannot be used for entropy, and there are two obvious alternatives: use entropy directly as cell value, or use entropy as the reward gained when visiting a cell. The first option is myopic and greedy, as it will simply choose the adjacent cell with the highest entropy, so the second option was selected. Then, as there are no obstacles to avoid in the underwater environment, the cost of moving to any adjacent cell will be the same. Therefore instead of having a movement cost function, I use a discount factor so that further away cells are valued less. This means the problem is posed as a standard MDP where value iteration is guaranteed to converge. These decisions define an algorithm I termed Entropy-MDP (or H-MDP).

The algorithm proceeds in three stages: first the entropy of each cell in the OG is calculated, using

$$H_c(b) = -P_b(m_c)\log_2 P_b(m_c) - (1-P_b(m_c))\log_2(1-P_b(m_c)) \tag{5.2}$$

where the notation $P_b(m_c)$ makes explicit that the occupancy probabilities come from the OG which is part of the belief state b. Second, value iteration over all cells is performed until cell values have converged, using the update equation:

$$V'(c) = \max_a \{H_a + \gamma V(a)\} \tag{5.3}$$

where a indexes over all cells adjacent to c. Note that for a "normal" MDP, the update would be given by $V'(s) = \max_a \{H_a + \gamma \sum_{s'} P(s'|s,a)V(s')\}$, but we have deterministic actions. This means we do not need to consider the probability $P(s'|s,a)$, but can instead treat actions as equivalent to the cell the agent ends up in. Third and last, the cell to move to, a, is chosen greedily with respect to the value function:

$$a = \operatorname*{argmax}_c \{H_c + \gamma V(c)\} \tag{5.4}$$

where c indexes over all cells adjacent to c_{AUV}, the agent's location. The full algorithm is shown in Algorithm 5.1, and an example of the agent's behaviour using this algorithm is presented in Figure 5.2.

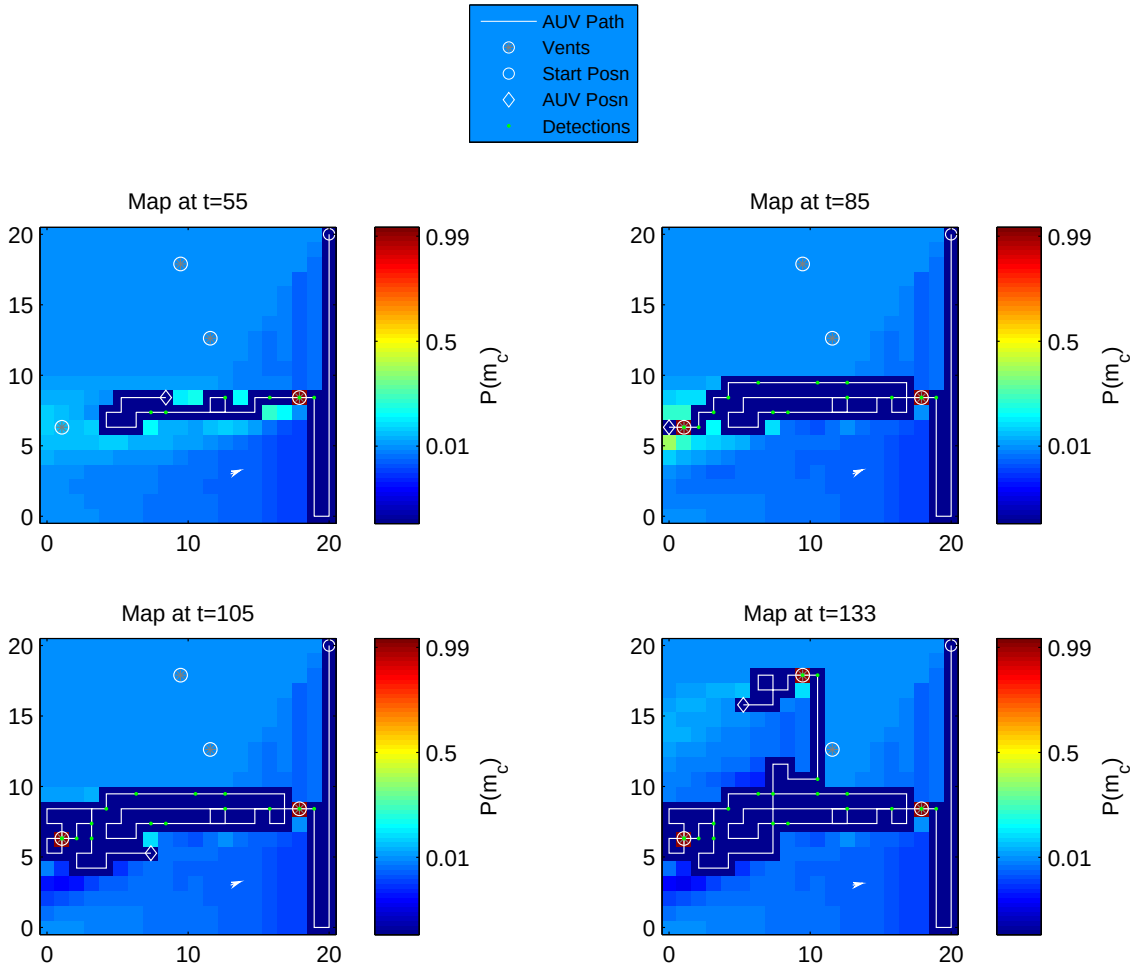


Figure 5.2: Example of a typical path generated by the H-MDP algorithm, illustrated by four snapshots of the same trial at different times. The agent starts at the white circle in the top-right, and its path is shown by the white line terminating in a diamond. The vent locations, which are initially unknown to the agent, are shown by grey stars in white circles, and the green dots represent points when the agent detected a plume. Finally, the map colouring indicates cell occupancy probabilities $P(m_c)$.

Algorithm 5.1 The H-MDP algorithm in pseudo-code. The procedure `choose-action-h-mdp` is applied at each timestep to select the action to take.

```
Procedure choose-action-h-mdp(b,c_curr,γ,θ): cell
  For each cell s
    Set rewards(s) = −P(m_s) log P(m_s) − (1 − P(m_s)) log (1 − P(m_s))
  Endfor
  Set values = 0..0
  Set oldValues = 0..0
  Set δ = 100
  While δ > θ
    Set δ = 0
    For each cell c
      values(c) = max_{a∈adjacent_cell(c)} {rewards(a) + γvalues(a)}
      δ = max(δ, |values(c)-oldValues(c)|)
    Endfor
    Set oldValues = values
  Endwhile
  Return argmax_{c∈adjacent_cell(c_curr)} values(c)
Endproc
```

5.4 ΣH (Infotaxis) Algorithm

The key deficiency of the H-MDP algorithm is that it fails to take into account the different amount of information that will be gained from taking different actions. As map uncertainty is what makes this problem hard, a planner that uses the observation function to reason about the effect of actions on the belief state is likely to do better than one that ignores observations. Therefore I decided to adapt the infotaxis algorithm of [Vergassola *et al.*, 2007] to the vent prospecting domain.

With an occupancy grid, the entropy of each cell is given by Equation 5.2. An OG does not represent a probability distribution, as the occupancy of each cell is calculated independently and (in general) $\sum_c P(m_c) \neq 1$. However, this independence of cell probabilities means we can sum cell entropies to find the total entropy of the OG,

$$H(b) = \sum_c H_c(b) \tag{5.5}$$

This result is derived by [Rocha *et al.*, 2005] as follows: first the joint entropy chain rule [Cover and Thomas, 2006]

$$H(X_1, X_2) = H(X_1) + H(X_2|X_1) \tag{5.6}$$

can be extended to a set of more than two random variables, $\mathscr{X} = \{X_1, \ldots, X_n\}$, to give

$$H(\mathscr{X}) = H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i|X_1, \ldots, X_{i-1}) \tag{5.7}$$

If all the random variables in $\mathscr{X}$ are independent, then $H(X_i|X_1, \ldots, X_{i-1}) = H(X_i)$ for all i, so Equation 5.7 becomes

$$H(\mathscr{X}) = \sum_{i=1}^{n} H(X_i) \tag{5.8}$$

which can be applied to the occupancy grid to give Equation 5.5.

Algorithm 5.2 The ΣH (infotaxis for vent prospecting) algorithm in pseudo-code. The procedure `choose-action-sh` is applied at each timestep to select the action to take.

```
Procedure choose-action-sh(b,c_curr): cell
  For each cell a ∈ adjacent_cell(c_curr)
    For each observation z
      calculate P(z|b,a)
      Set b' = srog(b,a,z)
      Set E_ΣH(a) = Σ_z P(z|b,a) Σ_c H_c(b')
    Endfor
  Endfor
  Return argmin_a E_ΣH(a)
Endproc
```

Thus the expected entropy after action a is given by

$$\mathrm{E}[H(b,a)] = \sum_z P(z|b,a) \sum_c H_c(b') \tag{5.9}$$

where $b' = \mathrm{SE}(b,a,z)$ is the belief state that results from applying Jakuba's algorithm to the OG component of belief state b, given an action a and observation z. Note that instead of allowing multiple detections per timestep as in [Vergassola *et al.*, 2007], we have summed over our observation set $z = \{l, p, n\}$ (where l represents locating a vent, p detecting a plume, and n detecting nothing). Vergassola *et al.* point out that the first term in Equation 5.1 makes the agent exploit its current map, as the entropy drops to zero if the source is found. While we have an equivalent term in Equation 5.9 (the summation element where $z = l$), allowing multiple sources means the entropy does not drop to zero when we find a vent, so our implementation of infotaxis has slightly less motivation to actually visit sources. Finally, as in [Vergassola *et al.*, 2007], we select the action that minimises Equation 5.9 (i.e., maximises the reduction in entropy). The full algorithm is known as ΣH (or infotaxis for vent prospecting), and is shown in Algorithm 5.2, with results using this algorithm presented in Section 5.5.

5.5 $\Sigma\Delta H$ Algorithm

As the vent prospecting domain uses an occupancy grid, it is significantly different from the domain that Vergassola *et al.* applied infotaxis to, and infotaxis can be improved upon for this domain. The OG allows for multiple sources, but we do not expect to find many sources in a given search area, so the prior probability of occupancy of cells in the OG is very low (we use 0.01). This leads to a peculiarity of the domain: when the agent receives useful information, the OG update causes the entropy of affected cells to *increase* rather than decrease. In particular, when the agent gets a plume detection, which is much more useful to it than no detection, the occupancy probability of all cells that could potentially contain the source vent rises. As the prior occupancy probability of these cells is very low, increasing $P(m_c)$ also increases their entropy, as illustrated by Figure 5.3 which shows a plot of cell entropy together with the low prior. Note that because the OG does not represent a distribution, increasing the probability of some cells does not mean the probability of other cells must decrease to compensate, and plume detections cause the entropy of the OG as a whole to increase.

A possible solution to this problem is to initialise the OG with a uniform prior, setting the prior occupancy probability to the maximum entropy value of 0.5. Unfortunately this throws away a lot of information, and the maps produced by the OG algorithm using a uniform prior are inferior to the

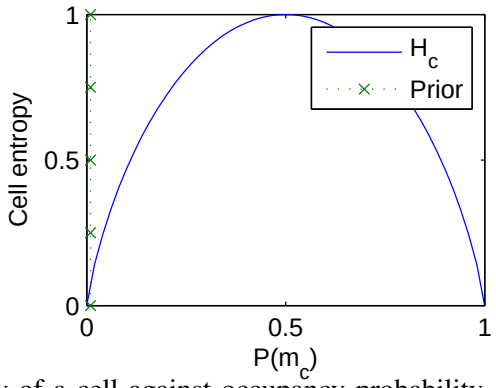


Figure 5.3: Plot of entropy of a cell against occupancy probability, where the entropy of a cell is calculated using $H_c = -P(m_c)\log_2 P(m_c) - (1-P(m_c))\log_2(1-P(m_c))$. A prior occupancy probability of 0.01 also plotted (the dashed green line running alongside the y-axis), illustrating the fact that whenever $P(m_c)$ values increase from their prior, cell entropy is almost guaranteed to increase.

Algorithm 5.3 The $\Sigma\Delta H$ algorithm in pseudo-code. The procedure `choose-action-sdh` is applied at each timestep to select the action to take.

```
Procedure choose-action-sdh(b,c_curr): cell
  For each cell a ∈ adjacent_cell(c_curr)
    For each observation z
      calculate P(z|b,a)
      Set b' = srog(b,a,z)
    Endfor
    Set E_ΣΔH(a) = Σ_z P(z|b,a) Σ_c |H_c(b') − H_c(b)|
  Endfor
  Return argmax_a E_ΣΔH(a)
Endproc
```

extent that infotaxis based on these maps is barely better than exhaustive search (infotaxis with a 0.5 map prior finds 38% of vents, compared to 66% with a prior of 0.01, and 33% for MTL). Instead the $\Sigma\Delta H$ algorithm fixes this issue by preferring actions that *change* the entropy of cells in the OG the most, regardless of whether the entropy of each cell actually increases or decreases. More precisely, we select the action a that maximises $\mathrm{E}_z[\Sigma\Delta H]$, where

$$\Sigma\Delta H(b,a,z) = \sum_c \left| H_c(b') - H_c(b) \right| \tag{5.10}$$

Equation 5.10 calculates, for every cell in the OG, the absolute value of the change in entropy between belief state b and belief state b' (where b' results from observing z in cell a), and then sums up the entropy change from all cells. This makes sense because the desirable observations (finding a vent or detecting a plume) result in large changes in $P(m_c)$ values, whereas detecting nothing results in only a small shift in $P(m_c)$ values. The full $\Sigma\Delta H$ algorithm is given in Algorithm 5.3. Some of the algorithms presented in this chapter have slightly non-intuitive names; Table 5.1 shows the relationship between all of the entropy-based algorithms (including $\Sigma\Delta H$-MDP which is introduced in Section 5.6), classifying them in terms of whether or not they are myopic, and what the key entropy calculation is.

Results for the H-MDP, ΣH (infotaxis), and $\Sigma\Delta H$ algorithms are shown in Figure 5.4. This figure shows the mean percentage of vents found over 600 experimental trials, performed using the setup

Table 5.1: Overview of entropy algorithms, showing how the algorithms differ in terms of whether or not they are myopic, and the core calculation used to find the value of cells.

Core calculation / Myopic?	Entropy (H)	ΔH	$\lvert\Delta H\rvert$
Yes (one-step lookahead)	[not tested]	ΣH (infotaxis)	$\Sigma\Delta H$
No (MDP)	H-MDP	ΣH-MDP	$\Sigma\Delta H$-MDP

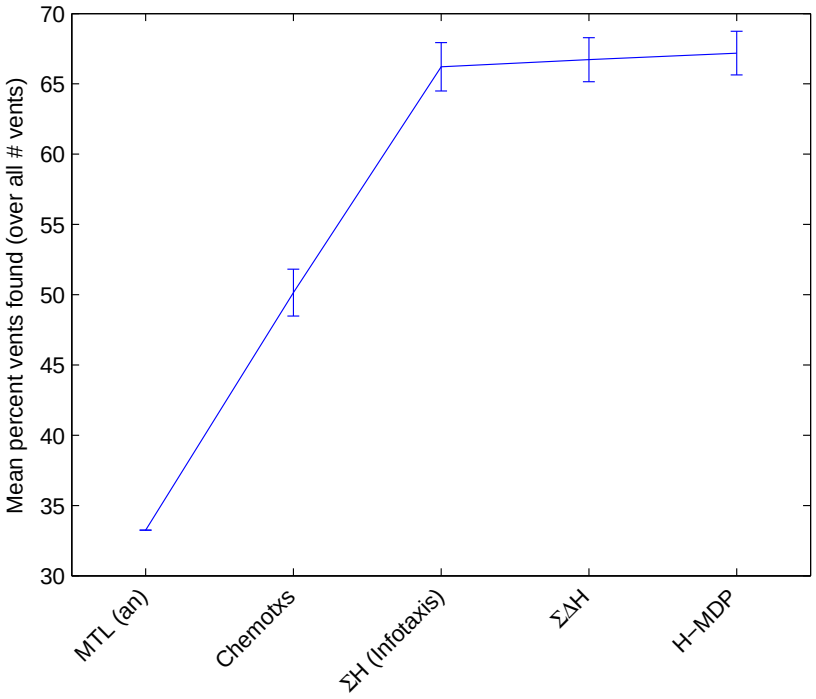


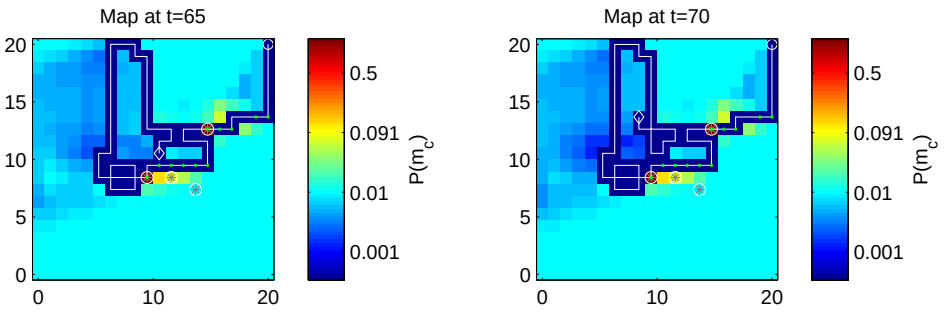
Figure 5.4: Results of H-MDP, ΣH (infotaxis), and $\Sigma\Delta H$. Also shown for comparison are MTL and chemotaxis. All results are means of 600 trials – 150 each with 3,4,5, and 6 vents. 95% confidence interval bars are also plotted. Note the lines connecting data points are purely to aid visualisation.

described in Section 5.2. Also shown are the comparison algorithms MTL and chemotaxis (described in Section 2.4). The figure shows that the novel entropy-based algorithms perform significantly better than both MTL and chemotaxis, with H-MDP, ΣH and $\Sigma\Delta H$ all finding more than double the number of vents found by MTL. $\Sigma\Delta H$ is, as expected, an improvement on ΣH but the non-myopic H-MDP algorithm is slightly better than both. However, the differences between the entropy-based algorithms are not statistically significant for a confidence level of 0.05.

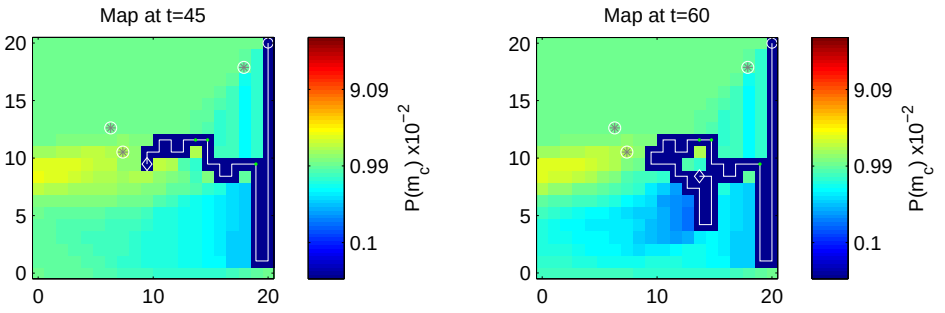
5.6 $\Sigma\Delta H$-MDP Algorithm

$\Sigma\Delta H$ was effective at locating vents, but not quite as good as the simpler H-MDP approach. Examination of its behaviour showed it often took myopic decisions, for example it would fail to explore high-probability areas of the grid if it would have to cross its path (consisting of zero-probability cells) to get there (see Figure 5.5).

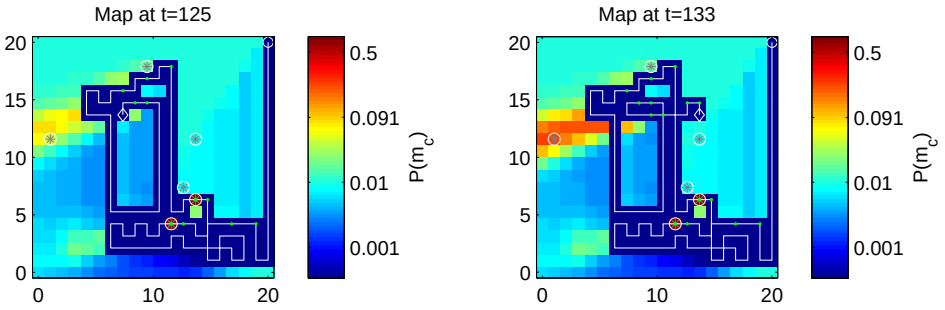
Infotaxis and $\Sigma\Delta H$ only consider the effects of one action, and better plans can generally be for-



(a) In the left-hand plot the agent is very close to an area of high probability where there is a vent, but by the right-hand plot it has turned away from that area because it is 'blocked' by the zero-probability cells it has already explored.



(b) Here in the left-hand plot the agent is very close to an area of raised probability where there is a vent, but it initially chooses to search east (where there is also raised probability). In the right-hand plot we can see that the agent searched two or three cells to the east, but had then exhausted the raised probability cells to the east of its previous location. However it still continued exploring east, as it was too far from the western area to take it into account.



(c) In the left-hand plot the agent is very close to an area of high probability where there is a vent, but by the right-hand plot it has turned away from that area because it is 'blocked' by the zero-probability cells it has already explored.

Figure 5.5: The agent's path (in white) plotted on top of OG values, showing three examples of the myopic behaviour of $\Sigma\Delta H$. The agent's location is shown by a white diamond, and the (unknown to the agent) vent locations are shown by white circles containing grey crosses.

med by anticipating outcomes further into the future. Therefore I developed the $\Sigma\Delta H$-MDP algorithm, which attempts to maximise $\Sigma\Delta H$ values over a discounted infinite horizon. The Bellman optimality equation for action-values Q to maximise $\Sigma\Delta H$ on each step is given by

$$Q(b,a) = \left(\sum_z P(z|b,a)\Sigma\Delta H(b,a,z) \right) + \gamma \sum_z P(z|b,a) \max_{a'} Q(b',a') \tag{5.11}$$

where γ is a discount factor, $0 \leq \gamma < 1$, that weights rewards lower the further into the future they are. Note that the $\Sigma\Delta H$ algorithm is a special case of this where $\gamma = 0$. Equation 5.11 cannot be solved iteratively as a set of $|\mathscr{B}|$ equations because the belief state b is continuous and cannot be enumerated. It can be solved by forward search through the belief space, but this is exponential in the number of steps of lookahead and therefore practical only for very small lookaheads. Instead we opt to separate the belief space into two components, $b = (\mathbf{O}, c)$, the OG plus the agent's location, and assume that the OG component is fixed. Given a fixed OG, Equation 5.11 can be collapsed to

$$Q(c,a) = \left(\sum_z P(z|b,a)\Sigma\Delta H(b,a,z) \right) + \gamma \max_{a'} Q(a,a') \tag{5.12}$$

where a' represents a cell index adjacent to a (i.e. a possible action), and we have used the starting belief state b to calculate $\mathrm{E}_z[\Sigma\Delta H]$. Equation 5.12 can be solved efficiently as a normal, discrete-state MDP, with the 'rewards' $\mathrm{E}_z[\Sigma\Delta H]$ being calculated for all grid cells a before running value iteration. The complete algorithm is shown in Algorithm 5.4; note that it is very similar to the H-MDP algorithm, but uses $\mathrm{E}_z[\Sigma\Delta H]$ instead of cell entropy as the rewards.

The intuition behind the $\Sigma\Delta H$-MDP algorithm is that we have ignored the effect of observations on the map when the robot moves to a new location, and instead we use the observation function when calculating the 'rewards' $\Sigma\Delta H$ we expect at each location. This allows us to make decisions based on expected observations far away, under the assumption that the mean OG will not change much as we travel to that location, without having to search over an information-state tree with a large branching factor.

Results are shown in Figure 5.6, where experiments were conducted according to the procedure described in Section 5.2. Figure 5.4 shows that both infotaxis and $\Sigma\Delta H$ are far better at finding vents than chemotaxis or MTL. $\Sigma\Delta H$-MDP improves on $\Sigma\Delta H$, validating our intuition that $\Sigma\Delta H$ would benefit from a longer planning horizon. Further, $\Sigma\Delta H$ improves on ΣH, and $\Sigma\Delta H$-MDP improves on the MDP version of infotaxis (ΣH-MDP) to a statistically significant extent.

5.7 Results and Discussion

5.7.1 Human Prospecting Results

This section presents results of the 40×40 grid version of the human-controlled vent prospecting experiments described in Section 2.4.2. The 40×40 experiments had 10 subjects who performed 38 trials in total, and the distribution of how many subjects completed how many trials is given in Figure 5.7a. The number of trials performed for each vent-count is given in Figure 5.7b, and is in line with the Normal distribution the vent-count was sampled from, where the lack of any trials with seven vents is just a random outcome. Finally, Figure 5.7c shows the performance of the human subjects in the experiments, where the percent of vents found was averaged over all of each subject's trials, and the subjects have been ranked in performance order. Figure 5.7c also shows the number of trials completed by each subject, and there is a slight trend for subjects who performed more trials to have higher average scores, but this is at best a small effect.

Algorithm 5.4 The $\Sigma\Delta H$-MDP algorithm in pseudo-code. The procedure `choose-action-sdh-mdp` is applied at each timestep to select the action to take.

```
Procedure choose-action-sdh-mdp(b,c_curr,γ,θ): cell
  Set rewards = calculate-sdh(b)
  Set values = 0..0
  Set oldValues = 0..0
  Set δ = 100
  While δ > θ
    Set δ = 0
    For each cell c
      values(c) = max_a{rewards(a) + γvalues(a)}
      δ = max(δ, |values(c)-oldValues(c)|)
    Endfor
    Set oldValues = values
  Endwhile
  Return argmax_{c∈adjacent_cell(c_curr)} values(c)
Endproc

Procedure calculate-sdh(b): values
  For each cell c
    For each observation z
      calculate P(z|b,c)
      Set b' = SE(b,c,z)
      Set E_ΣΔH(c) = Σ_z P(z|b,c) Σ_i |H_i(b') − H_i(b)|
    Endfor
  Endfor
  Return E_ΣΔH
Endproc
```

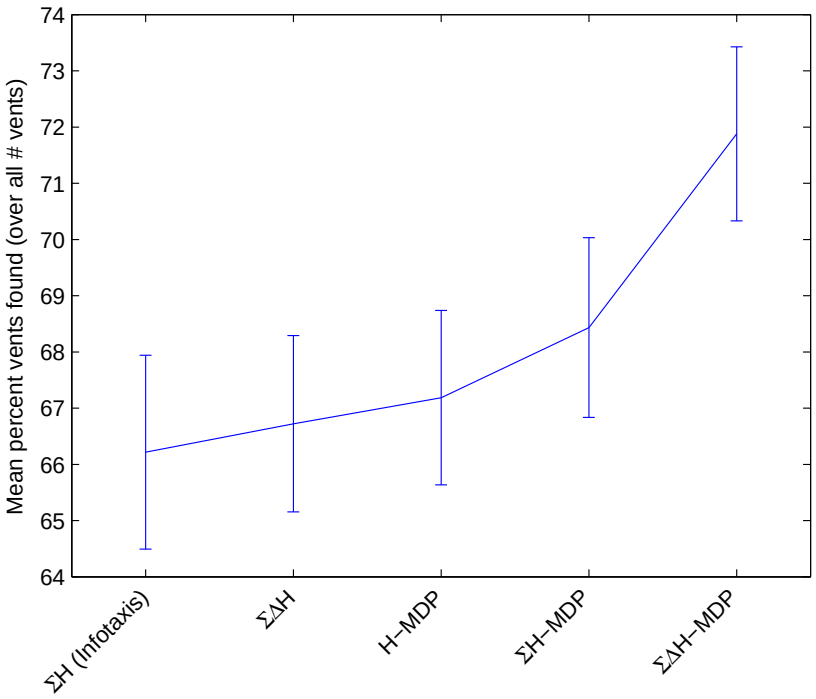


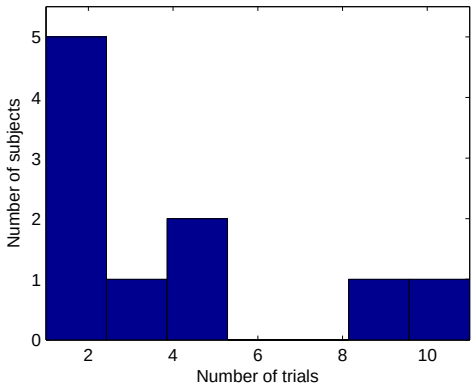
Figure 5.6: Results of H-MDP, infotaxis, infotaxis-MDP, $\Sigma\Delta H$, and $\Sigma\Delta H$-MDP. All results are means of 600 trials – 150 each with 3, 4, 5, and 6 vents. 95% confidence interval bars are also plotted.

While the 40×40 results were useful, the subjects varied considerably in their performance, and did not carry out enough trials to attach much statistical significance to. Further, using a different experimental environment to the one used to evaluate the novel algorithms developed in this thesis was in hindsight not a good decision. Therefore I decided to execute 100 trials myself, after some practise time, using the experimental conditions described in Section 5.2. The same random seeds used for other experiments were used, which necessitated selecting the number of vents randomly but keeping track of the next seed to be used for each vent-count. The alternative, executing all the 3 vent trials followed by all the 4 vent trials and so on, would have meant the experimenter always knew how many vents to expect, useful information that is not available to the automated algorithms.

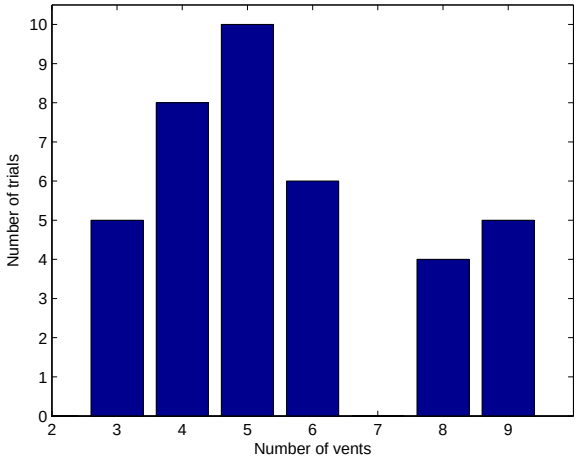
The results from the 20×20 setup experiments are discussed in Section 5.7.2, and Figure 5.11 shows the human performance relative to the novel entropy-based algorithms.

5.7.2 Novel Algorithm Results

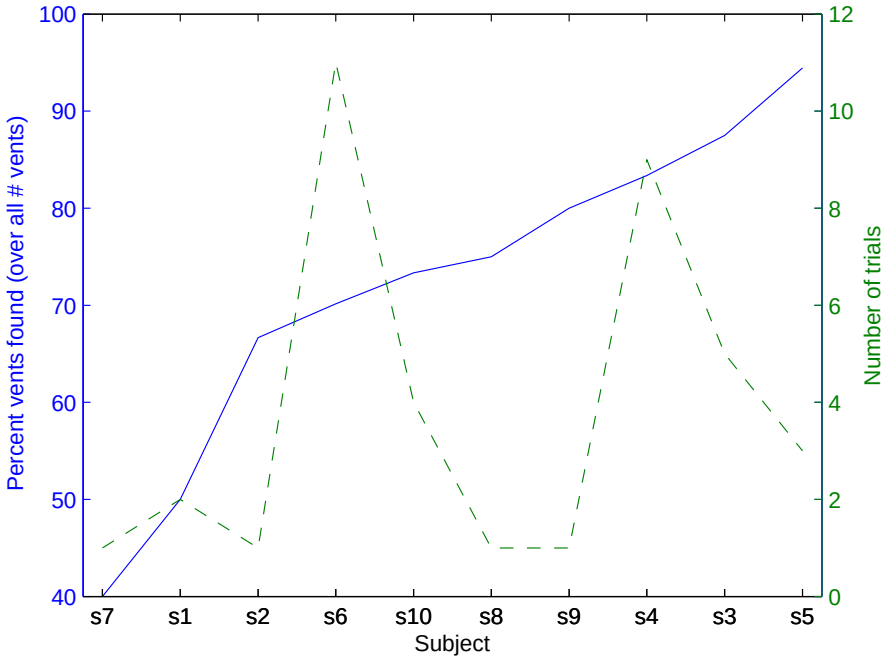
This chapter has introduced a variety of similar algorithms, and Table 5.1 helps to distinguish them by whether they are myopic or not, and by how cell values are calculated. Also, Table 5.2 shows the core mathematics underlying each of the algorithms. Results for MTL, Chemotaxis, $\Sigma\Delta H$ and $\Sigma\Delta H$-MDP, including the run-time of the algorithms in my test environment, are shown in Figure 5.8. The figure clearly shows the improvement over conventional techniques provided by the entropy-based algorithms, and in the case of $\Sigma\Delta H$ this benefit comes at very little computational cost. $\Sigma\Delta H$-MDP is better than $\Sigma\Delta H$ by more than five $\bar{f}_v$-points, but it is also much slower. The big cost in run-time is not solving the MDP, which is fast and only needs be done once per timestep, but in calculating the $\mathrm{E}_z[\Sigma\Delta H]$ values. Finding a $\mathrm{E}_z[\Sigma\Delta H]$ value requires that the occupancy grid algorithm be run three times, once for each observation, and this must be done for every cell in the grid. This occupies the



(a) Histogram showing that most subjects performed only a small number of trials on the vent-prospecting simulator.



(b) Number of trials performed for each vent-count.



(c) Performance of the human subjects in terms of the percent of vents found, ordered from worst-to-best performing subject. Also shown as a dashed line (and corresponding to the right-hand y-axis) is the number of trials each subject completed, which is only loosely correlated with their performance.

Figure 5.7: Statistical analysis and performance results from the human-driven vent prospecting experiments with a 40×40 grid.

Table 5.2: Summary and core mathematics for all entropy-based algorithms. Here a indexes over possible actions the agent can take (or equivalently possible cells it can move to), u indexes over cells adjacent to s, and the notation $\doteq$ is used to represent equality after iteration to convergence.

Name	Core mathematics (action selection)	Summary
H-MDP	$\text{argmax}_a \{H_a + \gamma V(a)\}$, where $V(s) \doteq \max_u \{H_u + \gamma V(u)\}$	Plans path to maximise entropy of cells visited
ΣH (infotaxis)	$\text{argmin}_a \{\sum_z P(z\|b,a) \sum_c H_c(b')\}$, where $b' = SE(b,a,z)$	Chooses action that maximises the expected reduction in entropy of the OG
ΣH-MDP	$\text{argmax}_a \{\text{E}_{\Sigma\text{H}}(a) + \gamma V(a)\}$, where $V(s) \doteq \max_u \{\text{E}_{\Sigma\text{H}}(u) + \gamma V(u)\}$ and $\text{E}_{\Sigma\text{H}}(s) = \sum_z P(z\|b,s) \sum_c (H_c(b) - H_c(b'))$ with $b' = SE(b,s,z)$	Plans a path that maximises the expected reward, where cell rewards are given by the expected reduction in entropy for making an observation from that cell (given the initial belief state)
$\Sigma\Delta H$	$\text{argmax}_a \{\sum_z P(z\|b,a) \sum_c \|H_c(b') - H_c(b)\|\}$, where $b' = SE(b,a,z)$	Chooses action that maximises the expected sum of *changes* in entropy of cells in the OG
$\Sigma\Delta H$-MDP	$\text{argmax}_a \{\text{E}_{\Sigma\Delta\text{H}}(a) + \gamma V(a)\}$, where $V(s) \doteq \max_u \{\text{E}_{\Sigma\Delta\text{H}}(u) + \gamma V(u)\}$ and $\text{E}_{\Sigma\Delta\text{H}}(s) = \sum_z P(z\|b,s) \sum_c \|H_c(b') - H_c(b)\|$ with $b' = SE(b,s,z)$	Plans a path that maximises the expected reward, where cell rewards are given by the expected *change* in entropy (summed over the whole OG) for making an observation from that cell (given the initial belief state)
Obs-Reward-MDP	$\text{argmax}_a \{\text{R}_{\text{obs}}(a) + \gamma V(a)\}$, where $V(s) \doteq \max_u \{\text{R}_{\text{obs}}(u) + \gamma V(u)\}$ and $\text{R}_{\text{obs}}(s) = P(z = l\|b,s) R_{vent} + \eta P(z = p\|b,s) R_{vent}$	Plans a path that maximises the expected reward, where cell rewards are given by a weighted sum of the probabilities of finding a vent and detecting a plume in that cell (given the initial belief state)

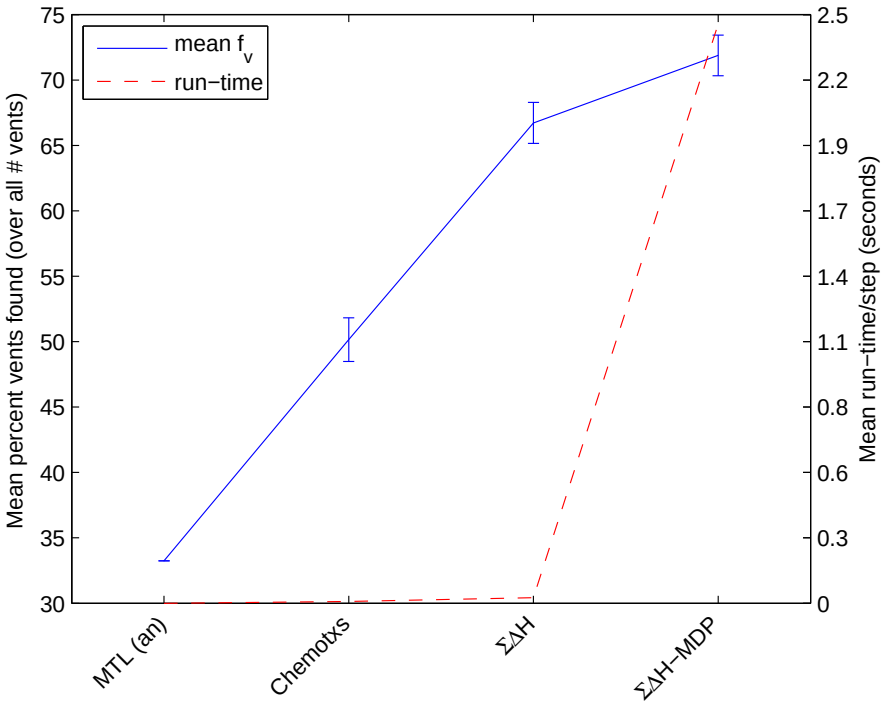


Figure 5.8: Results of MTL, chemotaxis, $\Sigma\Delta H$ and $\Sigma\Delta H$-MDP. All results are means of 600 trials – 150 each with 3, 4, 5, and 6 vents. The solid line shows the mean percent of vents found with 95% confidence interval bars, and the dashed line shows the run-time per timestep.

vast majority of the processing time needed to run $\Sigma\Delta H$-MDP, and is a problem also encountered by the ΣH-MDP algorithm (but not H-MDP).

A further comparison between ΣH, $\Sigma\Delta H$, and $\Sigma\Delta H$-MDP was performed by repeating the experiments with a single vent in the search area. This is a much less interesting test from the point of view of the objectives of this thesis, which are to enable multiple vents to be found efficiently, but it makes the vent-prospecting domain more similar to the domain infotaxis was developed in. The results are shown in Figure 5.9, and while $\Sigma\Delta H$-MDP is again the most efficient algorithm, the order of ΣH and $\Sigma\Delta H$ is reversed. I hypothesise that this is because the OG behaves much more like a true probability distribution when there is only one source, as once a plume has been detected, no plumes from other vents can be found subsequently. This will prevent the total entropy of the OG from oscillating as much, as $\sum_c P(m_c)$ will be less variable than for multiple sources.

Returning to the multiple-vent case, of the three MDP algorithms, $\Sigma\Delta H$-MDP was the best (at a statistical confidence level of 0.05), and $\Sigma\Delta H$ was also the best of the non-MDP entropy-inspired algorithms (although not to a statistically significant extent). The reason that $\mathrm{E_z}[\Sigma\Delta H]$ is actually a good measure to optimise can be understood better with reference to Figure 5.10, which compares change in entropy (as used by $\Sigma\Delta H$ algorithms) with change in occupancy probability, for a single cell. Figure 5.10 shows that changes in cell probabilities that frequently occur with useful observations – detecting a plume or locating a vent – result in higher rewards using entropy change than using probability change. In general, if a plume is detected, affected cells will change from low probability (for example the prior of $P(m_c) = 0.01$) to medium probability of around 0.2-0.5. For these cells, $|\Delta H_c|$ will give near to the maximum reward, whereas $|\Delta P(m_c)|$ gives less than half the reward. For changes of $P(m_c)$ from around 0.01 to ≤ 0.2, the difference in reward is more pronounced, and more

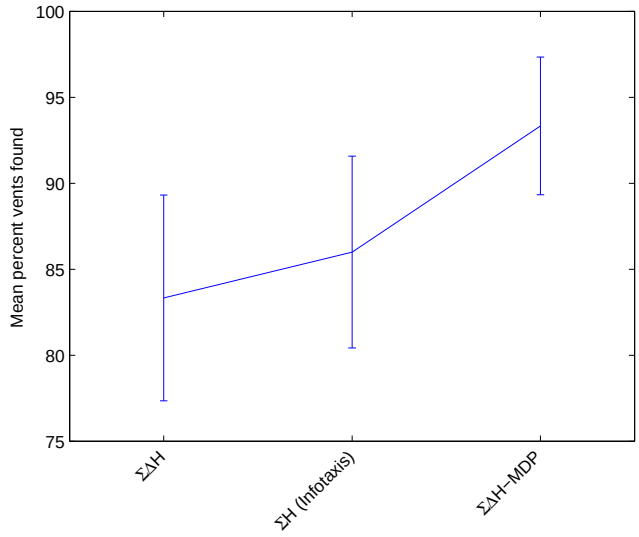
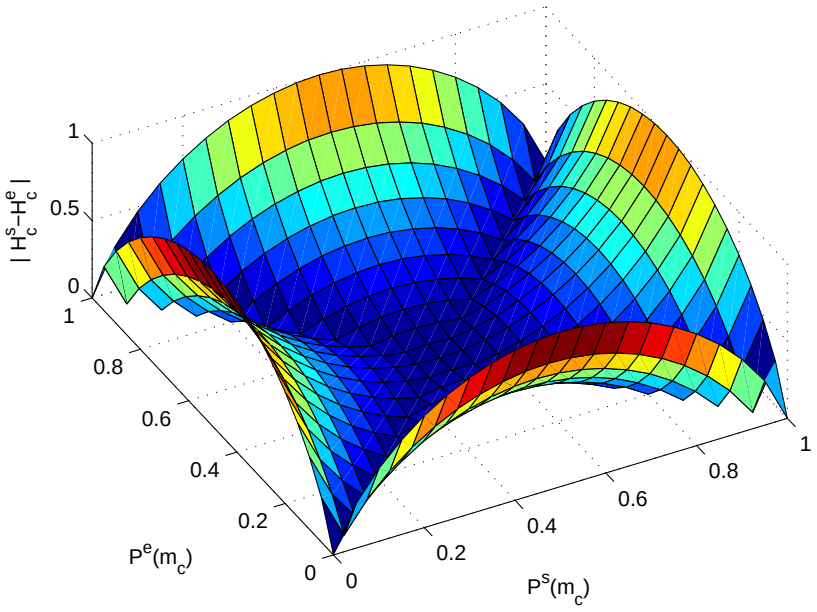


Figure 5.9: Results of $\Sigma\Delta H$, ΣH and $\Sigma\Delta H$-MDP with only one vent in the search area. Results are means of 150 trials with the vent located in a random position on each trial.

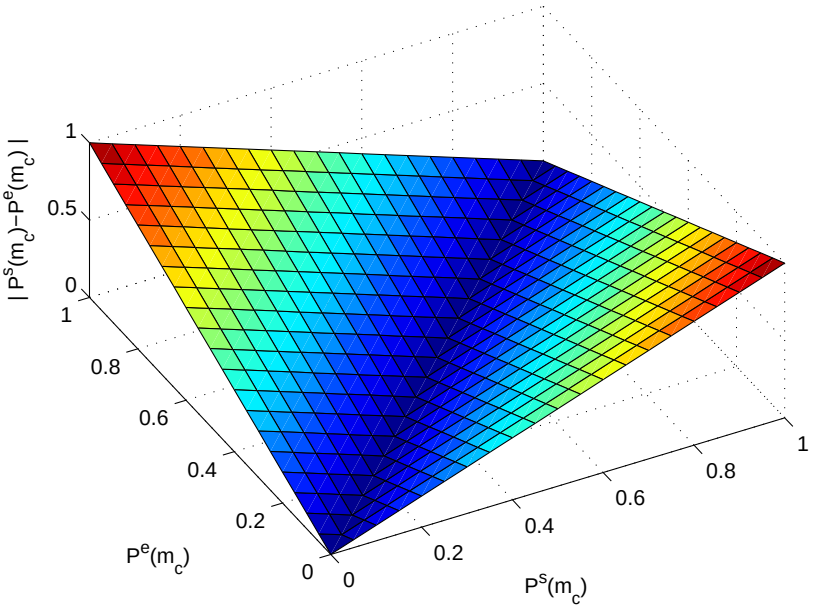
cells will end up in the range of $P(m_c) \leq 0.2$ than in the higher range following a plume detection. The difference when locating a vent is also important: if a vent is found in an unexpected location, and $P(m_c)$ changes from nearly zero to one, then $|\Delta P(m_c)|$ gives a maximal reward. So the probability-change reward is higher for finding a vent where one is not expected than for finding a vent where one is expected, and while this may encourage exploration, it is unlikely to lead to optimal behaviour as it fails to take advantage of the OG map. The entropy-change reward by contrast will produce a maximal reward if a cell with start probability $P(m_c) = 0.5$ is subsequently found to contain a vent (note that in practise, the OG algorithm almost never assigns cells $P(m_c) > 0.5$ unless they are known to contain a vent). So in summary, the reward function used by $\Sigma\Delta H$ encourages both exploration (by rewarding plume detections, making it worthwhile to visit unexplored areas) and exploitation (by rewarding highly when cells that the map indicates are likely to contain vents are found to actually contain a vent).

Given the preceding discussion, the question can be asked: why not directly reward the agent for both finding vents and detecting plumes, instead of using an esoteric entropy-inspired reward function? The idea of direct rewards for finding vents is an obvious one (and is part of the problem formulation presented in Section 4.3), but this means treating the problem as a POMDP with the associated tractability problems. However I take this approach in Chapter 6, and achieve results that are comparable to those presented in this chapter. In fact, the good performance of the entropy-based algorithms is a validation of the idea of using entropy for map-based problems with an exploration element, even when the ultimate goal is visiting or finding items of interest.

To illustrate the value of $\Sigma\Delta H$, additional experiments were performed with an algorithm that does reward directly for locating a vent or detecting a plume, the *obs-reward-MDP* algorithm. Like $\Sigma\Delta H$-MDP, obs-reward-MDP assumes the agent could 'teleport' to any cell in the grid, and calculates the expected reward at that cell. It then solves an MDP in these expected reward values to find the best action. The difference is that instead of using $\mathrm{E}_z[\Sigma\Delta H]$ as the cell reward, obs-reward-MDP uses the sum of $R_{vent} \times P(z = locate\,vent)$ plus $\eta \times R_{vent} \times P(z = detect\,plume)$, where $0 < \eta < 1$ and was set to $\eta = 0.05$ in experiments. Figure 5.11 shows the results for obs-reward-MDP, and it can be seen that all the entropy-based MDP algorithms outperform it. There are two reasons that rewarding for detecting a plume turns out to be a bad idea: first it causes the agent to try to follow plume particles



(a) Entropy change



(b) Probability change

Figure 5.10: Illustration of the per-cell reward using the $\Sigma\Delta H$ and $\Sigma\Delta H$-MDP algorithms (Subfigure a), as compared to using the change in occupancy probability (Subfigure b). The graphs show the start probability $P^s(m_c)$ and end probability $P^e(m_c)$ for a single cell, representing before and after an action. Subfigure a shows the absolute value of the difference in cell entropy, and Subfigure b shows the absolute value of the difference in occupancy probability. Note that both graphs are symmetric about the $y = 1 - x$ as well as $y = x$ planes, and differences in the colours of the peaks of the top graph are just shading effects.

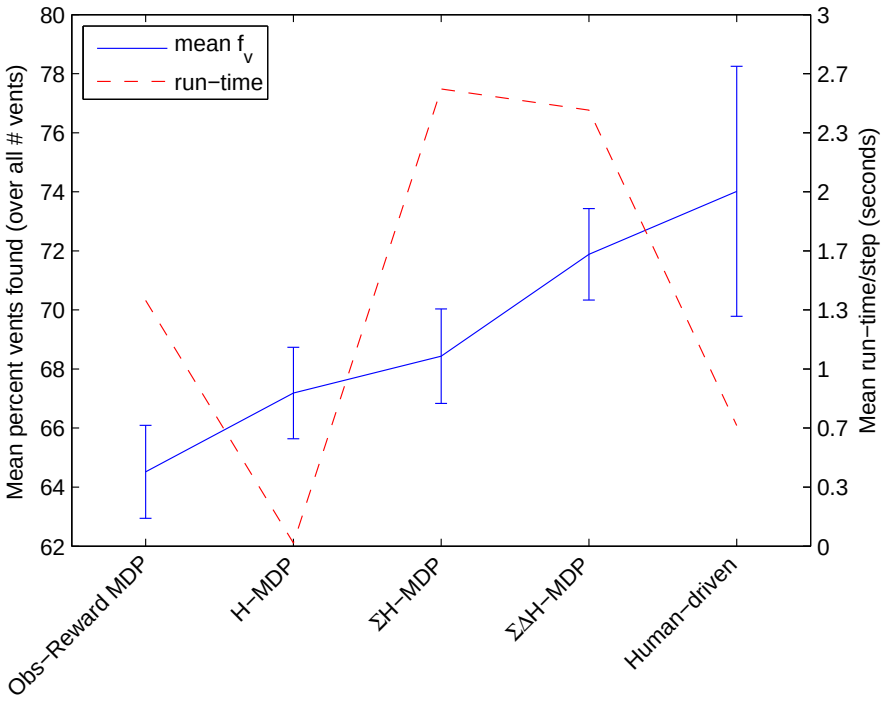


Figure 5.11: Results of obs-reward-MDP, H-MDP, ΣH-MDP, $\Sigma\Delta H$-MDP and human-driven vent prospecting. All results are means of 600 trials – 150 each with 3, 4, 5, and 6 vents – with the exception of human-driven which is the mean of 100 trials, 25 with each vent-count. The solid line shows the mean percent of vents found with 95% confidence interval bars, and the dashed line shows the run-time per timestep.

down-current, instead of searching up-current where the vent is likely to be, as the environment model can accurately predict a high probability of encountering the plume again. The second reason is that it discourages exploration; even with no reward for visiting a cell the agent has already been to, the algorithm tends to stay around raised probability cells instead of exploring new areas of the map.

Figure 5.11 also shows the results of human-driven vent prospecting using experimental conditions identical to those described in Section 5.2. While the human managed to find 74% of the vents compared to 71.9% for $\Sigma\Delta H$-MDP, this is not a statistically significant improvement for a confidence level of 0.05, due to the relatively small number of trials in the human-driven results. As mentioned in Section 2.4.2, human performance may be close to optimal, so for the automated algorithms to be near human performance is actually encouraging.

A key issue with analysing the performance of the algorithms developed in this thesis is that it is hard to see where an algorithm has made better or worse action choices simply from observing the OG and resulting action at each timestep. For example, from examining Figure 5.2, it is not easy to say where the agent could have improved the path it took. Even for humans familiar with the interactive simulation described in Section 2.4.2, the 'strategy' being pursued by an algorithm is usually not obvious, and even if it is, it is not clear if a given strategy is in fact better than some other strategy or not. This problem is known as the *credit assignment problem*, and it is one of the central challenges of reinforcement learning [Sutton, 1984; Sutton and Barto, 1998]. This makes it tricky to develop any intuition about why some algorithms do better than others.

Chapter 6

Planning Using Information Lookahead

6.1 Previous Work Using MDPs and POMDPs

For problems that can be posed as a POMDP, the POMDP policy provides the optimal way for an agent to act given the incomplete information it receives, and means the agent will choose between greedy actions and information-gathering actions so as to maximise its total expected reward. However, solving POMDPs exactly is intractable for realistic problem sizes, because for a POMDP with $|\mathscr{S}|$ states, the belief-state MDP has $|\mathscr{S}|$ continuous variables (see Section 3.2). This section reviews methods from the literature that avoid having to solve the POMDP exactly.

First, methods for finding an approximate solution to POMDPs are discussed (Section 6.1.1). These *point-based methods* use a sample of belief points to build up an approximate solution, which is guaranteed to have bounded error with respect to the optimal solution. However, point-based methods are still computationally intensive, and have only been applied to problems with $\sim 10^5$ states. For problems that are larger than that, a second possibility is to try to create a simplified version of the problem, and solve that (Section 6.1.2). These methods require the problem to have some structure that can be exploited in order to create a relaxed version of the problem which has solutions that are also good solutions to the full problem. However, the state space of the vent prospecting domain has very little structure that can be exploited in this way. A final option is the *online POMDP* approach, where instead of trying to calculate a policy that will be optimal for any belief state, the algorithm tries to find the optimal action for the current belief state only (Section 6.1.3). With online POMDP solvers, the planning has to be interleaved with execution, as the agent's actual belief state will only be known at run-time. This means that the agent has to have the processing power and time available for on-board planning, whereas with a pre-calculated policy, the agent can be relatively dumb and still act optimally in response to new observations.

6.1.1 Approximate Methods for Solving POMDPs

Methods using point-based updates are currently the most effective way of generating near-optimal POMDP solutions; that is, the most effective of the methods that produce a value function with bounded error with respect to the true POMDP value function. Point-based algorithms are much more efficient than exact solution methods such as witness (described in Section 3.2.2.3), because they only update the value function at a finite set of belief points. The underlying assumption is that the value function will be fairly smooth in belief space, so belief points that are near to each other will have similar values.

Given a set of belief points, point-based techniques maintain a solution as a set $\mathscr{V}_t$ consisting of one α-vector (and corresponding optimal action) per belief point. The solution for a one-timestep-longer horizon, $\mathscr{V}_{t+1}$, is generated by applying a belief-based update to each belief point b. This

update proceeds in three steps:

1. The best α-vector for every action and observation is found, by applying the transition function to find the new belief state $b' = \mathrm{SE}(b,a,z)$, and choosing the α-vector that maximises $V(b') = b' \cdot \alpha$.

2. The normal Bellman update is run for every action, summing over observations and using the action-observation α-vectors calculated in step (1). This creates one α-vector per action.

3. Finally the output vector (and associated action) is the one with the highest value at belief point b.

This is much more efficient than the update required in exhaustive algorithms such as witness, and can be computed in polynomial time. Finally, as for exact algorithms, incrementally increasing the horizon t allows the discounted infinite-horizon value function to be approximated arbitrarily closely.

Specific examples of point-based algorithms include *point-based value iteration* (PBVI) [Pineau *et al.*, 2003; Pineau *et al.*, 2006], *heuristic search value iteration* (HSVI) [Smith and Simmons, 2004; Smith and Simmons, 2005], and *Perseus* [Spaan and Vlassis, 2005]. These algorithms all make use of the belief-point update outlined above; the key difference between them is in how they select the set of belief points used. While an exact POMDP value function will define the optimal action for all possible belief states, the point-based algorithms focus updates on the most important belief states. All the algorithms use only belief states that are reachable from the initial belief state. Perseus finds belief points by generating random action/observation trajectories, but instead of backing up all belief points on each iteration, only backs up randomly chosen belief points until the value function has improved for all belief points (including the ones that were not explicitly backed up). PBVI uses a more complex procedure for selecting the belief points to include in the solution set, aiming to add new points that will reduce the expected error in the value function the most. It also alternates between performing value function updates, and expanding the set of belief points, in order to provide anytime performance. HSVI differs in that it maintains both a lower and upper bound on the value function, where the lower bound is defined by the set of α-vectors, and the upper bound is defined by a set of belief points. It expands belief states using a depth-first search through the action/observation space, continuing until the expected error falls below a threshold value. The nodes to expand are chosen so as to maximise the upper-bound value function for actions, and maximise the uncertainty defined by the gap between the upper and lower bounds for observations.

An interesting recent extension to point-based methods is given by [Ong *et al.*, 2010], who define the *mixed observability Markov decision process* (MOMDP), where some of the state variables are fully observable and some are partially observable. Ong *et al.* factor out the observable variables from the unobservable ones, and store MOMDP solutions as one set of α-vectors for every possible value of the observable variables. The α-vector sets are calculated in the same way as for most point-based algorithms, but the reduced set size gives a significant speed improvement for many tasks generally treated as POMDPs. This work is potentially relevant to the vent prospecting problem, as only the vent map $\mathbf{m}$ is partially observable and the other state variables are observable, but the contribution of $\mathbf{m}$ to the total state space size (2^C states) dominates that of the other variables.

While point-based methods have enabled problems with tens of thousands of states to be solved, the problem of large, continuous state spaces has not been successfully addressed.

6.1.2 Methods for Approximating POMDPs

Early methods for solving POMDPs concentrated on relaxing certain aspects of the POMDP to create a simpler problem that can be solved exactly. The first step is often to solve the 'underlying MDP', which is a problem with the same states, actions, transition function and rewards as the POMDP, but

where the state is assumed to be fully observable. The underlying MDP can be solved very quickly, for example [Wingate and Seppi, 2005] are able to solve problems with millions of states in less than a dozen seconds. The MDP solution gives $Q(s,a)$ values indicating which action is optimal for every state.

An obvious simplification is to maintain a full belief state, but choose the action that is optimal for the most likely state (the MLS policy in [Cassandra *et al.*, 1996]), in other words choose

$$\operatorname*{argmax}_a Q(\operatorname*{argmax}_s b(s), a)$$

A slightly more sophisticated variant is the Q_{MDP} algorithm [Littman *et al.*, 1995], where the Q-values are weighted by the probability of being in each state, so the action chosen is

$$\operatorname*{argmax}_a \left(\sum_s b(s) Q(s,a) \right)$$

Note that here the $Q(s,a)$ values can be thought of as $|\mathscr{A}|$ α-vectors, one for each action.

Both the MLS and Q_{MDP} policies perform very well in environments with limited uncertainty and few states [Cassandra *et al.*, 1996], with MLS often performing better, but neither approach chooses actions to gain information, so they do not cope well with environments where the informational effects of actions are important.

Several attempts have been made to alleviate the deficiencies of Q_{MDP} by trying to take account of uncertainty in the belief state. [Apostolikas and Tzafestas, 2004] build on Q_{MDP} by encouraging exploratory actions, and also include a term to keep track of visited states in the MDP formulation. [Hauskrecht, 2000] presents the *fast informed bound* (FIB) algorithm, which represents the value function by a set of $|\mathscr{A}|$ α-vectors. These α-vectors are initialised with the Q_{MDP} values, and updated using:

$$\alpha^a_{t+1}(s) = R(s,a) + \gamma \sum_z \max_{\alpha_t \in \mathscr{V}_t} \sum_{s'} P(z|s',a) P(s'|s,a) \alpha_t(s')$$

This is significantly more computationally demanding than calculating Q_{MDP} values, but produces a closer bound on the exact value function, as it takes some account of the agent's uncertainty.

Other work attempts to simplify the POMDP by exploiting structure in the problem. [Poupart and Boutilier, 2004] manage to solve a problem with 33 million states by mapping the belief state to a lower-dimensional space, and then using a belief-point update algorithm to produce a value function. [Sridharan *et al.*, 2008] separate the state space into a two-level hierarchy, and are able to solve the resulting hierarchical POMDP much more efficiently than the original POMDP.

6.1.3 Online POMDP Solvers

The methods discussed in Sections 6.1.1 and 6.1.2 are offline: they consist of an offline planning phase, where a policy specifying an action for every belief state is calculated, and an acting phase where the agent takes actions, updates its belief state, and uses the policy to select the next action. An alternative approach is to solve the POMDP online, as described in a survey paper by [Ross *et al.*, 2008], where planning and acting are interleaved. This work allows much larger POMDP problems to be addressed.

The fundamental idea behind online POMDP algorithms is to solve the POMDP for only the belief state that the agent is currently in. After taking the best action for this belief state, the agent receives an observation, updates its belief state, and then executes the online POMDP solver again to find the best action for the new belief state. The methods are closely related to point-based approaches: the algorithm simulates actions and observations to generate belief points, starting from the current belief

state, and then uses a Bellman update to find the value of predecessor belief states:

$$V(b) = \max_a \left[\rho(b,a) + \gamma \sum_z P(z|b,a) V\left(SE(b,a,z)\right) \right] \tag{6.1}$$

Actions and observations are generated to a fixed depth D, and the long-term values of belief points at the leaves of the tree are estimated using a heuristic.

Most approaches, for example the *real-time belief space search* (RTBSS) algorithm of [Paquet *et al.*, 2005; Ross *et al.*, 2008], maintain both a lower and upper bound heuristic. These bounds are kept for the leaf nodes, and propagated up the tree using Bellman backups. They enable efficient choices to be made about the best nodes to expand in order to minimise the error of the estimated value function.

Lower bounds can be found by calculating the value function for a *blind policy*, where the agent chooses the same action regardless of what belief state it is in. The upper bound can be found by solving the underlying MDP to produce Q_{MDP} values, or FIB values (see Section 6.1.2). However, these methods require a state space where the underlying MDP can be solved efficiently, which is not the case in my vent prospecting domain. In [Paquet *et al.*, 2005], the authors use RTBSS in a domain with a state space of comparable size to the vent prospecting problem; however they do not specify the heuristic used for leaf nodes.

6.2 Information Lookahead

The *information lookahead* (IL) algorithm applies online POMDP techniques (as discussed in Section 6.1.3) to the vent prospecting domain. Online POMDPs are appropriate for this domain because the state space is far too large for the full problem to be solved even approximately (using the methods described in Section 6.1.1), and there is very little structure in the domain that can be easily exploited in order to reduce the size of the problem. Indeed, using an occupancy grid has already compressed the belief space considerably, from 2^C map variables to just C variables, and any further compression is likely to lose valuable information.

The IL algorithm solves a belief MDP approximately by evaluating the value of actions N steps into the future, and relying on a heuristic function to estimate the value of actions beyond that. Once the best action has been found, the agent performs this action, updates its belief state using the observation received, and then calculates N steps of lookahead from this new belief state. Action values are calculated in a recursive way: for each action, we find the new belief states we may end up in, and then for each new belief state, find the value of each possible action. The steps of lookahead remaining, k, is initially set to N and is reduced by one on each recursive evaluation. When the recursion ends, at $k = 0$, we use a heuristic function $Q_h(b,a)$ to provide an estimate of the expected long-term value of being in state b and taking action a. IL calculates Q-values using a slightly modified form of the Bellman equation:

$$Q(b,a,k) = \begin{cases} \rho(b,a) + \gamma \sum_z P(z|b,a) \max_{a'} Q(b',a',k-1) & \text{if } k > 0 \\ Q_h(b,a) & \text{if } k = 0 \end{cases} \tag{6.2}$$

where γ is a discount factor, $0 \leq \gamma < 1$, $\rho(b,a)$ and $P(z|b,a)$ are the reward and observation functions defined in Section 4.5, and $b' = \mathrm{SE}(b,a,z)$ is calculated using Jakuba's OG update algorithm. Figure 6.1 shows graphically how Q-values are calculated by IL: the calculation starts with the leaf nodes on the far right of the diagram, using $Q_h(b,a)$ estimates. Values are then propagated to the left by taking the maximum of action values, and the expectation of observation values, using Equation 6.2. Note that there are only three possible actions and three possible observations on each step, both relatively few, so sampling trajectories through the action-observation tree instead of exhaustively enumerating them was not expected to offer enough performance benefit to outweigh the potential loss of accuracy. The algorithm is given in full in Algorithm 6.1.

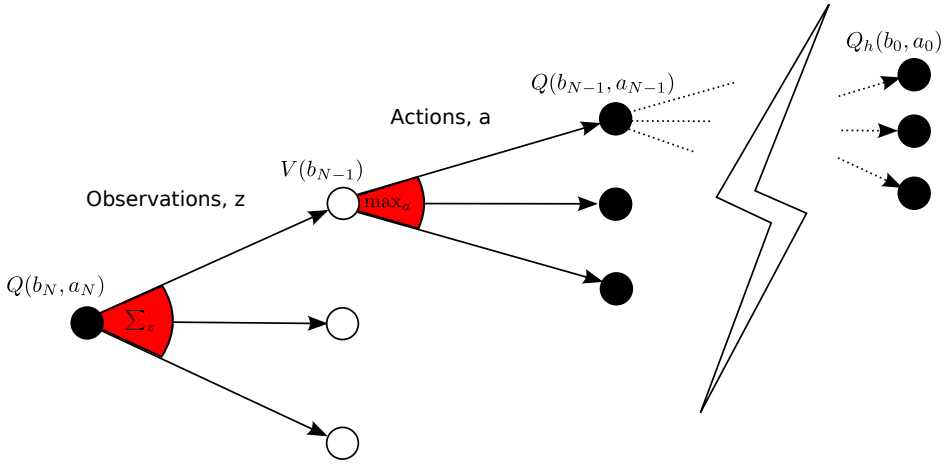


Figure 6.1: Calculation of the information-lookahead value of actions. The subscript on belief state b and actions a indicates the number of steps of lookahead remaining; when there is no lookahead remaining, the heuristic function $Q_h(b,a)$ is used to supply action values. These values are then propagated backwards by taking the maximum of action values at belief state nodes, and the weighted sum of observation values at action nodes.

Algorithm 6.1 The IL algorithm in pseudo-code. The procedure `choose-action-info-lookahead` is applied at each timestep. The argmax operation over a is performed over cells $\{forward, left, right\}$ from c_{AUV} given c_{prev} as the previous cell. $P(z|b,a)$ is calculated using the formulae given in Section 4.5.4. Also note that for efficiency reasons the procedure value-approx returns an approximation to the belief state value, rather than the $Q(b,a)$ action value, so the code differs slightly from the format implied by Equation 6.2.

```
Procedure choose-action-info-lookahead(b, c_prev, c_AUV, γ, N): cell
  Return argmax_a {q-value(b, c_AUV, γ, N, a)}
Endproc

Procedure q-value(b, c_curr, γ, k, a): qValue
  For each observation z
    <calculate P(z|b,a)>
    Set b' = SE(b,a,z)
    If k = 1
      Set futureVal(z) = value-approx(b', c_curr, γ, a)
    Else
      Set futureVal(z) = max_a' {q-value(b', a, γ, k − 1, a')}
    Endif
  Endfor
  Return ρ(b,a) + γ Σ_z P(z|b,a)·futureVal(z)
Endproc
```

Algorithm 6.2 The approximation function for the basic heuristic, defining the `value-approx` function called in Algorithm 6.1. Note a' is maximised over cells $\{forward, left, right\}$ from a given c_{curr} as the previous cell.

```
Procedure value-approx(b,c_curr,γ,a): value
  Return max_a' ρ(b,a')
Endproc
```

The heuristic function $Q_h(b,a)$ should approximate the expected future reward following on from action nodes at the leaves of the belief state/action tree. If the heuristic provided a perfect estimate of this future reward, there would be no need to perform N steps of lookahead – the heuristic itself could just be used. Simulating and evaluating future states is guaranteed to reduce the error in the estimate of Q-values compared to the true value function [Ross *et al.*, 2008], but IL is exponential in the number of steps of lookahead. The limited lookahead makes Q, the value of an action a given a belief state b, also a function of k, the number of steps left to take – the same belief state may be worth more if the agent can take more steps following it. As can be seen from Figure 6.1, the algorithm has a branching factor of nine: for each extra step taken, nine times as many Q-values must be evaluated. For example, with $N = 4$ the $Q_h(b,a)$ function must be called approximately 6500 times. This places two key requirements on the heuristic function: it should execute quickly, as it must be called many times, and it should approximate the true value function closely, so that good values can be found using small values of N.

A major issue with using online POMDP methods in the vent prospecting domain is that the underlying MDP is too large to be solved (as outlined in Section 4.8). This meant I had to adopt alternative approaches for $Q_h(b,a)$ heuristics. A simple heuristic is described in Section 6.3 below, and a more advanced heuristic is presented in Section 6.4.

6.3 Basic Heuristic

A basic heuristic is to simply use the reward gained at the agent's final location as the estimate of the long-term reward, so Q_h is given by

$$\begin{aligned} Q_h(b,a) &= \rho(b,a) \\ &= \begin{cases} P(m_a)R_{vent} & \text{if } a \text{ has not been visited} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{6.3}$$

This is certain to be an underestimate of the true expected reward, but has some intuitive justification in that cells with high probability are likely to be surrounded by other cells with high probability. In the remainder of this thesis, information lookahead with the basic heuristic will simply be referred to as IL, or ILN where N indicates the number of steps of lookahead. The heuristic function is shown in Algorithm 6.2, and results for ILN are presented in Section 6.5.

The exponential complexity in N limits the lookahead that can be practically used; most experiments have been performed with $N \leq 4$, and Figure 6.2 shows the run-time for a complete trial on a 20×20 grid for different values of N. Note that the algorithm is myopic – it takes no account of OG values more than $N+1$ steps away from the agent's starting cell. If the number of timesteps in a mission (L) was equal to N, then the ILN algorithm would be optimal. As $L \gg N$, the underlying problem can be thought of as infinite horizon, so we are conceptually approximating an infinite horizon problem with a discounted finite horizon problem.

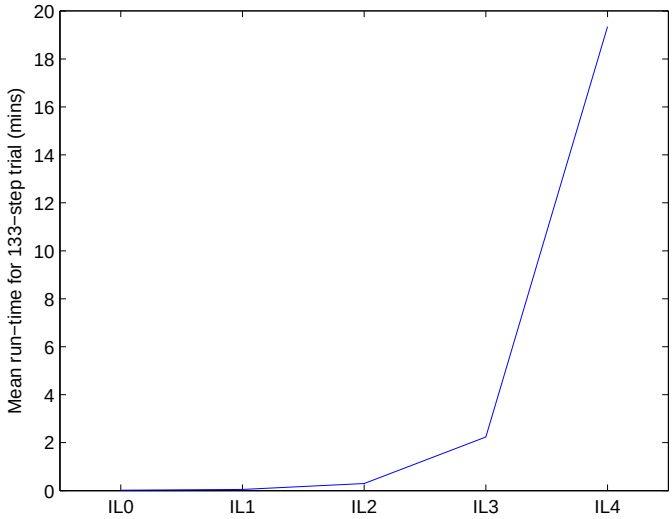


Figure 6.2: Run-time of the IL algorithm for increasing values of N, the number of steps of lookahead performed.

6.4 CE Heuristic

A more sophisticated heuristic I developed is the *certainty equivalent* (CE) heuristic, which finds a value for each cell in the grid by assuming that the agent's current belief state is the true state of the world. In other words, it assumes that future observations will not improve the agent's estimate of vent locations. This differs from the underlying MDP (commonly used as a heuristic for online POMDP solvers) because instead of dropping the observation function from the POMDP, we have dropped the observation function from the belief-MDP. Thus the reward obtained by visiting each cell will be $P(m_c)R_{vent}$, instead of 0 or R_{vent}, as it will be in the underlying MDP. To make the problem tractable, we further drop the list of already-found vents ($\mathbf{v}$) from the state space. This collapses the problem to an MDP where the state s is just the cell the agent is in (as the OG is fixed), and this MDP can be solved by value iteration much faster than the IL version of the problem. For a belief state b_0, the value update is given by

$$V(c) = \max_a \{R(b_0,a) + \gamma V(a)\} \tag{6.4}$$

where a maximised over the set of cells adjacent to c. $R(b_0,a)$ is given by

$$R(b_0,a) = \begin{cases} P_{b_0}(m_a)R_{vent} & \text{if } a \notin \mathbf{v}_0 \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

where $P_{b_0}(m_a)$ is the occupancy probability of cell a in belief state b_0, and $\mathbf{v}_0$ is the list of found vents in b_0.

Equation 6.4 is slightly simpler than the standard dynamic programming algorithms, where the value update is given by $V(c) = \max_a \{R(b_0,a) + \gamma \sum_{c' \in 1:C} P(c'|c,a)V(c')\}$, because we have a deterministic transition function so do not have to sum over future states c'. Also, note that we are solving the infinite-horizon version of the MDP, so we are assuming that the agent can take an infinite number of steps and relying on the discount factor $\gamma < 1$ to produce finite Q-values. The CE heuristic is shown in Algorithm 6.3.

CE is not optimal mainly because it takes no account of potential information gain, but also because it allows the agent to be rewarded multiple times from visiting the same cell. With the IL

Algorithm 6.3 The approximation function for the CE heuristic, defining the `value-approx` function called in Algorithm 6.1. The main body performs value iteration based on standard dynamic programming, and ε specifies the convergence tolerance.

```
Procedure value-approx(b,c_curr,γ,c_next): value
  Set ε = 0.0001
  Set P(m_c) = 0 for all c ∈ v
  Loop
    For c = 1 : C
      For a ∈ A
        Q(c,a) = P(m_a)R_vent + γV(a)
      Endfor
      V(c) = max_a Q(c,a)
    Endfor
  Until ΔV < ε
  Return V(c_next)
Endproc
```

algorithm, visited cells are known to either be empty or contain a vent, so either $P(m_c) = 0$, or c is added to the list of known vents $\mathbf{v}$ for which the re-visitation reward is zero. With the CE heuristic, the reward function is approximated by $P(m_c)R_{vent}$ for all cells c. Before running the $Q_h(b_0, a_0)$ function, we set $P(m_c) = 0$ for all vents that have been found either during the mission to date, or during the current branch of the IL action-observation tree, so the reward will be correct for all these cells. However, the dynamic programming in Algorithm 6.3 implicitly assumes that the reward for all cells that are unvisited in b_0 can be claimed multiple times. The underlying issue here is that the true reward function from Section 4.5 is non-Markovian with respect to the reduced state space consisting of just c_{AUV}.

Fixing this problem is not straightforward; keeping track of the path of the agent does not result in any extra computational complexity in IL, because paths are explicitly evaluated and the belief state is updated after each step. However, with the CE heuristic, adding a list of previously visited cells to the state would increase the state space size by a factor of 2^C (as we would have to be able to represent any of the C cells as having been visited), which would make the problem intractable. An alternative would be to make the OG itself part of the MDP's state space, and set $P(m_c)$ to zero for cells when they are visited (in a similar way that $\mathbf{v}$ is dropped from the POMDP state in Section 4.4), but this has an even worse impact on the state space size as it adds C continuous variables. The consequences of allowing rewards for re-visiting cells are discussed further in Section 6.5.

A final point about the CE heuristic not keeping track of the agent's path is that it allows immediate backtracking, as there is no way of ruling out actions that return to the previous cell. This is not thought to cause any problems however, because the immediate backtracking action is only excluded from the action set because it will never be optimal.

The IL algorithm with the CE heuristic is referred to in the remainder of this thesis as IL-CE or ILN-CE, and results using IL-CE are presented in Section 6.5. IL-CE can be thought of as the IL analogue of the $\Sigma\Delta H$-MDP algorithm, as both rely on a heuristic value and an MDP solver; however in $\Sigma\Delta H$-MDP the MDP is solved only once, after the $\Sigma\Delta H$ values have been calculated, whereas in IL-CE the MDP is solved many times (for every leaf node), and the final action values are calculated afterwards based on the MDP solution.

Given we have a procedure to find CE values for every cell in the grid (Algorithm 6.3), a *CE*

Algorithm 6.4 The CE value iteration algorithm in pseudo-code. The procedure `choose-action-CE` is applied at each timestep, and the `value-approx` function called is the one given in Algorithm 6.3. Note a is maximised over cells $\{forward, left, right\}$ from c_{AUV} given c_{prev} as the previous cell.

```
Procedure choose-action-CE(b, c_prev, c_AUV, γ): cell
  Return argmax_a {ρ(b,a) + γ·value-approx(b, c_curr, γ, a)}
Endproc
```

algorithm that uses these values directly to find the best action can be defined. The CE dynamic programming provides $V(c)$, so to find the best action we must perform one step of lookahead, to find

$$\underset{a}{\text{argmax}}\{\rho(b,a)+\gamma V(a)\} \tag{6.6}$$

where a is maximised over the set of cells adjacent to c_{AUV}. This algorithm is equivalent to IL0-CE, but is classed as a separate algorithm because, unlike IL, it does not perform any simulation of future belief states. The algorithm is shown as Algorithm 6.4, and again results are given in Section 6.5.

6.5 Results and Discussion

6.5.1 General

Experiments with IL4-CE, IL1 and IL4 were conducted using the experimental setup described in Section 5.2, with a discount factor of $\gamma = 0.9$ used for both the lookahead and the CE dynamic programming, and $R_{vent} = 1$. Results for these algorithms together with MTL, chemotaxis, $\Sigma\Delta H$, and $\Sigma\Delta H$-MDP for comparison are shown in Figure 6.3. The most surprising result was that IL4-CE performs extremely badly, worse than even an exhaustive search (MTL), and takes over four times as long to run as IL4. IL4 was the best algorithm, outperforming $\Sigma\Delta H$-MDP (although not at a statistical confidence level of 0.05). While IL4 was over three times slower than $\Sigma\Delta H$-MDP, even for smaller lookahead values IL was remarkably effective, with IL3 being statistically equivalent to $\Sigma\Delta H$-MDP but running more than twice as fast, and IL1 finding 97% as many vents as $\Sigma\Delta H$-MDP but running two orders of magnitude faster.

The effect of increasing lookahead (N) in the IL algorithm is shown in Figure 6.4. Apart from a dip with IL2, which is well within the margins of experimental error, this shows a consistent improvement in algorithm performance as N is increased (but at the cost of exponential increase in run-time).

6.5.2 IL-CE Issue

The CE heuristic actually hurts information lookahead to such an extent that not only is IL-CE much worse than IL, it is much worse than CE without any lookahead. This surprising result is due to the fact that the IL-CE algorithm actively avoids visiting very high probability cells, as shown in Figure 6.5. This behaviour arises because when CE values are computed, a reward is given when a state is revisited multiple times, as described in Section 6.4. Therefore high-probability cells can acquire values significantly higher than R_{vent}, which is set to one, because the CE value function defines a policy where the agent simply moves back-and-forth between the high-probability cell and one of its neighbours (see Figure 6.5 for example CE values). If the agent plans to visit such a high-probability cell during its N steps of lookahead, the maximum reward it can receive for that cell is $R_{vent} = 1$, and re-visiting the cell will have no reward. So instead it avoids high probability states, thinking it can get more value for visiting them later, during the CE phase of planning. It is similar

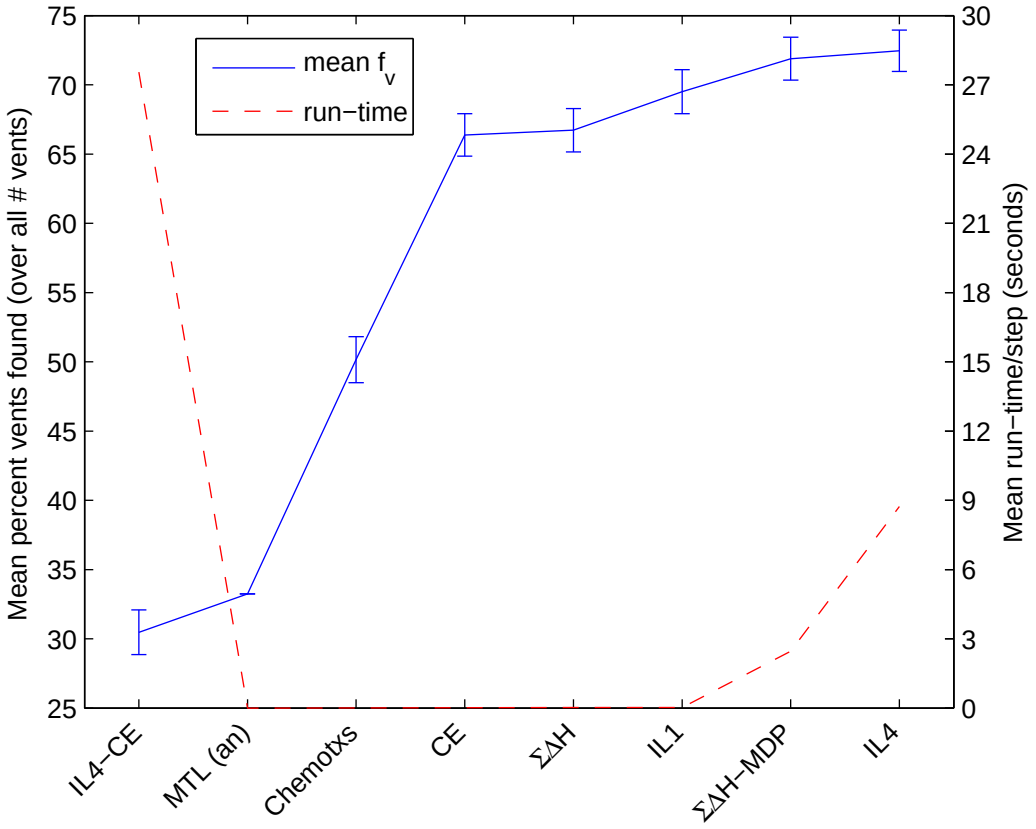


Figure 6.3: Results of IL1, IL4, and IL4-CE, with MTL, Chemotaxis, $\Sigma\Delta H$, and $\Sigma\Delta H$-MDP shown for comparison. The average time in seconds taken to compute each timestep is shown as a dashed line (against the right-hand y-axis). All results are means of 600 trials – 150 each with 3,4,5, and 6 vents. 95% confidence interval bars are also plotted. Note that the MTL result was calculated analytically, so has zero-height confidence bars, and that the lines connecting data points are purely to aid visualisation.

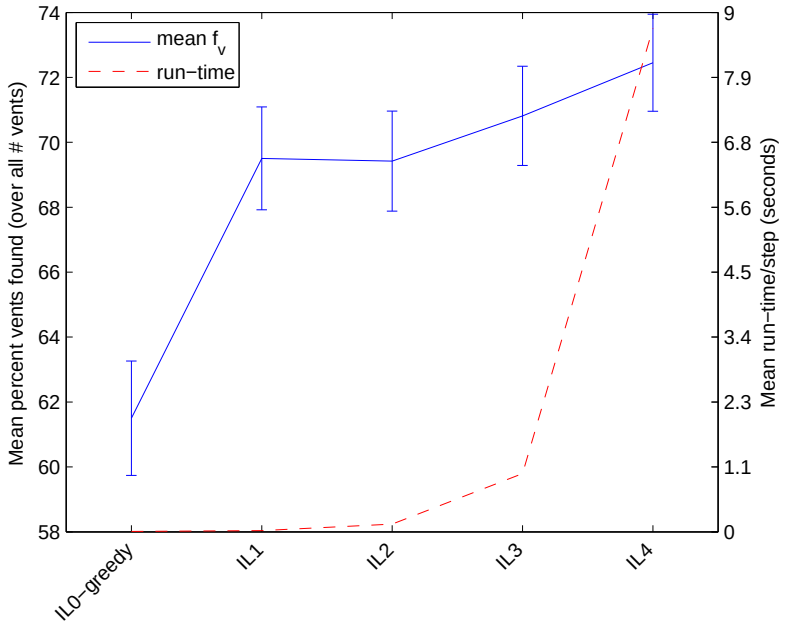


Figure 6.4: Results of IL0 to IL4, showing the effect of the number of steps of lookahead. The average time in seconds taken to compute each timestep is shown as a dashed line (against the right-hand y-axis). All results are means of 600 trials – 150 each with 3,4,5, and 6 vents. 95% confidence interval bars are also plotted.

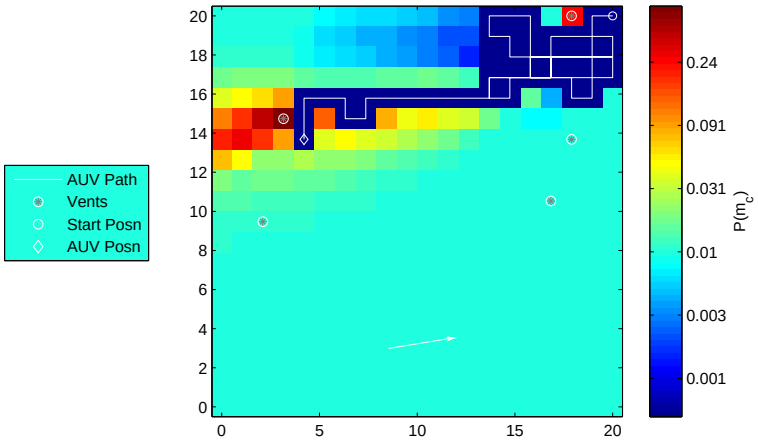
to a person always saving a tasty treat for tomorrow, and therefore never actually eating it. Chapter 7 shows how the problem with revisiting states can be prevented by using an orienteering problem formulation.

6.5.3 Human-directed CPT Results

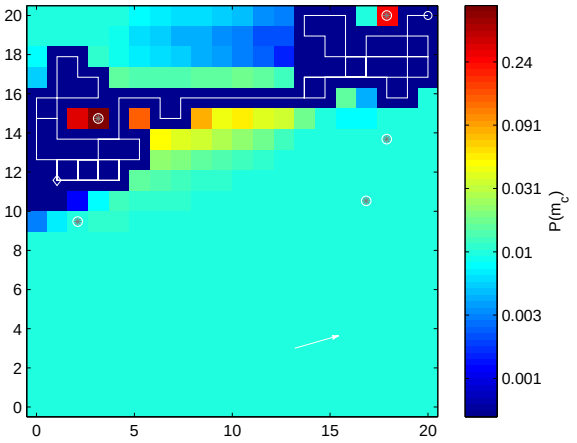
Figure 6.6 compares results for IL4 with results from the human-controlled simulation experiments described in Section 2.4.2, for both the standard experimental conditions described in Section 5.2, and the 40×40 grid setup described in Section 2.4.2.

The average results over all numbers of vents were that with the 40×40 setup IL4 found 81% of the vents and the humans found 76%, whereas with the 20×20 setup humans did better, finding 74% of the vents compared to 73% for IL4. This reversal hopefully indicates a lack of bias in the 20×20 experiments, as they were executed solely by the author, whose novel algorithms would have looked better had he performed badly in the human-controlled simulation. The 40×40 experiments were conducted by members of the Intelligent Robotics Lab, and the trials executed by the author were deliberately excluded from the 40×40 results. Note that far more 20×20 trials were completed than 40×40 (100 human trials and 600 IL4 trials, compared to 38 human trials and 29 IL4 trials for the 40×40 setup), which means we cannot have the same confidence in the 40×40 results.

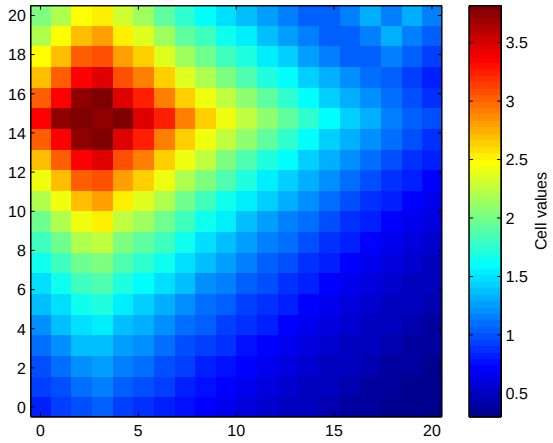
The results show that the 40×40 problem was actually easier than the 20×20 problem, as both IL4 and humans found a higher percentage of vents using the 40×40 setup. This is thought to be mostly because the mean vent density was much lower in the 40×40 setup, being 5 vents over 1600 cells (0.3125%) compared to 4.5 vents over 400 cells (1.125%). A further reason was that the 560 timesteps allowed a slightly higher proportion of the grid to be covered in the 40×40 setup, 35% as opposed to 33.25%. This may also explain why performance decreases sharply as the number of



(a) Plot of OG and agent path at t=70.

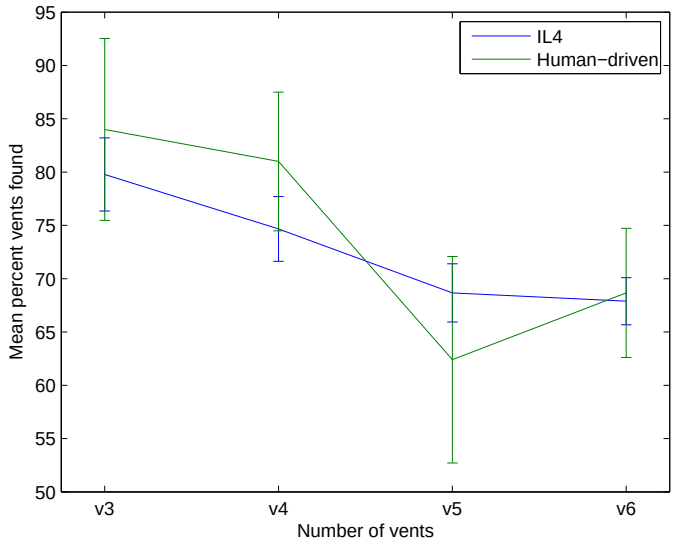


(b) Plot of OG and agent path at t=133.

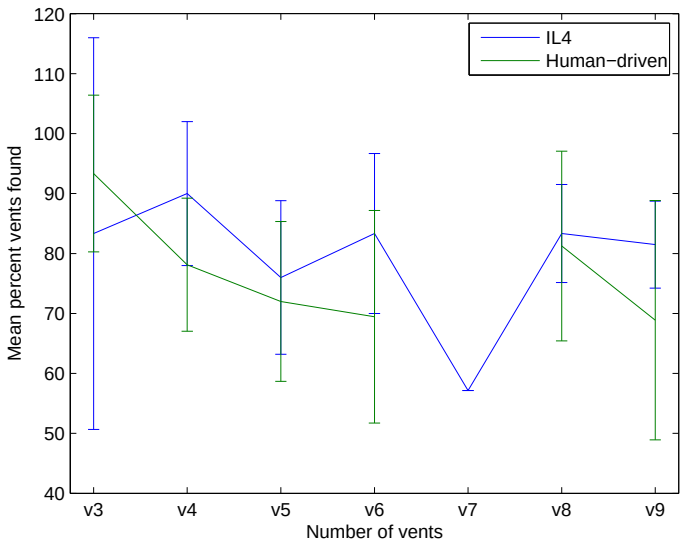


(c) CE values found by value iteration for the OG in Subfigure b.

Figure 6.5: Illustration of the reward problem with the IL-CE algorithm, showing a trial where the vent-avoiding behaviour is particularly obvious. In Subfigures a and b, the colour of each cell indicates its occupancy probability $P(m_c)$, the white arrow near the bottom of the grid represents the direction of the current, and the other symbols are as shown in the legend to the left of Subfigure a. For Subfigure c, cell colours indicate their MDP values. In Subfigure a, the agent has already avoided a vent near the top-right corner of the grid, and is very close to a second vent. But by the end of the mission, shown in Subfigure b, it has failed to visit the second vent, despite repeated visits to many of the surrounding cells. Subfigure c shows the CE cell values corresponding to the OG in Subfigure b, demonstrating that the agent will get more reward for the vent cell if it delays visiting it until the CE phase, after the N steps of lookahead.



(a) Results of the 20×20 grid experiments, showing percent of vents found against number of vents for both IL4 and human-driven vent-prospecting.



(b) Results of the 40×40 grid experiments, showing percent of vents found against number of vents for both IL4 and human-driven vent-prospecting. Note for 7 vents there were no human trials, and only one IL4 trial, which is why the confidence bar has zero height for that datapoint.

Figure 6.6: Results from the human-controlled vent prospecting experiments, compared with results for the IL4 algorithm (using an identical setup). Subfigure a shows results using a 20×20 grid, with 150 trials for each vent-count for the IL4 algorithm and 25 trials for each vent-count for the (single) human subject. Subfigure b shows results with a 40×40 grid, where the data were calculated from 38 human trials and 29 IL4 trials. In all cases the agent's starting position was fixed and the vent locations selected randomly, and in all cases except IL4 on the 20×20 grid the number of vents was chosen randomly. 95% confidence interval bars are also plotted.

vents increases using the 20×20 conditions (as shown in Figure 6.6a), but for the 40×40 setup the percent of vents found is almost constant as number of vents increases (Figure 6.6b). Both setups suggest humans do better when there are fewer vents, especially when there are just 3 vents, and the advantages of IL4 show up more when the vent density is higher.

Chapter 7

Using Orienteering Methods as a Correction

This chapter explains how an orienteering problem (a form of the travelling salesman problem) solver is used to address the multiple rewards issue with the IL-CE algorithm. The correction is also applied to the $\Sigma\Delta H$ algorithm, with better results than obtained for the IL algorithm.

7.1 IL-CE Issue and OP Solution

In Chapter 6, we saw that the IL algorithm performs worse with the certainty equivalent heuristic than with a basic myopic heuristic. The problem is that the MDP formulation allows the agent to gain reward from re-visiting cells, which leads to MDP values for some cells becoming much larger than the maximum one-step reward. When this happens, the optimal plan for the IL lookahead (which was performed for four steps) is to avoid these high-reward cells, so that they can be visited repeatedly once the agent enters the MDP phase. The resultant behaviour is that the agent never visits high-value cells because it re-plans on every step, always planning to visit these high value cells in four steps time.

The obvious way to avoid this problem would be to encode a list of visited cells in the state space of the MDP presented in Section 6.4, so the revised state would consist of the agent's current cell (C possible values) plus a boolean vector indicating, for every cell, whether it has been previously visited or not (2^C possible values). Then the MDP transition function would not only change the agent's current cell, but also mark the previous cell as visited. The true reward function would be Markovian with respect to this expanded state, re-visiting a cell would have zero reward, and the problem would be avoided. However, adding the list of visited cells would increase the state space size from C to $C \cdot 2^C$ possible states, meaning $\sim 10^{123}$ states given our experimental settings. While algorithms such as LAO* [Hansen and Zilberstein, 2001] and RTDP [Barto *et al.*, 1995] are able to solve very large MDPs, 10^{123} states is some way from being possible. Further, such algorithms rely on a heuristic to estimate the distance to the goal. As we define a reward for every cell in the grid, there are no obvious heuristics to estimate how close a given path through the grid is to the globally optimal path.

An alternative way of solving the re-rewarding problem is to use forward search over the action space of the problem, i.e. Equation 6.4 can be solved recursively, provided we keep track of visited cells during the recursion. Computationally this is a much less demanding problem than forward search in the action-belief space (the IL algorithm), because the occupancy grid does not have to be updated on each step (plus the branching factor is lower as only actions, which are deterministic, need be considered).

This revised problem is in fact identical to the *orienteering problem* (OP) [Tsiligirides, 1984; Golden *et al.*, 1987], a variant of the travelling salesman problem where cities are assigned a score. The agent's aim is to choose a subset of cities to visit together with a path between these cities so as to maximise the total score, while respecting an upper limit on the distance it can travel. In our OG setting, each cell represents a city, and the distance between all connected cities is the same (as cities are only connected to adjacent cities one grid cell north, east, south and west). The mission duration sets a limit on the number of steps that we wish to maximise our score over.

7.2 Previous Work on the Orienteering Problem and Self-Avoiding Walks

This section describes previous work on firstly the OP, and secondly the *self-avoiding walk* (SAW). SAWs are interesting because when solving an OP in a grid setting such as found in my domain, it is useful to generate a path that does not cross itself. The development of my OP algorithm is explained more fully in Section 7.3 below.

The OP is derived from the *travelling salesman problem* (TSP) first discussed in Section 4.8, which is a classic example of an NP-complete problem [Garey and Johnson, 1979]. In the standard TSP, the aim is to find the shortest possible circuit that visits all of the cities in a set, where the pair-wise distances between all cities are supplied. The *prize-collecting travelling salesman problem* (PCTSP, [Balas, 1989]) is a variant where each city has a reward associated with it, but not all cities need to be visited. Skipping a city incurs a penalty however, and the objective is to minimise the sum of distance travelled plus penalties, while collecting at least a minimum amount of reward. Finally, the OP has rewards for cities visited, but no penalties for missing cities, and the path does not have to end at the start point. The objective is to gain the maximum amount of reward possible, given an upper limit on the total distance that can be travelled. [Feillet *et al.*, 2005] present a review of prize-collecting variations of the TSP and the algorithms that have been applied to these problems.

Solving the OP exactly is still NP-hard [Golden *et al.*, 1987], so previous work has focused on approximate methods. [Tsiligirides, 1984] presents several early algorithms for the OP, as well as a canonical set of problems that have been used for evaluating other OP algorithms. The input for the OP is a list of rewards or scores $S(j)$ for each city j, and a list of pairwise distances between cities, $C(i,j)$ being the distance between i and j. The cities are assumed to be fully connected to each other, and the distances in C are just straight-line distances, so C can be specified by giving the Cartesian coordinates of all cities. The first of Tsiligirides' algorithms is the *deterministic algorithm*, where paths are generated within the arc of a circle that is gradually expanded, and the second is a Monte-Carlo algorithm, which proves to be the better of the two. The Monte-Carlo approach samples a set of 3000 randomly-generated paths, and was found to produce near-optimal results. The path is built up incrementally by assigning a 'desirability' $A(j)$ to potential nodes j to visit following node $LAST$:

$$A(j) = \left\{ \frac{S(j)}{C(LAST, j)} \right\}^4 \tag{7.1}$$

A shortlist of four potential next nodes j is created, and j is selected with probability

$$P(j) = \frac{A(j)}{\sum_{q=1}^{4} A(q)} \tag{7.2}$$

Better solutions can be found by taking the output from either the Monte-Carlo or deterministic algorithms and using them to seed a route improvement stage. To improve a route, firstly swaps are attempted between a pair of nodes on the route. Then if one of these swaps results in decreased total distance travelled, attempts are made to add a new node into the route.

Many of the TSP algorithms found in the literature also rely on route improvement by swapping two cities, the so-called 2-opt heuristic [Lin, 1965]. As the OP does not have to include every city in

the path, OP solutions also often include heuristics that swap cities on the path with 'unused' cities [Golden *et al.*, 1987; Chao *et al.*, 1996]. Golden *et al.* present a typical algorithm, where an initial path is generated by adding nodes greedily, then a route improvement step is performed, which first applies 2-opt heuristics, and next attempts to add new nodes to the route. The final stage of the algorithm is a step where the centre-of-gravity g of the nodes is found by weighting their coordinates according to their score, and then a new path is created by adding nodes greedily in order of their weighted distance from g, $S(j)/C(g,j)$. This is repeated until the route ceases to change between iterations. Chao *et al.* also present a multi-step approach that starts with a greedy initial path, but improve on the results of Golden *et al.* by maintaining a set of paths that cover all the nodes, and using a carefully-selected series of improvement steps that swap nodes between or within paths.

An approximation algorithm for the OP with a guarantee of obtaining reward within a constant factor of the optimal reward is presented by [Blum *et al.*, 2007]. If the optimal reward is Π^*, they guarantee a reward of Π^*/α, where their implementation has $\alpha = 4$. The Blum *et al.* approach relies on an algorithm by [Chaudhuri *et al.*, 2003] for finding a minimum cost tree containing nodes s and t, which is used to find a constant factor approximation to the MIN-EXCESS-PATH problem. The MIN-EXCESS-PATH is the path from s to t that gains at least reward k and has the minimum 'excess' distance over the shortest path from s to t. The MIN-EXCESS-PATH is found by splitting a path into segments of two types: those that can be solved optimally by incrementally adding further away nodes, and those that can be approximated using the Chaudhuri *et al.* algorithm. Given this MIN-EXCESS-PATH procedure, the OP is solved by first guessing the optimal reward k. Then the MIN-EXCESS-PATH collecting k from the start node to every other node is found, and at least one of these paths will satisfy the maximum distance requirement of the OP. This process is then repeated with a new guess of k, using a binary search, until the best solution is found. This is a very interesting approach, but relies on a reward function that can be discretised to return integers, and also involves considerable computational overhead for a fairly loose approximation guarantee.

My approach to solving the OP is based on Tsiligirides' Monte-Carlo algorithm, which requires the generation of a self-avoiding walk [Madras and Slade, 1996; Hayes, 1998] through the grid cells (see Section 7.4 for details of the implementation). A SAW is a path connecting vertices on a regular (hyper-)cubic lattice in D-dimensions, such that vertices are only connected to their nearest neighbours, and the same vertex is never visited twice. SAWs have been extensively studied by the chemical physics and mathematics communities because they provide an extremely good model of linear polymer molecules, and are used to investigate statistical properties of such molecules, which can have chain lengths of up to 10^5. However, generating a SAW is itself a complex problem, with many algorithms being exponential in the length of the walk [Sokal, 1995].

Monte-Carlo methods are commonly used for generating SAWs, and some of the key Monte-Carlo algorithms (as described in [Sokal, 1995]) are outlined next. Monte-Carlo SAW algorithms can be either static, which produce statistically independent sample paths, or dynamic, which create a series of correlated paths using a Markov process. The most basic static method is *simple sampling*, where samples are created one node at a time with uniform probability of choosing any of the neighbouring nodes at each step, and then all samples that cross themselves are discarded. The problem with this is that the number of crossing samples grows exponentially with path length, approximately according to $e^{0.42N_{OP}}$ in two dimensions, where N_{OP} is the path length. An improvement is *r-step stride sampling*, where all possible SAWs of length r are used as building blocks for longer paths. Generally r is ≤ 10, and this algorithm is much more efficient but still exponential in N_{OP}. *Dimerisation* is a divide-and-conquer approach where an N_{OP}-step SAW is generated by creating two $(N_{OP}/2)$-step SAWs and concatenating them. If the result is a SAW, it is accepted, and otherwise the process is begun again. The $(N_{OP}/2)$-step SAWs are similarly split into two, until a small N_{OP} value is reached and a more basic method is used. This algorithm is almost polynomial in N_{OP}. Finally, dynamic methods can be polynomial in N_{OP}; for example the *pivot algorithm* generates a sequence of SAWs by either

reflecting or rotating the section of a path following a randomly-chosen pivot point. If the new path is self-avoiding it is accepted, and otherwise a different pivot point is chosen.

7.3 IL-OP Algorithm

This section describes the IL-OP algorithm, which replaces the CE heuristic in IL-CE with an OP solver. As with IL-CE, the OP heuristic assumes that the OG given by the belief state b_0 at the leaf node is fixed and will not change in the future. The instance of the OP is defined as follows:

- The maximum length of the path is N_{OP}.
- Each cell in the OG defines a city, with reward $P(m_c)R_{vent}$ where $P(m_c)$ is given by the OG in b_0, or reward zero if $\mathbf{v}(c) = true$ in b_0. As in Section 6.4, this reward model assumes that the agent will receive no further observations after the current timestep, so it must act on the basis that the occupancy probabilities in b_0 represent the true probabilities of finding vents in the corresponding cells.
- Each city is connected only to the four cities to the north, east, south and west of it. The distance between all pairs of connected cities is one unit.
- The start cell is fixed, and the first cell in the path cannot be the cell the agent occupied in the previous timestep.
- The path may terminate at any cell.

Thus the IL-OP algorithm is to perform information lookahead in the action-observation space for N steps, and then solve an N_{OP}-step OP for every leaf node, using the belief state from that node as input. The framework for the IL-OP algorithm is still Algorithm 6.1, but the procedure `value-approx` implements an OP solver, as discussed later in this section. Applying the OP heuristic in place of the CE MDP should solve the IL-CE problem whereby it avoids high-value cells, as now re-visiting cells has no value in either the lookahead phase or the leaf-node heuristic phase of planning. The OP solution is basically solving an N_{OP}-horizon MDP instead of the infinite-horizon MDP solved for the CE solution.

Ideally, if there are L_{rem} steps remaining in the mission, the OP should be solved for $N_{OP} = L_{rem} - N$ steps; for example, at the start of a 133-step mission with IL performed for $N = 4$ steps, we should solve a 129-step OP. However, our OP implementation is exponential in the number of steps, which limits the OP path length we can use despite the fact that OP is much faster than IL. Given that there is a branching factor of nine for the action-observation lookahead (three non-backtracking actions and three observations), for four steps of IL lookahead the OP must be solved about 6500 times on every timestep of the simulation. Section 7.6 discusses results of the ILN-OPN_{OP} algorithm for $N \in \{2,4\}$ and $N_{OP} \in \{2,4,6,8,10,12,14\}$.

7.4 OP Implementation

We now consider the implementation of the OP solver. The key feature of our OP problem as compared to those in the literature is that we do not have full connectivity of the graph, that is we do not have all nodes connected to all other nodes. This means many of the heuristic methods used for solving OPs (as summarised in Section 7.2) cannot be applied, as they rely on the ability to replace an arbitrary city in the candidate path with some other arbitrary city. Instead we use a Monte-Carlo (MC) approach similar to [Tsiligirides, 1984], in which a set of sample paths is generated, and the best path is chosen from these samples. However even with this approach the limited connectivity of the graph

causes problems, as we must prevent the path from crossing itself, so that we avoid counting the same cell twice in the path value. Note that the optimal path will always be a self-avoiding walk, because for any path that does cross itself, there is a non-crossing path visiting the same cells plus one extra cell that must be higher-value than the crossing path. We generate SAWs by incrementally adding randomly-chosen actions to the path, provided at least one of the possible actions is not already part of the path, or starting again if the path has reached a 'dead end'.

Overall this Monte-Carlo algorithm is very similar to Tsiligirides' algorithm, but differs in several ways: firstly we can consider *all* possible next nodes, as there will be at most three candidates. Secondly, if there are no candidate nodes (the path end-point is surrounded by cells it has already visited), we abandon that path and start with a new one. In fact this *inversely restricted sampling* method introduces a bias compared to simple sampling [Sokal, 1995], but this does not matter for our purposes as we do not rely on any statistical properties of the sample of paths generated. Inversely restricted sampling is much more efficient than simple sampling methods, but is still exponential in the walk length N_{OP}. The next difference to Tsiligirides' method is that we weight each node evenly, rather than according to a desirability criterion as in Equation 7.2. This is because for large N_{OP} (greater than about 45), most of the computation time for the OP solver is spent generating paths that have to be discarded. Figure 7.1 shows the run-time of the MC algorithm for increasing path length, together with the number of paths that have to be discarded to generate 3000 valid paths. For small N_{OP}, the run-time looks almost linear, but as N_{OP} increases far more crossing paths are generated, and the exponential nature of the algorithm begins to show. Therefore we expect that weighting some actions more than others will result in significantly increased run-time, as paths will vary less and be more likely to become 'trapped', so any benefit in choosing better paths will be outweighed by the longer running times. Finally we note that, in line with the IL and CE algorithms, we apply a discount factor γ so that the n^{th} step in the SAW is discounted by $\gamma^{(n-1)}$.

To confirm that the Monte-Carlo algorithm produces acceptable approximations to the true optimal path value, I implemented an exhaustive enumeration procedure that finds the optimal path. Figure 7.2a shows the rewards for the two OP solvers plotted against path length up to values of $N_{OP} = 18$, and for path lengths up to $N_{OP} \approx 9$ the MC procedure produces results that are indistinguishable from optimal. Figure 7.2b plots the reward found by the Monte-Carlo algorithm as a percentage of the optimal reward, for MC algorithm variants where 1500, 3000 and 6000 sample paths are generated. The graph indicates that, after a threshold N_{OP} value, the Monte-Carlo reward starts to tail off fairly linearly as N_{OP} increases, and sampling more paths alters the threshold but does not significantly change the gradient of decline. Finally, Figure 7.2c shows the run-times of the two algorithms against N_{OP}, and demonstrates clearly why an approximate solver is needed. However, for very short walks ($N_{OP} \leqslant 8$), the exhaustive enumeration method is actually faster than the Monte-Carlo method, so the optimal procedure was used for walks of these lengths. For $9 \leq N_{OP} \leqslant 14$, the MC method with 1500 path samples was used, and for $N_{OP} \geq 15$ MC with 3000 path samples was used.

The leaf-node heuristic for the IL-OP algorithm is shown in Algorithm 7.1 (where the parent function is given by Algorithm 6.1), and the two OP solvers (Monte-Carlo and optimal) are shown in Algorithm 7.2.

As was the case for the CE heuristic, we can also use the OP heuristic by itself to create the *OP algorithm*, where no information lookahead is performed. This pure-OP algorithm simply selects the path that maximises $\sum_{c \in path} P(m_c) R_{vent}$, where $P(m_c)$ is defined to be zero for all previously-visited cells, and is presented as Algorithm 7.3. Results for this OPN_{OP} algorithm, as well as for ILN-OPN_{OP}, are presented in Section 7.6.

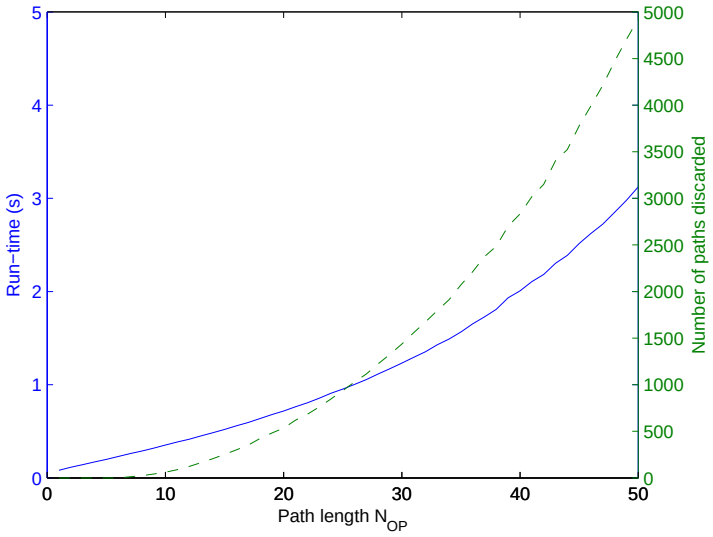
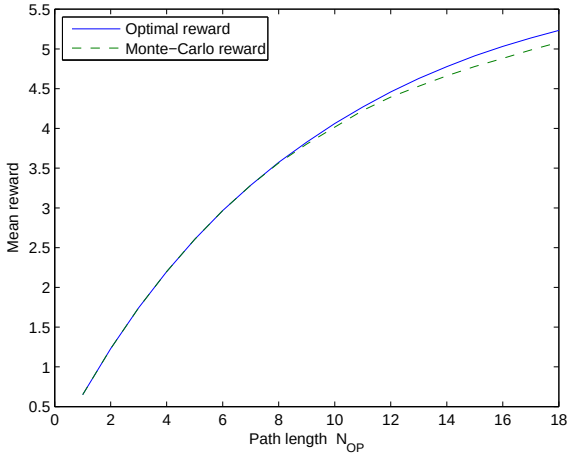


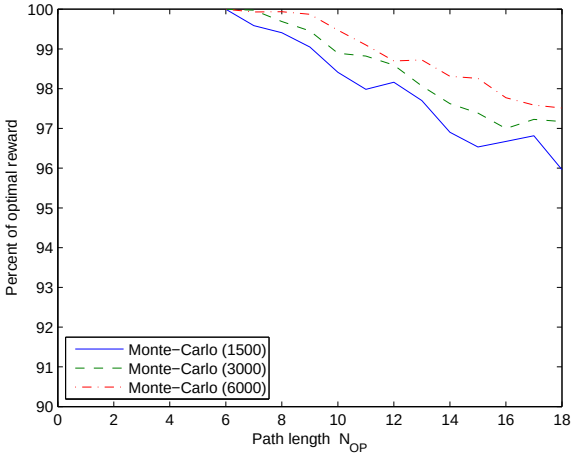
Figure 7.1: Run-time and number of discarded paths versus path length N_{OP} for the Monte-Carlo OP algorithm. The dashed line (corresponding to the right-hand y-axis) is the number of paths that cross themselves and therefore had to be skipped when generating exactly 3000 valid paths. Note that the run-time is nearly linear in N_{OP} for small N_{OP} values, but the number of discarded paths increases exponentially with N_{OP}, and starts becoming significant around $N_{OP} = 30$. This means the Monte-Carlo algorithm as a whole is exponential in N_{OP}, although the gradient is still relatively shallow at $N_{OP} = 50$. Each data-point making up the lines is the mean of 24 trials, each with a different rewards grid.

Algorithm 7.1 The IL-OP10 heuristic in pseudo-code. The OP solver tries to maximise the total discounted $P(m_c)R_{vent}$ value of a path of length 10 steps (for clarity, the discounting by a factor of γ on each step is not shown in the pseudo-code). Here L is the total mission length (133 timesteps in experiments), and t is the current timestep.

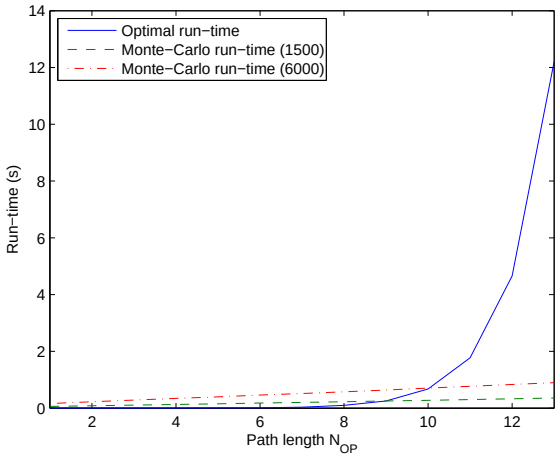
```
Procedure value-approx(b, c_curr, γ, c_next): value
  For each cell c
    If (c has been visited)
      Set rewards(c) = 0
    Else
      Set rewards(c) = P(m_c)R_vent
    Endif
  Endfor
  Set N_OP = max(1, min(10, L − t))
  If N_OP > 8
    Set path = solve-op-monte-carlo(c_curr, c_next, rewards, N_OP)
  Else
    Set path = solve-op-optimal(c_curr, c_next, rewards, N_OP)
  Endif
  Return Σ_{c∈path} rewards(c)
Endproc
```



(a) Mean reward found by the Monte-Carlo algorithm (using 3000 samples) against the optimal reward.



(b) Percent of the optimal reward attained by the Monte-Carlo algorithm, showing experiments where the Monte-Carlo algorithm used 1500, 3000 and 6000 path samples.



(c) Run-time of the Monte-Carlo and optimal algorithms, where variants of the Monte-Carlo algorithm using 1500 and 6000 path samples are shown.

Figure 7.2: Performance of the Monte-Carlo and optimal (exhaustive search) OP algorithms. All figures are the mean of 24 trials, each with a different reward grid, where each reward grid consists of $\mathrm{E}_z[\Sigma\Delta H]$ values calculated from a different OG. Note the run-times are only plotted up to $N_{OP} = 13$ because the times for larger N_{OP} values with the optimal algorithm would make the y-scale too big to see the Monte-Carlo run-times (for example, the mean run-time with $N_{OP} = 15$ is 83s, and with $N_{OP} = 18$ it is 24 minutes).

Algorithm 7.2 The Monte-Carlo and optimal OP solvers in pseudo-code. The OP solver tries to maximise the total discounted V values on a path of `num-steps` steps (for clarity, the discounting by a factor of γ on each step is not shown in the pseudo-code). Note that `solve-op-optimal` is implemented here using a recursive subprocedure, but the actual code uses a loop, which makes it harder to understand but considerably faster. Also in `solve-op-optimal`, a iterates over cells $\{forward, left, right\}$ from `curr` given `prev` as the previous cell.

```
Procedure solve-op-monte-carlo(prev,curr,V,num-steps): array
  Set bestValue = -1
  Set bestPath = []
  Loop 3000 times
    Set trial = {randomly-generated SAW of length num-steps
                 starting at a cell adjacent to curr but ≠ prev}
    If ∑_{c∈trial} V(c) > bestValue
      Set bestValue = ∑_{c∈trial} V(c)
      Set bestPath = trial
    Endif
  Endloop
  Return bestPath
Endproc

Procedure solve-op-optimal(prev,curr,V,num-steps): array
  Set bestValue = -1
  Set bestPath = []
  Set path = []
  For a ∈ 𝒜
    eval-action(curr,a,V,num-steps)
  Endfor
  Return bestPath
  Procedure eval-action(prev,curr,V,num-steps)
    If num-steps = 0
      If ∑_{t∈path} V(t) > bestValue
        Set bestValue = ∑_{t∈path} V(t)
        Set bestPath = path
      Endif
    Else
      Set path(num-steps) = curr
      For a ∈ 𝒜
        eval-action(curr,a,V,(num-steps - 1))
      Endfor
    Endif
  Endproc
Endproc
```

Algorithm 7.3 The pure-OP algorithm in pseudo-code. The procedure `choose-action-OP` is applied at each timestep, and the `value-approx` function called is the one given in Algorithm 7.1. Note a is maximised over cells $\{forward, left, right\}$ from c_{AUV} given c_{prev} as the previous cell.

```
Procedure choose-action-OP(b, c_prev, c_AUV, γ): cell
  Return argmax_a {ρ(b,a) + γ·value-approx(b, c_curr, γ, a)}
Endproc
```

7.5 ΣΔ*H*-OP Algorithm

The ΣΔ*H*-MDP described in Section 5.6 does not suffer from the same performance problems that IL-CE does, but instead outperforms the myopic ΣΔ*H* algorithm it is based on. However, the MDP formulation does make the same implicit assumption: that repeated observations from the same cell have the same value as the first observation. In reality this is not the case, as the same plumes will be detectable from any given location. We find that (for areas of the OG unaffected by plume detections) previously-visited cells have $E_z[\Sigma\Delta H]$ values about half of those of unvisited cells, a discrepancy which is exaggerated in MDP value functions, as re-visiting a cell with slightly higher reward than surrounding cells will lead to it having a much higher value. This led me to postulate that a better assumption is that repeated observations have zero value.

To achieve this, the OP solver described in Section 7.4 is used to create a variant of the ΣΔ*H*-MDP algorithm: instead of solving an MDP in $E_z[\Sigma\Delta H]$ values, we solve an N_{OP}-step OP where the reward for each step is the $E_z[\Sigma\Delta H]$ value of that cell (where $\Sigma\Delta H(b,a,z)$ is given by Equation 5.10). As the OP solver is exponential in the path length N_{OP}, experiments have only been performed with $N_{OP} = 30$ at most, whereas the agent has 133 steps available at the start of the mission. However the effect of truncating the OP path length is small given that, as in Equation 5.12, we use a discount factor of $\gamma = 0.9$. Pseudo-code for the ΣΔ*H*-OP algorithm is given in Algorithm 7.4.

Note that the limited path length the OP is solved for also means that for ΣΔ*H*-OPN_{OP}, the algorithm is independent of grid size for all grids larger than $2N_{OP} \times 2N_{OP}$ cells (as ΣΔ*H* values further than N_{OP} steps from the agent do not have to be calculated).

7.6 Results and Discussion

Figure 7.3 shows comparative results for ΣΔ*H*-OP10 and OP35 using the same experimental conditions described in Section 5.2. It is clear that ΣΔ*H*-OP10 is an improvement on ΣΔ*H*-MDP for only a small increase in run-time, with the results being better at a significance level of 0.05 in a one-tailed test. ΣΔ*H*-OP10 is also better than the previous best algorithm, IL4 (although not by a statistically significant margin), and runs much faster.

The variation in performance of ΣΔ*H*-OP with path length N_{OP} is shown in Figure 7.4. Figure 7.4a shows N_{OP} values from 4 to 30, where the exact OP solver was used for $N_{OP} \leq 8$ and the Monte-Carlo solver used for N_{OP} of 10 and above. This graph clearly shows that for $N_{OP} > 10$ the performance of the algorithm actually gets worse, and the switch to the Monte-Carlo algorithm is an obvious explanation for this, as Figure 7.2b shows that the quality of solutions produced by the MC solver will decline steadily with increasing N_{OP} (for $N_{OP} > 8$). However, if that is the explanation then the $N_{OP} = 10$ result (which is better than $N_{OP} = 8$ despite using the MC algorithm) would have to be considered a statistical fluctuation. Figure 7.4b shows ΣΔ*H*-OP performance using only the optimal OP solver, for N_{OP} values between 4 and 12 (the run-time of the optimal algorithm meant experiments with $N_{OP} > 12$ were not practical). While all the $\bar{f}_v$ values shown in Figure 7.4b (including for ΣΔ*H*-MDP) are sta-

Algorithm 7.4 The $\Sigma\Delta H$-OP algorithm in pseudo-code. The procedure `choose-action-sdh-op` is applied at each timestep to select the action to take. The OP part of the algorithm tries to maximise the total discounted $\Sigma\Delta H$ value of a path of length N steps (for clarity, the discounting by a factor of γ on each step is not shown in the pseudo-code).

```
Procedure choose-action-sdh-op(b, c_prev, c_AUV, γ, N): cell
  For each cell s
    If (s has been visited)
      Set E_ΣΔH(s) = 0
    Else
      For each observation z
        calculate P(z|b,s)
        Set b' = srog(b,s,z)
      Endfor
      Set E_ΣΔH(s) = Σ_z P(z|b,s) Σ_c |H_c(b') − H_c(b)|
    Endif
  Endfor
  Set N_OP = max(1, min(N, L − t))
  If N_OP > 8
    Set path = solve-op-monte-carlo(c_prev, c_AUV, E_ΣΔH, N_OP)
  Else
    Set path = solve-op-optimal(c_prev, c_AUV, E_ΣΔH, N_OP)
  Endif
  Return path(1)
Endproc
```

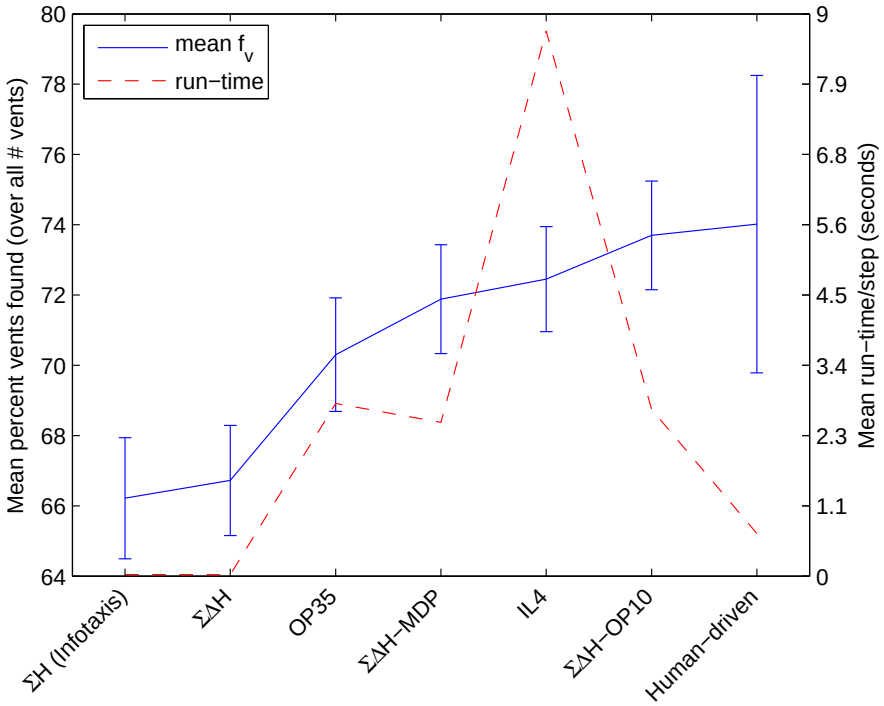
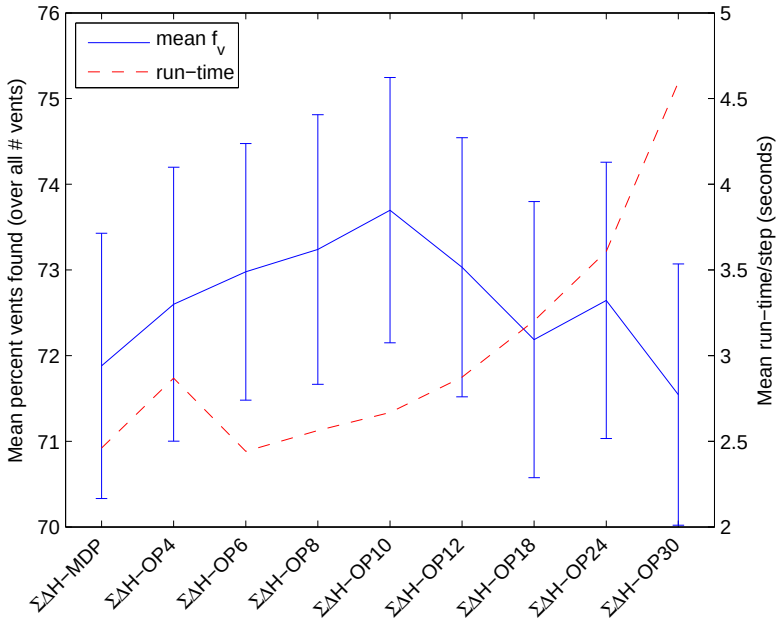
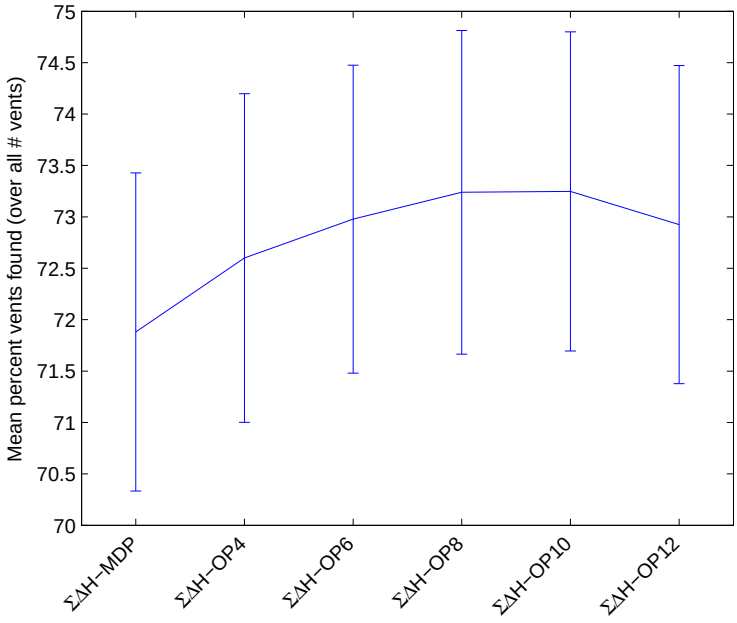


Figure 7.3: Results for the $\Sigma\Delta H$-OP10 algorithm compared to human-driven search, OP35, IL4, Infotaxis and the other belief-change-maximisation algorithms. The solid line is the mean percentage of vents found, and the dashed line is the run-time per timestep. All results are means of 600 trials – 150 each with 3,4,5, and 6 vents – except the human result, which is the mean of 100 trials. 95% confidence bars are also plotted.



(a) ΣΔH-OP results where the optimal OP solver is used for $N_{OP} \leq 8$, and the Monte-Carlo OP solver is used for $N_{OP} \geq 10$. The anomalous run-time for ΣΔH-OP4 is probably due to the computing cluster being overloaded while those experiments were running, as there is no technical reason it should be slower than ΣΔH-OP6 and ΣΔH-OP8.



(b) ΣΔH-OP results where the optimal OP solver is used for all N_{OP} values.

Figure 7.4: Results of ΣΔH-OPN_{OP} for values of N_{OP} from 4 to 30, with ΣΔH-MDP shown for comparison. Note that the x-axis is not to scale in terms of N_{OP}.

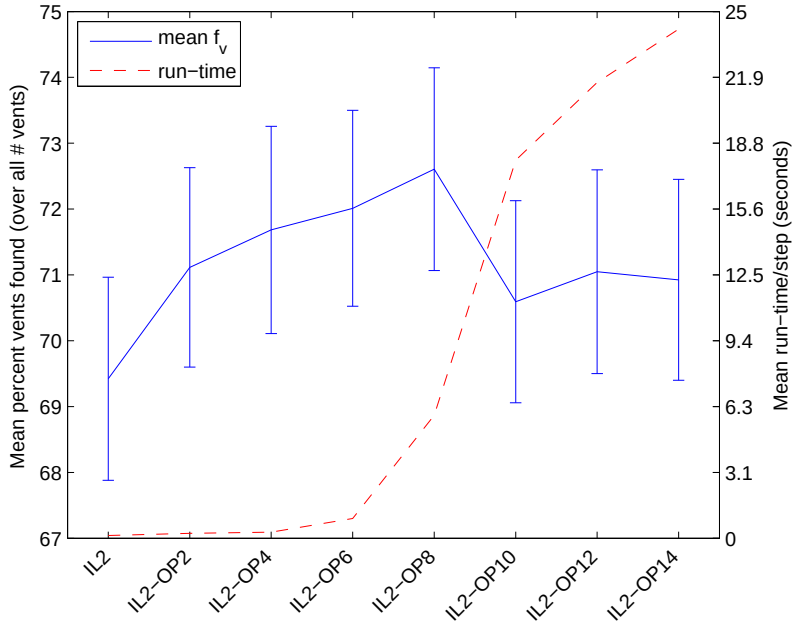
tistically equivalent, the graph does suggest that there is a peak in algorithm performance, after which vent-finding performance decreases with increasing path length. One hypothesis for this phenomenon is that large N_{OP} values bias the algorithm towards exploitation of high-probability cells near the agent, rather than encouraging exploration of further away locations that may lead to discovering new vents.

Results for applying the OP correction to IL are promising, but less clear-cut than for the $\Sigma\Delta H$ case, as Figure 7.5 shows. For IL2 (two steps of information lookahead), adding an OP correction invariably improved the performance, with a steady increase from $N_{OP} = 2$ to $N_{OP} = 8$. The sharp drop from IL2-OP8 to IL2-OP10 again suggests that the suboptimal OP solutions provided by the MC solver, which is used for $N_{OP} \geq 10$, are damaging vent finding performance. The lack of any trend in performance between IL2-OP10 and IL2-OP14 could also be explained by the deterioration of MC results with increasing N_{OP}, as shown in Figure 7.2b, or simply by statistical fluctuations.

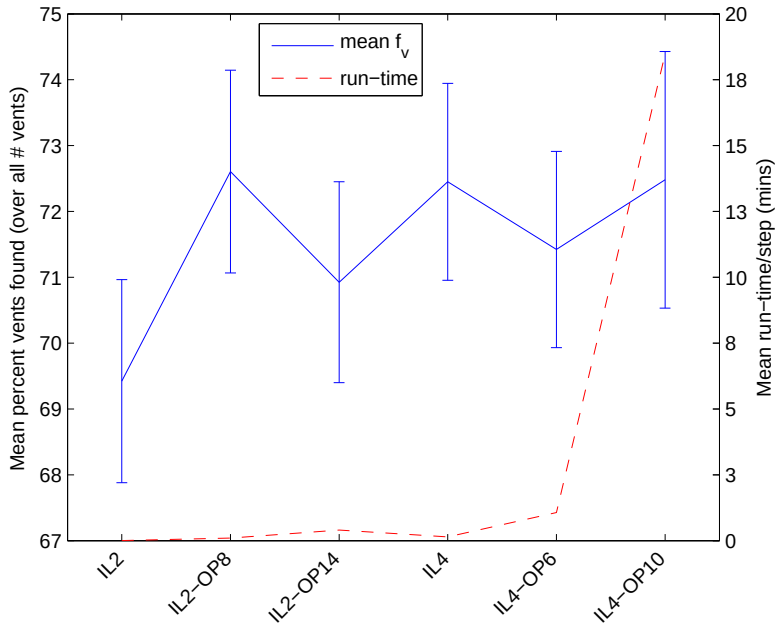
With four steps of IL, adding an OP6 correction actually results in worse performance, but early results indicate this is reversed for IL4-OP10, as shown in Figure 7.5b (where some of the IL2 results are also included for comparison). Given that each trial for IL4-OP10 required approximately 41 hours of CPU time, only 340 instead of 600 trials were performed, so less confidence can be attached to the IL4-OP10 results.

I hypothesise that the OP correction interacts in a complex manner with the exploration / exploitation trade-off inherent in the IL algorithm. Note that IL would be optimal if it could be run with a large enough lookahead, whereas the OP correction is greedy – it will try to visit all cells in the vicinity of a vent with $P(m_c)$ even slightly higher than the prior, instead of choosing to abandon that area and search for more vents. My intuition is that having a larger OP correction partly cancels out the drive to perform exploratory actions in IL, causing the agent to spend too much time near vents it has already visited.

Figure 7.6 shows the effect of increasing the OP path for the pure-OP algorithm. This shows that a simple gradient-ascent algorithm (OP1) is far from optimal in this environment, but also that lengthening the OP path does not consistently improve performance.



(a) Effects of the OP correction on IL2, with increasing N_{OP} shown along the x-axis.



(b) Effects of the OP correction on IL4, with the best and worst IL2-OP results shown for comparison. Note the 95% confidence intervals are larger for IL4-OP10 because they are over 340 trials rather than 600 for all other algorithms, due to computational constraints (the run-time scale is in minutes).

Figure 7.5: Effects of the OP correction for IL2 and IL4, with algorithm run-time per timestep shown as a dashed red line (against the right-hand y-axis).

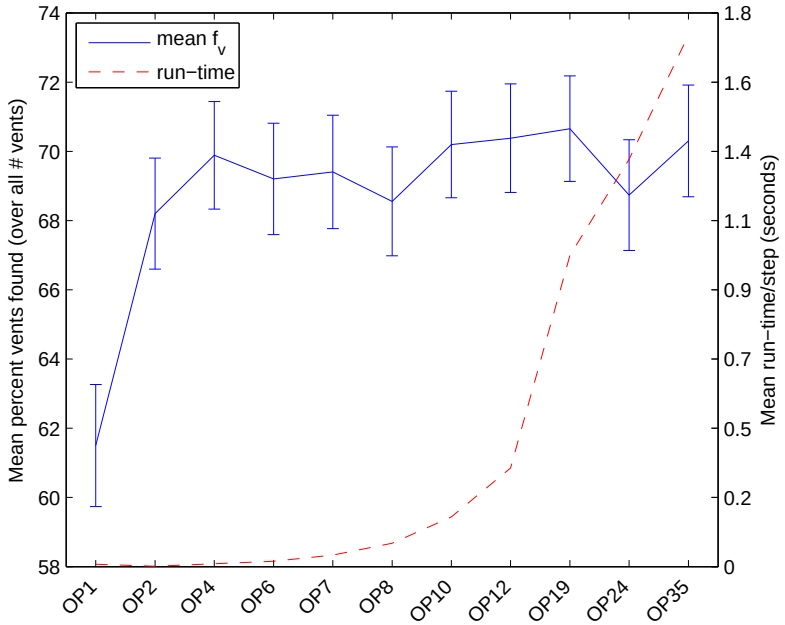


Figure 7.6: Performance of the pure-OP algorithm against N_{OP} (note the x-axis is not to scale in terms of path length N_{OP}). Algorithm run-time per timestep is shown as a dashed red line (against the right-hand y-axis).

Chapter 8

Discussion and Conclusion

In this chapter, first I review the performance of all algorithms and cover some general issues with them (Section 8.1). Next Section 8.2 seeks to evaluate the contributions of this thesis. I cover the improvement on conventional vent prospecting methods promised by my novel algorithms, and the ways in which the model is simplified from the true domain. Section 8.3 describes possible future work on this topic, and finally, Section 8.4 summarises this thesis and its contributions.

8.1 Discussion

8.1.1 Algorithm Comparisons

The performance of all algorithms is shown in Figure 8.1, where only the best-performing variant of parameterised algorithms is plotted, for example only IL2-OP8 from the IL-OP class of algorithms. Table 8.1 lists the performance of a slightly larger set of algorithms, and also gives the complexity class for each algorithm, whether or not the algorithm is myopic, and the standard deviation in the $\bar{f}_v$ result. Algorithms are classed as myopic in Table 8.1 if they only examine the grid four cells or fewer away from the agent's location, 'partly myopic' if they look more than nine steps away but not as far as the edges of the search area, and non-myopic if they examine all cells in the search area. An evaluation of the novel algorithms against the comparison algorithms is given in Section 8.2; here I discuss some aspects of the novel algorithms.

First, $\Sigma\Delta H$-OP, IL2-OP8, and IL4 were the best performing algorithms overall, and with performance that was statistically equivalent. This was unexpected because $\Sigma\Delta H$ rewards for the 'wrong' thing (information gain), as opposed to IL, which rewards for the same measure algorithms are evaluated on (visiting vents). Further, the OP correction assumes zero reward for re-visiting cells, which is accurate when used as a correction for IL, but not for $\Sigma\Delta H$ as useful observations can still be made by re-visiting a cell.

While $\Sigma\Delta H$-OP, IL2-OP8, and IL4 were the best algorithms in terms of finding vents, the run-time and scalability of the algorithms is also important. The main contributor to the run-time of my novel algorithms is running the OG update, which is approximately linear in C, the number of cells in the grid. The time taken to solve an MDP is linear in C, but is negligible for algorithms where only one MDP is solved per timestep. However, the time for the OP solver to run is relevant, and this scales with $e^{\lambda N_{OP}}$ where λ was experimentally determined to be 0.034. The IL algorithms are exponential in the number of steps of lookahead, N, and as the branching factor in the state-observation space is nine, they scale approximately with $9^N C$ (because a new occupancy grid must be calculated at every leaf node). Calculating a $\Sigma\Delta H$ value for a specific cell is linear in C, as it requires three OG updates (one for each observation), but as the $\Sigma\Delta H$-MDP and $\Sigma\Delta H$-OP algorithms require $\Sigma\Delta H$ values to be found for all C cells, they scale with C^2 (plus the OP contribution in the case of $\Sigma\Delta H$-OP).

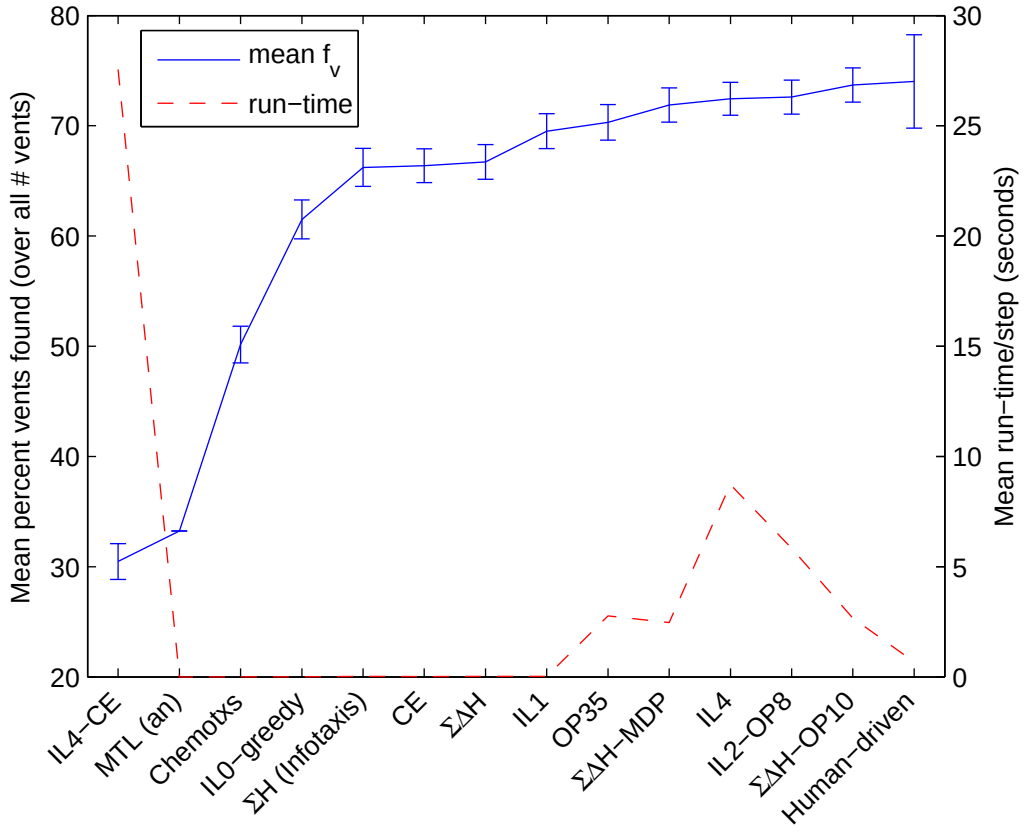


Figure 8.1: Results of the best algorithm of each class, in terms of the percent of vents found (blue line and left-hand axis) and the run-time (dashed red line and right-hand axis). 95% confidence intervals for the percent found are also shown. Note that the human-driven confidence intervals are larger than the other algorithms because only 100 as opposed to 600 trials were completed, and the confidence interval for the MTL result has zero height because this result was found analytically rather than experimentally.

Table 8.1: Results of all novel algorithms presented in this thesis. The first column gives the algorithm name, and the reference of the Algorithm listing defining it, or failing that the section that it is described in. The second column indicates if the algorithm is myopic, where myopic means it only looks at the map within a few steps of the agent when planning. The third column gives the computational complexity order of the algorithm (ignoring constants), where C is the number of cells in the OG, and the λ parameter for OP (which must be less than one) was found experimentally to be $\lambda = 0.034$ for the Monte-Carlo algorithm. The final two columns give the mean percent of vents found (over 600 trials) and the standard deviation (σ) in this value.

Algorithm (and reference)	Myopic?	Complexity	$\bar{f}_v$, % vents found	Std. Dev
MTL (Sec 2.4)	N/A	1	33.25	N/A
Chemotaxis (Sec 2.4)	yes	1	50.15	20.82
Human driven (Sec 2.4)	N/A	N/A	74.02	21.59
H-MDP (5.1)	no	C	67.19	19.38
ΣH (Infotaxis) (5.2)	yes	C	66.22	21.54
$\Sigma\Delta H$ (5.3)	yes	C	66.72	19.58
ΣH-MDP	no	C^2	68.43	19.98
$\Sigma\Delta H$-MDP (5.4)	no	C^2	71.88	19.34
IL0 (greedy) (6.2)	yes	1	61.50	22.03
IL1 (6.2)	yes	$9C$	69.51	19.81
IL4 (6.2)	yes	9^4C	72.45	18.68
IL4-CE (6.3)	no	$9^4 \cdot 2C$	30.47	20.19
CE (6.4)	no	C	66.37	19.13
Obs-Reward-MDP (Sec 5.7)	no	C^2	64.80	19.89
IL2-OP8 (7.1)	partly	$9^2(C+e^{8\lambda})$	72.61	19.25
IL2-OP14 (7.1)	partly	$9^2(C+e^{14\lambda})$	70.92	19.07
IL4-OP6 (7.1)	partly	$9^4(C+e^{6\lambda})$	71.42	18.63
OP10 (7.3)	partly	$e^{10\lambda}$	70.20	19.23
OP35 (7.3)	no	$e^{35\lambda}$	70.30	20.18
$\Sigma\Delta H$-OP10 (7.4)	partly	$C^2+e^{10\lambda}$ †	73.70	19.35

† Note that for $C > (2N_{OP})^2$, the $\Sigma\Delta H$-OP algorithm is actually linear in C, and scales with $(2N_{OP})^2C + e^{N_{OP}\lambda}$.

As shown in Figure 8.1, IL-CE was not only the worst performing algorithm, it was also by far the slowest, taking over three times as long as IL4. While Table 8.1 shows that IL-CE has the same complexity class as IL, $\mathcal{O}(9^N C)$, the constant factor is much greater in the case of IL-CE as it must solve 9^N MDPs instead of simply looking up 9^N values in an array. IL-OP scales slightly worse than IL, but for the relatively small grid size experiments were performed on, it was generally faster than IL with a slight performance improvement (although this was only the case for IL2-OP, and not IL4-OP). Similarly, $\Sigma\Delta H$-MDP performed worse than $\Sigma\Delta H$-OP8 and had almost identical run-time, so given that $\Sigma\Delta H$-OP will scale better, it is considered to be the superior algorithm. In the middle of the performance range was a group of algorithms which all have $\mathcal{O}(C)$ complexity, including $\Sigma\Delta H$, ΣH, CE, H-MDP and IL1, and these offer a good compromise between run-time and vent-finding effectiveness.

Of the $\mathcal{O}(C)$ algorithms, IL1 was the best, and this demonstrates that IL can be effective even for small lookahead values. I believe this to be because when the agent gets a plume detection, the OG algorithm increases the occupancy probability of cells in a cone starting from the agent's location and extending up-current (as shown in Figure 5.1). Therefore this cone provides a 'probability gradient' that leads to the vent. This does not mean that a simple gradient ascent algorithm would do well in this domain – in fact IL0 is a gradient ascent algorithm and it performs significantly worse than any other novel approach (with the exception of IL-CE). Further, IL4 improves on IL1 by three $\bar{f}_v$ points, and Figure 8.1 clearly shows that the myopic algorithms are all found on the left-hand-side of the graph with low $\bar{f}_v$ numbers.

IL compares well with the more heuristic approaches such as $\Sigma\Delta H$-OP in this domain partly because the action/observation tree (as shown in Figure 4.2) has a branching factor of only nine for each timestep, three possible actions and three possible observations. Most problems have many more actions and observations, which makes information-lookahead methods less attractive, for example [Ross *et al.*, 2008] find in a domain with 128 actions that exhaustive information-lookahead performs worse than several different methods using heuristic sub-sampling. In fact, having a small branching factor makes IL more similar to the point-based methods discussed in Section 6.1.1, as sampling belief states has less advantage over exhaustive enumeration. IL is still conceptually quite different from point-based methods, because it finds exact small-horizon values rather than an approximate general value function. For the vent prospecting problem, while $\Sigma\Delta H$-OP outperforms IL4 (although not to a statistically significant extent) and runs considerably faster, it is worth noting that IL has a slight advantage in terms of computational complexity so could scale better, because while IL is linear in OG size, $\Sigma\Delta H$-OP is only linear for small values of N_{OP} and is polynomial in C otherwise. The final point in favour of IL is that its performance degrades slowly as N is decreased, but its run-time drops exponentially.

8.1.2 Practitioners' Guide

This section aims to provide some guidance on which of the novel algorithms presented in this thesis would be the best to use for a given practical application. I assume that the objective will be to maximise the number of vents found; given this, the characteristics of the application that determine what algorithm will be most suitable are the size of the grid, and the compute time and computational resources available for running the planner. It is possible that the behaviour produced by certain algorithms will be better suited to some types of domain, but without concrete details of the domain and further experimental study, it is impossible to draw any conclusions about differences in algorithm behaviours.

The choice of grid cell size (together with the speed of the AUV, which is not considered to be a free parameter) will determine how long the AUV takes to traverse a grid cell, which equates to one timestep in the model. Section 4.4.2 suggests real-world values for the search area size and cell size, producing a grid with $C = 40000$ and a simulation timestep of 5 s. Figure 8.1 shows that most

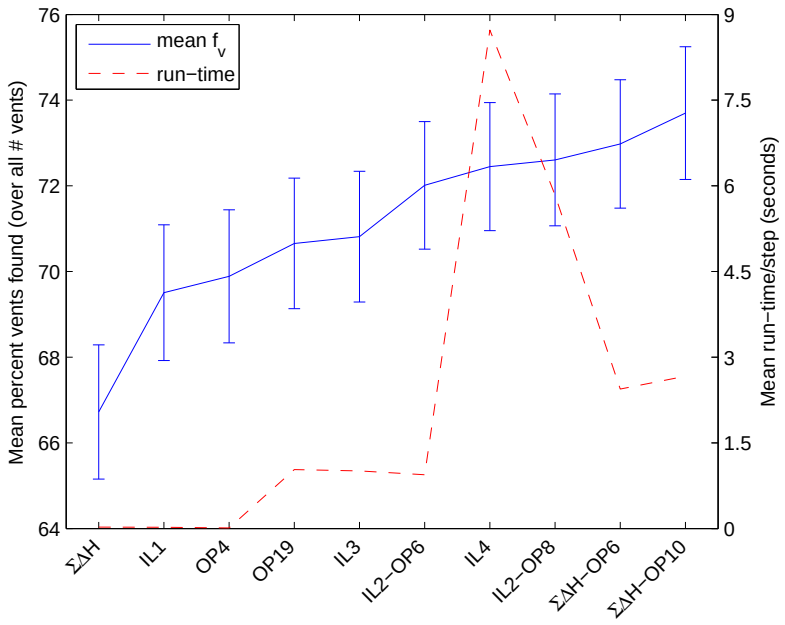


Figure 8.2: Results for algorithms suggested for practical use, in terms of the percent of vents found (blue line and left-hand axis) and the run-time (dashed red line and right-hand axis). 95% confidence intervals for the percent found are also shown.

algorithms complete a timestep within 5 s, but that is for a grid with $C = 400$ and using a relatively fast processor. Even then, algorithms such as OP4 have relatively high performance but are (almost) completely independent of the grid size, so would be practical for grids with 40000 cells. If the search area size is assumed to be fixed, to enable any of the algorithms that perform better than OP4 to be used, the number of grid cells would have to be reduced. This is doubly beneficial, because as well as the reducing C, the bigger cells will mean the timestep will be longer.

Figure 8.2 shows a selection of potentially appropriate algorithms, when run-time as well as vent-finding performance is taken into account. Some of the novel algorithms are in fact completely dominated by others, because they not only perform worse, but have worse computational complexity too. In Figure 8.2, the $\Sigma\Delta H$ and IL1 (and therefore the CE algorithm too) are dominated by OP4, which not only runs very quickly in the test environment, but also has $\mathcal{O}(1)$ complexity in grid size. The ΣH (infotaxis) and IL0 (greedy) algorithms perform even worse than CE, and have no computational advantage over OP4. Finally, $\Sigma\Delta H$-MDP is completely dominated by the $\Sigma\Delta H$-OP algorithm (for $N_{OP} \in \{4,6,8,10\}$), as $\Sigma\Delta H$-OP performs better, runs faster in the experimental environment used, and has the same complexity in C.

The remaining algorithms that are worth considering are:

- OP, especially with relatively small values of N_{OP}, for cases where computation time is very limited. Figure 7.6 shows that the performance of OP is relatively constant for N_{OP} values between 4 and 19.

- IL is never the best choice for any value of N in Figure 8.2 – there is always another algorithm that is better and faster – but it does have two advantages. First, it scales better with grid size than $\Sigma\Delta H$-OP, and IL4 at least has comparable performance to $\Sigma\Delta H$-OP. Second, it performs well with small values for the lookahead, N, which means it is a more dependable choice of

algorithm. As shown in Section 7.6, the performance of IL-OP (and $\Sigma\Delta H$-OP) can actually get worse as the lookahead is increased, so the choice of N_{OP} for these algorithms is critical.

- IL2-OP is likely to be the best algorithm for grids significantly larger than the $C = 400$ grid used in experiments in this thesis. This is because it performs consistently well for $2 \leq N_{OP} \leq 8$ (even though IL2-OP8 is the best variant, statistically IL2-OP6 is equivalent to $\Sigma\Delta H$-OP10), and it has better computational complexity than $\Sigma\Delta H$-OP. Compared to IL, it scales essentially the same with grid size, but performs better for shorter run-times.

- $\Sigma\Delta H$-OP is the best algorithm to use with grid sizes of around 400 cells, provided the run-time of approximately 3 s per timestep is acceptable. Figure 8.2 shows that $\Sigma\Delta H$-OP10 and $\Sigma\Delta H$-OP6 *both* have superior performance to any other algorithm, and have run-times much better than the two next best algorithms.

Other aspects of deploying the methods presented in this thesis on a real vehicle are considered in Section 8.3.5.

8.1.3 Model Issues

This section covers two aspects of the model presented in Chapter 4: first that the current plays a critical role in the performance of algorithms, and second that the need for a deterministic vent detector is not completely clear.

The current is the only clue used by the occupancy grid algorithm to assign probability to vent locations (given a plume detection), so the movement of the agent with respect to the current direction is very important for how easily it can find vents. For example, if the agent moves down-current, it will be sampling some of the same volume of water it has already sampled (depending on the relative velocities of the agent and current), so will not be gaining much information. Also, choosing to start the agent at the most down-current part of the grid may well favour certain algorithms over others, as the agent is starting in the best possible location to gather information about the search area given the (generally north-east) current direction. Early experiments comparing up-current starting location with down-current starting locations indicated that the myopic algorithms (for example chemotaxis and IL1) performed considerably worse when the agent started up-current compared to starting down-current. Experiments varying the agent's starting location were not pursued however, for the following reasons:

- Finding the explanation for algorithms performing differently given different current-relative starting locations is extremely hard (as discussed in Section 5.7).

- It was thought that a more fair comparison between algorithms could be made by fixing the start position and randomising the vent locations, as opposed to defining a small set of configurations of vents and start position. This should avoid cases where a particular algorithm happens to choose a path that encounters a vent from biasing the performance results.

- In the real world, the search area can simply be defined to be up-current from wherever the AUV is when it reaches its target depth, as the aim would be to assist the agent in finding vents as much as possible.

The deterministic vent detector was introduced in Section 4.3.4 in order that the agent knows if it has found a vent and therefore whether it will receive a reward. However, for most of the planning algorithms, it does not make any difference whether or not the agent knows when it finds a vent, and the vent detector is an extension of Jakuba's observation model that would be tricky to implement on a real vehicle (although an idea is given in Section 4.7.3). Further, the extra observation $z = l$

invalidates some of the assumptions underlying Jakuba's OG algorithm, which depends on a binary observation model.

The vent detector was introduced to help prevent the agent from gaining reward by re-visiting high probability cells; but in fact this functionality was implemented by simply making the reward for re-visiting *any* cell zero. If the vent detector was removed from the model, the only impact it would have on planning is that the calculation of $P(z|b,a)$ would be less accurate, because visited vents would have $P(m_c)$ values less than one (often only in the region of 0.5). For the IL algorithms, planning would be otherwise unchanged, as all visited cells have zero reward; similarly for the entropy-based algorithms, we could implicitly assume there was still a vent detector, and set the entropy of all visited cells to zero (as the entropy of cells with $P(m_c) = 1$ or $P(m_c) = 0$ is zero). One potential benefit of assuming a vent detector is that it would be possible to extend the model to include actions that the agent should take when it discovers a vent, such as photographing the area or collecting water samples.

8.1.4 Mapping Issues

This section covers some drawbacks and problems with the OG mapping algorithm used in this thesis. The first issue is one of numeric precision when updating occupancy grid probabilities, as is pointed out in [Jakuba, 2007] §2.4.1. The problem is that evaluating Equation 3.7 requires calculating a product over cells of the form $\prod_c \left(1 - P_c^d P(m_c)\right)$, and as for most cells $P_c^d \ll 1$ and $P(m_c) \ll 1$, a loss of significant digits is unavoidable.

The problem was first noticed due to a discrepancy between the probability of a cell being occupied before running the OG algorithm and the sum of the weighted probabilities under each observation after running the OG algorithm. The following equality was found not to hold:

$$P(m_c|\mathbf{O}) = \sum_z P(z)P(m_c|z,\mathbf{O}) \tag{8.1}$$

where m_c represents cell c being occupied, $\mathbf{O}$ is the occupancy grid, and z is a possible observation. Equation 8.1 should hold by the basic laws of probability, and applying the OG update equations by hand for a very small (2×2) grid indicated that it will hold under the OG update mathematics, meaning the problem was a floating-point precision one. I attempted to find a workaround for this problem without success[1], but as the errors were of the order of double-precision floating point accuracy for $\mathcal{O}(1)$ calculations, approximately 10^{-14}, the effect on action choice should be very small and only manifest due to cumulative errors (if at all).

One property of the occupancy grid approach that makes it a less than ideal partner to some of the entropy-based techniques developed in Chapters 5 and 7 is that an occupancy grid $\mathbf{O}$ is *not a probability distribution*; that is, $\sum_c P(m_c) \neq 1$. This means that increasing the probability of some cells containing a vent does not automatically reduce the probability for other cells. While we can sum the entropy of grid cells to get the joint entropy of the cell variables (see Section 5.4), this is only a proxy for the entropy of the map, and so plume detections can result in an increase in summed entropy. This is the reason $\Sigma\Delta H$ had to be introduced using the modulus of the entropy change, instead of the much more theoretically desirable reduction in entropy.

Another key drawback of OGs is that they cannot express a multi-modal distribution over vent locations, which would allow more informed planning decisions to be made. For example, the data may indicate either one vent relatively close, or else two vents further away, but this exclusive-or choice cannot be represented by an OG map.

[1] Workarounds tried included taking logs and using MATLAB's `log1p` function, and using the *vpa* arbitrary-precision toolbox from The Mathworks. The *vpa* package did reduce the error slightly, but at the cost of two orders of magnitude greater computation time.

Finally, there are some characteristics of Jakuba's OG algorithm that mean it does not produce the best possible maps for vent prospecting. As the updates are calculated recursively, it cannot revise the probability of cells when a new observation accounts for previous observations. The exact OG algorithm developed in [Jakuba, 2007] is capable of this 'explaining away': when a vent is found, it revises the probability of nearby cells downwards on the basis that the vent was likely responsible for any prior plume detections that caused those cells to have higher probability. Unfortunately, the optimal algorithm is exponential in the number of plume detections, which makes it impractical for use in realistic conditions. A related aspect is that sometimes the algorithm gets a plume detection, but then the agent travels away from the vent and no further detections are received. In these circumstances the OG values for the area surrounding the vent drop back to nearly the prior, whereas to a human it is obvious that there must be a vent somewhere, as otherwise the agent could not have detected a plume. The last characteristic of the algorithm is that it throws away all information on the magnitude of hydrothermal tracers when it encounters a plume, as it requires the observations to be converted into a binary detection or non-detection of a plume. While tracer concentration is not proportional to vent distance, it still incorporates useful information about the possible source locations, and it should be possible to develop better maps by taking this information into account.

8.2 Evaluation

8.2.1 Evaluation Using Metrics

The primary aim in developing the novel algorithms presented in Chapters 5, 6, and 7 was to outperform conventional vent prospecting approaches (as outlined in Section 2.4) in terms of number of vents found during a finite-duration AUV mission. In this aim and judged against the criterion of percent of vents found ($\bar{f}_v$), the novel algorithms have been very successful, as shown by Figure 8.1. The jointly-best-performing algorithms, $\Sigma\Delta H$-OP and IL4, were able to find more than twice as many vents as an exhaustive search (MTL), and almost 50% more than the reactive comparison algorithm chemotaxis. Even a greedy method that made use of the OG (IL0) generated $\bar{f}_v$ results more than 10 points better than chemotaxis, and the other novel algorithms were at least an improvement of 15 $\bar{f}_v$ points on chemotaxis (apart from IL-CE, for the reasons discussed in Section 6.5). Having the agent controlled by a human produced better $\bar{f}_v$ results than $\Sigma\Delta H$-OP, but not at a statistically significant level; in any case, as discussed in Section 2.4.2, human performance on problems similar to this is often close to optimal, so achieving 98% of the human $\bar{f}_v$ value is a positive result. Further, AUVs are useful especially because they do not have a human operator, which enables them to go to less accessible places, further from the mother ship, and without direct supervision, so it is not necessary to match human performance on the problem for autonomous control algorithms to be very valuable to the AUV community.

However there are some drawbacks to using $\bar{f}_v$ as the criterion for algorithm comparison, starting with the fact that the percent of vents found is not equivalent to the total discounted reward R_t received by the agent. This is because $\bar{f}_v$ does not take discounting into account, whereas R_t (given by Equation 3.11) weights vents found early in the mission higher than vents found later on. R_t was considered as an evaluation measure, but experimental results showed that with the rapid discounting used ($\gamma = 0.9$), total return was completely dominated by how quickly the first vent was found. This differed significantly from the overall aim of finding as many vents as possible, making R_t inappropriate for an evaluation measure. However using $\bar{f}_v$ may penalise reward-based algorithms such as IL because they are optimising for a different quantity to the one they are being judged on.

Another aspect of the evaluation measure is that a reward is only given for visiting the same cell that a vent occupies. Jakuba's algorithm can often locate vents accurately without actually visiting the cells they occupy [Jakuba, 2007; Jakuba and Yoerger, 2008], given a sensible survey path, where

'locate' implies that very high probability cells correspond to vent locations, and all vent locations are covered by high probability cells. In fact, for some grid cell sizes, visiting a cell containing a vent may be dangerous for the vehicle given the very high-temperature fluid emanating from vents.

Instead of evaluation by deploying the algorithm on a vehicle (which was outside the scope of this work, but the tasks involved are considered in Section 8.3.5), an alternative would be to use oceanographic data from research cruises and AUV deployments as input to the AUV algorithms. I was given access to an extensive repository of such data (courtesy of Dr Bramley Murton of the National Oceanography Centre, Southampton), and investigated this approach. It was found to be not feasible because every decision the algorithm makes will change the path of the AUV, and AUV data will only be available for the exact path taken by the AUV. Data from towed platforms and CTD casts allows a picture of a wider area to be built up, but the resolution of this data means that some model must be created to allow interpolation between data points. At this stage a better approach would be to adopt a more accurate theoretical model of currents and hydrothermal tracer propagation, as discussed in Section 8.3.1.

One of the challenges with this work is that there is no obvious way to compute the optimal behaviour, even for relatively small occupancy grids. Given this, the $\bar{f}_v$ metric and the comparison algorithms chosen provide a reasonable picture of the utility of my novel algorithms.

8.2.2 Wider Impact

For scientists interested in hydrothermal vents, and AUV operators in general, it is hoped that the advances presented in this thesis will be of practical use. As well as developing several computationally-intensive algorithms that are very effective in finding vents in a search area, I have presented myopic variants that attain most of the performance of the slower algorithms in a fraction of the time. For example, the IL1 algorithm finds approximately 95% as many vents as $\Sigma\Delta H$-OP in slightly less than 1% of the planning time, meaning that for most likely computational limitations imposed by the AUV hardware, an appropriate and effective algorithm can be used.

The algorithms developed are likely to be effective in other similar fields, and in particular where the goal is to find multiple chemical sources in a turbulent flow environment. Such domains are common in the military (mine detection), oil industry (oil prospecting and finding leaks in pipelines), and search-and-rescue fields (finding people trapped in collapsed buildings or mine shafts). Existing algorithms for these domains generally only reason about a single chemical source, whereas in the real world it is usually possible for there to be multiple sources in close proximity.

While the focus of this work has been on the chemical plume tracing problem, the comparison of entropy-based (heuristic) algorithms against information-lookahead approaches may be interesting in other domains. I have shown that while entropy may be optimising for the wrong thing (assuming a domain where locating *and visiting* targets is the goal), it is at least as effective and as fast as truncated search in the belief space. However it cannot be predicted if my conclusions will apply to other domains without significant experimental work, which was beyond the scope of this thesis with its focus on the oceanographic domain.

In fact the novel algorithms presented in Chapters 5, 6, and 7 do not have any explicit dependence on the vent-prospecting domain. The $\Sigma\Delta H$-based algorithms are implicitly tailored to the domain, because they use a change-in-entropy heuristic that makes sense in domains with low priors, but the IL algorithm has no implicit or explicit dependencies. This can be seen as a limitation of the work presented here, in that an obvious approach faced with a computationally intractable POMDP is to leverage properties of the domain to gain tractability; but given that the aim of this research was always to maximise vent-hunting performance, it is likely that the algorithms and parameters that have proved effective are only effective due to the nature of the map generated by the occupancy grid updates.

Finally, as noted in Section 1.4, the work in this thesis has resulted in three conference publications and a technical report.

8.2.3 Limitations of the Model

This section describes some limitations of the model of the problem that was used in this thesis (where many aspects of this model were dictated by the decision to use Jakuba's OG algorithm). These limitations do not invalidate any of the results presented, but may mean they do not transfer directly to the real AUV domain.

First, several continuous and partially-observable variables in the state space have been approximated as being discrete and completely known to the agent. Specifically, a small set of discrete, deterministic actions is assumed; the AUV location is assumed to be known exactly; the underwater environment is considered as 2D; and the vehicle and vents are assumed to only occupy spatial coordinates defined by a relatively coarse grid. For implementation on a real vehicle, most of these could be overcome by relying on macro-actions and waypoint-based navigation (as noted in Section 4.7), but it is vital that the AUV knows its own location accurately in order to build correct maps. Addressing this localisation issue would be a key aspect of deploying the system on a real vehicle, and Section 8.3.5 notes some of the ways localisation accuracy can be maintained on AUVs. However, localisation could still be a problem when using low-cost AUVs. A final variable that has been only superficially modelled is the energy constraints of the vehicle, and this is important for real missions as a low battery would make it imperative to end the mission and return to the surface rather than risk losing the AUV.

Second, the value of all algorithms developed depends on the environment model used being a reasonable model of hydrothermal plumes – otherwise the algorithms may be completely ineffective when faced with the dynamics of a real plume. This is both because algorithms have only been validated using a simulation based on this model, and because the mapping algorithm relies on this model to infer vent locations from plume detections, so errors in the model will cause it to place high likelihood of vents being found in places other than their true locations.

Third, the OG algorithm will produce the most accurate vent maps if the current is relatively constant on the timescale of the AUV mission. If the current reverses during a mission, plumes may be detected due to a vent actually located down-current from the vehicle. Although the agent stores the history of the current to allow this to be accounted for, a rapidly fluctuating current direction will mean the potential source locations given a plume detection will cover a much larger area. Ocean currents are known to change due to periodic processes as well as less well understood random processes [German *et al.*, 2008; Berdeal *et al.*, 2006], and an even worse outcome from the point of view of the OG algorithm is for there to be no current, as then it has no way of remotely predicting the location of a vent.

Finally, the OG is *not* an exact representation of vent locations given the observation history, because the cell independence assumption central to Jakuba's algorithm is only approximately valid. This means the reward function used in the IL algorithms is biased, as it is not based on a completely accurate Bayesian estimate of the belief state. The effect of this is hard to predict, but it could introduce a systematic error in the reward-based planning algorithms. This issue also affects the entropy-based algorithms, because summing cell entropies to find the entropy of the full map is only valid if the cell occupancy probabilities are truly independent, which they are not.

8.2.4 Choice of Mapping Algorithm

Jakuba's OG algorithm is well suited to mapping an area containing an initially unknown number of hydrothermal vents, and provided a good foundation for the work in this thesis, which was one of the first applications of AI planning methods to vent prospecting. However, given the requirements of

planning the path of an agent as well as creating a map, it has transpired that the OG approach has several shortcomings. Section 8.1.4 describes several key issues: OG maps cannot represent a multi-modal distribution over locations, their recursive nature means they cannot use the current observation to explain previous observations, and they do not behave as a probability distribution. The last point makes them inherently ill-matched with the use of entropy to drive exploration: much less meaning can be attributed to changes in entropy when the total probability mass of the map is not fixed.

An issue not covered in Section 8.1.4 is the state space defined by an OG map, which causes problems for decision-theoretic planning algorithms. This is chiefly because the state space of an OG map is larger than for many other possible mapping methods, but also because the OG has a large number of variables (C real-valued variables), which makes it hard to compress or summarise the state in any way. Section 8.3.2 discusses potential future directions for the mapping algorithm.

8.3 Future Work

8.3.1 Environment/Simulation

Improvements in the fidelity of the environment model are desirable for two reasons: to have a better idea of how algorithms will perform in the real world, and to improve the performance of the planning algorithms by making their predictions of future states closer to the true states.

The environment model implemented for this work could be enhanced by firstly extending it to three dimensions. While the model described in Jakuba's thesis is three-dimensional, the implementation requires significant modifications to work in three-dimensions.

The model also assumes a simple Gaussian dispersion of plume particles. Higher fidelity models are available, for example [Speer and Rona, 1989; McDougall, 1990], that would probably improve the real-world performance of the planning systems documented in this thesis. As well as improving the plume propagation model, the simulation environment could be made more similar to expected real-world conditions by including a current composed of different components that vary with different periodicities. A further enhancement could be to include random elements in the current to represent flows dictated by the specific topology of mid-ocean ridges. The characteristics of deep ocean currents are discussed in [Berdeal *et al.*, 2006; German *et al.*, 2008].

Lastly, Jakuba's model is a general one that does not consider the different characteristics of the various biogeochemical tracers that can be detected. Some tracers are conservative and have long residency times in the water column, whereas others can only be detected very close to the source vent.

The other aspect of the simulation environment is that it does not model stochastic action outcomes or resource (specifically battery power) depletion, and these could be included. Similarly, the environment used in experiments divides the search area into a 20×20 grid, which is smaller than ideal for real-world deployment. Using a larger grid would promote the development of fast, effective algorithms that are more likely to perform well on a real AUV. Of course, changes to the action and resource model and (possibly) environment size would necessitate alterations to the planning algorithms to cope with these.

8.3.2 Mapping

The current mapping algorithm assumes all grid cells are independent, and enforces a binary detection model, neither of which is ideal as noted previously in this chapter. While my planning algorithms should be agnostic about improvements to the mapping algorithm, a better map may make a more significant performance improvement than better planning.

Most non–occupancy-grid mapping methods would require the belief state to be factored by the number of vents in the search area: there would be a distribution over how many vents there are,

and then for each vent-count there would be a separate distribution over the location(s) of the vents. This would complicate planning, in that there would be N separate maps if we allow between one and N vents, but this could be addressed in a variety of ways. For example, the planner could simply plan on the basis of the map corresponding to the most likely number of vents, or the best action for each vent-count could be found and these actions weighted by the likelihood of the corresponding vent-count.

Specific mapping techniques that are worthy of investigation are factored particle filters [Doucet *et al.*, 2000; Montemerlo *et al.*, 2003], where each particle would encode the locations of all vents in the area, and Gaussian process [Rasmussen and Williams, 2006] maps.

Finally, it may be possible to create better maps (in combination with OGs or some other mapping method) by making use of bathymetry data, which are relief maps of the ocean floor generated using sonar, either ship-based or on-board the AUV. Hydrothermal vents often form large chimneys, which can show up on bathymetry data, and even in the absence of chimneys experts can predict areas that are likely to contain vents from the topography. If bathymetry data could be interpreted and merged into maps created from plume detections, the quality of map is likely to be superior, but this is a hard problem. An easier way to approach it could be to use data from previous surveys – either ship-based or using a submersible platform – to set the prior occupancy probabilities for the AUV's the OG map.

8.3.3 Extensions to the Current Planning Approach

There are several immediate avenues to explore for improving performance within the current framework of planners that I have developed, and this section describes some of these. Considering the IL algorithm first, the key to near-optimal performance is having a good leaf-node heuristic function. In Section 6.5, we saw that the CE heuristic had characteristics that made it perform worse than the myopic 'basic' heuristic, which was the occupancy probability of the leaf cell in the lookahead tree. So an obvious approach to improving performance is to develop a better non-myopic heuristic to approximate the expected long-term value of leaf nodes. One idea for such a heuristic would be to define a small region in the OG surrounding the leaf-node cell, and either sum the probabilities of all these cells, or of the top X highest-probability cells. This sum would indicate the overall value of the area around the cell at the end of the lookahead path, but as the lookahead is only for $N \leq 4$ steps, the leaf cells from different paths will be close to each other and the sum will include many of the same cells. As the optimal policy will avoid less high-value cells anyway, the summed-region heuristic may not differentiate well between different paths. It is also still a local heuristic, and until the last 20 timesteps in a mission of 133 timesteps, the agent is still capable of reaching any point in the grid, so the difference in true expected reward between heading in one direction or another may not show up. Further evidence that finding a good leaf-node heuristic is not a trivial problem comes from the IL-OP results presented in Section 7.6: OP is likely to be one of the better (and more computationally expensive) heuristics for this domain, as it solves the relaxed problem exactly, and yet IL4-OP does not provide a clear performance advantage over IL4. Finally, developing a new heuristic without a bound in performance with respect to the optimal value is unlikely to be of general scientific interest beyond the specific domain under consideration.

A minor enhancement that could be made to the IL implementation is to re-use the calculations of new belief states between timesteps. When performing information lookahead, the action-observation tree (as shown in Figure 4.2) is expanded down to N levels. When the agent then selects an action and receives an observation, it is following a specific path one level down that tree, and so child belief states down to $N-1$ levels could be re-used. However, this means only one-ninth of the action-observation tree can be re-used, so the speed improvement is likely to be less than 10% which is not very significant for an exponential algorithm.

An interesting algorithm to try out would be the CE-only algorithm, but with a small probability of taking a random, exploratory action instead of the optimal action from the MDP solution. This

approach is unlikely to outperform IL4 or $\Sigma\Delta H$-OP, but may shed some light on the importance of exploratory actions in this domain.

Moving on to the OP correction, it is worth noting that the OP solver implemented is exponential in the path length N_{OP}, and more efficient algorithms exist. While an OP algorithm that scales much better with N_{OP} is very desirable, a significant constraint is that we need the algorithm to run very quickly. For example, the current code takes a lengthy 2.5 hours to execute an IL4-OP6 trial, but to achieve even this performance the OP solver has to run in less than 0.01 seconds (due to the large number of belief states generated by looking ahead four steps). More sophisticated OP algorithms are given in [Blum *et al.*, 2007], but given these performance requirements, a better approach may be an improved algorithm for self-avoiding walks that could be used with the existing Monte-Carlo OP implementation, such as the dimerisation algorithm. Alternatively, it may be possible to leverage the limited-connectivity manhatten nature of our OP graph to create novel heuristics that can be used to incrementally improve an initial OP solution.

While it would be useful to have a faster OP implementation, its main purpose would be to enable larger N_{OP} values to be used, and the results given in Chapter 7 do not conclusively support the idea that increasing N_{OP} invariably leads to better-quality plans.

An idea for a new algorithm is to combine the IL and $\Sigma\Delta H$ algorithms by using $\Sigma\Delta H$ values as the leaf-node heuristic, creating an IL-$\Sigma\Delta H$ algorithm. It would be very interesting to find the effectiveness of this algorithm, and the only drawback with it would be that calculating $\Sigma\Delta H$ values is relatively costly, so the number of steps of IL that are practical may be small.

An interesting extension to the work presented in this thesis would be to investigate why the $\Sigma\Delta H$ calculation is so effective for this domain, given that $\Sigma\Delta H$ is really a heuristic (because $\sum_c |\Delta H_c|$ is not the true joint entropy of the OG cells). On the mapping side, Jakuba's OG algorithm performs very poorly when the OG priors are set to $P_c^p = 0.5$. It would be useful to find out if there is any reason for this (other than 0.5 being a poor estimate of the true priors), and if so, whether there is a workaround. If priors of 0.5 could be used, the need for the $\Sigma\Delta H$ heuristic would be removed.

Lastly, in the case when the agent receives no plume detections, the problem reduces to simply one of exploration. It may be possible to calculate the optimal behaviour exactly for this case, and if that can be done, the behaviour of the planning algorithms from Chapters 5, 6, and 7 could be compared to this optimal behaviour.

8.3.4 New Planning Approaches

This section describes requirements and ideas for new planning methods that could be developed to address the vent prospecting problem, including a hierarchical POMDP idea.

First I cover some problem characteristics that may assist in creating good plans, if a planning algorithm were to take account of them. A basic property of the domain is that if the agent detects a plume, there must be a vent located somewhere up-current from the agent (ignoring the possibility of a false positive detection, as this was set to zero in all experiments). However, if the agent explores further and does not receive another plume detection within the order of a dozen timesteps, the OG probabilities for cells up-current from the agent fall back to nearly the prior level. In this case the algorithms discussed in this thesis often fail to find the vent responsible for the detection, and a better planner might keep track of plume detections and continue exploring the area up-current from the original detection. Of course this is fundamentally a weakness in the mapping algorithm, and additionally, in the real world a plume detection may be due to a vent outside the search area, in which case it would not be worthwhile attempting to locate that vent.

As mentioned previously, the ocean current is central to this problem. While the vehicle's speed will generally be significantly greater than the current, the current will have an effect on how efficiently the vehicle can reach different locations. Some work on planning for AUVs has leveraged

predictions of the future current to plan the vehicle's path so that it is assisted rather than impeded by the current [Kruger *et al.*, 2007].

Developing an algorithm that plans in continuous space would be a useful direction for research, as noted in Section 8.2.3, and this is not necessarily mutually exclusive with an OG map.

Finally, the only concrete approach for a new planning algorithm that I will discuss is that of a hierarchical online POMDP. The idea is to use two levels of OG map: a high-level map with large cells, and a set of finer-grained low-level maps, one for each large cell. Unlike [Sridharan *et al.*, 2008], for the vent prospecting domain it makes sense to solve a POMDP over the high-level map first, and then solve two or three of the low-level maps, corresponding to the large cell containing the agent and the one or two highest-value adjacent large cells. The high-level OG will allow an online POMDP algorithm to be used to find the optimal path non-myopically, as only a few steps of lookahead will be needed to reach the edge of the grid, and the low-level OG will allow actual actions for the agent to be planned.

The key implementational issue to solve is how the observation model, environment model, reward model and OG updates will work for the high-level map. The most practical idea appears to be to maintain the high-level map independently of the low-level maps, rather than by aggregating cells from the low-level maps, and to re-scale the observation and reward functions to the larger cell size. These functions could be calculated by estimating a parameter ϕ, the probability that the agent will visit a vent in a large cell that it passes through, given that there is one there.

8.3.5 Deployment on a Real Vehicle

All work for this thesis was conducted in simulation, but care was taken to avoid algorithms that would not be practical on a real vehicle. To implement the code on an AUV, the key issues that would need addressing are to:

- Ensure that the environment simulation is accurate enough (as discussed in Section 8.3.1). This may require the planner to plan in 3D and continuous space, but it is hoped that the use of wrapper actions would allow the existing 2D OG model to be used.
- Interface the planner to the vehicle's navigation system. This could be straightforward if the vehicle was capable of running MATLAB on-board, as for example Autosub6000 should be (Stephen McPhail, personal communication, 21 May 2009), and if the AUV's navigation system accepts waypoints as input, which could easily be output from MATLAB (as done by [Harris, 2010]). If the planning system needed to be re-written in another language, for compatibility or execution speed reasons, much more work would be involved.
- A deterministic vent detector would have to be implemented (see Section 4.7.3).
- The localisation accuracy would have to be acceptable.
- The grid size and scale would have to be chosen appropriately, as discussed in Section 4.4.2.
- The most appropriate algorithm from the ones presented in this thesis would have to be selected (as discussed in Section 8.1.2).

As regards localisation accuracy, for vehicles such as Autosub, once it is within a few hundred metres of the bottom localisation by Doppler sonar is accurate enough, but during the descent the localisation essentially drifts with the current [McPhail *et al.*, 2009; Morice and Veres, 2010]. There are several methods that can be used to ensure the AUV knows where it is, such as deploying seafloor transponders to communicate acoustically with the AUV [Yoerger *et al.*, 2007b], which is known as a long baseline (LBL) system. Such systems are expensive however, and an alternative is the short

baseline (SBL) method where dead-reckoning position fixes are improved by measuring the distance to a single acoustic transponder located on the mothership. New software methods can significantly improve the accuracy of SBL [McPhail *et al.*, 2009; Morice and Veres, 2010], avoiding the need to resort to LBL beacons. A final idea that requires only a sidescan sonar (which is carried by almost all research AUVs) is to use AI techniques to match topography recorded from ship-based sonar with topography recorded from the AUV, in order to infer the exact location of the AUV [Di Massa and Stewart, 1997].

Many of the issues involved in using the planning software described in this thesis to search for vents are considered by [Harris, 2010], who targets Autosub6000 as the deployment platform. She reconciles the occupancy grid representation with Autosub6000's dynamics and command format, and implements a vent detector by making the vehicle descend and search a cell. She also assumes the vehicle is equipped with a camera, and develops a PRM [Kavraki *et al.*, 1996; Kavraki and Latombe, 1998] algorithm to take the vehicle on a path allowing a vent to be photographed from all angles. However, the limited turning circle of the vehicle proves to be a problem for this approach.

Deploying the software in the real world would provide the best way of evaluating the algorithms developed in this thesis against their primary goals, subject to the issues of only being able to gather data for a small number of missions, and the difficulty of finding ground-truth vent locations, as mentioned in Section 1.2.1.

8.4 Conclusions

This thesis has addressed the problem of searching the ocean floor for hydrothermal vents using an autonomous underwater vehicle. This is an example of a real-world problem where AI techniques would be very beneficial, allowing AUVs to search a larger area with less human supervision than would otherwise be possible. However the problem is a challenging one, because observations from the vehicle's sensors provide only limited information on the location of vents, rather than constraining them to lie in a small region of the search area. In addition to this partial observability, the spatial nature of the problem means it has a continuous state space, and an important consideration is the limited duration possible for AUV missions. Given these problem characteristics, the goal was to develop (in simulation) an algorithm that maximises the number of distinct vents the agent finds during a mission.

The problem was divided into mapping and planning, and for the mapping component I adopted the occupancy grid algorithm of [Jakuba, 2007]. The planning problem therefore takes the OG map as an input, and is isolated from the details of the vent mapping method (although clearly properties of the map are critical to the effectiveness of the planning algorithms). This planning problem is very hard due to the large, partially-observable state space; for example, [Smith and Simmons, 2005] use state-of-the-art POMDP methods to solve a problem with $\sim 10^5$ states, and note that this is larger than most POMDP solvers can manage but their algorithm would run out of memory given a problem 10 times larger. For comparison, the experimental setup used in this thesis has $\sim 10^{123}$ states.

The contributions of this thesis are:

- A POMDP model of the vent prospecting problem (Chapter 4). This model uses an OG to represent beliefs about vent locations, has a deterministic transition function allowing the agent to move one cell forwards, left or right, and incorporates an observation model based on the advection of hydrothermal plume particles by the current. It also includes a reward function where a constant reward is provided for each distinct vent visited by the agent, and defines a finite mission length.

- Experimental results suggesting that, for the vent prospecting domain and possibly for other

similar domains, using a heuristically-based entropy approach is as effective as and more computationally efficient than using a more theoretically grounded POMDP approach.

- A novel family of entropy-inspired algorithms for finding landmarks of interest in a low-prior occupancy grid environment (Chapter 5). The best performing of these algorithms was $\Sigma\Delta H$-MDP, which chooses actions that result in the largest mean *change* in entropy of OG cells. It is able to plan ahead over a discounted infinite horizon by ignoring belief-state changes for movement actions but utilising expected belief-state changes for making observations at distant locations.

- The application of online POMDP methods to the vent prospecting problem (Chapter 6). The ILN algorithm was presented, which performs a forward search for N steps through the belief space of the problem, and then propagates heuristic values backwards from the leaf nodes to calculate optimal action-values (given the approximate values for leaf nodes). Two heuristics were introduced, a basic heuristic that simply uses the occupancy probability of the final cell as a proxy for future reward, and a CE heuristic that generates leaf-node values by solving a deterministic MDP in occupancy probabilities from the OG. Experiments showed that the CE heuristic resulted in very bad vent prospecting performance, but the basic IL algorithm worked well for this domain.

- A correction procedure for the entropy and IL algorithms based on using an orienteering problem solver (Chapter 7). The OP solver replaced the MDP in $\Sigma\Delta H$-MDP, to improve the valuation of repeated visits to grid cells, and the resulting $\Sigma\Delta H$-OP algorithm was found to perform at least as well as IL but with considerably reduced computation time. I also applied the OP solver as a correction to information lookahead, where it shows promise in improving the algorithm's performance. However there appear to be complex interactions here, possibly due to the exploration/exploitation trade-off that must be made in this domain, and these are worthy of further research.

In conclusion, the vent hunting performance of the IL, IL-OP, and entropy-based approaches was statistically very similar. This is an interesting result, because the IL algorithm calculates an optimal N-step strategy, whereas using $\Sigma\Delta H$ values as rewards is a heuristic method, given that map quality is not the goal in this problem. It indicates that for partially-observable domains where the state space is too large for even approximate solution methods such as PBVI or Perseus to be applied, using a well-chosen heuristic together with a mechanism for including distal action effects can lead to effective planning. Of course, this thesis focused on only one domain, and my results will depend to a greater or lesser extent on the peculiarities of this domain.

The IL algorithm performed very well in the vent prospecting domain for several reasons: first because the OG algorithm creates a path of approximately increasing probability from the agent to the vent, which can be followed fairly effectively, especially if the agent plans more than one step ahead. Second, the relatively small branching factor of the action-observation space makes exhaustive search in that space more efficient than for many similar POMDP domains. However, overall the entropy-based algorithms and in particular $\Sigma\Delta H$ combined with the OP solver were the best algorithms for this domain, matching IL4 in performance but running three times faster.

Appendix A

Implementation Notes

All algorithms were implemented using MATLAB® version 7.4 (R2007a). Development work was done on Microsoft Windows, but experiments were performed under Linux (CentOS 5.5) on a multi-core Intel® Xeon® cluster running at 2.33GHz with 8GB RAM per eight cores.

A.1 Parameters

Algorithm parameters were as shown in Table A.1. The discount factor of $\gamma = 0.9$ was used for all algorithms. While this is a large discount to use, trials using a higher value for γ produced very similar results at the cost of significantly longer computation times.

The false positive plume detection rate was set to zero for all experiments.

Random numbers were generated using MATLAB's 'state' algorithm, which corresponds to the ziggurat algorithm of [Marsaglia and Tsang, 2000]. The random number generator was seeded with a different seed for each trial, and the seeds were generated by `http://www.random.org/`. The same set of 600 seeds (150 for each vent-count) was used to evaluate all algorithms.

A.2 Chemotaxis

This section describes the details of the chemotaxis algorithm implementation, which has to ensure the agent only moves between grid-cell-centre coordinates. The algorithm is reactive, in that the only input is whether or not there was a plume detection on each timestep. It proceeds as follows:

1. Prior to any detection the agent follows an MTL pattern with a trackline separation of three grid cells.
2. After any plume detection, the agent starts a 'surge' up-current for a duration of six timesteps.
3. After completing the surge, the agent moves in a spiral, with the radius increasing on each timestep.
4. If the agent's next move would take it outside the grid, it is instead sent on a triple-length surge at a randomly-chosen angle ϕ, $\phi \sim N(\Phi, 20\frac{\pi}{180})$ where Φ is the exact angle from the agent's location to the centre of the grid.
5. On every timestep when the agent detects a plume, the surge is re-started as in step (2).

For the surges, both up-current and when the agent is redirected toward the centre of the grid, a path along a given angle must be calculated. Unlike most line rasterising algorithms, we require the line to be composed of cells with their sides touching, in other words the line cannot go directly diagonally.

Table A.1: Algorithm parameters used for experiments in this thesis.

Parameter	Value	Notes
Grid Size	20x20	
Mission Length in Timesteps	133	
Agent Start Location	(20,20)	Top-right cell
EDDY_DIFFUSIVITY	0.04	Noise σ in the current
Plume Particle Diameter	1	Determines when agent detects a particle
Prior Occupancy Probability	0.01	
Current	$\begin{pmatrix} 0.7 \\ 0.3\sin 0.02t \end{pmatrix}$	Function of the timestep t
γ, Discount Factor	0.9	Used by all algorithms: MDP, IL, and OP
PF	0	False plume detection probability
IP_GROUPING_THRESHOLD	0	Values >0 invoke the extended IP algorithm
PUBAR	0	Values >0 group detections together

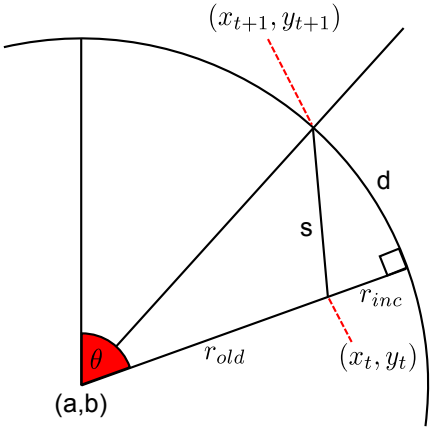


Figure A.1: Calculation of the trajectory for the spiral phase of the chemotaxis behaviour. On each timestep the spiral radius is increased by r_{inc}, and the new position of the agent (x_{t+1}, y_{t+1}) is calculated using the coordinates of the centre of the circle (a,b), the arc θ to the current position (x_t, y_t), and the agent's speed s. The area bounded by r_{inc}, s and d is approximated as a right-angled triangle in order to find an approximate value for d.

This is achieved by projecting the line onto the grid, and then choosing a path consisting of all the cells the line passes through. In the case that the line passes exactly through a grid 'cross-hair', an extra cell is added to ensure the path consists of valid moves.

When a surge is completed, the agent must start moving in an anticlockwise spiral. At the start of the spiral phase, the centre of the spiral is calculated as a point *MOTH_START_RADIUS* units (see Table A.2) up-current from the agent's location. Then on each timestep, the radius is incremented by *MOTH_RADIUS_INCREMENT*, and a desired new location is found using the agent's speed, s, the coordinates of the centre of the circle (which is fixed), and the angle to the old location. The actual location is then the cell-centre coordinates nearest to the desired location. The required calculations are shown graphically in Figure A.1.

Table A.2 shows the parameters used by the chemotaxis algorithm.

Table A.2: Parameters used for trials with the chemotaxis algorithm.

Parameter	Value	Notes
MOTH_SURGE_TIME	6	Number of timesteps to surge up-current for
MOTH_START_RADIUS	2	Start radius for the spiral behaviour
MOTH_RADIUS_INCREMENT	0.1	Radius increment for the spiral behaviour

A.3 Modified Observation Model

As mentioned in Section 4.3.4, my model introduced a deterministic vent detector and an additional observation, that of locating a vent ($z = l$), that was not present in Jakuba's model. This was implemented as an 'overlay' consisting of a list of cells that were known to contain vents, and a list of cells that were known to be empty. The actual OG data structure storing $\log r_{c,t}$ values (which map to $P(m_c)$ values) was therefore not altered at all; instead, before the reward function was applied or the calculation of $P(z|b,a)$ values was performed, the overlay was merged with the $\log r_{c,t}$ values to produce a $P(m_c)$ vector with zeros for empty cells and ones for vents (or, for the reward function, zero in both cases).

Therefore Jakuba's IP algorithm itself (Algorithm 3.1) was run with the unmodified $\log r_{c,t}$ values, i.e. as if the deterministic vent detector had not existed. In the case that a cell containing a vent was visited, and the observation in my model was $z = l$, the IP algorithm was always run with $z = p$, as we would always expect to detect a plume from the vent when we visit a cell containing a vent anyway.

Appendix B

Glossary of Oceanographic Terms

ABE Autonomous Benthic Explorer, an AUV operated by WHOI.

Advection The motion of particles being carried along by a fluid flow.

Asthenosphere A deformable layer of the Earth just below the lithosphere. The asthenosphere is contained within the upper mantle.

Autosub The AUV operated by NOCS.

Autosub6000 The newest incarnation of Autosub, developed and operated by NOCS and capable of diving to 6000 m.

AUV Autonomous Underwater Vehicle, a submersible robot.

Bathymetry Study of the relief of the ocean floor.

Buoyant plume The plume of seawater enriched by chemicals and mineral particles as it rises from a hydrothermal vent, and before it has attained neutral buoyancy.

Black Smoker A particular kind of hydrothermal vent, where particles precipitate out as the hot hydrothermal fluid mixes with cold seawater, and cause the appearance of black smoke coming from the vent.

CTD Conductivity, Temperature and Depth data and/or sensors.

GMT Generic Mapping Toolkit, an open-source command-line driven toolkit for producing graphics by processing geographic data.

Hydrothermal Vent A point in the ocean bed, often on a spreading centre, where hot water escapes out into the sea.

LBL A Long Base-Line acoustic navigation system, where beacons are positioned at known locations on the sea floor, near the AUV's search area, and acoustic transmissions from these beacons are used by the AUV to fix its position.

Lithosphere The rigid upper layer of the Earth, which is split into continental plates. This layer includes the crust and part of the upper mantle.

LSS Light scattering sensor or optical backscatter sensor. LSS readings indicate the density of particles in the water. Also called a nephelometer, sometimes measured in nephels.

MAPR Miniature Autonomous Plume Recorder, a small CTD sensor. 6 of them are attached to the same cable as TOBI.

NOCS National Oceanography Centre, Southampton, a division of the University of Southampton.

Non-buoyant plume The plume of seawater enriched by chemicals and mineral particles produced by a hydrothermal vent, when it has attained neutral buoyancy and has been dispersed over a wide area.

OBS Optical Back-Scattering. See LSS.

Potential Temperature A standardised measure of the temperature of a fluid: the temperature the fluid would attain if it was adiabatically brought to a pressure of 1000 millibars.

Side-scan sonar Acoustic imaging system where a wide-angled sound pulse is bounced off the ocean floor, and a detailed picture of the sea bed to either side of the vessel is built up.

SROG Semi-recursive occupancy grid algorithm, a generic term for a family of OG algorithms developed in [Jakuba, 2007].

TOBI Towed Ocean Bottom Instrument, a side-scan sonar and CTD platform towed several miles behind a mother ship, and kept at a constant altitude of a few hundred metres above the ocean floor.

The Vent Prospecting Problem Shorthand term for the complete problem of creating a probabilistic map of hydrothermal vents based on noisy sensor observations, and concurrently planning a path for the AUV to locate and visit as many vents as possible given the resource constraints of a mission.

WHOI Woods Hole Oceanographic Institution, an ocean research and education organisation located in Massachusetts, USA.

Appendix C

List of Symbols

a An action $a \in \mathscr{A}$, the set of all actions. Equivalently the index of a cell that an action moves the agent into.

$\mathscr{A}$ The set of possible actions, specifically $\{N, E, S, W\}$.

α Used to represent α-vectors, which are components of POMDP solutions, where $\alpha(s) = V(s)$.

b A belief state, $b \in \mathscr{B}$.

$\mathscr{B}$ The (infinite) set of all possible belief states b.

c_{AUV} The index of the cell containing the agent, and a component of the state space s.

c_{prev} The index of the cell that the agent was in on the previous timestep.

C The number of cells in the grid; for a 20×20 grid, $C = 400$.

d The event of detecting a plume, regardless of the source.

d_c The event of detecting a plume due to a vent at cell c. Note that $P(m_c|d_c) = 1$.

d^F The event of a false positive plume detection.

$\bar{f}_v$ The mean percent of vents found.

γ The discount factor, $0 \leq \gamma < 1$, which causes rewards to be count for less the further into the future they are.

H_c The entropy of cell c in the OG.

$H(b)$ The joint entropy of all C cells in the OG defined by b.

k Remaining number of steps to perform forward-search in the belief space for.

l The observation of locating a hydrothermal vent at the present position.

L The mission length in timesteps.

$\mathbf{m}$ The vent map, consisting of *true*/*false* values for all cells indicating whether or not they contain a vent.

m_c A boolean value: *true* if cell c contains a vent, *false* otherwise.

M The true number of vents in the search area.

$\mathscr{M}$ The set of all cells containing a vent.

n The null observation - no detections.

N Number of steps (in total) that forward-search in the belief space is performed for, with a given algorithm.

N_{OP} Number of steps available in the orienteering problem.

ν Side-length of grid, $\nu^2 = C$.

$\mathbf{O}$ The occupancy grid map, consisting of $P(m_c)$ values for all cells.

p The observation of detecting a plume.

P_c^d The probability that a particle from cell c is detected, given that c contains a vent, i.e. $P_c^d = P(d_c|m_c)$

P^F The probability of a false alarm detection, $P^F = P(d^F)$.

P_c^p The prior probability that cell c is occupied by a vent.

$P(m_c)$ Probability that cell c is occupied by a vent. This symbol is used as shorthand for the posterior probability of occupancy given the observations to date, $P(m_c|\mathbf{z}_{1:t})$.

$P(\neg m_c)$ Probability that cell c does not contain a vent. This symbol is used as shorthand for the posterior probability of occupancy given the observations to date, $P(\neg m_c|\mathbf{z}_{1:t})$.

Q The action-value function in an MDP/POMDP, $Q(s,a)$ or $Q(b,a)$.

$r_{c,t}$ Odds-ratio of occupancy probability for cell c at time t, $r_{c,t} = \frac{P(m_c|\mathbf{z}_{1:t})}{P(\neg m_c|\mathbf{z}_{1:t})}$

τ Timesteps elapsed since the emission of a particle from the source.

R Reward function in an MDP/POMDP, $w = R(s,a)$.

R_{vent} Fixed reward value the agent receives for visiting a previously unknown vent.

ρ Reward function in a belief-MDP, $w = \rho(b,a)$.

s The state of the system/agent.

$\mathscr{S}$ The set of all possible states.

$\mathbf{SE}$ The state estimator component of a POMDP, $b' = \mathrm{SE}(b,a,z)$.

T Transition function in an MDP/POMDP, $T = P(s'|s,a)$.

$\mathbf{U}$ The ocean current vector at a particular time.

$\underline{\mathbf{U}}$ The complete history of the ocean current vector.

$\mathbf{v}$ The agent's list of confirmed vent locations, consisting of *true/false* values for all cells indicating whether or not they are known to contain a vent.

V The value function of an MDP/POMDP, $V(s)$ or $V(b)$.

$\mathscr{V}$ A set of α-vectors used as a (partial) POMDP solution.

w The reward gained on a timestep, $w = R(s,a)$.

ω Identifier for a plume particle (Section 4.6).

x The true distance to an obstacle measured by a range-finding sensor (Section 3.1).

$\mathbf{x}_{AUV}$ The Cartesian coordinates of the vehicle, i.e. the coordinates of cell c_{AUV} (Section 4.6).

$\mathbf{x}_s$ The Cartesian coordinates of a source vent (Section 4.6).

z Used to indicate a specific observation.

$\mathbf{z}_{1:t}$ The vector of all observations from time 1 to time t.

Z Observation function in a POMDP, $Z = P(z|s,a)$.

$\mathscr{Z}$ The set of possible observations, specifically $\{l, p, n\}$.

Bibliography

[Apostolikas and Tzafestas, 2004] Giorgos Apostolikas and Spyros Tzafestas. Improved Qmdp policy for partially observable Markov decision processes in large domains: Embedding exploration dynamics. *Intelligent Automation and Soft Computing*, 10(3):209–220, 2004.

[Arulampalam *et al.*, 2002] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[Astrom, 1965] K. J. Astrom. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, February 1965.

[Baker and German, 2004] Edward T. Baker and Christopher R. German. On the global distribution of hydrothermal vent fields. In C.R. German, J. Lin, and L.M. Parson, editors, *Mid-Ocean Ridges: Hydrothermal Interactions Between the Lithosphere and Oceans*, number 148 in Geophysical Monograph Series, pages 245–266. American Geophysical Union, 2004.

[Baker *et al.*, 1995] Edward T. Baker, Christopher R. German, and Henry Elderfield. Hydrothermal plumes over spreading-center axes: Global distributions and geological inferences. In S. Humphris, R. Zierenberg, L. Mullineaux, and R. Thomson, editors, *Seafloor Hydrothermal Systems: Physical, Chemical, Biological, and Geological Interactions*, number 91 in Geophysical Monograph, pages 47–71. American Geophysical Union, Washington, D.C., 1995.

[Baker *et al.*, 2007] Maria Baker, Eva Ramírez-Llodra, Paul Tyler, Christopher German, Amy Baco-Taylor, Antje Boetius, Monika Bright, Daniel Desbruyères, Chuck Fisher, Yoshihiro Fujiwara, Françoise Gaill, Andrey Gebruk, Kim Juniper, P.A. Lokabharathi, Anna Metaxas, Lisa Levin, Ashley Rowden, Ricardo Santos, Tim Shank, Lúcia de Siqueira Campos, Craig Smith, Cindy Van Dover, Anders Warén, and Craig Young. ChEss protocols: Exploration and investigation of deep-water chemosynthetic ecosystems. ChEss Project, September 2007. http://www.noc.soton.ac.uk/chess/docs/chess_protocols.pdf.

[Balas, 1989] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[Barto *et al.*, 1995] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1-2):81–138, 1995.

[Berdeal *et al.*, 2006] Irene Garcia Berdeal, Susan L. Hautala, Leif N. Thomas, and H. Paul Johnson. Vertical structure of time-dependent currents in a mid-ocean ridge axial valley. *Deep Sea Research Part I: Oceanographic Research Papers*, 53(2):367–386, 2006.

[Blum *et al.*, 2007] Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal on Computing*, 37(2):653–670, 2007.

[Bourgault *et al.*, 2002] F. Bourgault, A. Makarenko, S. Williams, B. Grocholski, and F. Durrant-Whyte. Information based adaptive robotic exploration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-02)*, pages 540–545, 2002.

[Bresina *et al.*, 2002] John Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramkrishnan, David Smith, and Rich Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proceedings of the Eighteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 77–84, San Francisco, CA, 2002. Morgan Kaufmann.

[Bush *et al.*, 2008] Lawrence A. Bush, Brian Williams, and Nicholas Roy. Computing exploration policies via closed-form least-squares value iteration. In *Doctoral Consortium, Eighteenth International Conference on Automated Planning & Scheduling (ICAPS-08)*, 2008.

[Cassandra *et al.*, 1994] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. *Proceedings of the National Conference on Artificial Intelligence*, 2:1023–1028, 1994.

[Cassandra *et al.*, 1996] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: discrete bayesian models for mobile-robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-96)*, volume 2, pages 963–972, Osaka, Japan, 1996.

[Cassandra *et al.*, 1997] Anthony Cassandra, Michael L. Littman, and Nevin L. Zhang. Incremental Pruning: A simple, fast, exact method for partially observable Markov decision processes. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 54–61, San Francisco, CA, 1997. Morgan Kaufmann.

[Chao *et al.*, 1996] I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88:475–489, 1996.

[Chaudhuri *et al.*, 2003] Kamalika Chaudhuri, Brighten Godfrey, Satish Rao, and Kunal Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the fourty-fourth IEEE Symposium on Foundations of Computer Science (FOCS-03)*, pages 36–45, 2003.

[Corliss *et al.*, 1979] John B. Corliss, Jack Dymond, Louis I. Gordon, John M. Edmond, Richard P. von Herzen, Robert D. Ballard, Kenneth Green, David Williams, Arnold Bainbridge, Kathy Crane, and Tjeerd H. van Andel. Submarine thermal sprirngs on the galapagos rift. *Science*, 203(4385):1073–1083, 1979.

[Cover and Thomas, 2006] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley, second edition, 2006.

[Dantzig *et al.*, 1954] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, November 1954.

[Dearden *et al.*, 2003] Richard W. Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David E. Smith, and Rich Washington. Incremental contingency planning. In *Proceedings of ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*, Trento, Italy, June 2003.

[Dearden *et al.*, 2007] Richard W. Dearden, Zeyn A. Saigol, Jeremy L. Wyatt, and Bramley J. Murton. Planning for AUVs: Dealing with a continuous partially-observable environment. In *Workshop on Planning and Plan Execution for Real-World Systems, 17th International Conference on Automated Planning & Scheduling*, 2007.

[Di Massa and Stewart, 1997] Diane E. Di Massa and Jr. Stewart, W. K. Terrain-relative navigation for autonomous underwater vehicles. In *Proceedings of OCEANS 1997*, volume 1, pages 541–546. MTS/IEEE Conference Proceedings, 1997.

[Dissanayake *et al.*, 2001] M. W. M. Gamini Dissanayake, Paul Newman, Steven Clark, Hugh F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation*, 17(3):229–241, June 2001.

[Doucet *et al.*, 2000] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the Sixteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 176–1, San Francisco, CA, 2000. Morgan Kaufmann.

[Elfes, 1989] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[Farrell *et al.*, 2003] Jay A. Farrell, Shuo Pang, Wei Li, and Richard Arrieta. Chemical plume tracing experimental results with a REMUS AUV. *Oceans Conference Record (IEEE)*, 2:962–968, 2003.

[Feillet *et al.*, 2005] Dominique Feillet, Pierre Dejax, and Michel Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39(2):188–205, 2005.

[Feng *et al.*, 2004] Zhengzhu Feng, Richard Dearden, Nicolas Meuleau, and Richard Washington. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 154–161, Arlington, Virginia, United States, 2004. AUAI Press.

[Ferri *et al.*, 2008] Gabriele Ferri, Michael V. Jakuba, and Dana R. Yoerger. A novel method for hydrothermal vents prospecting using an autonomous underwater robot. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA-08)*, pages 1055 – 1060, Piscataway, NJ 08855-1331, United States, 2008.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman & Co Ltd, 1979.

[Garrison, 2002] Tom Garrison. *Oceanography: An Invitation to Marine Science*. Thomson Brooks/Cole, fourth edition, 2002.

[German and von Damm, 2003] C. R. German and K. L. von Damm. Hydrothermal processes. In Heinrich D. Holland and Karl K. Turekian, editors, *Treatise on Geochemistry*, pages 181–222. Pergamon, Oxford, 2003.

[German *et al.*, 1998] C. R. German, K. J. Richards, M. D. Rudnicki, M. M. Lam, and J. L. Charlou. Topographic control of a dispersing hydrothermal plume. *Earth and Planetary Science Letters*, 156:267–273, 1998.

[German *et al.*, 2008] Christopher R. German, Dana R. Yoerger, Michael Jakuba, Timothy M. Shank, Charles H. Langmuir, and Ko-ichi Nakamura. Hydrothermal exploration with the autonomous benthic explorer. *Deep-Sea Research Part I: Oceanographic Research Papers*, 55(2):203–219, 2008.

[Ghallab *et al.*, 2004] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[Gold, 1992] Thomas Gold. The deep, hot biosphere. *Proceedings of the National Academy of Sciences of the United States of America*, 89(13):6045–6049, July 1992.

[Golden *et al.*, 1987] Bruce L. Golden, Larry Levy, and Rakesh Vohra. The orienteering problem. *Naval Research Logistics*, 34(3):307–318, 1987.

[Hansen and Zilberstein, 2001] Eric A. Hansen and Shlomo Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[Harris, 2010] Catherine Harris. A hydrothermal-vent exploration algorithm for Autosub6000. BSc. Dissertation, School of Computer Science, University of Birmingham, April 2010.

[Hauskrecht, 2000] Milos Hauskrecht. Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13:33–94, 2000.

[Hayes, 1998] Brian Hayes. How to avoid yourself. *American Scientist*, 86(4):314, 1998.

[Hert *et al.*, 1996] Susan Hert, Sanjay Tiwari, and Vladimir Lumelsky. A terrain-covering algorithm for an AUV. *Autonomous Robots*, 3:91–119, 1996.

[Jakuba and Yoerger, 2008] Michael V. Jakuba and Dana R. Yoerger. Autonomous search for hydrothermal vent fields with occupancy grid maps. In *Proceedings of the 2008 Australasian Conference on Robotics & Automation (ACRA-08)*, 2008.

[Jakuba, 2007] Michael Jakuba. *Stochastic Mapping for Chemical Plume Source Localization with Application to Autonomous Hydrothermal Vent Discovery*. PhD thesis, MIT and WHOI Joint Program, February 2007.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[Kavraki and Latombe, 1998] Lydia E. Kavraki and Jean-Claude Latombe. Probabilistic roadmaps for robot path planning. In K. Gupta and A. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 33–53. John Wiley, 1998.

[Kavraki *et al.*, 1996] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[Kollar and Roy, 2008] Thomas Kollar and Nicholas Roy. Efficient optimization of information-theoretic exploration in SLAM. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1369–1375, 2008.

[Krause and Guestrin, 2005] Andreas Krause and Carlos Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 324–331, Arlington, Virginia, 2005. AUAI Press.

[Krause and Guestrin, 2007] Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1650–1654, 2007.

[Kruger *et al.*, 2007] Dov Kruger, Rustam Stolkin, Aaron Blum, and Joseph Briganti. Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-07)*, pages 4265–4270, 2007.

[Kushmerick *et al.*, 1995] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76(1-2):239–286, 1995.

[Kveton *et al.*, 2006] Branislav Kveton, Milos Hauskrecht, and Carlos Guestrin. Solving factored MDPs with hybrid state and action variables. *Journal of Artificial Intelligence Research*, 27:153–201, 2006.

[Lin, 1965] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44(10):2245–2269, 1965.

[Lindley, 1956] D. V. Lindley. On a measure of the information provided by an experiment. *The Annals of Mathematical Statistics*, 27(4):986–1005, December 1956.

[Littman *et al.*, 1995] M. Littman, A. Cassandra, and L. Kaelbling. Learning policies for partially observable environments: scaling up. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pages 362 – 70, 1995.

[Littman, 1996] Michael L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Department of Computer Science, Brown University, March 1996.

[Loredo, 2004] Thomas J. Loredo. Bayesian adaptive exploration. *AIP Conference Proceedings (Twenty-Third International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering)*, 707(1):330–346, 2004.

[Low *et al.*, 2009] Kian Hsiang Low, John M. Dolany, and Pradeep Khosla. Information-theoretic approach to efficient adaptive path planning for mobile robotic environmental sensing. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS-09)*, pages 233–240. The AAAI Press, Menlo Park, California, 2009.

[MacGregor and Ormerod, 1996] J. N. MacGregor and Thomas Ormerod. Human performance on the traveling salesman problem. *Perception & Psychophysics*, 58(4):527–539, 1996.

[Madras and Slade, 1996] Neal Madras and Gordon Slade. *The Self-Avoiding Walk*. Birkhäuser, 1996.

[Marsaglia and Tsang, 2000] George Marsaglia and Wai Wan Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 10 2000.

[Martin and Moravec, 1996] Martin C. Martin and Hans Moravec. Robot evidence grids. Technical Report CMU-RI-TR-96-06, CMU Robotics Institute, 1996.

[Martinez-Cantin *et al.*, 2007] R. Martinez-Cantin, N. de Freitas, A. Doucet, and J. A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Proceedings of Robotics, Science and Systems (RSS-07)*, 2007.

[Masson *et al.*, 2009] J.-B. Masson, M. Bailly Bechet, and M. Vergassolav. Chasing information to search in random environments. *Journal of Physics A: Mathematical and Theoretical*, 42(43):434009, 2009.

[Mausam *et al.*, 2005] Mausam, E. Benazera, R. Brafman, N. Meuleau, and E.A. Hansen. Planning with continuous resources in stochastic domains. *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 19:1244, 2005.

[McDougall, 1990] T. J. McDougall. Bulk properties of "hot smoker" plumes. *Earth and Planetary Science Letters*, 99(1-2):185–94, 1990.

[McGann *et al.*, 2008a] Conor McGann, Frédéric Py, Kanna Rajan, John Ryan, and Richard Henthorn. Adaptive control for autonomous underwater vehicles. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*, pages 1319–1324. The AAAI Press, Menlo Park, California, July 2008.

[McGann *et al.*, 2008b] Conor McGann, Frédéric Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. A deliberative architecture for auv control. *IEEE International Conference on Robotics and Automation (ICRA-08)*, pages 1049–1054, May 2008.

[McGann *et al.*, 2008c] Conor McGann, Frédéric Py, Kanna Rajan, Hans Thomas, Richard Henthorn, and Rob McEwen. Preliminary results for model-based adaptive control of an autonomous underwater vehicle. *Eleventh International Symposium on Experimental Robotics 2008 (ISER-08)*, 2008.

[McPhail and Stevenson, 2009] Stephen McPhail and Peter Stevenson. Autosub6000 specification and LARS interface to host ship. Technical report, National Oceanography Centre, Southampton, 2009.

[McPhail *et al.*, 2009] Stephen McPhail, Maaten Furlong, Miles Pebody, James Perrett, and Peter Stevenson. Autosub6000 - results and enhancements. In *Oceans 2025 Annual Science Meeting*, 2009.

[Montemerlo *et al.*, 2003] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003. IJCAI.

[Morice and Veres, 2010] Colin P. Morice and Sandor M. Veres. Geometric bounding techniques for underwater localisation using range-only sensors. *Journal of Systems and Control Engineering*, 224, 2010.

[Murphy and Russell, 2001] Kevin Murphy and Stuart Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In A. Doucet, N. de Freitas, and N. J. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.

[Nemhauser *et al.*, 1978] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, 1978.

[Ng *et al.*, 1999] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-99)*, pages 278–287, San Francisco, CA, USA, 1999.

[Ong *et al.*, 2010] Sylvie C. W. Ong, Shao Wei Png, David Hsu, and Wee Sun Lee. Planning under uncertainty for robotic tasks with mixed observability. *The International Journal of Robotics Research*, 29(8):1053–1068, 2010.

[Papadimitriou, 1977] Christos H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4(3):237–244, 1977.

[Paquet *et al.*, 2005] Sebastien Paquet, Ludovic Tobin, and Brahim Chaib-Draa. An online POMDP algorithm for complex multiagent environments. In *Proceedings of the International Conference on Autonomous Agents (AAMAS-05)*, pages 1101–1108, Utrecht, Netherlands, 2005.

[Parson *et al.*, 1995] L. M. Parson, C. L. Walker, and D. R. Dixon, editors. *Hydrothermal Vents and Processes*. Number 87 in Geological Society Special Publication. Geological Society Publishing House, 1995.

[Pêtrès *et al.*, 2007] Clément Pêtrès, Yan Pailhas, Pedro Patrón, Yvan Petillot, Jonathan Evans, and Dave Lane. Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics*, 23(2):331–341, April 2007.

[Pineau *et al.*, 2003] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, Acapulco, Mexico, 2003. IJCAI.

[Pineau *et al.*, 2006] Joelle Pineau, Geoffrey Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *Journal of Artificial Intelligence Research*, 27:335–380, 2006.

[Pirajno, 2009] Franco Pirajno. Submarine hydrothermal mineral systems. In *Hydrothermal Processes and Mineral Systems*, chapter 7, pages 581–726. Springer Netherlands, 2009.

[Poupart and Boutilier, 2004] Pascal Poupart and Craig Boutilier. VDCBPI: An approximate scalable algorithm for large scale POMDPs. In *Advances in Neural Information Processing Systems 17 (NIPS-04)*, pages 1081–1088, 2004.

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

[Rasmussen and Williams, 2006] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[Rocha *et al.*, 2005] Rui Rocha, Jorge Dias, and Adriano Carvalho. Cooperative multi-robot systems: A study of vision-based 3-d mapping using information theory. *Robotics and Autonomous Systems*, 53(3-4):282–311, 2005.

[Ross *et al.*, 2008] Stephane Ross, Joelle Pineau, Sebastien Paquet, and Brahim Chaib-draa. Online planning algorithms for POMDPs. *Journal of Artificial Intelligence Research*, 32:663–704, 2008.

[Russell *et al.*, 2003] R. Andrew Russell, Alireza Bab-Hadiashar, Rod L. Shepherd, and Gordon G. Wallace. A comparison of reactive robot chemotaxis algorithms. *Robotics and Autonomous Systems*, 45(2):83–97, November 2003.

[Saigol *et al.*, 2009a] Zeyn A. Saigol, Richard W. Dearden, Jeremy L. Wyatt, and Bramley J. Murton. Information-lookahead planning for AUV mapping. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.

[Saigol *et al.*, 2009b] Zeyn A. Saigol, Richard W. Dearden, Jeremy L. Wyatt, and Bramley J. Murton. Information-lookahead planning for AUV mapping. Technical Report CSR-09-01, School of Computer Science, University of Birmingham, April 2009.

[Saigol *et al.*, 2010a] Zeyn A. Saigol, Richard W. Dearden, Jeremy L. Wyatt, and Bramley J. Murton. Belief change maximisation for hydrothermal vent hunting using occupancy grids. In Tony Belpaeme, Guido Bugmann, Chris Melhuish, and Mark Witkowski, editors, *Proceedings of the Eleventh Conference Towards Autonomous Robotic Systems (TAROS-10)*, pages 247–254. University of Plymouth, 2010.

[Saigol *et al.*, 2010b] Zeyn A. Saigol, Frédéric Py, Kanna Rajan, Conor McGann, Jeremy L. Wyatt, and Richard W. Dearden. Randomized testing for robotic plan execution for autonomous systems. In *Proceedings of 2010 IEEE/OES Autonomous Underwater Vehicles (AUV2010)*, Monterey, California, September 2010.

[Singh *et al.*, 2009] Amarjeet Singh, Andreas Krause, and William J. Kaiser. Nonmyopic adaptive informative path planning for multiple robots. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 1843–1850, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

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

[Smith and Simmons, 2004] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *Proceedings of the Twentieth Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*, pages 520–527, Arlington, Virginia, 2004. AUAI Press.

[Smith and Simmons, 2005] Trey Smith and Reid Simmons. Point-based POMDP algorithms: Improved analysis and implementation. In *Proceedings of the Twenty-First Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 542–549, Arlington, Virginia, 2005. AUAI Press.

[Sokal, 1995] Alan D. Sokal. *Monte Carlo and Molecular Dynamics Simulations in Polymer Science*, chapter Monte Carlo Methods for the Self-Avoiding Walk. Oxford University Press, New York, 1995. arXiv:hep-lat/9405016.

[Sondik, 1978] Edward J. Sondik. The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs. *Operations Research*, 26(2):282–304, Mar. - Apr. 1978.

[Spaan and Vlassis, 2005] Matthijs T. J. Spaan and Nikos Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[Speer and Rona, 1989] Kevin G. Speer and Peter A. Rona. A model of an atlantic and pacific hydrothermal plume. *Journal of Geophysical Research*, 94(C5):6213–6220, 1989.

[Sridharan *et al.*, 2008] Mohan Sridharan, Jeremy L. Wyatt, and Richard W. Dearden. HiPPo: Hierarchical POMDPs for planning information processing and sensing actions on a robot. In *Proceedings of the Eighteenth International Conference on Automated Planning & Scheduling (ICAPS-08)*, 2008.

[Stachniss *et al.*, 2005] C. Stachniss, G. Grisetti, and W. Burgard. Information gain-based exploration using Rao-Blackwellized particle filters. In *Proceedings of Robotics, Science and Systems (RSS-05)*, pages 65–72, 2005.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[Sutton, 1984] Richard S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, Department of Computer Science, University of Massachusetts, Amherst, MA 01003, 1984.

[Thompson and Wettergreen, 2008] David R. Thompson and David Wettergreen. Intelligent maps for autonomous kilometer-scale science survey. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS-08)*, 2008.

[Thrun *et al.*, 2005] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[Thrun, 2002a] Sebastian Thrun. Particle filters in robotics. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 511–51, San Francisco, CA, 2002. Morgan Kaufmann.

[Thrun, 2002b] Sebastian Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[Tsiligirides, 1984] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[Vergassola *et al.*, 2007] Massimo Vergassola, Emmanuel Villermaux, and Boris I. Shraiman. 'Infotaxis' as a strategy for searching without gradients. *Nature*, 445(7126):406–409, January 2007.

[Vickers *et al.*, 2001] Douglas Vickers, Marcus Butavicius, Michael Lee, and Andrei Medvedev. Human performance on visually presented traveling salesman problems. *Psychological Research*, 65:34–45, 2001.

[Vickers, 2000] Neil J. Vickers. Mechanisms of animal navigation in odor plumes. *Biological Bulletin*, 198(2):203–212, 2000.

[Weissburg, 2000] Mark J. Weissburg. The fluid dynamical context of chemosensory behavior. *Biological Bulletin*, 198(2):188–202, 2000.

[WHOI, 2005] WHOI. Dive and discover: Hydrothermal vents. *Woods Hole Oceanographic Institution website*, 2005. http://divediscover.whoi.edu/vents/vent-infomod.html.

[Wingate and Seppi, 2005] David Wingate and Kevin D. Seppi. Prioritization methods for accelerating MDP solvers. *Journal of Machine Learning Research*, 6:851–881, 2005.

[Yoerger *et al.*, 2007a] Dana R. Yoerger, Albert M. Bradley, Michael Jakuba, Christopher R. German, Timothy Shank, and Maurice Tivey. Autonomous and remotely operated vehicle technology for hydrothermal vent discovery, exploration, and sampling. *Oceanography*, 20(1):152–161, 2007.

[Yoerger *et al.*, 2007b] Dana R. Yoerger, Michael Jakuba, and Albert M. Bradley. Techniques for deep sea near bottom survey using an autonomous underwater vehicle. *International Journal of Robotics Research*, 26(1):41–54, October 2007.

[Zhang and Liu, 1996] Nevin L. Zhang and Wenju Liu. Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUSTCS9631, Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, 1996.