UNIVERSITY OF CALIFORNIA
SANTA CRUZ

A COMPUTATIONALLY TRACTABLE INFORMATION FORAGING ALGORITHM THAT SATISFIES TIME-TO-GO CONSTRAINTS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

Doctor of Philosophy

in

COMPUTER SCIENCE

by

Jeremy Gottlieb

December 2014

The Dissertation of Jeremy Gottlieb
is approved:

Professor Gabriel Elkaim, Chair

Professor Renwick Curry

Professor Alex Pang

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

Abstract

A Computationally Tractable Information Foraging Algorithm that Satisfies Time-to-Go Constraints

by

Jeremy Gottlieb

Robots are frequently being used to explore environments that are largely inaccessible to humans. Most people are familiar with the ground rovers such as *Spirit*, *Opportunity*, and *Curiosity* that are on Mars. Autonomous underwater vehicles are becoming more widely used to conduct oceanographic surveys, particularly in places that people cannot go, such as under the polar ice caps. A large part of increasing the utility of these robotic missions will involve giving the robots significantly enhanced autonomy, particularly when it comes to planning precisely what aspects of the environment to survey and sample.

Much of the research on robotic path planning has focused on finding minimal cost paths between two points, and being able to rapidly re-plan when environmental conditions change. Less research has been done into how robots should plan paths that allow them to maximize utility, such as information gathered. One of the primary reasons for this is that finding the optimal solution to the maximization problem is NP-hard, and thus cannot be computed in a tractable amount of time, much less in real time.

This dissertation presents a computationally tractable method for allowing robots to plan paths that can approximately maximize information gathered while satisfying any relevant horizon constraints, such as a deadline. It builds atop existing algorithms known to be capable of rapid planning and re-planning, such as D*. The algorithm utilizes these methods for computing lowest-cost paths between nodes in the environment, and then planning which nodes to search based on a function of both the cost of exploration and the expected amount of information contained at the nodes. Experimental results in small environments show that the algorithm will mostly generate paths that are at least 75% of optimal.

This dissertation is dedicated to my daughters, Phoebe and Eleanor, in the hopes that they will realize it is never too late to determine what you really want to do in life.

Acknowledgments

I would like to extend my sincere gratitude to my advisor, Dr. Gabriel Elkaim, who has been able to mentor me through the process of becoming an engineer, while being very understanding of the difficulties involved with doing so while starting a family.

Many humble thanks are also due to Dr. Renwick Curry, who has been an invaluable mentor throughout the research and dissertation process.

I would also like to thank Dr. Alex Pang, who agreed to serve on my committee at the last minute.

Overall, this dissertation would probably never have been finished without the mentoring, understanding, and forebearance of these fine committee members.

Lastly, I will never be able to fully express the gratitude I feel for my wife, Kate Lockwood. It was her idea for me to go back and get a second Ph.D. in computer science after my having, for many years, said that I wished I had gone into robotics instead of psychology. Throughout the process she has been an invaluable partner, taking on more than her share of the child-rearing and home making when circumstances required it. It is not an understatement that none of this would have happened without her support, and that myself, and our entire family, would be lost without her unwavering hand at the tiller.

Chapter 1

Introduction

Advances in artificial intelligence (AI) and robotics allow us to create machines that can explore areas that are difficult, if not impossible, for humans to explore. This is primarily due to the ability of machines to operate with significantly more autonomy than was previously possible. This enhanced autonomy means that robots can be sent on longer-duration missions with more vaguely defined mission goals and still be trusted to gather useful information and return safely. As a result, robotics researchers are beginning to focus on enhancing the planning capability of autonomous vehicles to enable them to engage in such long-term autonomy missions to places where humans are unable to provide much direct control to the vehicle, such as under the ocean surface or on distant planets.

One of the key requirements of such capabilities is that they must be computationally tractable such that a vehicle can execute them in real-time or near real-time. If a rover on Mars can operate fully autonomously but requires an hour to plan every single step, it would be better to simply let humans control it, since operational decisions would be made more quickly. Likewise, an autonomous underwater vehicle (AUV) mapping the ocean under the ice pack is constantly in motion, meaning the amount of time it can take to plan is limited. However, safety cannot be sacrificed to gain computational speed.

Much of the robotic planning research has thus far focused on improving the ability of robots to operate safely and at lowest cost in increasingly ambiguous environments. Minimal cost

solutions have long been mathematically known to be computationally tractable for a large subset of planning problems (e.g., , [29, 35, 44, 62]). However, as robots are asked to make more decisions for themselves, and planning problems get more complicated, research is starting to shift towards finding tractable methods for generating "good enough" solutions to intractable problems.

The research described in this dissertation is of that character. As we envision mobile robots being sent to gather scientific information, they are going to need the capability to make decisions about where to explore. Then they will need to be able to plan routes through the environment that allow them to maximize the number of places they can explore, and thus the amount of information they are able to collect. Making these decisions and planning these routes is computationally difficult. This dissertation presents a method for simplifying these processes to make them tractable, particularly in a real-time robotic system.

Let us start by examining the state-of-the-art for two different robotic platforms actively being used to gather scientific information: AUVs operating in our oceans and ground rovers working on Mars.

Autonomous Underwater Vehicles

Autonomous underwater vehicles (AUVs) are becoming an increasingly popular tool for oceanographers to use for studying the oceans. AUVs have many advantages over traditional ship-based sampling methods. Among the most significant is lower cost. Time aboard a ship can cost many tens of thousands of dollars per hour (or more, if the researcher does not have direct access to one). While custom-built AUVs can have high up-front capital costs (see [88]), several companies have in recent years started offering low-cost off-the-shelf options, such as the EcoMapper [1].

Thus far, most AUVs operate by following GPS waypoints pre-programmed by oceanographers. The range of missions that they are used for has expanded significantly as their capabilities have improved. Initially they were used largely for mapping the ocean floor with sonar [52, 53]. This required very little autonomy on the part of the AUV as it was generally programmed with

a depth and a lawnmower pattern to follow for it to fully map the patch of ocean floor scientists were interested in. These types of applications are still make up a large number of AUV missions, as witnessed by the recent use of the *Bluefin 21* to assist in finding Malaysian Airways 370.

More recently, AUVs have been mounted with standard oceanographic sensors, such as CTD (conductivity, temperature, depth) sensors (e.g., [88, 67, 105]) and mass spectrometers ([51, 60]) to do high frequency data collection. Sibenec et al. [88] describe an AUV that can even collect water samples and return them to shore.

As both the data collection payloads and computational power available onboard have improved, researchers have been actively examining mechanisms for allowing AUVs to analyze the collected data in real-time to identify ocean features as they fly through them (e.g., [39, 26, 59, 25, 46]). This can then be leveraged into methods that allow AUVs to construct their own plans for where they should go and what kind of data they should collect so as to satisfy broad mission parameters, as in [69, 68, 79].

This expanded autonomous capability will become more important as two classes of vehicles become more prevalent. First, there is significant active research into how to extend the mission lifetime of AUVs. The *Dorado* class of AUV described in [88] and other torpedo-sized AUVs typically have a mission lifetime of 24 hours or less. Bellingham et al. [10] describe an AUV with a theoretical mission life that currently stands at about two weeks, with a goal of increasing that to 30 days. There is also a second class of vehicle that is a hybrid AUV and autonomous surface vessel (ASV) called a glider. The best known of the glider class of vehicles is the Wave Glider from Liquid Robotics [45, 66] which functions more as an ASV, though there are also gliders that operate completely as AUVs (e.g., [92]).

Wave Gliders get their propulsive power from the wave action of the ocean (hence the name), and thus do not require any internal power for propulsion. This means that with a small set of solar panels on the surface to power its payload, a glider can theoretically stay in the water in perpetuity.

Liquid Robotics has driven Wave Gliders from California to Australia completely autonomously [2]. Other gliders manipulate buoyancy for propulsion. These platforms provide the possibility of ocean monitoring missions that can last for months, or even years, providing continual streams of data. If they live up to their promise, they are likely to significantly alter how oceanographic research is conducted.

Rovers

Ground rovers conducting long-term scientific missions have received publicity in recent years due to the successes of the Mars Expeditionary Rovers (MERs), *Spirit* and *Opportunity*, along with the more recent *Curiosity* rover. *Opportunity* has the distinction of having been working on the surface of Mars, conducting science and returning data, for over 10 years as of this writing[1]. *Curiosity* is a larger vehicle that does not depend on solar panels for its power. Thus, it's capabilities are significantly greater, and it is hoped that it will last as long as *Opportunity*, if not longer.

However, despite their amazing success, the Mars rovers are limited in what they can accomplish by the fact that they have very limited autonomous capabilities [14]. While the rovers have the capability to operate fully autonomously, that capability has rarely been used during their missions [13]. This is primarily a function of the need for absolute safety as these vehicles drive around. Should something happen to a rover on Mars, such as when *Spirit* got stuck in the sand in 2009, there is no recourse to correct the issue.

As expensive as they are to build, launch, and land, rovers are still the cost-effective mechanism for exploring terrestrial bodies other than Earth. As larger rovers like *Curiosity* become the model, the types of sensor suites rovers can carry will expand significantly [41, 15]. As a result, the more capabilities a rover has the more possible mission goals it can attempt to address. Balancing these mission goals and making intelligent decisions about where to explore and what to sample so as to maximize goal satisfaction will only increase in importance as rovers are able to do more things.

[1]Note that *Spirit* and *Opportunity* were originally designated for 90 day missions. *Spirit*, got stuck and stopped working after "only" six years.

1.1 Long Term Autonomy

To a large extent, this lack of autonomy is a somewhat manageable problem when dealing with rovers on the Moon or ASVs on the ocean surface. There is very little communications latency, meaning the vehicles can be constantly monitored. Thus any problems that arise can be quickly addressed. The robots can even be teleoperated when necessary. However, robots outside the immediate vicinity of Earth or underneath the ocean surface have significant communications latencies that makes such operations infeasible. Round trip communications to Mars range anywhere from 10-50 minutes, and are impossible when Mars is on the other side of the Sun.

AUVs are nearly impossible to communicate with when they are submerged unless there is a surface vessel equipped with an acoustic modem in their immediate vicinity (which eliminates the cost benefits of using an AUV in the first place). Most AUVs surface at regular intervals to communicate via satellite, but this has to be balanced with the fact that AUVs are at their most vulnerable when they are at the surface, especially if there are other boats in the area. Thus most missions only surface briefly every 20-30 minutes, usually to check in with shore and get a GPS fix. At all other times the AUV is completely out of contact.

The communications problem only gets worse when sending robots farther afield in our solar system, or to inaccessible places on Earth. For example, there have been AUV missions to conduct sonar surveys of the underside of both the Arctic and Antarctic ice packs (e.g., [32]) for the purposes of better measuring ice thickness. For the duration of these missions, there were no communications with the AUV; it had to operate completely autonomously.

In this circumstance, as with the MERs, the autonomy of the AUV was limited to simply navigating between way points. While this gives the robot the maximum likelihood of safely navigating the environment, it significantly constrains what kind of missions it can undertake. Even if the robot finds something that would be of significant interest to its operators, it does not possess the capability to react to this new information. Because it is out of communications range, it cannot

call back to the ship to ask if it should modify its mission. Thus, a potentially valuable opportunity could be lost.

As a result, there is a line of robotics research that is getting more attention that specifically focuses around the issues associated with robotic planning in the context of long-term autonomous missions (e.g., [9, 80, 92]). Much of this research is focused on the problem of how to give a robot the capability to understand broad mission goals, and to then use those broad goals to make its own decisions about how to adjust its behavior to maximize achievement of those goals.

For example, consider a rover on Mars that has been tasked to drive towards a particular rock formation some distance away. Under the current planning regime, the robot will drive a short way towards the rock formation, then send pictures back to Earth and wait for an all clear and instructions about what point to drive to next. This is necessarily time-consuming, and information almost certainly gets missed between the moments when the robot takes the pictures.

On the other hand, suppose the rover has the capability to visually determine whether a particular rock might be interesting with regards to a broad mission goal to, "find evidence that liquid water once flowed on Mars." A rover with the ability to make autonomous decisions about where to drive and which rocks to examine could still start off following a human-determined path. However, while driving it can be constantly scanning its environment. Then, if it sees a rock it determines is interesting, it would have the ability to break from the human-determined path and make its own decision to go sample that rock, before continuing to its original goal location.

A rover with this capability would have two significant advantages over the current state-of-the-art. The first is that this rover could make faster progress across the terrain. Without having to stop as frequently to check with Earth and receive instructions it can move more continuously, and thus make faster progress, even if its velocity of movement stays the same as it is now. The second major benefit would that information a scientist might find interesting and relevant to the mission goals is less likely to be missed. If the rover is constantly scanning and making decisions about what

to sample, information is less likely to be overlooked during the periods between communication with Earth.

1.2 The Information Gathering Problem

In this context, the information gathering problem is a central piece of the larger puzzle of enabling effective long-term autonomy. Put simply, the problem is this: Given a map of the environment, a destination, and operational constraints (if any), how can a robot plan a path through that environment that satisfies the constraints while maximizing the amount of scientific information it will gather?

There are, of course, complicating factors. The most important is the nature of the map. The most effective use of a robot for exploration is in environments that are inaccessible to people. In all likelihood, this means that there are no maps of the environment that are detailed or certain enough for a robot to generate a perfect plan and then execute it. More typical is that the robot will have access to a map generated by processing low-resolution[2] remote sensing imagery, such as satellite imagery (e.g., [86]).

From the perspective of the robot, this map will have two faults. The first is that, in the best case, the resolution of the map will be roughly the size of the robot (meaning the robot would take up one pixel in an image). Usually it will be less (i.e., the robot is much smaller than a single pixel), often significantly less. This means that the remote-sensing map will not show many of the obstacles the robot needs to avoid, such as boulders or holes/depressions that are some significant fraction of the robot's size. For successful autonomy these factors must be accounted for by on-board processing of the robot's sensor data.

The second fault of the remote-sensing maps is that the processing of these images rarely gives definitive answers. Usually it gives stochastic estimates, such as: This pixel (or group of pixels) as a 75% likelihood of containing the information or object you are interested in. Thus, the path the

[2]Low-resolution relative to the sensors on board the robot.

robot generates is not just one that tries to visit as many information-containing pixels as possible, but rather one that has to be selective and choose the path that will maximize the *expected value* of the information it will accumulate across the entire path. As will be shown in future chapters, this problem is fundamentally intractable (NP-hard).

1.3 Outline of the dissertation

This dissertation presents the research and development of the Information Foraging Algorithm (IFA). The IFA is a computationally tractable means by which a robot can make rapid, intelligent decisions about where to forage for information, while still managing to satisfy a deadline or other constraint on its operations. Chapter 2 will summarize common robotic path planning algorithms and discuss why they are or are not good approaches to solving the information gathering problem. Chapter 3 describes the IFA and how it is specifically applied to the information gathering problem. Chapter 4 presents a number of simulations performed to test the performance of the IFA both relative to optimal performance and to demonstrate its ability to enable fast decision making even in large environments. Finally, Chapter 5 summarizes the results and presents future directions for improvement of the IFA.

Chapter 2

Related Work

There is a long history of research into the problem of robotic path planning. As with most fields, the research has evolved over the years, addressing the simplest cases in the beginning, getting increasingly more complex over the years. Most recently, with programs such as the DARPA Grand Challenge and Urban Challenge ([47, 97, 98, 99]) as well as the Google Car project ([30, 70]), the advances in route planning capability have been significant.

However, traditional path planning research has generally been focused on generating minimal paths, i.e., ones that minimize a cost function, such as time, fuel, or control effort. Some of the reasons for this focus center on the intractability of most versions of maximization problems. There is also a divide between the path planning efforts of the artificial intelligence (AI) community, which typically are derived from broader planning methodologies and depend on a level of discretization, and the types of route-generation research that comes from control theory. Control theoretic algorithms are also generally trying to minimize costs within a set of constraints, but do so within the context of continuous state spaces. As a result, the routes generated by control theory approaches also tend to explicitly take into account the vehicle dynamics, which are frequently either ignored or simplified in the AI literature.

In general, though, all path planning systems (and planning systems more generally) are representing the environment (including the vehicle) at any given point in time as a set of values associated with a vector of variables known as a *state vector*. Fundamentally, the purpose of any

planning system is to determine the sequence of actions that will transform some initial state, which we will call the *Start*, into some final state (*Goal*) subject to constraints on the values the state variables can have and how they can evolve.

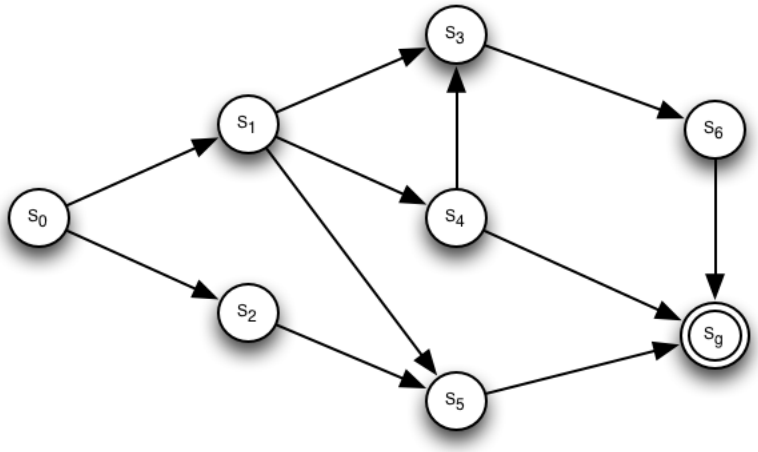


Figure 2.1: Simple state space example. Start = S_0; Goal = S_g

This means that path planning algorithms have three phases to them. First, the initial plan is constructed based on the initial knowledge the robot has about the environment. The second phase involves plan execution, namely navigating towards the goal. The third phase is replanning that takes place as a result of new information about the environment being gathered from the robot's sensors.

Ideally the initial plan construction should be relatively fast — the state might change if the robot spends hours developing its initial plan — but it does not need to be instantaneous. As a result, this phase usually occurs while the robot is stationary. The construction of the initial plan also tends to involve doing as much of the work to support the other stages as can possibly be done now. For example, if replanning is going to involve knowing the shortest path from every point in the environment to the goal, then as many of those as possible would be computed during the initial planning phase instead of while the robot is moving.

Once the plan is constructed, the robot needs to execute it by navigating through the

environment. As it moves it will gather new sensor data that needs to be integrated into its understanding of the environment. Because the robot is moving, data integration needs to occur in real time. If the robot is too slow at processing sensor information then its state might change significantly in a manner it is not aware of.

This can happen in two ways. The easiest to understand is that the robot is moving while it is processing data. If a vehicle traveling 60 miles per hour takes 2 seconds to process its sensors, it will have traveled almost 60 yards in that time span. For a ground vehicle the difference in terrain could be significant. Likewise, for an AUV that could be the difference between water that allows a sufficient operating depth and water that is too shallow. Thus, the faster the vehicle moves the faster its sensor processing needs to be.

The second manner in which vehicle state might change involves the rate at which the sensors are gathering information. If a vehicle's sensors are operating at 1Hz, meaning they take one new measurement per second, then a two second processing cycle means that half the information gathered by the sensors must be ignored. Thus, sensor processing needs to occur as rapidly as the sensor with the highest sampling frequency.

Finally, if this sensor processing indicates significant differences between what the robot expects to see in the environment and what it actually sees, it will need the capability to rapidly develop a new plan of action taking into account this new information. Given that things can change often, especially in environments that are dynamic or where the robot's information is inherently incomplete, replanning also needs to be a fast process so that the robot does not spend the bulk of its time sitting around and planning.

Virtually all of the research into robotic path planning focuses on improving capabilities in one or more of these three areas. For the most part, path planning research itself focuses on the planning and replanning phases, while sensor processing and integration is the focus of research into processes such as sensor fusion and SLAM (simultaneous localization and mapping), which this

dissertation will not be addressing further.

2.1 Traditional Path Planning

Initial path planning research focused almost exclusively on the path generation phase. Typically the environment had polygonal obstacles, and the vehicle dynamics were highly simplified: constant velocity, instantaneous direction changes, etc. These simplifications have the benefit of reducing the state space to something computationally tractable. The environments were also static and completely known, which eliminated the need for replanning while the vehicle was navigating. Figure 2.2 shows one simple example.

Figure 2.2: Deterministic planning problem.
Move from S to G while avoiding the black areas.

To further streamline the problem, traditional path planning mechanisms impose two other simplifications on the environments in which they operated. The first is to assume that the environment was completely known *a priori*. Thus, every region of the environment is either known to be safe for the vehicle or is an obstacle, which is any unsafe region. The second simplification is to discretize the environment. Continuous environments are more computationally difficult to process

simply because there are now an infinite number of steps along the path. Optimization problems in continuous environments have to be solved either by integrating a cost function over a given path, or by differentiating a cost function over the entire environment. While these are the methods used by optimal control theory approaches (discussed in Section 2.2.2), any environment with obstacles is going to be difficult to represent in this fashion due to the discontinuities introduced by the obstacles' existence.

The other advantage to discretizing the environment is that path planning systems can then leverage existing graph theoretic search mechanisms to find lowest-cost paths through the environment. There are known polynomial-time algorithms, some as good as $\mathcal{O}(n \log n)$, for finding lowest cost paths through graphs. The research on these problems tends to focus on three interrelated areas: (1) The most efficient way to construct a graph from an environment, (2) How to reduce the size of the graph without losing important information about the environment, and (3) More efficient methods for replanning when new information is gained about the environment.

2.1.1 Graph Construction

There are multiple approaches to partitioning an environment into some sort of graph. The trade-off between methods is usually capturing the salient details of the environment within its structure versus the complication of building the graph. For example, the simpler the graph is to build, the less the structure of the graph will relate to the structure of the environment. Several of the most popular methods for graph construction are visibility graphs, Voronoi diagrams, and grids. Details of each are provided below.

Visibility Graphs

Visibility graphs try to avoid the problem of choosing a resolution by constructing a graph out of the obstacles in the environment. The graph that is constructed has as its vertices the start and end points as well as all the corners of the objects in the environment. Edges are constructed between any two vertices that have a straight line-of-sight (see Fig. 2.3). Navigation then proceeds

along these edges. In an environment with obstacles, visibility graphs generate optimal shortest paths through the environment that avoid the obstacles. Safety margins can be achieved by simply extending the boundaries of the obstacles by an appropriate amount and computing the graph based on these enhanced obstacles.

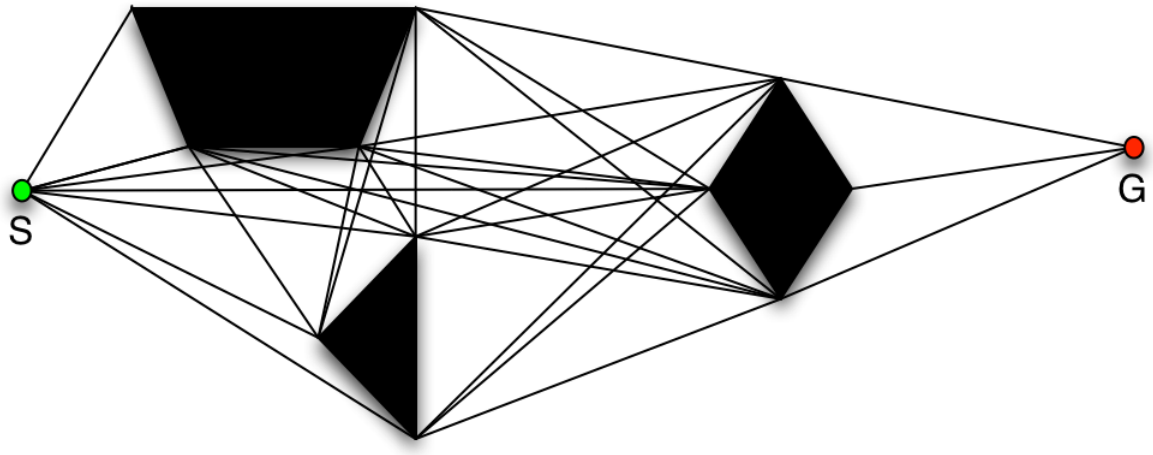


Figure 2.3: Simple visibility graph. Start = S; Goal = G

Visibility graphs can be good in simple, relatively uncluttered environments. However, the graphs are expensive to compute. If obstacles are not regular polygons, it can be difficult to determine if a given arc between vertices intersects a small part of an intervening obstacle. There can also be difficulties with curved obstacles in that there are potentially infinite vertices. Typically, the arcs are drawn as either tangent lines or connecting nearest points, but calculating these tangent lines can also be difficult if the curved part of the obstacle cannot be represented or approximated functionally.

One drawback to graph-based methods generally is that, in order to simplify the planning process, each arc is usually treated as a single, consistent piece of the pathway. Otherwise, one possibility is the potential would exist for a given arc to have multiple possible costs. This effectively increases the number of arcs in the graphs, which increases the complexity of the planning process. Another possibility would be to include at least some aspects of vehicle dynamics (such as velocity).

However, graphs do poorly as the dimensionality of the planning space increases, primarily because the number of arcs in the graph starts to get very large. Thus, graphs tend to oversimplify both the terrain and the vehicle dynamics. In general, it is expected that the vehicle will travel at a single velocity along a given arc, as if all the terrain along that arc were consistent.

The requirement can be especially problematic in visibility graphs. Because the arcs are connecting the vertices of obstacles, some of the arcs in a sparse environment might be very long. Either the cost of the arc must be based on the slowest maximum velocity the vehicle can achieve along the arc, or new nodes must be inserted to allow for different velocities to be chosen along its length. As a result, like most planning algorithms, visibility graphs tend to have their best success in constrained, relatively consistent environments such as an office or a warehouse. However, in environments where there might need to be frequent changes in direction or velocity, or where such changes would lead to more efficient routes, visibility graphs perform poorly by virtue of generating extremely complicated graphs.

Visibility graphs would not be an appropriate solution to the information gathering problem. The main reason is that they are specifically designed to generate optimal solutions to cost minimization problems in environments with obstacles. Within that context they are guaranteed to generate optimal, lowest cost paths between two points. However, there is no obvious mechanism by which they could be applied to a maximization problem. Because generating the graphs is costly, they are not ideal for environments that will require any sort of pre-planning, which will be true of any kind of dynamic environment or one where there is uncertain knowledge about the environment.

Voronoi Diagrams

A Voronoi diagram is constructed by dividing the environment into regions defined such that every point within a region is closer to the center of its region than the center of any other region, as in Figure 2.4. In an environment with obstacles, this effectively creates a graph where each point along the boundary between two regions is equidistant from the obstacles at the centers

of those regions. Navigation then takes place along the edges of the Voronoi graph. This has the advantage of guaranteeing that the robot will always be as distant as possible from any obstacles along its route. Whereas a visibility graph will guarantee an optimal lowest-cost path between two points, a Voronoi diagram will generate the safest possible path for navigation between those points [23], but this path will be longer than the one generated by a visibility graph.

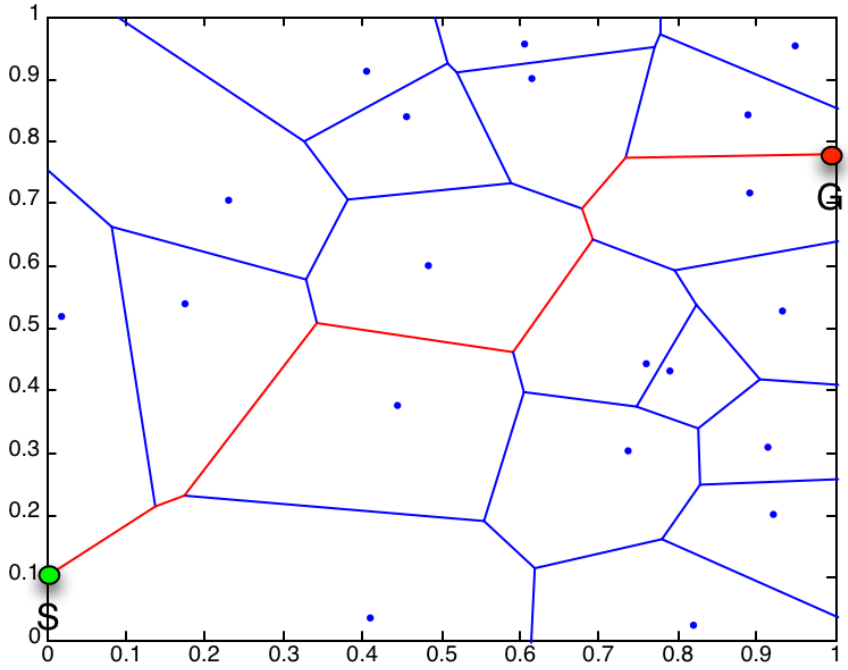


Figure 2.4: Voronoi Diagram example
Red path is the solution.

As with visibility graphs, Voronoi diagrams are very useful at generating paths that avoid specific regions of the environment, but less useful for generating paths that actively seek to traverse or search regions. They have many of the same drawbacks, such as treating each arc as a homogenous bit of terrain and vehicle dynamics. Their primary utility is in guaranteeing the safety of the vehicle.

Voronoi diagrams are also expensive to generate, rendering them less useful for environments where replanning is necessary. They are especially difficult to generate if the obstacles being avoided cannot be represented as points. Research on how to generate Voronoi diagrams relative to polygonal [100] and elliptical [73] obstacles is ongoing. It is not clear how computationally difficult

it would be to generate such diagrams from arbitrarily shaped obstacles.

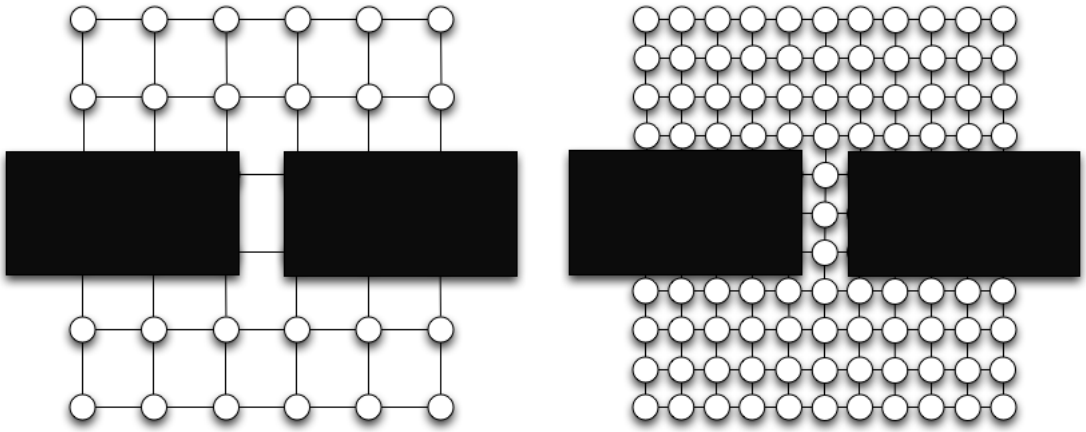
Grids

The simplest mechanism is to impose a uniform grid on the environment. This generates a graph where all the nodes are equally spaced horizontally, though there may not be equal costs for traveling between adjacent nodes. It is simple because the generation of the graph itself does not depend in any way on the structure of the environment, meaning it can be computed very quickly. On the other hand, the graphs generated by making a grid tend to have many more nodes and edges than either visibility graphs or Voronoi diagrams, meaning the route planning process is more computationally expensive.

If the nodes are spaced far apart, (i.e., the grid has low resolution) there will be fewer nodes and edges in the graph, meaning solutions can be generated more quickly. However, if nodes are too far apart, the paths generated may be sub-optimal. In the worst case, if the valid paths between obstacles are small, no solution may be found because the grid will not have sufficient resolution to locate these narrow paths (see Figure 2.5a).

On the other hand, a grid with high resolution will be better able to find paths through tightly packed obstacles, but it will be computationally more difficult because there will be that many more nodes and edges to deal with (see Figure 2.5b). For example, if grid A has twice the resolution of grid B (i.e., the space between nodes is half as much in grid B as in grid A), grid B will have four times as many nodes and roughly four times as many edges, which will generally increase the computation time from 6 to 16 times depending on the exact implementation being used. Thus, the primary difficulty with grid-based methods is determining the proper resolution to use.

While grids are the simplest method to construct, they can also be the most robust. With Voronoi diagrams and visibility graphs, the locations of the nodes and the lengths of the arcs are determined by gross physical features of the environment. In other words, the less cluttered the environment, the fewer nodes there will be, and the longer the arcs connecting them will be.



(a) Low resolution grid (b) High resolution graph

Figure 2.5: Different grid resolutions relative to closely space obstacles.

Difficulties arise when these methods are applied in non-uniform environments, i.e., ones where the traversal cost for the robot varies with the underlying terrain, even if the robot is continuing to move in a straight line. Because a grid places its nodes and edges without regard to the physical structure of the environment, the costs of the edges can be more easily manipulated based on data about the underlying terrain, and even changed by the robot as it navigates through the environment.

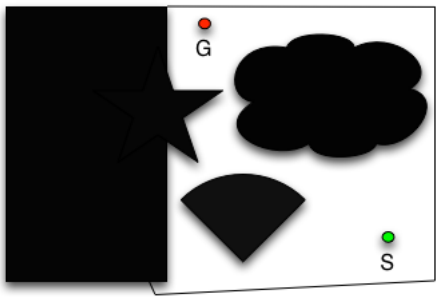


Figure 2.6: Crowded environment.

Grids lack robustness in tightly packed environments. For example, in Figure 2.6 it should be obvious that the optimal path goes between the obstacles, but the grid necessary to generate such a path either does not exist or must have such a high resolution that searching it will be very

costly. However, grids can be very effective in the types of outdoor environments that in which scientific robots are operating. As will be seen in Chapter 3, this is one of the primary reasons why the Information Foraging Algorithm uses a grid representation.

2.1.2 Planning and Replanning

The primary reason that classical planning techniques focus on mechanisms for creating a graph is so that they can leverage traditional graph search techniques for the actual route planning itself. Djikstra's algorithm [29] was initially popular, and the A* algorithm [44] was developed primarily to try to improve upon the search times that Djikstra's algorithm required.

In so far as a minimum cost path needs to be generated through a known environment, these algorithms work perfectly fine. However, it is extremely rare for environments to be completely known. Typically, either some aspect of the horizon beyond the robot's sensor horizon is not known, or the robot's position in the environment is not perfectly known, or both. As a result, some sort of replanning is usually needed as the robot gains information about the environment, its position, or other state variables that may be relevant.

Much of the literature related to replanning has been based on improving performance. In particular, an area of significant focus has been how to target the replanning only to those parts of the environment that are relevant to the robot's path. For example, if the goal is to minimize the time it takes to get to the goal, then once a robot has moved closer to the goal it should not retreat from the goal unless that reduces the total cost to the goal, such as if an unknown obstacle is blocking the robot's path.

The D* algorithm

One of the most popular algorithms in use for robotic path planning and fast replanning is the D* family of algorithms [93, 95, 94, 54]. D* is short for "dynamic A*" to highlight how the algorithm derives its similarities from A*. D* (and it's relatives, Focused D*, Field D*, and D* Lite) encompasses both planning and replanning with the same algorithm. Like A*, D* is an incremental

search algorithm that use heuristics to guide which node will be explored next in the process of finding the lowest cost path.

In its initial planning phase, it operates in essentially the same manner as A*. Based on this initial path, the robot starts driving through the environment. It continues along the initially planned path until it senses some aspect of the environment that is different from what it has stored in its internal map, such as an obstacle that is not represented. When it detects such a change, it replans based on the new information and then continues. This replanning step is where the power of D* lies in that it has been shown to be more efficient than simply performing repeated applications of A* [93].

The main advantage that the replanning system in D* has over A* comes in the number of nodes it examines when it detects something different. In particular, if A* is run to find the shortest cost path between a node and the goal, it must continue to run until the goal node has a lower cost than any of the nodes in the A* priority queue. The goal could be on the queue but skipped if there are lower cost nodes on the presumption that those lower cost nodes might lead to a lower cost path to the goal, even if those nodes can only utilize pathways that were shown in the initial planning phase to ultimately have a higher cost than the one already known to lead to the goal.

D*, on the other hand, utilizes the information it generated in the initial planning phase such that each node remembers the exact cost between itself and the goal. What this means is that D* does not need to continue to search and update until it establishes the goal as the lowest cost node in the queue. It only has to search until it finds the set of nodes whose cost-to-goal is unaltered by the new information the robot has gained. At that point, it is not possible for the algorithm to discover a new, shorter path to the goal. Thus, D* just connects the updates it has done to the already existing information in the unaffected nodes and stops. This means it ultimately examines fewer nodes during its replanning phase, and thus is more efficient than repeated applications of A*.

Other replanning systems

D* was developed and has been used as a generic planning system that can operate in any environment. If one is willing to sacrifice this generality for algorithms that are specifically tailored to particular environments, then one can take advantage of regularities in those environments to generate ad-hoc but efficient planning and navigation systems.

One of the more popular environments for which to develop different types of navigation systems is the office building environment. This environment is well contained, can be relatively well known ahead of time, and it is highly structured. We will explore in more detail about the office building environment when we discuss partially observable Markov decision processes in Section 2.3. However, one example is Kim et al. [50] where they have used visual information to generate the navigation graph as the robot moves around the office building.

Similarly, Djikstra's algorithm, A*, and D* all operate by searching in one direction — either from the start to the goal or vice versa. Other algorithms attempt to gain performance by searching in both directions simultaneously (e.g., [83]). This has the advantage of reducing average case search times (though not worst case times) at the expense of making the stopping conditions much more complicated. In particular, such a system is unlikely to have any advantages in replanning over D* because the backwards search from the goal to the robot will mostly be traversing nodes and arcs that are unaffected by the new data that has been discovered in the immediate vicinity of the robot.

2.1.3 Path smoothing

There are two major drawbacks to discretizing the environment. The first is that, except in the case of visibility graphs, the paths being generated are not strictly optimal. For example, path planning through a grid is necessarily going to force the robot to follow the grid lines, even if the center of the grid square can also just as easily be traversed. Likewise, traveling through a Voronoi diagram will constrain the robot to follow the graph lines, even if a node could be safely skipped

because it is far away from any obstacles. However, unless the environment is structured such that one of the continuous path planning mechanisms discussed in Section 2.2 can be used, there is no computationally simple (or even tractable) way to avoid discretizing the environment.

The second drawback is that the paths are not smooth. No vehicle can make the type of instantaneous 90-degree turn that would be required for it to strictly follow a path along the lines of a grid without stopping. Thus, there must be some sort of smoothing process that takes the disjointed, edge-following path generated from the graph and converts it into a path that can actually be followed by a robot and actually takes into account constraints on vehicle dynamics.

We will be ignoring this facet of route planning in the algorithms presented later, so we will only mention two of the most popular methods for accomplishing this smoothing. One is to generate the plan and then smooth out the transitions between graph segments using a curve generating algorithm, such as Bezier curves [104, 102, 82] or continuous curvature deformation [72]. The other is to attempt to build curves into each of the corners of the graph that are consistent with the dynamic constraints of the vehicle, such as in [77].

2.2 Continuous Path Planning

In the real world, robots do not operate in discrete domains. The primary advantage to discretization — allowing for intractable problems to be solved, or at least well-approximated — is significant enough that the methods still see widespread usage, even in robots being deployed in the field. Where possible, though, it would be ideal to solve the path-planning problem over the continuous environment.

Most of the continuous methods in common use come from the realm of control theory. The basic premise is to represent the environment, and ideally the vehicle mechanics, as part of a multi-dimensional set of differential equations (or difference equations if time must be discretized).

These types of continuous path planning systems have numerous advantages over discrete

planning systems. The most obvious is that they will generate a globally-optimal path. That is to say, the path generated by an optimal, continuous algorithm is guaranteed to be the very best path that can be generated. This contrasts with the discrete planners, where the path generated is only optimal relative to the discretization imposed on the environment. If the cost function has a finite maximum, continuous planning algorithms can find that maximum as easily as they can find minimums. Thus, for the types of maximization problems such as the information foraging problem, where the expected maximum amount of information is known, these methods might, in principle, generate an exact solution to the problem.

2.2.1 Potential Fields

Potential fields are constructed to be analogous to the idea of physical potentials, such as electrical potential or potential energy. In the physical cases, objects want to move towards the bottom of the potential well. A simple example is a ball on top of a hill. At the top of the hill, the ball has maximum potential energy. Entropy dictates that the ball attempt to move to a state of minimum potential energy, namely the bottom of the hill. Problems can arise if the ball gets trapped in a local minimum, such as a hole in the side of the hill.

Potential fields for path planning are constructed on the same principle, only now they assign potentials to states in the state space. As the goal is the desired state for the robot to eventually end up in, it is given a strong negative potential. Since the obstacles are areas the robot needs to avoid, they are given positive potentials. The other areas of the state space are given potential values based on some function of their distance from the areas with assigned potentials [48].

How the potentials are calculated and propagated through the state space is the subject of a significant amount of research (e.g., [4, 101]). Regardless of how they are assigned, navigation through the potential field is just gradient descent, where the robot follows the path of steepest descent through the state space. Potential fields have demonstrated great success in, for example,

enabling real time obstacle avoidance as a robot moves through a cluttered environment [16].

Part of the power of potential fields is that they can be used to encompass not just the physical environment itself, but also the constraints on robotic motion and desired states for the robot to be in. They can also be used to designate desirable non-goal states that the robot should want to pass through by giving those states negative potentials. For this reason, potential fields have been used not just for mobile robot navigation, but also have seen use in robotic arm manipulation (e.g., [49]).

As with any gradient descent algorithm, the primary failures of the algorithm are the possibility of getting trapped in a local minimum without reaching the goal state. Years of experience with gradient descent based learning algorithms such as back propagation [81] show that this problem only gets worse as the dimensionality of the potential surface increases. Obviously, the more variables are included in the state space, the higher its dimensionality will be. The possibility of getting caught in a local minimum will also increase the more convoluted the potential surface is. In other words, the more obstacles and desirable states are contributing to the total potential of the various points in the state space, the more bumps and ridges the potential surface will have, meaning more local maxima and minima [12, 38]. While there have been attempts to develop potential field algorithms that eliminate local minima (e.g., [63]), they have thus far met with limited success.

The types of techniques that can be used to overcome this failure are similar with path planning algorithms as they are with other gradient descent techniques. These include taking random sized steps down the gradients so as to be certain that the minimum discovered is global. There are also heuristic approaches similar to A* that take into account in what direction, or even exactly where, the goal state is within the state space and bias the robot to move in that direction, even when a minimum has been achieved. None of these are guaranteed to solve the problem. Interestingly, the same authors who used potential fields to such success in [16] later abandoned the approach due to increasing problems as they tried to use their algorithms in more complicated state spaces, both

with local minima and with other issues [57].

In principle, there is no particular reason why the potential field could not be inverted, such that desirable areas have high potential and the robot performs gradient ascent instead of gradient descent. The same issues still exist, but this visualization lends itself more naturally to maximization problems. As we shall see in Chapter 3, a version of this concept is used in the development of the Mass metric (Section 3.2) to help the Information Foraging Algorithm determine which node it should navigate towards next. Primarily for the reasons outlined above, however, a full potential field implementation was not used.

2.2.2 Optimal Control Theory

Optimal control theory is concerned with how to determine the set of control inputs to a system to satisfy some optimality criteria, subject to constraints that are relevant to the problem. A simple example might be what control inputs should be given to a vehicle traveling in a straight line across known terrain so as to minimize its travel time such that it comes to a complete stop precisely at its destination.

Mathematically, control systems are defined by a state vector and a set of differential equations that describe how the state variables evolve. The system becomes a control system when control variables are added to the differential equations, thus describing how the control variables combine with the natural evolution of the state variables to generate the ultimate state of the system:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t) \tag{2.1}$$

Where $\mathbf{x}(t)$ is the vector of state variables at time t and $\mathbf{u}(t)$ is the vector of control inputs at time t. Given a control input function and an initial condition $x(0)$, the evolution of the system is completely determined.

We can then define a cost (or benefit) function over this that describes what the costs

(benefits) are of a particular sequence of states and control actions:

$$J = \int_0^T F(\mathbf{x}(t), \mathbf{u}(t), t)^T F(\mathbf{x}(t), \mathbf{u}(t), t) dt \tag{2.2}$$

The goal of an optimal control system is to determine the set of control inputs, $\mathbf{u}(t)$ that minimizes (or maximizes) J.

The primary mathematical difficulty with optimal control generically is that systems of any sort of complexity are non-linear, which means that there is no analytic solution to the problem. Thus, most optimal control systems work with either a linear approximation of the system, or they try to restrict the control horizon (how far ahead the system looks) to be close enough that the system looks linear, and then continuously perform this kind of local optimization (this is the underlying basis of receding horizon control discussed below).

For our purposes, this mathematical complication is actually less relevant than the requirement that we be able to describe both the system and the cost function in terms of differential equations (or difference equations in the discrete time case) where time is the only dependent parameter. In particular, this requires that both costs and state variables be able to be represented as continuous functions. In the information gathering problem, the nature of the problem is such that the presence of information will be inherently discontinuous, and not necessarily a function of time.

2.2.3 Hybrid Approaches

Receding Horizon Control

Receding horizon control (RHC) is a mechanism that tries to achieve many of the benefits of using optimal control theoretic techniques, while both constraining the state space to make the problem more tractable as well as reducing the effects of the systems non-linearities. The basic premise of receding horizon control is simple — instead of conducting an optimal control analysis all the way out to time T, pick some shorter time horizon, t_i, and generate an optimal control derived path from now until then, t_0 until t_i. Take one step along this optimal path to time t_1. Then

perform the optimal control analysis from t_1 to t_{i+1}. Repeat this process until the step taken is the one that satisfies the goal [61].

As with optimal control, RHC requires a mathematical model of the entirety of the state space being manipulated, including not just vehicle state variables but environmental variables as well. Furthermore, RHC is not itself optimal over the entirety of the path traveled through the state space. It is only locally optimal with respect to each step it takes. Obviously the farther out the receding horizon is, the closer to global optimality the final result will be, but the more complicated the calculations will be at each step. It is also not required that only one step be taken before updating the optimal control analysis. Taking multiple steps saves on computation, but at the cost of the resulting path being even less optimal.

Since it uses optimal control as its base, receding horizon control has many of the same difficulties, including local minima and complicated state spaces. Some of the significant attempts to address these problems have involved Mixed Integer Linear Programming (MILP) [84, 11, 85]. As a full MILP problem is also known to be NP-complete, the approach is generally combined with RHC so as to make problems more tractable [21].

However, Schouwenaars et al. [85] in particular highlights one of the problems with RHC, namely that in certain domains a plan to a particular horizon might leave the vehicle in a state that makes solutions beyond the horizon infeasible. In this instance they are working with unmanned aerial vehicles (UAVs) that are planning trajectories to avoid obstacles and satisfy other dynamic and deadline constraints. UAVs that are like airplanes (as opposed to, for example, quad-coptors) have a unique dynamic constraint — they need to keep moving at a minimum speed in order to maintain lift. This means that the UAV could plan for a particular horizon without being aware of (or taking into consideration) an obstacle just beyond the horizon, and then be constrained by its dynamics such that it cannot maneuver to avoid the obstacle when it moves within the planning horizon. Their solution was to plan in such a fashion that the goal was achievable, but also so that

a default safe trajectory within the visible horizon was also always achievable. This obvious came at a cost, both with regards to computation and optimality.

2.3 Stochastic Path Planning

The problem of stochastic planning in static environments is relatively simple. Frank [35] and Mirchandrani [71] showed that in a graph where the arc costs are represented by independent probability distribution functions (PDFs) that do not change over time, the shortest path can be found by setting the cost of each arc to the expected value of its cost PDF and using any of the standard deterministic shortest-path algorithms (e.g., Djikstra's, A*, etc.).

Much of the more recent research on path planning has been exploring how to account for stochastic uncertainties in the environment. There are three types of uncertainties that this research tries to address. The first is uncertainty about the environment. The second is uncertainty about the robot's exact state, such as its exact position. The third is uncertainty about the effects of the robot's actions on its state.

Which type of uncertainty a research group is interested in depends largely on the environment their robot will be operating in. For robots operating in structured environments such as roadways and office buildings, the focus tends to be on the uncertainty of the robot's exact position in the environment. This is largely because the environment is highly structured, and thus can be extremely well known ahead of time (for example, [90, 50]). This type of research explores complex statistical models that derive probabilistic estimates of the robot's state based on the previous state, the action taken from that state, and the current sensor information. A successful system is one where the highest probability state is always the one that most accurately reflects the real state of the robot.

Among the more popular methods for solving these types of problems are Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs) [55, 56, 91, 33, 34].

These methods generally require some mechanism for specifying ahead of time what the states of the system are, as the assumption for solving MDPs (which underly POMDPs) is that the set of states and the set of actions are both finite.

This feature is why these techniques have had the most success in structured environments, such as in an office building [56] and in a robot museum tour guide [17, 18, 96]. Because the dynamic requirements can be highly simplified, possible states are constrained and can be fully specified in advance. For example, in an office building you can assume that the floor over which the robot will move is relatively uniform. Thus, all of the locations along a hallway can be collapsed into a single state like *InHallway*. Then the planner need simply specify that when the robot is in this state it should move in the required direction until it enters another state. The other states of interest (e.g., *InDoorway, InCubicle*, etc.) are relatively small in number and can also capture a large number of actual physical states of the robot.

This aspect is also why these types of techniques have not seen a lot of use or success in unconstrained, open-world environments such as the ones being considered for the information gathering problem. The manner in which these environments impacts the dynamics of the robot is non-uniform, unlike the office building floor, meaning that each location in the environment needs to be represented as a distinct state, requiring its own set of possible actions distinct from those of adjacent physical states. Since MDPs and POMDPs become significantly more difficult to solve as the state space increases, this tends to render those approaches as being computationally intractable without significant simplifications that can hinder the ability of the robot to operate.

For robots in these types of unconstrained environments, uncertainty research focuses more on the robot's uncertainty about the environment, in particular about navigating while reacting to obstacles that it did not know about ahead of time, as in [13, 22, 28, 36, 54, 58, 72, 87, 89]. This type of research focuses on rapid, reactive replanning that can happen in as close to real-time as possible. However, it tends to avoid more complicated statistical models, mainly because finding

the shortest path through graphs where the arc cost PDFs change over time has been shown to be computationally extremely difficult (see [37]).

2.4 Optimal Foraging Theory

Optimal Foraging Theory (OFT) is an area of study in ecology that tries to understand how predators determine where to search for prey, and for how long. Fundamentally, any animal wants to maximize the calories it takes in relative to the calories it expends. With respect to predators, this means they want to eat as many prey as possible while minimizing the effort spent locating and killing that prey. Thus, when a predator locates a patch of the environment that contains prey, its behavior needs to balance two things. First, it needs to acquire enough calories to compensate for the effort spent searching for the prey-containing patch. On the other hand, it needs to be certain that it is ingesting calories at a sufficient rate to compensate for the effort being expended catching the prey within the patch.

One of the primary effects of this balancing act is that predators never end up eating all of the prey within a particular patch. This is primarily because as the predator eats prey within the patch, the density of prey within the patch shrinks, meaning that more effort must be expended finding and catching the prey. Eventually, a tipping point will be reached where it makes more sense for the predator to move to the next patch than to keep trying to locate new prey within the current, depleted patch.

2.4.1 Marginal Value Theorem

The Marginal Value Theorem (MVT) attempts to describe when this tipping point is reached [20]. Summarized in Figure 2.7, the MVT assumes that the energy a predator can gain by catching prey over time can be represented by a logarithmic curve similar to other types of diminishing-return curves, such as the power law of learning. Thus, the amount of time a predator should spend in a patch is determined by a line tangent to this curve which starts at the time the

predator started searching for a patch of prey. The area under this line is the amount of energy the predator expends both getting to the patch and hunting within the patch.

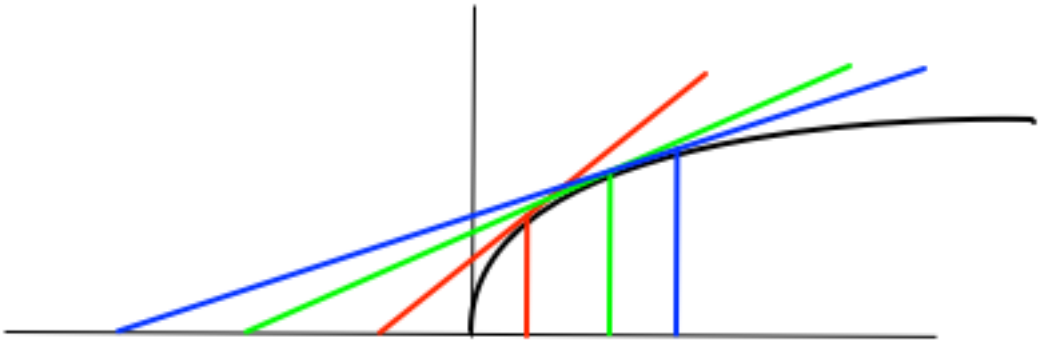


Figure 2.7: Marginal Value Theorem
Different color lines correspond to different travel times to patch,
and thus different times spent foraging the patch.

As we will see, the primary difficulty with using the MVT in a robotics task is that the precise nature of the diminishing-returns curve for a particular task may not be known in advance. This means that the robot will need to have some way to learn the details of this curve as it moves through the environment. With regards to information gathering, it may be possible to apply information-theoretic concepts such as entropy to give the robot the capability of determining for itself when to move on to a new patch.

2.4.2 Applications

Within robotic planning, variations on OFT have been applied to a number of different problems. One example is the use of OFT to make decisions about how fast a robot should move through different parts of the environment [75, 5]. The robot is searching for tasks to perform in the environment so as to maximize the rewards it receives for performing those tasks. OFT can be used to model the expected rewards within a patch of the environment based on the perceived density of tasks and the quantity of tasks already performed within a given patch. In other words, if the robot can learn the relevant diminishing-returns function, then it can apply the MVT to its behavior and

make decisions about how carefully to search for tasks in a particular patch relative to moving on to the next patch.

In many ways, the OFT problem is similar to the information gathering problem. In our problem, the "predator" is the robot and the "prey" is the information being gathered. In both cases, the predator is exploring patches of the environment trying to maximize the amount of prey that it gathers. However, for the most part, OFT has not yet been applied to comprehensive information gathering systems. There are many reasons for this, not least of which is the domain-specific nature of the diminishing-returns curves. Preliminary research has been done trying to understand how an artificial system might learn these curves and make good decisions ([19, 76]), but as yet the theory has not been deployed on a full robotic foraging system.

Much of this has to do with several key differences between the foraging problem and the information gathering problem. The primary one is that OFT primarily concerns itself with how long a predator should forage in a particular patch. When a predator leaves a patch, it is assumed that it finds the next one by engaging in what amounts to a random walk.[1] The information gathering problem, on the other hand, has to address not just when to abandon a particular patch, but also which patches to explore in the first place. If the robot has no prior knowledge of where information containing patches are likely to be, then its behavior until it discovers one ought to resemble a random walk (with a bias for heading towards its eventual goal). However, if the robot does have prior probabilities for information (such as from remote sensing imagery), then it needs to be able to choose which patches to explore and which ones to ignore. It is this particular aspect of the problem that this dissertation is addressing.

[1] This walk can obviously by guided by the predator's senses, such as smell, but OFT does not try to address patch finding or the various things that could impact it.

2.5 Information Foraging

The problem of developing a robotic system that can forage for information combines a number of the issues that the algorithms described here are trying to address — such as path planning, uncertainty, and foraging — that have not previously been integrated. By themselves, none of the previous approaches are able to provide a comprehensive solution to the information foraging problem.

For example, as described in Section 2.4, systems that utilize concepts from foraging theory are almost entirely concerned with how much effort should be expended foraging within a particular patch. Navigation between patches has not yet been strongly considered. OFT also does not deal at all with any operational constraints, such as a deadline or resource constraints.

Route-planning algorithms, on the other hand, are concerned almost entirely with finding paths that minimize costs. The reasons for this mostly center around the intractability of the generic maximization problem, but also around the fact that most of the research is concerned mainly with developing better ways for robots to maneuver from a starting point to a destination. In so far as recent work has started to concern itself with the robots doing something useful while traveling, the focus has been on coverage of an area (such as with rescue robots) or opportunistic tasks.

2.5.1 The Information Foraging Algorithm

Chapter 3 details the Information Foraging Algorithm (IFA), which integrates aspects of several of the techniques thus far described. In particular, it not only tries to capture how long a given patch should be foraged, but also to take into account the selections of which patches to forage, how to navigate between those patches, and how to take into account uncertainties about the environment as they pertain both to patches and to navigation.

The IFA accomplishes this by determining the shortest paths between the start and all nodes in the graph and then from those to the goal using whichever shortest-path algorithm is

appropriate for the task, e.g., , Djikstra's algorithm, A*, etc. Then, for every node that contains information, the IFA assigns a value, called *Mass*, that contains both the information content of the node and the time available for further exploration after that node has been visited and its information harvested. The algorithm then visits the node with the largest Mass, harvests that node's information, and recalculates all of these values for the remaining information containing nodes.

When calculating shortest paths from the starting point to all nodes, the IFA explicitly accounts for any areas of uncertain traversability. Instead of simply treating an uncertain arc as untraversable, it makes what amounts to a cost-benefit judgment regarding whether enough time would be saved by traversing that arc to make it worth trying to gain more information about its traversability, versus the cost of simply avoiding the arc entirely.

The IFA takes a greedy approach by always navigating to the node with the largest Mass. A completely new plan is generated after information is gathered, or if the environment changes. The primary benefit of this approach is computational efficiency. The optimal versions of the information maximization problem are essentially versions of the Traveling Salesperson problem, which is known to be NP-hard and have a computational complexity of roughly $\mathcal{O}(n!)$ for brute-force methods, and $\mathcal{O}(n^2 2^n)$ for more exotic approaches like dynamic programming (where n is the number of nodes). By only looking ahead one node, in the worst case the IFA only needs to perform two runs of Djikstra's algorithm, which is $\mathcal{O}(n^2)$ in the worst case, plus calculate the Mass for each node, which is obviously $\mathcal{O}(n)$.

As will be shown in Chapter 4, the IFA generally performs quite well, finding paths that gather over 75% of the information that the optimal path would gather. However, the price for using a greedy approach is that there is no minimum performance guarantee. This means that, on rare occasions, the IFA generates paths that gather less than 50% of the optimal amount of information. This happens on less than 1% of trials, though, and worse performance is even more rare. Thus,

accepting such a small number of failed pathways to get such a significant boost in computational performance should be acceptable.

Finally, the IFA is guaranteed to satisfy a deadline (or other horizon constraint) as a result of the initial calculation of the shortest path from each node to the goal. Before traveling to a node N, the IFA checks that the total of the time to travel to N and the time to travel from N to the goal will not violate the deadline. Any node where such a violation would occur is excluded from any subsequent consideration.

2.5.2 Contributions of this thesis

Robotic foraging algorithms have thus far focused on planning surrounding a single patch in which to forage. Most of the literature focuses on determining how to maximize the robot's exploration of a particular patch, and when the robot should go forage in a new patch. This thesis expands the robotic foraging research to make several new contributions, including:

- Developed a mechanism for finding the shortest expected time to any node in a graph while guaranteeing the robot will satisfy a deadline to the goal.
- Developed a novel metric (mass) for judging the utility of a node N based not just on the benefit from N but on the potential to gain benefits from the opportunity to visit other nodes after N.
- Adapted the nearest neighbor heuristic for solving maximization problems to choose the order of patches to visit and forage. This makes the maximization problem computationally tractable.
- Explicitly took into account the uncertain traversability of arcs in the graph both when measuring mass and planning the route to the next patch to forage.
- Developed a foraging system that can explicitly account for operational constraints of the environment.

Chapter 3

The Information Foraging Algorithm

This chapter describes the Information Foraging Algorithm (IFA) developed to address the information gathering problem. Each component of the algorithm will be described in more detail below, but the basic premise is that it takes as its inputs a map (in graph form) where each node has an expected information content and each arc has a traversal cost and a traversal probability between 0 and 1, as well as a goal location and a deadline time. Figure 3.1 shows a simple example. In this map, arcs that are untraversable will be represented with an infinite cost.

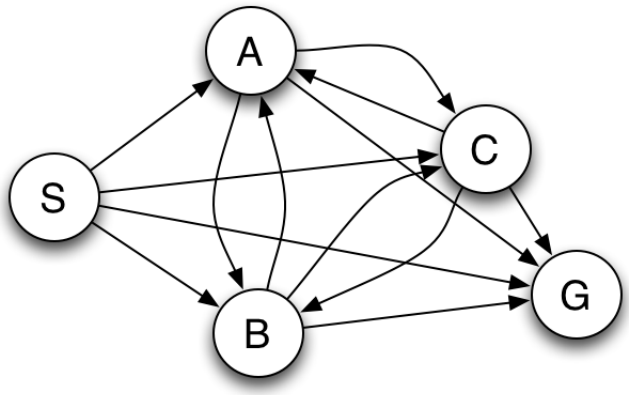


Figure 3.1: Simple environment map. A, B, and C are all information containing patches.

This map would be derived from processing remote-sensing imagery of the environment, such as satellite photos. Research on this problem is in its early stages as before autonomous, long-duration exploratory robots became more prevalent there was no significant need for systems

to autonomously process such imagery in this manner. However, there are currently some basic algorithms in place. For example, the Belkin-O'Reilly algorithm identifies ocean fronts from satellite sea-surface temperature images using mechanisms similar to edge-detection algorithms from computer vision [8] and another uses ontologies to extract other types of ocean features from satellite imagery [3].

Since any maximization problem in a graph is going to be NP-hard, there needs to be some heuristic that will allow the robot to determine where it should go next in order to take a path that ends up being near optimal. In the IFA, the robot calculates a *Mass* (see Section 3.2) for each node in the graph from the input map and starts navigating toward the node with the highest Mass. As long as its sensor data is consistent with the data stored in its map, it will continue moving towards this node. Once it detects an inconsistency, either in the information content of a node or the cost to traverse an arc within its sensor radius, it will re-plan and start again. This continues until the robot reaches *Bingo Time*. Bingo Time is defined as the time when the only nodes the robot can reach without violating its deadline are those along the derministic path to the goal.

Many of the implementation details will be domain specific. In particular, how information will be gathered and how that impacts the planning and, especially, re-planning phases will be strongly dependent on the characteristics of the information being searched for. Likewise, issues of traversability will depend on the interaction between vehicle dynamics and the environment. Thus, this dissertation presents a framework that can accommodate a wide range of possible information and environment models.

The IFA is also accounting for uncertainties in the environment. Traditional path planning algorithms have largely viewed areas of the environment as being either safe or unsafe for the robot to traverse. Obviously, the real world is less binary than this. The IFA accounts for this uncertainty about traversability by judging paths based on their expected cost.

3.1 Shortest Paths

Traditional path planning algorithms treat areas of the environment in a binary fashion: they are either safe or unsafe. Path planning sticks to the safe areas, and decisions are relatively straightforward. The robot traverses only the safe areas, usually trying to minimize a cost function and possibly satisfy constraints such as a deadline. In general, if there is uncertainty about a region's traversability, it is treated as unsafe and avoided completely. However, in certain environments those regions might be very large, such as the rim of a crater on Mars, and there could be significant cost savings to be gained if these regions could be determined to be traversable (safe) upon closer examination by the robot. On the other hand, if the robot explores an ambiguous region and determines it is not safe, then it will have wasted time approaching the region and diverting around it. Thus there needs to be a mechanism for deciding whether it is worth the risk to explore the uncertain region.

Uncertainty can be modeled in a graph-based map of the environment by associating with each arc a traversal probability. Any arc with a traversal probability, P_t, of either 1 or 0 can be thought of as a *deterministic arc*. Deterministic arcs with $P_t = 0$ are completely untraversable, while those with $P_t = 1$ are known to be unambiguously safe. Any other arc can be thought of as a *stochastic arc*, since there is only a prior probability that the arc can be traversed. Figure 3.2 shows an example of a simple graph with both deterministic and stochastic arcs.

For example, Figure 3.2 shows a graph with both deterministic and stochastic arcs. Two types of paths through this graph can be defined. Deterministic paths utilize only arcs where $P_t = 1$. These are paths that the robot is completely certain can be traversed, and it is completely certain of the cost of these arcs. Any path that contains even one stochastic arc will be called a stochastic path. $T_d(A, B)$ is then defined as the time to get from A to B along the shortest deterministic path. In Figure 3.2, if assume the cost of each arc is assumed to be the same, then the red path is the path determined by $T_d(S_0, S_g)$.

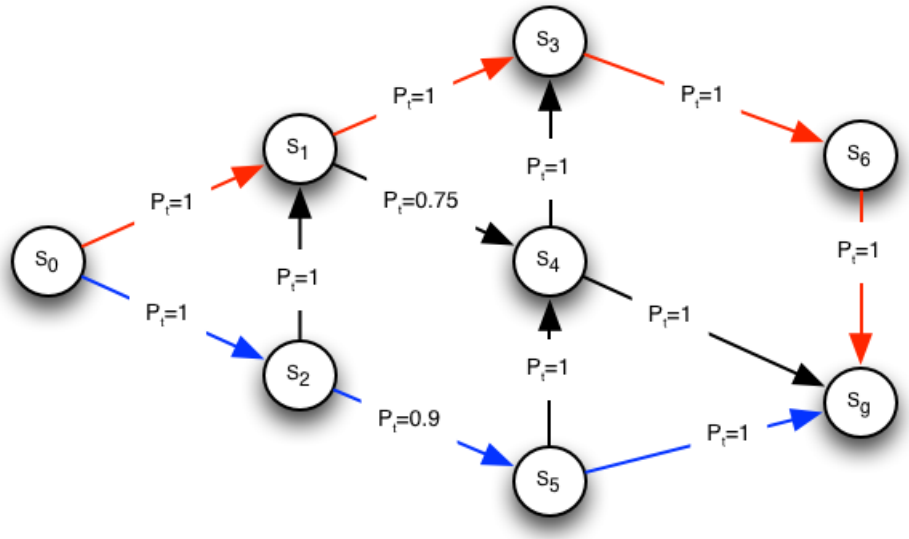


Figure 3.2: Graph with stochastic arcs
Red path is $T_d(S_0, S_g)$; Blue path is $T_s(S_0, S_g)$

$T_s(A, B)$ is defined as the time to get from A to B along the shortest path in a graph containing stochastic arcs. In order to path plan using stochastic arcs, there must be some mechanism for penalizing arcs with lower values of P_t. This is most easily achieved by multiplying $C(x, y)$ by $\frac{1}{P_t(x,y)}$. More formally, the cost of traversing a stochastic arc is defined as $\frac{C(x,y)}{P_t(x,y)}$. This yields a range of possible costs from $C(x, y)$ if $P_t(x, y) = 1$, which is what would be expectedfor deterministic paths, to ∞ for $P_t(x, y) = 0$, which is also what would be expected.

In Figure 3.2, the blue path is the path determined by $T_s(S_0, S_g)$. The path described by $T_s(A, B)$ is not required to have a stochastic arc. For any particular graph it may well be the case that the shortest path between two nodes in the graph is deterministic regardless of whether stochastic arcs are present in the graph. For example, in Figure 3.2, there are no stochastic arcs between S_3 and S_g. Thus, $T_s(S_3, S_g) = T_d(S_3, S_g)$. In the simplest case, the shortest path between two adjacent nodes is the arc connecting them, and the robot's sensors presumably have sufficient detection power such that this arc is always deterministic. Thus, $T_s(A, B) \leq T_d(A, B)$. If $T_s(A, B) > T_d(A, B)$, then $T_s(A, B)$ would not be the shortest path between A and B in the stochastic graph.

The question then becomes which cost should be used when deciding which nodes to visit. Using T_s would allow the robot to plan to visit more nodes, but if any of the arcs turn out to be untraversable then the plan could be severely disrupted. Imagine a robot that starts at S_0 in Figure 3.2 and starts following the stochastic path to S_g, and $T_D = 4$. It arrives at S_2 only to discover that the arc from S_2 to S_5 is not traversable. The robot drives to S_1 where it learns that the arc from S_1 to S_4 is also not traversable. Having consumed two time units to get to S_1, there is no path that will allow the robot to arrive at S_g before T_D. Thus, in order to guarantee that the deadline will be satisfied, if $T_R(N)$ is defined as the time remaining before the deadline when the robot is at N, it can only attempt a stochastic path if, for every node N along that path, $T_d(N, G) < T_R(N)$.

If this condition is satisfied, the robot can still be optimistic and use $T_s(S, N)$ to plan the route to N and still be guaranteed of satisfying the deadline. Since using T_S allows for the potential of exploring more nodes, it makes sense to use that for planning the route to N. This distinction highlights the first steps of the IFA: For all $N \in map$, calculate $T_d(N, G)$ and $T_s(S, N)$. If $T_s(S, N) + T_d(N, G) > T_D$, then there is no path the robot can take to explore N and still guarantee it can get to G before T_D, and so N should just be eliminated from all further consideration.

3.2 The Mass Metric

In order to plan its route, the robot needs some mechanism for determining the order of nodes it will explore. Utilizing the potential fields concept of how *attractive* a node is (see Section 2.2.1), a value called *Mass* is assigned to all information containing nodes ($I(N) > 0$). The Mass of a node is based on two factors. The first is the expected information content of the node, $I(N)$[1]. The second is the remaining *exploration time* after exploring the same node.

The exploration time of node N, $T_E(N)$, represents time left for the robot to explore other

[1]For clarity the $E\{\}$ notation will be dropped. It should be taken as implied whenever the information content of a node is referenced.

nodes after it travels to N and harvests the information from N. It is important to note that the time left for exploration is not simply the time remaining before the deadline, because the robot still also needs to navigate to G. Thus, $T_E(N)$ reflects the time it will take to travel to N, the time it will take to harvest $I(N), C_I(N)$, and the time it will then take in the worst case to get from N to G, $T_d(N, G)$.

Figure 3.3 shows a simple example. In this graph, assume $I(S_1) = I(S_2)$ and $C_i(S_1) = C_i(S_2)$. If the robot is at S_0, exploring either node will give it the same benefit. However, the cost of traveling to S_2 is less than the cost of traveling to S_1. Thus, if the robot visits S_2 it will have more time remaining before Bingo Time to explore other nodes than if it visits S_1. This difference in attractiveness is what $T_E(N)$ is capturing.

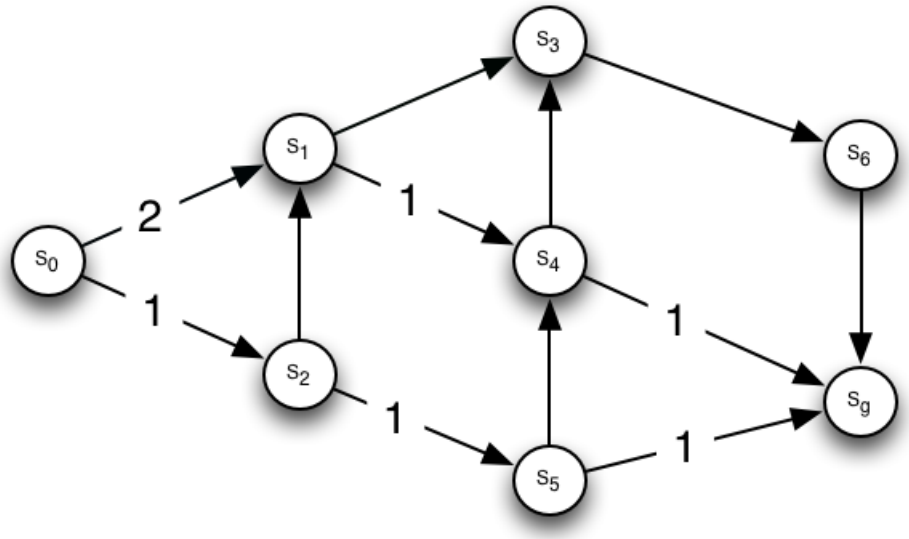


Figure 3.3: Exploration time example. $I(S_1) = I(S_2)$, and $C_i(S_1) = C_i(S_2)$, but $T_E(S_2) > T_E(S_1)$

$T_E(N)$ can be defined as:

$$T_E(N) = T_R(S) - (T_s(S, N) + \alpha C_I(N) + T_d(N, G)) \tag{3.1}$$

where α is simply a normalizing factor to account for different units between time and information.

The Mass of N, $M(N)$ can then be defined as:

$$M(N) = I(N) + \lambda T_E(N) \tag{3.2}$$

λ is a scaling factor that allows the relative importance of the information content of a node versus its exploration time to be adjusted. For example, in an environment where the information content of the nodes is very certain, a user might want to set λ to be low so the robot just visits as many high-information nodes as it can. On the other hand, in an environment where there is significant uncertainty about information content, a user might want to put more emphasis on exploration time by setting λ to be higher so as to cause the robot to pick nodes based more on how much time it will have for visiting other nodes later.

3.3 The Planning Phase

The IFA has two phases — A planning phase in which it selects the next information-containing node to visit, and an execution phase in which it actually navigates towards that node. These two algorithms trade control back and forth as appropriate until the robot has reached its goal.

Algorithm 3.1 shows the IFA planning phase. It takes as its inputs a Map of the environment in graph form, a start node S and a goal node G. In each planning step, the planner first determines three values for each node N in the Map: the deterministic cost to the goal, $T_d(N, G)$; the stochastic cost from the start, $T_s(S, N)$; and $T_E(N)$.

At this point Map is analyzed and any node where $I(N) = 0$ and $T_E(N) < 0$ is deleted because any node with a negative exploration time cannot be reached without violating the deadline.

To wit:

$$0 > T_E(N)$$

$$0 > T_R(S) - (T_s(S,N) + C_i(N) + T_d(N,G))$$

$$T_R(S) < T_s(S,N) + T_d(N,G)$$

If $T_E(N) < 0$ but $I(N) > 0$, this means that there is not time to both visit N and harvest its information before the deadline. Thus, N is made a non-exploration node by setting $I(N) = 0$ and $C_i(N) = 0$ and recalculating $T_E(N)$. Once this is complete, the transit costs and exploration times are recalculated on the new Map. These two steps alternate until $T_E(N) \geq 0$ for all $N \in Map$.

Once that is done, the planning phase calculates $M(N)$ for every information containing node remaining in the reduced Map and finds the node A with the largest Mass. If there is a tie between two or more nodes, that tie can be broken by any heuristic the user chooses. Then the IFA invokes the navigation phase for the robot to move towards A.

3.4 The Navigation Phase

The navigation phase is shown in Algorithm 3.2. It starts out by first retrieving the path from S to A. This would have been stored in Map as part of line 6 in Algorithm 3.1. It sets the current node to S and then executes a loop of actions until it either reaches A or detects that something in the environment is different from what it is expecting based on what is stored in Map. If any arc has changed, or if the expected information of any node has changed, it sets S to the current node and returns to the planning phase, whereupon the planning loop restarts.

If nothing has changed it updates T_R for the next step along the path based on the stochastic cost between the current node and the next node in the path,$T_s(Current, Path.next)$. It is worth noting that for any arc the robot is willing to travel between S and any adjacent node Z, $T_s(S,Z) = T_d(S,Z)$ because, in order to guarantee safety, the robot will only travel away from its current location along an arc that has been determined to be traversable, meaning that arc is now

Algorithm 3.1 The planning phase of the IFA

1: **function** IFA_PLAN(Map, S, G)
2: **while** $S \neq G$ **do**
3: **repeat**
4: **for all** $N \in Map$ **do**
5: Calculate $T_d(N, G)$
6: Calculate $T_s(S, N)$
7: $T_E(N) \leftarrow TR(S) - (T_s(S, N) + \alpha C_i(N) + T_d(N, G))$
8: **if** $T_E(N) < 0$ **then**
9: **if** $I(N) > 0$ **then**
10: $I(N) \leftarrow 0$
11: $C_I(N) \leftarrow 0$
12: **else**
13: $Map \leftarrow Map - N$
14: **end if**
15: **end if**
16: **end for**
17: **until** All $T_E(N) > 0$
18:
19: **for all** $N \in Map, I(N) > 0$ **do**
20: $M(N) = I(N) + \lambda T_E(N)$
21: **end for**
22:
23: $A \leftarrow \max_{I(N)>0} M(N)$
24: IFA_NAV(Map, S, A)
25: **end while**
26: **end function**

deterministic.

The robot then takes the next step along the path and updates *Current* appropriately. This loop continues and, so long as nothing has changed, it ends once the robot reaches A. When this happens, it will harvest the information from A, set $I(A) = 0$ so that it does not try to harvest A again later, update $T_R(A)$ to reflect the cost of information harvesting, $C_I(A)$, update S to be A, and return to the planning phase.

Algorithm 3.2 The navigation phase of the IFA

```
1:  function IFA_Nav(Map, S, A)
2:      Path ← GetPath(Map, S, A)
3:      Current ← S
4:      while Current ≠ A do
5:          Measure Environment
6:          if Any arcs or Information in Map change then
7:              Update Map
8:              S ← Current
9:              return to IFA_Plan with new S and new T_R(S)
10:         else
11:             T_R(Path.next) ← T_R(Current) − T_s(Current, Path.next)
12:             Current ← Path.next
13:         end if
14:     end while
15:     Harvest Info from A
16:     I(A) ← 0
17:     T_R(A) ← T_R(A) − C_i(A)
18:     S = A
19:     return to IFA_Plan with new S and new T_R(S)
20: end function
```

3.5 Information models

Nothing about the IFA requires that either the expected information of a node, the cost of traversing an arc, or the probability of traversal to remain constant for the entirety of the robot's journey. There are multiple factors that would change any of these values as the robot moves through

the environment, explores nodes, and updates its map. The mechanics of exactly how these updates are performed, however, would be domain specific, and thus outside the scope of this dissertation. However, some general expectations for how those effects would impact the IFA can be outlined

3.5.1 Specific Information

There are several ways that the specific nature of the information will impact the way information is both represented within a particular node and will change as the robot explores nodes. One major aspect is whether the information is specific or diffuse. Specific information is reflected in a particular object or feature.

For example, a rover on Mars might be tasked with looking for one sample of a specific type of rock. Its map would tell it that there is a 90% likelihood of the rock being in a particular patch, but the patch contains multiple nodes that could be explored. Once the rover finds a sample in a particular patch, it should stop searching that patch and move to the next one. This case is relatively simple to deal with as the rover simply explores the nodes of the patch until it finds the rock. Once it finds the rock, it re-plans based on the time to the deadline and searches the next patch.

However, there are two open questions that are actively being researched by statisticians interested in the field of space-time statistics (see, for example,[40, 6]). The first question has to do with the initial conversion of the low-resolution map into the graph the IFA uses to plan. If there are multiple nodes of the graph contained within the patch of interest, how should that 90% likelihood be divided up? Should every node in the patch get a 90% likelihood of information? Should likelihoods for each node be determined by a sample of a two-dimensional Gaussian distribution? If so, what are that distribution's parameters going to be?

The second questions has to do with how that likelihood should change for the other nodes in a patch once a particular node has been explored and the rock has not been found. Should the other nodes maintain the same likelihood? Should it go down? Should it go up? The answer to this

depends to some extent on what method is used for the original apportionment of the information likelihood. Most of the simulations described in Chapter 4 used a simplified information model that assigns the likelihood for a patch to all the nodes within the patch and does not change them as the robot explores the patch.

3.5.2 Diffuse Information

Another type of information would be diffuse information. This would exist in a scenario where the robot needs to sample one patch long enough to get a sufficient picture of its characteristics, and to then go sample another patch for contrast. For example, an oceanographer might want an AUV to take measurements on one side of a front and then cross the front and take measurements on the other side. Similarly, in using an unmanned aerial vehicle (UAV) to search for a given feature within a particular patch of the environment, there needs to be some metric for it to decide that continued searching is not likely to yield any new information, so it should give up and go look somewhere else.

Diffuse information can be more directly modeled with something like optimal foraging theory (see Sec. 2.4), where the curve in the marginal value theorem can be generated as a type of informatics-based entropy curve. Estimating the details of that curve would be difficult, but in theory it should be possible to measure when the new bit of information the robot has gained about the patch is not adding much to the overall amount of information gathered about the patch. It would be at this point that the robot would want to give up on the current patch and move on to the next one.

Chapter 4

Simulations

To test the Information Foraging Algorithm several different types of simulations were run using Matlab to model different aspects of the types of physical environments in which an information gathering robot might operate. The simulations were also designed to test different models of how scientific information might be distributed in these environments.

In all cases, the vehicle dynamics were ignored as being beyond the scope of this dissertation. In particular, it is assumed that the vehicle dynamics would be dealt with in two systems external to the IFA. The first is in the system that generates the initial map from remote imaging. Presumably, in developing estimates of both the cost for the vehicle to traverse a particular part of the environment, as well as a prior probability reflecting the likelihood that that part of the environment is traversable, the map making system will have to have at least a rudimentary model of the vehicle dynamics. The IFA then uses this map to generate a proposed route through the environment. This route will follow the grid lines in the graph generated in the initial phase of the IFA. Thus, some post-processing of the route using standard smoothing techniques, such as Bezier curves or interpolation (e.g., [102, 103, 31]) will be required. It is not atypical that this smoothing stage would also involve smoothing of the control inputs to the vehicle.

Vehicle dynamics can also be integrated into the sensor-based, realtime adjustments the robot will need to make while maneuvering through the environment. The map the robot is initially working with is based on low-resolution remote-sensing data. The sensors on-board the robot will

undoubtedly provide a much higher resolution of the immediately surrounding terrain, including roughness and smaller obstacles. The robot will need to integrate this sensor information to make decisions about specific control inputs. There already exists a significant body of research addressing this question(e.g., [99, 65, 64, 92, 28, 42]).

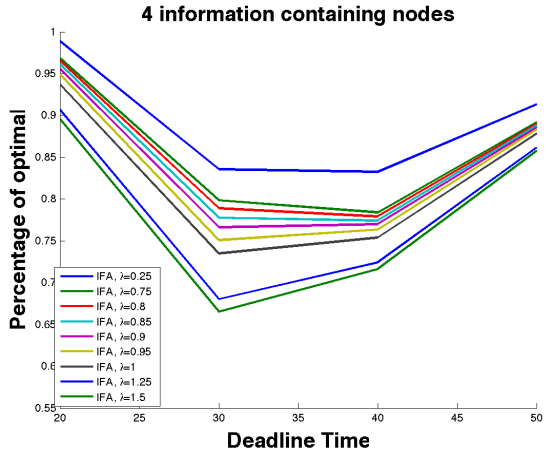
The first set of tests was designed to establish some measures of baseline performance. In these tests, physically plausible arrangements of information were constructed without trying to model every aspect of a physical environment. In all of these tests the performance of the IFA was compared to what the results of the optimal path through the environment would have been. The optimal path was simply computed with depth-first search with the constraint that no node could appear on a particular path more than once.

The second set of tests was designed to explore how well the IFA would perform with the existence of stochastic obstacles. For the most part, the environments used were the same as in the first set of tests, but with the introduction of arcs with a traversability probability less than 1.

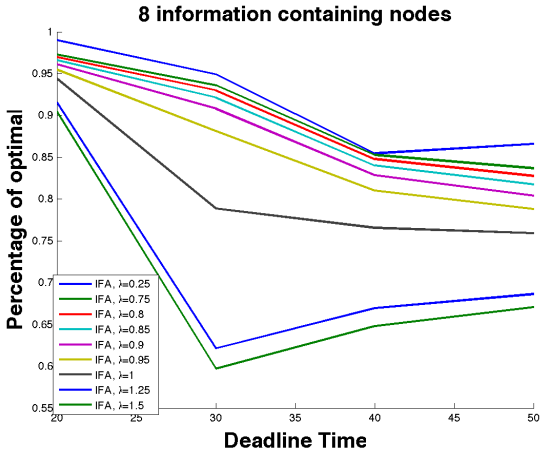
4.1 Monte Carlo Simulations

A Monte Carlo simulation was conducted where N nodes were randomly distributed within a 12×12 square. Their information contents were randomly assigned between 1 and $2N$ so as to reduce the likelihood of two nodes having the same $I(N)$. S and G were always at opposite corners of the square. We were interested in comparing how the IFA performed relative to optimal performance when $N = \{4, 8, 12\}, T_D = \{20, 30, 40, 50\}$, and $\lambda = \{0.25, 0.75, 1.0, 1.5\}$. If two or more nodes had the same Mass, the IFA would choose the node closest to the robot's current node. If there were still multiple nodes tied, it would choose one at random. There were 10,000 trials run for each condition.

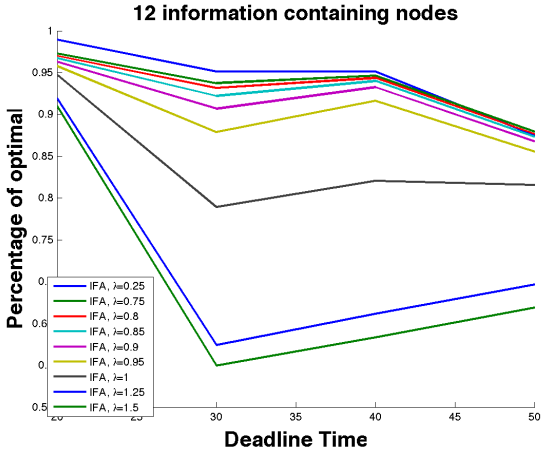
Figure 4.1 shows the average performance of the IFA for each of the various conditions expressed as an average percentage of the optimal amount of information that could have been collected on each run. As can be seen from the table and the figures, the IFA obviously did not



(a) 4 information containing nodes



(b) 8 information containing nodes



(c) 12 information containing nodes

Figure 4.1: Mean IFA performance as percentage of optimal: only deterministic arcs
Performance for $\lambda < 1.0$ is always over 75% of optimal.
Performance drops off substantially for $\lambda \geq 1.0$. See the text for details.

perform optimally, though it was not expected to. However, it did generally perform extremely well. The lowest average performance for any combination of N, λ, and T_D was 55%, and it generally performed better than 70% of optimal for all $\lambda \leq 1.0$.

In general, when $\lambda < 1.0$ the IFA performed extremely well (better than 85% of optimal), but performance dropped significantly for $\lambda > 1.0$. The most likely explanation for this trend has to do with the cost of harvesting the information from a node, $C_I(N)$. For the sake of simplicity,

all of these simulations have set $\alpha = 1$ and $C_I(N) = I(N)$. That is to say, the cost of harvesting information is equivalent to the amount of information in a node. Equations 3.1 and 3.2 are reprinted here as Equations 4.1 and 4.2 for reference:

$$T_E(N) = T_R(S) - (T_s(S,N) + \alpha C_I(N) + T_d(N,G)) \tag{4.1}$$

$$M(N) = I(N) + \lambda T_E(N) \tag{4.2}$$

Substitute Equation 4.1 into Equation 4.2 and simplify terms:

$$M(N) = I(N) + \lambda(T_R(S) - (T_s(S,N) + C_I(N) + T_d(N,G))) \tag{4.3}$$

$$= I(N) + \lambda T_R(S) - \lambda T_s(S,N) - \lambda C_I(n) - \lambda T_d(N,G) \tag{4.4}$$

Substituting $I(N)$ for $C_I(N)$ and collecting like terms:

$$= I(N) - \lambda I(N) + \lambda(T_R(S) - (T_s(S,N) + T_d(N,G))) \tag{4.5}$$

$$= (1-\lambda)I(N) + \lambda(T_R(S) - (T_s(S,N) + T_d(N,G))) \tag{4.6}$$

What Equation 4.6 implies is that for any $\lambda = 1$, the information content of a node is not relevant to its mass, and all that matters is the distance component of the exploration time. Even worse, for any $\lambda > 1$, having information in a node actually leaves it with a lower Mass than if it had no information at all.

Note that this is not inherently a difficulty with the fact that $\alpha C_I(N) = I(N)$. In fact, an argument could be made that this is an indication that the λ is doing the job it is supposed to do. Namely, it is there to allow the user to control the relative importance of the information content of a node, $I(N)$, and the cost of obtaining that information relative to the time remaining before the deadline, $ET(N)$. Put another way, λ allows us to determine the relative importance of simply gathering information regardless of where it is versus maximizing the opportunities to explore other nodes after the robot is finished with the next one.

One important factor upon which the manipulation of λ will depend is the level of certainty in the expected information at the nodes in the environment. If the level of certainty is high, then

it makes sense to set λ low so that the robot will gather from the high-information nodes with less concern about how much time is left for exploring nodes with less information. On the other hand, if there is significant uncertainty about $I(N)$, then the remaining exploration time becomes a more important factor in the decision making since choosing the wrong node might mean missing out on significant amounts of information entirely. This would lead to the choice of a higher value of λ.

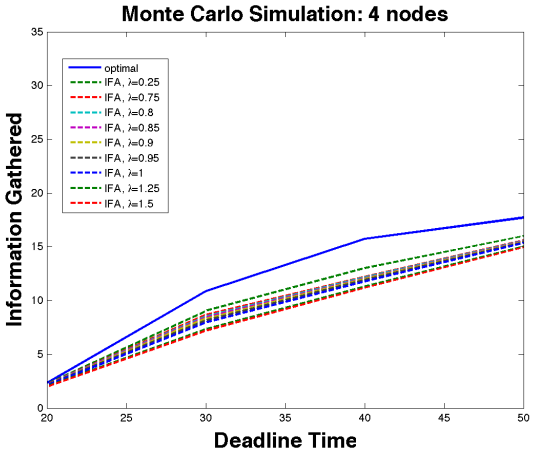
For any node node where $I(N) > 0$ and $C_I(N) > 0$, there will be some value of λ where $\alpha C_I(N)$ will essentially cancel out $I(N)$. For any individual node this is not necessarily a problem as it just means that, in the context of the chosen value of λ, that node is simply too expensive to go explore. On the other hand, if there is a proportional relationship between $I(N)$ and $C_I(N)$, then there will be a value of λ such that $\alpha C_I(N)$ will cancel out $I(N)$ for *all* information containing nodes, which would probably not yield useful results. Thus, when designing a domain-specific implementation of the IFA, it is going to be important to have at least some understanding of the relationship between $I(N)$ and $C_I(N)$, or some mechanism for a robot to learn that relationship.

Based on this realization, it makes sense to evaluate the IFA in the context of values of $\lambda < 1$. The reason for this is because when $\lambda = 1$ the information content of a node is irrelevant, meaning the IFA will essentially be navigating randomly towards the goal without actively seeking information. When $\lambda > 1$, nodes with information will actually have a lower Mass than equivalently distant nodes without information, meaning the IFA will actively avoid the information containing nodes.

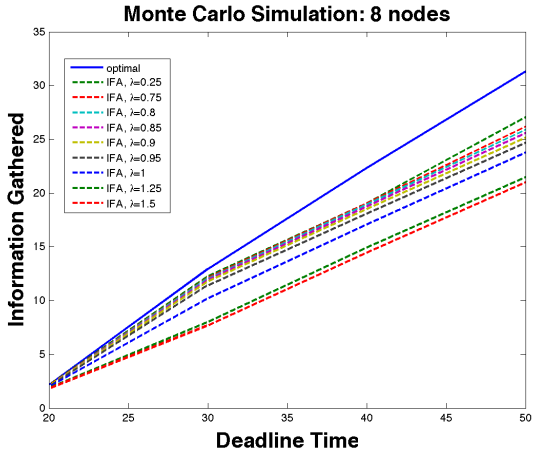
Given this, from the minimum performances we can see that there are occasions when the IFA collected less than 20% of the information of the optimal possible path. However, across the 4800 trials that were conducted with $\lambda < 1$, only 46 of them yielded paths that collected less than 50% of the information that the optimal paths collected. This is less than 1% of all trials. There were only 463 trials with paths that yielded less than 70% of the optimal information, which amounts to 9.65% of all paths. In other words, 9 times out of 10 the IFA will generate a path that

gathers at least 70% of the information the optimal path would have gathered, at a fraction of the computational cost.

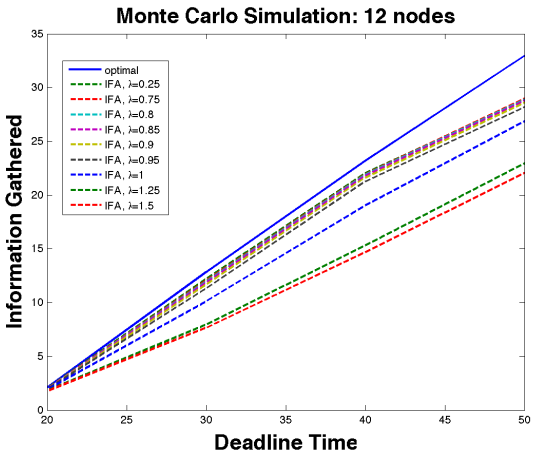
Figure 4.2 presents the results by showing the average difference for each condition between how much information the IFA harvests versus the optimal information that could be gathered. In general, lower values of λ had smaller raw differences from optimal than larger values. The raw differences from optimal also got larger the longer the deadline time.



(a) 4 information containing nodes

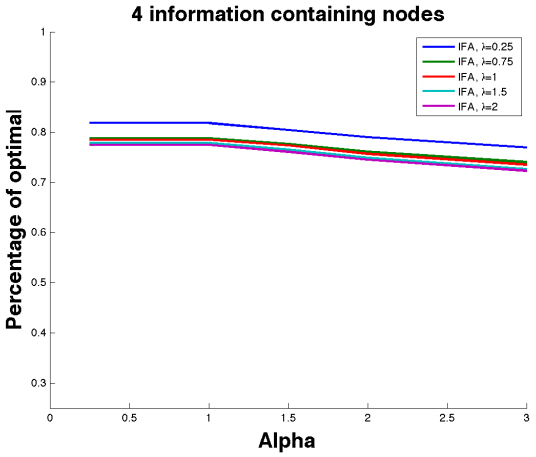


(b) 8 information containing nodes

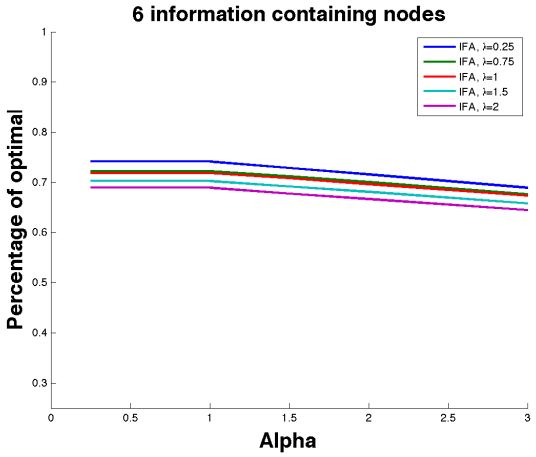


(c) 12 information containing nodes

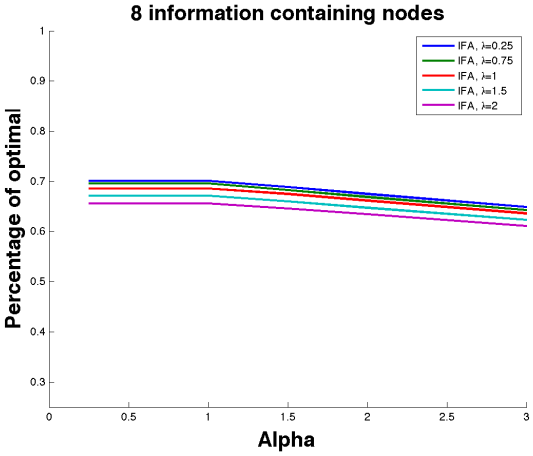
Figure 4.2: Mean Information Gathered: only deterministic arcs
For $\lambda < 1.0$, performance is largely the same.
When $\lambda \geq 1.0$, significantly less information is gathered. See text for details.



(a) 4 information containing nodes



(b) 6 information containing nodes

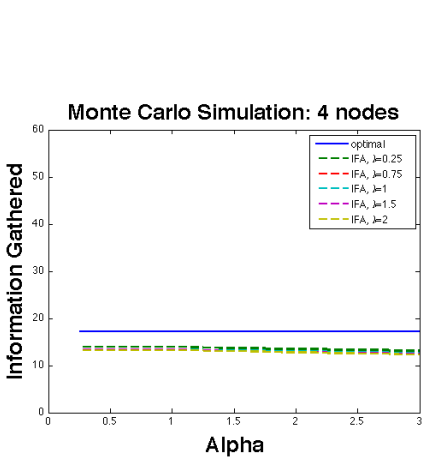


(c) 8 information containing nodes

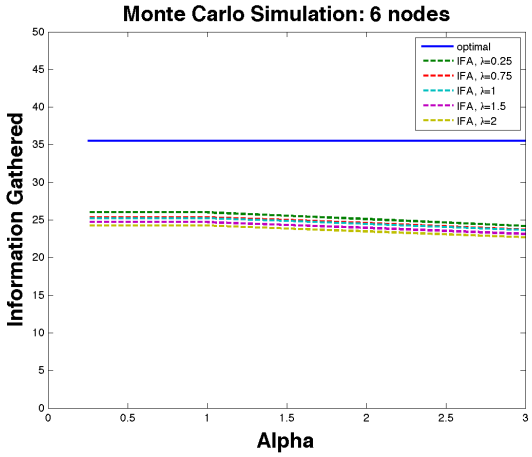
Figure 4.3: Mean IFA performance as percentage of optimal: α vs. λ, constant $C_i(N)$
For $\alpha \leq 1.0$, performance is largely the same.
When $\alpha > 1.0$, Performance slowly decreases.

A final set of simulations was performed to understand the relationship between α and λ. In the previous simulations, $C_i(N) = I(N)$, which has the effects described above. Those effects are true if there is any sort of linear relationship between $C_i(N)$ and $I(N)$. To eliminate that connection, two sets of simulations were performed. These simulations used 4, 6, and 8 nodes with $T_D = 30$ while varying both α and λ.

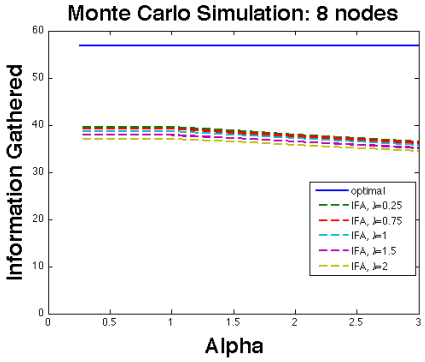
In the first, $C_i(N)$ was set as a constant (1) for all nodes N. Figures 4.3 and 4.4 show the



(a) 4 information containing nodes



(b) 6 information containing nodes



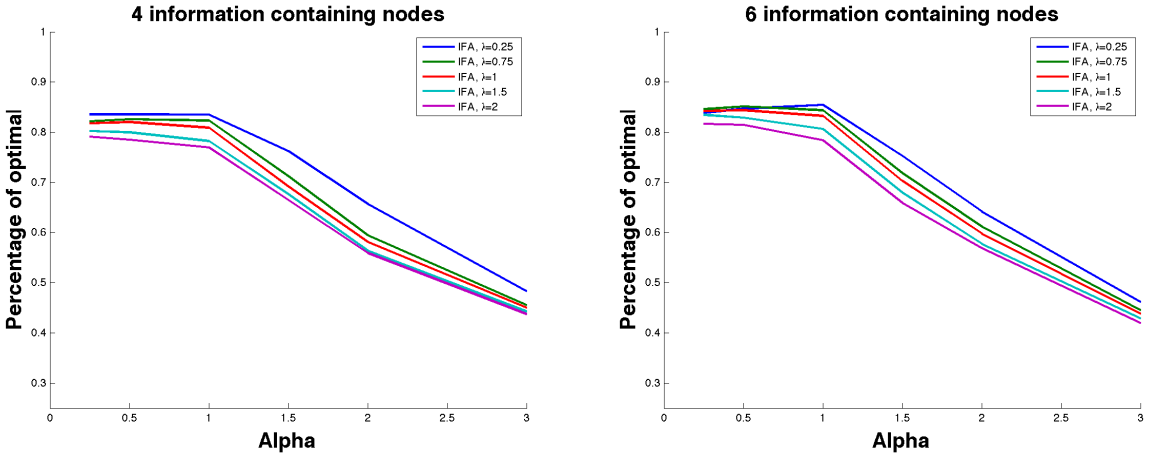
(c) 8 information containing nodes

Figure 4.4: Mean Information Gathered: α vs. λ, constant $C_i(N)$
For $\alpha \leq 1.0$, performance is largely the same.
When $\alpha > 1.0$, Performance slowly decreases.

percentage of optimal and the information gathered respectively as α and λ varied. All of the values of λ behaved in essentially the same fashion for all number of nodes. When $\alpha \leq 1$, performance was relatively steady. When $\alpha > 1$, performance declined slowly as α increased.

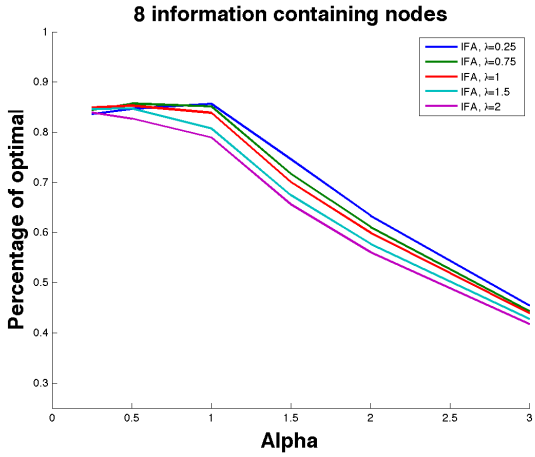
Performance holding steady while $\alpha \leq 1$ is almost certainly a ceiling effect reflecting the diminishing importance of $C_i(N)$. In particular, as $C_i(N)$ becomes less of a detriment when calculating $M(N)$, then $I(N)$ and the transit costs matter more. This means the IFA will start to seek out nodes that balance high information and low travel costs. At a certain point, this balance cannot be altered substantially, meaning the IFA will always find the same path even as α changes.

For these particular simulations, that point was $\alpha = 1$.



(a) 4 information containing nodes

(b) 6 information containing nodes

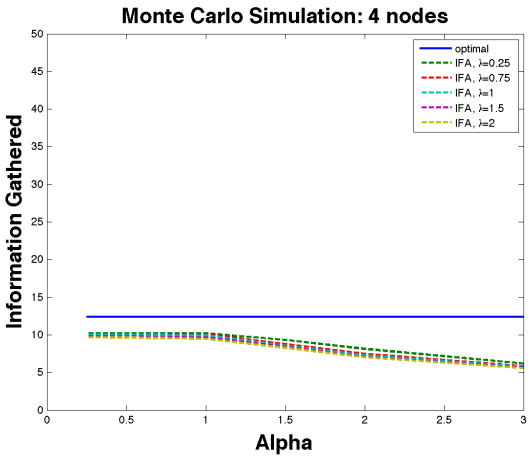


(c) 8 information containing nodes

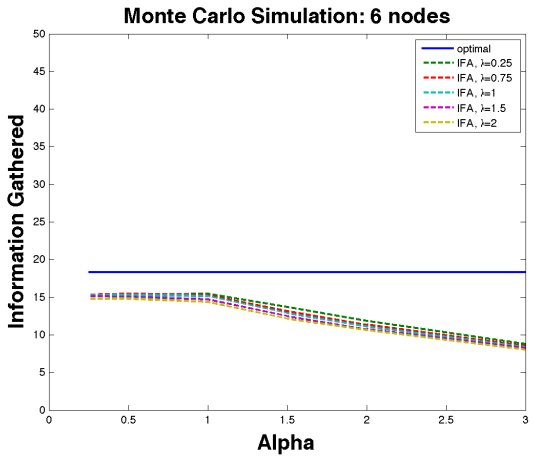
Figure 4.5: Mean IFA performance as percentage of optimal: α vs. λ, random $C_i(N)$
For $\alpha \leq 1.0$, performance is largely the same.
When $\alpha > 1.0$, Performance sharply decreases.

The more α increases above 1, the more the harvesting cost for a node negatively impacts T_E, and thus decreases $M(N)$. The travel times to N and from N to G become less important. Thus, as α increases, the IFA becomes biased towards visiting nodes with high information, even if those nodes also have high travel costs. This may prevent it from being able to visit lower information nodes later. In certain cases, the weighting of the harvesting cost can also get driven high enough

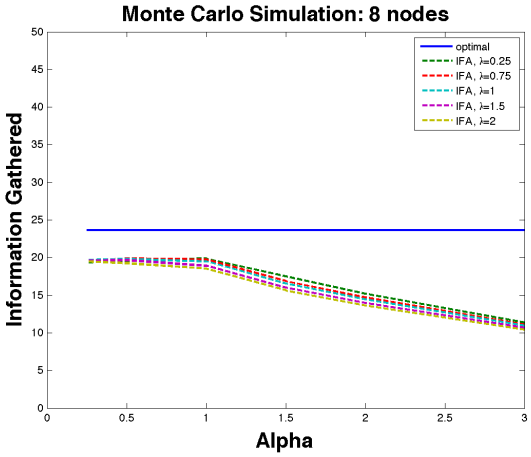
that the IFA is simply able to visit fewer nodes than for a lower value of α.



(a) 4 information containing nodes



(b) 6 information containing nodes



(c) 8 information containing nodes

Figure 4.6: Mean Information Gathered: α vs. λ, random $C_i(N)$
For $\alpha \leq 1.0$, performance is largely the same.
When $\alpha > 1.0$, Performance sharply decreases.

In the second, $C_i(N)$ was set randomly between 1 and $2N$ independently of $I(N)$. Figures 4.5 and 4.6 show the percentage of optimal and the information gathered as both α and λ varied. Similar to when $C_i(N)$ was constant, performance stays relatively stable for $\alpha \leq 1$, but then drops off fairly sharply. In fact, this case is the worst of all the various simulation cases. This result is also not surprising. As α increases the IFA is going to be biased towards nodes with lower

values of $C_i(N)$. Since $C_i(N)$ is assigned randomly to each node, the effect is that as α increases the sequence of nodes the IFA visits tends towards randomness and away from a pattern that can optimize the information gathered.

4.2 Monte Carlo Simulations with Stochastic Obstacles

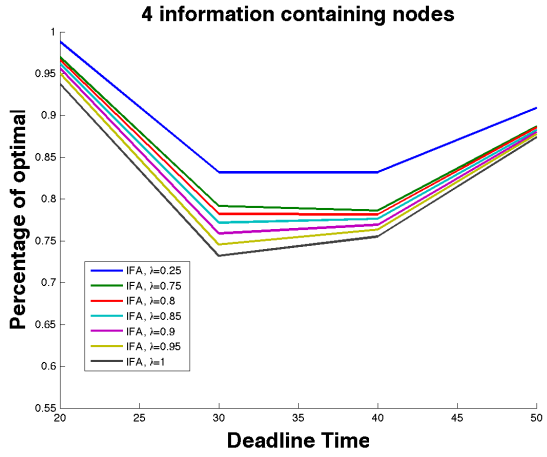
A second set of Monte Carlo tests was performed that was identical to the first in almost every respect. The major difference in this second set of tests is that one of the pathways between information containing nodes was randomly chosen to have a stochastic obstacle placed across it. This was implemented by designating one of the arcs connecting two information containing nodes as having a traversability probability, $P(T)$ chosen from a Gaussian normal distribution with $\mu = 0.75$ and $\sigma = 0.25$, $\mathcal{N}(0.75, 0.25)$, bounded to be between 0 and 1.

As the robot navigated through the environment, if it encountered this arc as the next arc in its path then at that point the weight of the arc was either set to ∞ (impassible) or to its initially estimated value with a probability equivalent to the traversability probability. If the arc was deemed to be impassible, the robot would re-plan before proceeding.

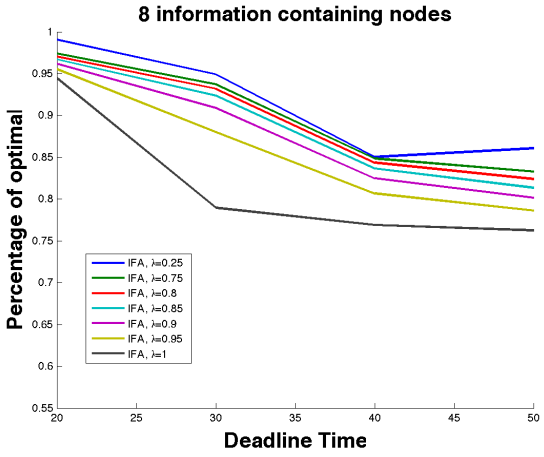
The optimal amount of information that could be gathered within the environment was determined by calculating the optimal amount of information assuming the stochastic arc was traversable (the stochastically optimal amount, O_s), then doing the same assuming it was not traversable (the deterministically optimal amount, O_d), and then taking a weighted combination where $O = P(T)O_s + (1 - P(T))O_d$.

As before, Figure 4.7 shows the average performance of the IFA as a percentage of the stochastically optimal path. The IFA performed very similarly in the stochastic case to how it performed in the completely deterministic tests performed above in that the averages are very similar, both within specific conditions as well as the overall averages.

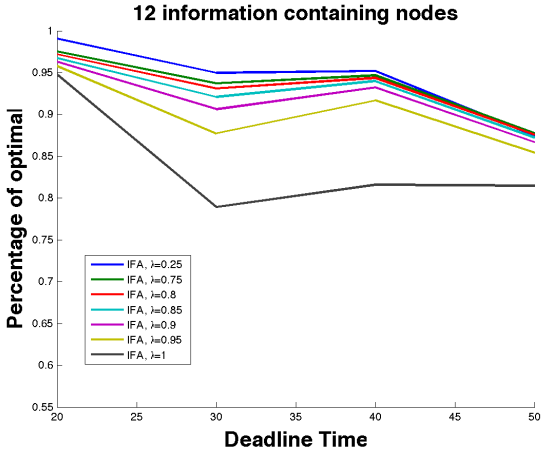
There are two ways in which the performance of the IFA in stochastic environments differed



(a) 4 information containing nodes



(b) 8 information containing nodes

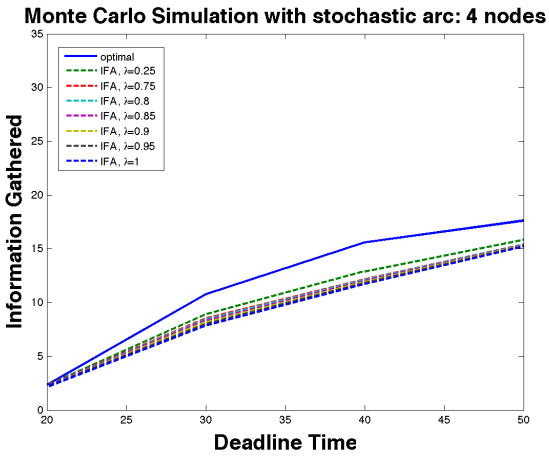


(c) 12 information containing nodes

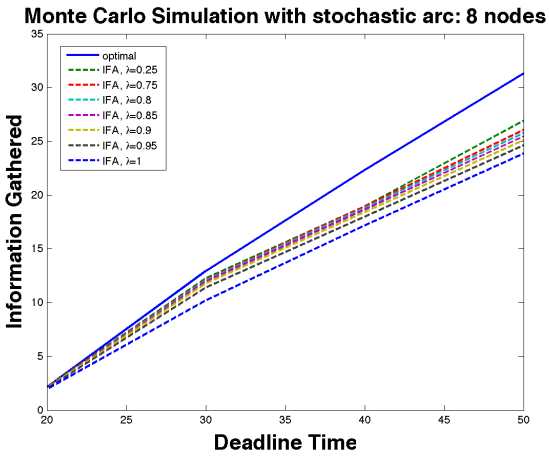
Figure 4.7: Mean IFA performance as a percent of optimal: 1 stochastic arc
Analyses are now restricted to $\lambda \leq 1.0$ (see text for details).
Performance is similar to the deterministic cases (see Fig. 4.1)

from its performance in deterministic environments. The first is that the minimum performance examples for the stochastic were somewhat worse, with several examples of minimum performance less than 10%. Despite that, the second difference is that there were, overall, fewer cases in which the IFA performed as poorly as in the deterministic version. Across the 4800 trials when $\lambda < 1$, only 48 trials resulted in performance less than 50% of optimal. This is only 1% of all the trials. Only 450 trials resulted in performance less than 70% of optimal, which represents only 9.4% of total trials.

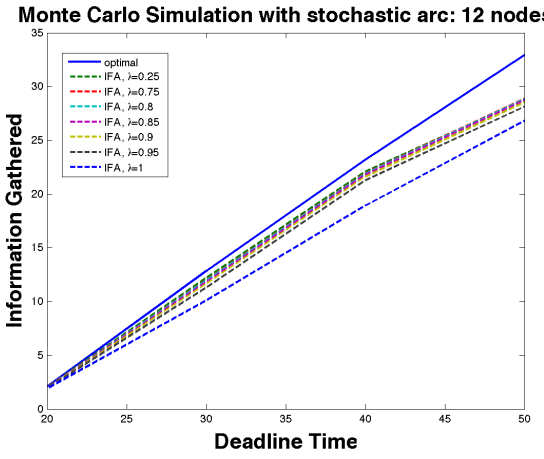
Figure 4.8 shows the information gathered by the IFA in the Monte Carlo tests with a stochastic obstacle relative to what the optimal path would have gathered. As was seen with the results based on the percentage of optimal, the raw information collected is also similar to the case where there was no stochastic obstacle.



(a) 4 information containing nodes



(b) 8 information containing nodes



(c) 12 information containing nodes

Figure 4.8: Mean Information Gathered: 1 stochastic arc
As before, performance mirrored the deterministic case (see Fig. 4.2).

4.3 Baseline tests

Overall, the results from the Monte Carlo simulations provide clear evidence that the IFA is a viable planning algorithm for the information gathering problem. In this second set of simulations, we attempted to model possible task environments into which a science gathering robot might be dispatched, and compare the results of the IFA to what a brute-force optimal path generator would have gathered[1]

The environments constructed in this section all share some basic properties. Information patches are modeled as nodes in the graph with an expected information content, $I(N)$, and an information harvesting time equal to the magnitude of $I(N)$. These nodes were then connected with bidirectional arcs in particular arrangements designed to loosely model potential patterns of information distribution within an uncluttered environment. For all environments, the start node, S, had only outgoing arcs and the goal node, G, had only incoming arcs. Once again, if multiple nodes had the same Mass the tie was broken by choosing the one closest to the robot's current node, and further ties were broken randomly.

4.3.1 Basic Simulation

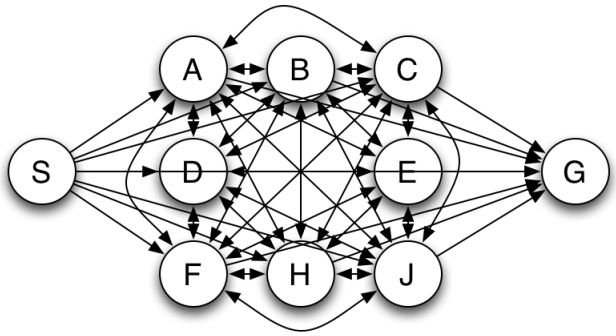


Figure 4.9: Basic Simulation environment
Every node is connected to every other node.
All arcs are deterministic with a cost of 1.
$I(N) = 1$ and $C_I(N) = 1$ for every node except S and G.

[1]Note: Because generating optimal paths gets exponentially more computationally expensive with each new node and set of arcs, the overall number of nodes and arcs was relatively restricted for all of these examples. The IFA by itself can handle extremely large environments very quickly. See Section 4.4.

This simulation was not designed to have any sort of physical analogue, but just as a reality check to establish how well the IFA would operate under very simplistic conditions. The environment had 8 information containing nodes, each of which contained an equal amount of information. S was connected to each node with an equal cost, and each node was further connected to G with an equal cost. Each node was also fully connected with every other node and, again, all of these arcs had equal costs. (In all cases the costs were 1, but since they were all equal the magnitude does not actually matter. See Figure 4.9.) The primary variable of interest here was T_D, essentially to ensure that the algorithm would behave as expected (visit as many nodes as possible before Bingo Time). For this simulation, λ was set to 0.25.

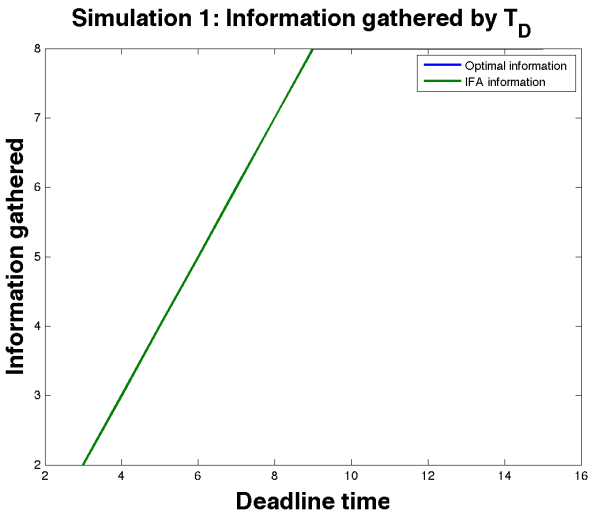


Figure 4.10: Basic Simulation results
Since all paths are basically the same, the IFA is always optimal.

In this simplistic case the IFA always harvested the optimal amount of information. Figure 4.10 shows the amount of information gathered as a function of T_D. There is only one line in the graph because both the optimal amount of information and the amount the IFA harvested are the same.

The IFA is always optimal because, with everything equivalent for all of the nodes in the graph, all of the nodes have the same Mass. The exception is the goal, which will have the lowest

Mass due to not having any information. Furthermore, every time the information gets harvested from a node, N, $I(N)$ gets set to 0, meaning that node will not get harvested again. Thus, the IFA simply keeps harvesting information from unharvested nodes until it either runs out of nodes to harvest or hits Bingo Time. This is, of course, precisely the optimal behavior.

4.3.2 Basic Simulation: Bottleneck

The environment in this simulation is precisely the same as the previous one, except now S and G are only connected to one (different) node each (see Figure 4.11. Once again, T_D was varied and the expected behavior was the same. In this case, though, the IFA did not quite conform to the expected behavior. Rather, as can be seen in Figure 4.12, the IFA visited one less information containing node than the optimal case.

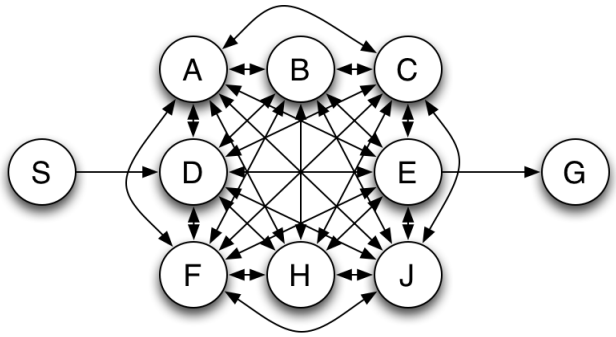


Figure 4.11: Bottleneck environment
S and G are each only connected to one other node.
All other nodes are fully connected to each other.
All arcs are deterministic with a cost of 1.
$I(N) = 1$ and $C_I(N) = 1$ for every node except S and G.

The explanation for why this happens is actually rather simple. Obviously the first node harvested in either case is the node connected to S (node D in Fig. 4.11). In the optimal case, given sufficient time, the last node harvested before traveling to the goal should be the one node connected to the goal (node E in the figure). Figure 4.13a shows such a path for $T_D = 5$.

However, in the context of the IFA, since the exploration time is calculated based on the time both to get from where the robot is to a given node and the time to get from that node to the

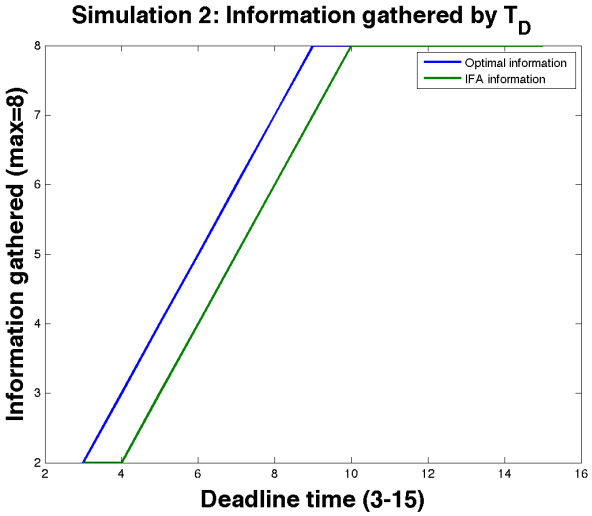
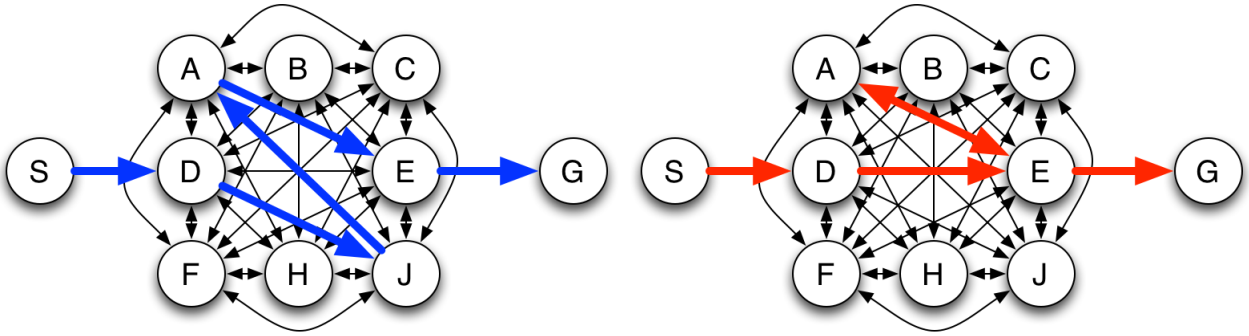


Figure 4.12: Bottleneck simulation results
The IFA is short of optimal because its second step was always to E.



(a) Optimal path (b) IFA path

Figure 4.13: Paths through the Bottleneck environment when $T_D = 5$.
Note the IFA path goes from E to A, then back to E because it has hit Bingo time.
Thus, the optimal path gathers information from four nodes, but the IFA path only gathers from 3.

goal, after harvesting node D, the node with the best T_E will be node E. This is because it is the same cost to get from node D to any of the other information containing nodes, but node E is the only node whose path to G does not have to go through another information containing node. Thus, its cost to go to G is 1 while the cost to G for all the other nodes is 2. Since all the information containing nodes have equal information, this means that E has the lowest exploration time, and thus the highest Mass (see Sec. 3.2). Thus with the IFA, node E is always the second node visited, which makes it slightly more costly to then proceed and visit other nodes, since the robot still has

to backtrack through E to get to G. Figure 4.13b shows a typical path generated by the IFA for this environment.

4.3.3 Linear Array 1

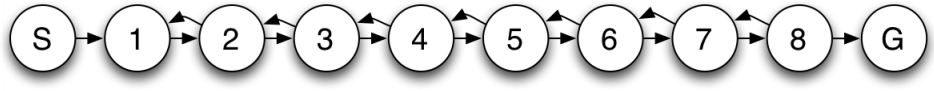


Figure 4.14: Linear environment with nodes farther from S having higher information content.

In this simulation the nodes are equally spaced along a straight line between S and G. The node with the most information was farthest away from S and the one with the least was closest, with the other nodes ordered in between (see Figure 4.14). One physical analogue for such an arrangement is if a rover is driving around looking for a particular kind of rock. Satellite imagery has generated a map with a 2-D Gaussian probability density function (PDF) centered at a point off in the distance, near G. Thus, the nodes near S are going to have a low probability of the rock of interest being present, meaning they will low expected information content. The ones near G, being near the center of the Gaussian PDF, will have a higher probability of the rock being there, and thus higher expected information content.

Here we were interested not just in varying T_D, which should again affect how many nodes are visited, but also λ. λ in particular should affect which node gets visited first by determining how big an effect the node's distance from S, via the exploration time, has on the node's Mass. Once again, $\alpha = 1$ and $C_I(N) = I(N)$.

As we saw in the Monte Carlo simulations, $\lambda \geq 1$ tended to significantly degrade performance as it eliminates the influence of information in the Mass function. In this particular configuration, though, favoring distance over information is actually advantageous, though. When $\lambda < 1.0$, the IFA picks the node with the highest information that it can reach first. This then forces it to backtrack to gather information from other nodes. When $\lambda \geq 1.0$, the IFA is biased to pick information containing nodes based on distance rather than information. This means it ends

up choosing nodes in order away from S, and it is able to actually gather more information.

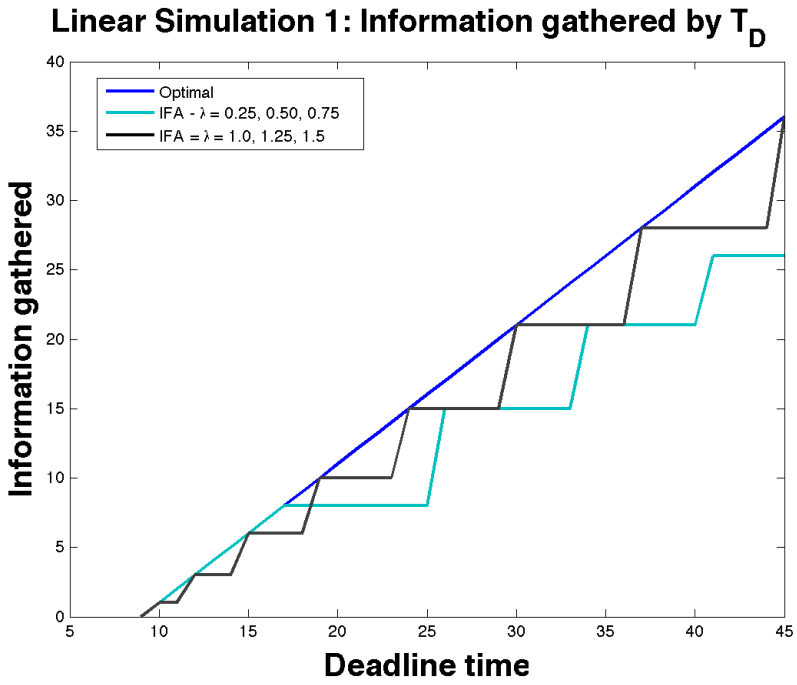


Figure 4.15: Linear simulation 1 results
When $\lambda < 1.0$ the IFA tends to go to the highest information node it can first and then backtrack. When $\lambda \geq 1.0$, it is biased to consider distance more, and so it travels to as many of the closest nodes first as it can.

4.3.4 Linear Array 2

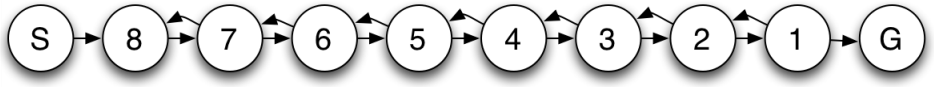


Figure 4.16: Linear environment 2 — the reverse of the previous environment

This simulation used the same set-up as Linear Simulation 1, but in this case the nodes were ordered with the highest information closest to S and the lowest information one farthest (see Figure 4.16. As a result, the optimal behavior is clearly that the nodes should be harvested in order between S and G with no backtracking. The physical analogue is obviously also the reverse of Linear Array 1. Here the expectation is that the robot should always stop at the highest information node that it can explore within the deadline time, then the second highest, etc.

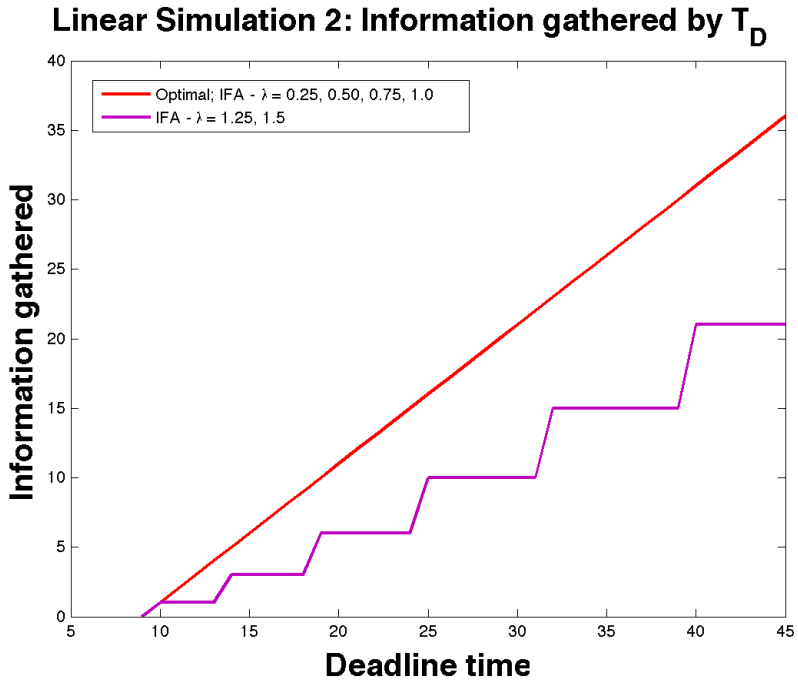


Figure 4.17: Linear simulation 2 results.
In this case, it is always optimal to travel to the highest information node possible first and then visit smaller ones along the way as possible. Thus for $\lambda \leq 1.0$, the IFA always generates optimal solutions. For $\lambda > 1.0$, since it visits closer nodes first it generates suboptimal solutions.

As can be seen from Figure 4.17, the results depended on λ. As we have seen with previous simulations, performance tends to change once $\lambda \geq 1.0$. Also as before, the higher λ performed substantially worse than the lower values. The reason for this is that higher values of λ put more of an emphasis on the exploration time. The nodes closer to the goal will always have better exploration times because they are close to the goal and have very low information harvesting costs. The nodes near the start will have worse exploration times because they are farther from the goal and have higher information harvesting costs. Lower values of λ put more of a premium on information when calculating Mass, and thus the higher-information nodes near the start will become more attractive.

4.3.5 Outlier 1

This simulation has multiple low information nodes near S and one very high information node far away from both S and G. There were 7 low information nodes that all had an expected

information value of 1. Their distance from S and from each other was 1. All the nodes, including the outlier, had a distance of 5 from G. We then varied both the distance of the outlier from S (from 1-10) as well as the expected information of the outlier (2-20). The outlier was also connected to the 7 nodes in the low-information cluster with the distance as from S (see Figure 4.18). We then, based on the previous simulations, chose three values of λ (0.25, 0.75, 1.0, and 1.50) that have tended to represent divisions between different types of plans the IFA generates. T_D was set to 35, which was the cost of traveling from S to G while harvesting the outlier in the highest-distance + highest-information case ($T_d(S, O) = 10, C_I(O) = 20$).

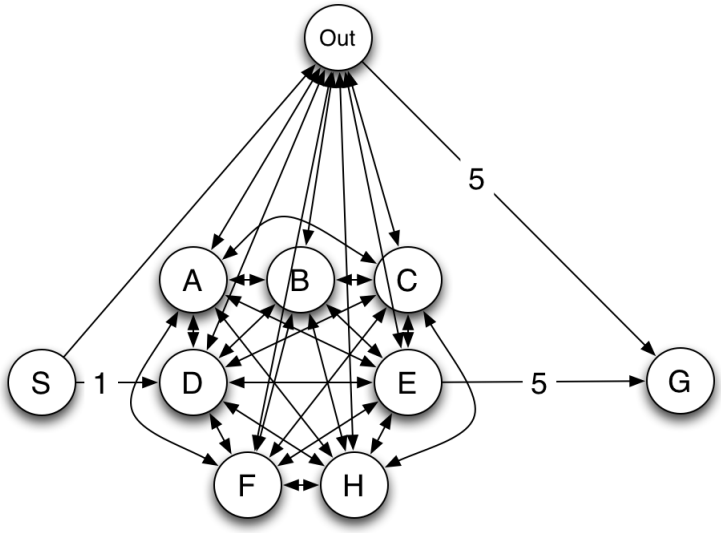


Figure 4.18: High information outlier

Figure 4.19 shows the results. As we would expect, when a lower value of λ puts the emphasis more on information, then the algorithm tends to choose the outlier node most of the time. Once the information content of the outlier exceeds 7 (the total information of all the nodes in the low-information cluster), then the IFA chooses the outlier every time (see Fig. 4.19a).

When the value of λ is such that there is more balance in the Mass metric between information and exploration time, then the selection of the outlier is much more dependent on how far the outlier is from S. In Figure 4.19b we can see that the farther the outlier is from S, the more

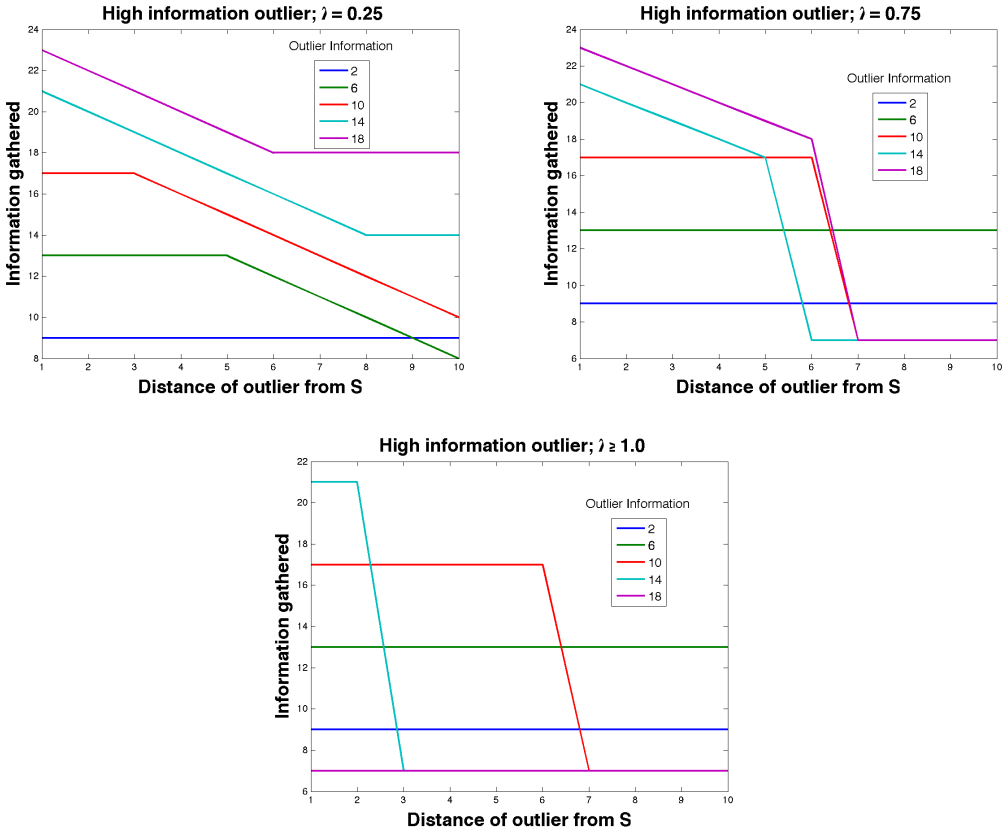


Figure 4.19: Outlier 1 results for three values of λ.
The IFA tries to visit the high information outlier if it can. Otherwise it visits as many nodes in the low information cluster as possible.

information it needs to contain before the IFA will decide to go explore it. This makes sense from a practical perspective. In many real-world situations like this, going to explore the outlier essentially means giving up on the multiple lower-information patches that are right next to the robot. Thus, the potential payoff for making that sacrifice needs to outweigh the information being given up in the more immediate vicinity of the robot.

Finally, if the value of λ is set to emphasize exploration time over information, then the IFA will always stay in the low-information cluster, at least for outlier information ≤ 20 (see Fig. 4.19c). As the outlier information increases we will eventually find a threshold where the IFA will choose to explore it, meaning that the algorithm largely behaves as we would intuitively expect.

4.3.6 Outlier 2

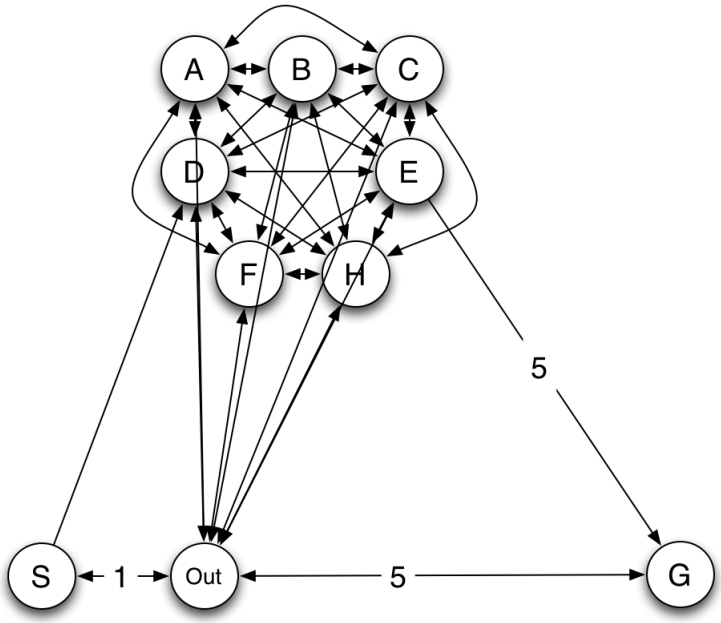


Figure 4.20: Low information cluster outlier

This is the complement of the Outlier 1 simulation, where the positions of the high-information node and the low-information cluster are reversed (see Figure 4.20). This simulation was done primarily as a reality check for understanding the results of the first Outlier simulation. The basic parameters were the same, except $T_D = 26$, which was the cost of getting information from the high-information node in the case where its expected information was 20.

As with Outlier 1, the behavior of the IFA depended upon whether more emphasis was placed on information or exploration time in the Mass metric. Figure 4.21 shows the results for $\lambda = \{0.75, 1.0, 1.50\}$. (The results from $\lambda < 0.75$ were exactly the same as for $\lambda = 0.75$, so they are not presented here.) When the emphasis was on information (Fig. 4.21a), then the IFA visited as many nodes in the low-information cluster as possible before visiting the high-information node and proceeding to the goal. As the emphasis was shifted to the cost of exploring a particular node, the IFA became more likely to focus on the closest nodes, which in most cases was just the

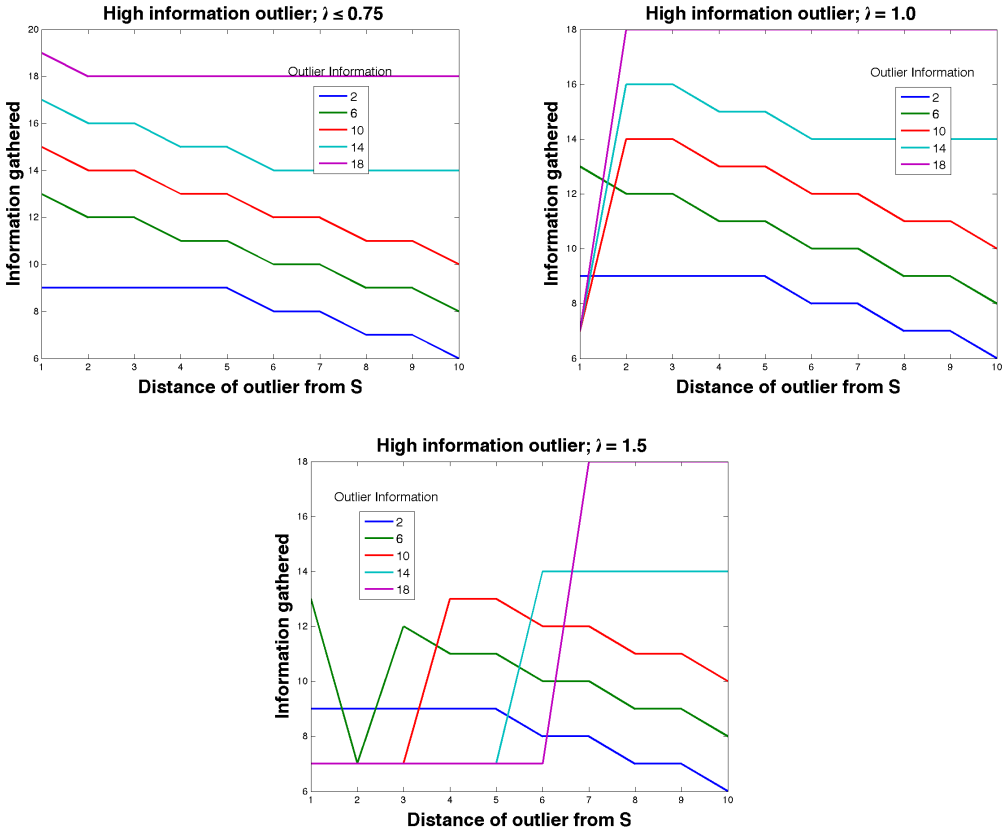


Figure 4.21: Outlier 2 results for three values of λ.
Similarly to the Outlier 1 simulation, the IFA always visits the high-information node when it can, and if there is sufficient time it tries to visit nodes in the low information cluster as well.

high-information node.

4.3.7 Scout the base

In the last of the baseline tests, S and G are next to each other in the center of a circle of information containing nodes (see Figure 4.22). The information value of each of these nodes is set randomly, and each node is equidistant from both S and G. This simulation is analogous to a robot needing to conduct a survey of the area around a central base. T_D and λ were both varied and, as before, $\alpha = 1.0$ and $C_I(N) = I(N)$.

The results are similar to those of the other simulations presented thus far. Smaller values of λ tend to generate more optimal results because they lead to the IFA choosing to visit nodes

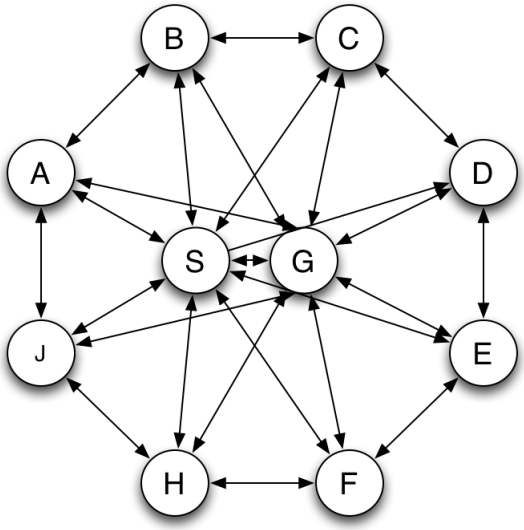


Figure 4.22: Scout the base environment

roughly in order from the most possible information to the least. Larger values of λ cause the robot to focus on nodes that are easier to visit and harvest from its current location, which means sometimes it does not go to the optimal node but to the easiest one.

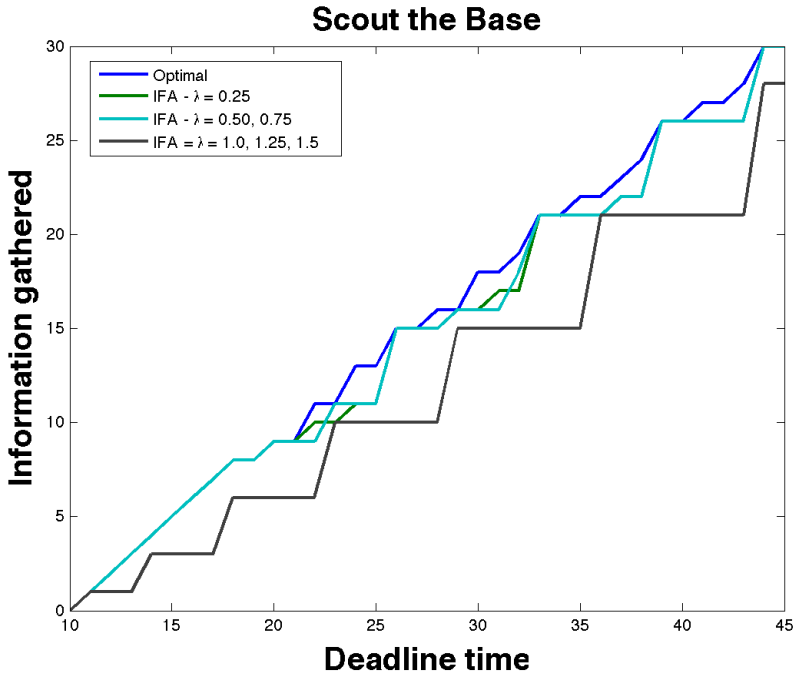


Figure 4.23: Scout the base results

4.4 Large Environment Tests

A number of large environment tests were also conducted to explore the performance of the IFA in more realistic scenarios. In these tests a square grid was constructed that was 50 nodes on a side. Within this grid rectangular patches of information were placed where each patch contained multiple information-containing nodes. This more closely approximates what a real-world information foraging scenario might look like. There are three examples described below: a simple environment with just information, and environment with deterministic obstacles, and one with a mix of deterministic and stochastic obstacles. Unfortunately, with 2500 nodes, the size of these environments precludes finding the exact optimal solutions for comparison. 170! is the largest number that Matlab will return an answer for, on the order of 10^{306}.

For the planning in these simulations, each node in the graph was treated as an independent entity. In other words, the IFA has no specific notion of a patch of information containing nodes. Thus, when it performs its initial calculations of $T_s(S, N)$ and $T_d(N, G)$, it is doing that for all 2500 nodes in the graph. It then calculates $T_E(N)$ and $M(N)$ for all of the 412 information containing nodes, and decides which one to visit next without regard to what nodes are surrounding it. This is something that will be addressed in the discussion of future directions (Section 5.2).

4.4.1 Simple environment

Figure 4.24 shows the basic environment with information patches and no obstacles. The robot starts in the bottom left corner and navigates to the upper right corner, trying to maximize the information it collects along the way. Figure 4.25 shows the paths generated by the IFA for several values of T_D. Based on the Monte Carlo analysis showing only small differences when $\lambda < 1.0$, and poor performance for $\lambda \geq 1.0$, these simulations simply used $\lambda = 0.5$.

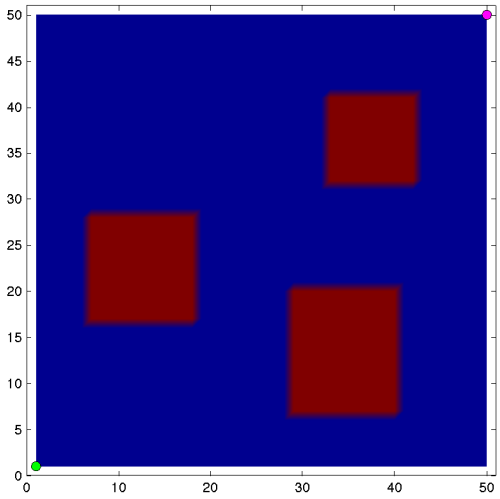
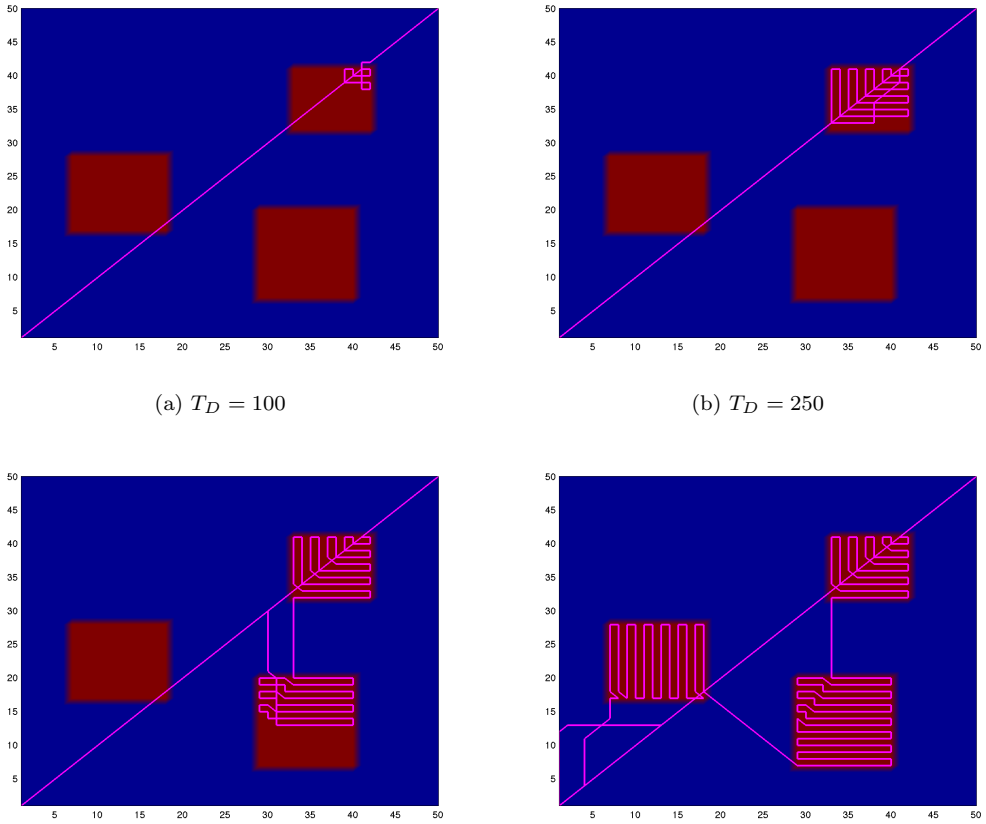


Figure 4.24: Simple environment with three information patches.
The green dot is S and the pink dot is G.

4.4.2 Deterministic obstacle

Figure 4.26 shows the same environment with the addition of a deterministic obstacle across the centerline path taken in all of the simple environment simulations. Figure 4.27 shows the paths generated for multiple values of T_D. Unsurprisingly, the IFA is easily able to compensate for the obstacle and generate paths that are largely similar to those generated in the environment without the obstacle.

In Figure 4.27d, the IFA has enough time before T_D to gather all of the information in the environment. One implicit bias of the IFA is that it will tend to maximize the amount of time it utilizes before T_D. This is a result of the fact that it will continue to explore nearby nodes as long as there are ones that can be explored, which is to say ones where $T_E(N) < T_D$. As time advances, the set of nodes where this is true shrinks until eventually, at Bingo Time, the only nodes left in the set are the ones along the shortest deterministic path to G. Any heuristic can be used to determine exactly which nearby nodes to visit. Figure 4.27d chooses randomly. There could also be heuristics specific to the domain or the information model, such as visiting nodes on the fringes of previously



(a) $T_D = 100$ (b) $T_D = 250$

(c) $T_D = 500$ (d) $T_D = 1000$

Figure 4.25: Paths generated by IFA through large simple environment.

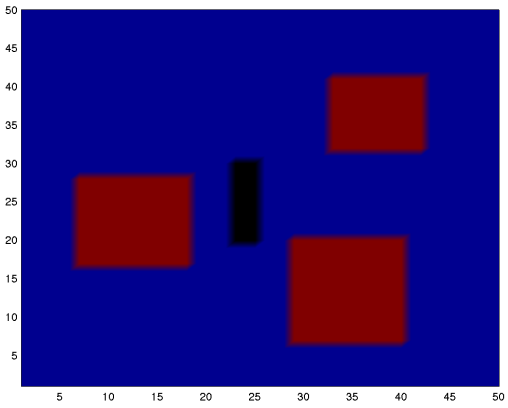
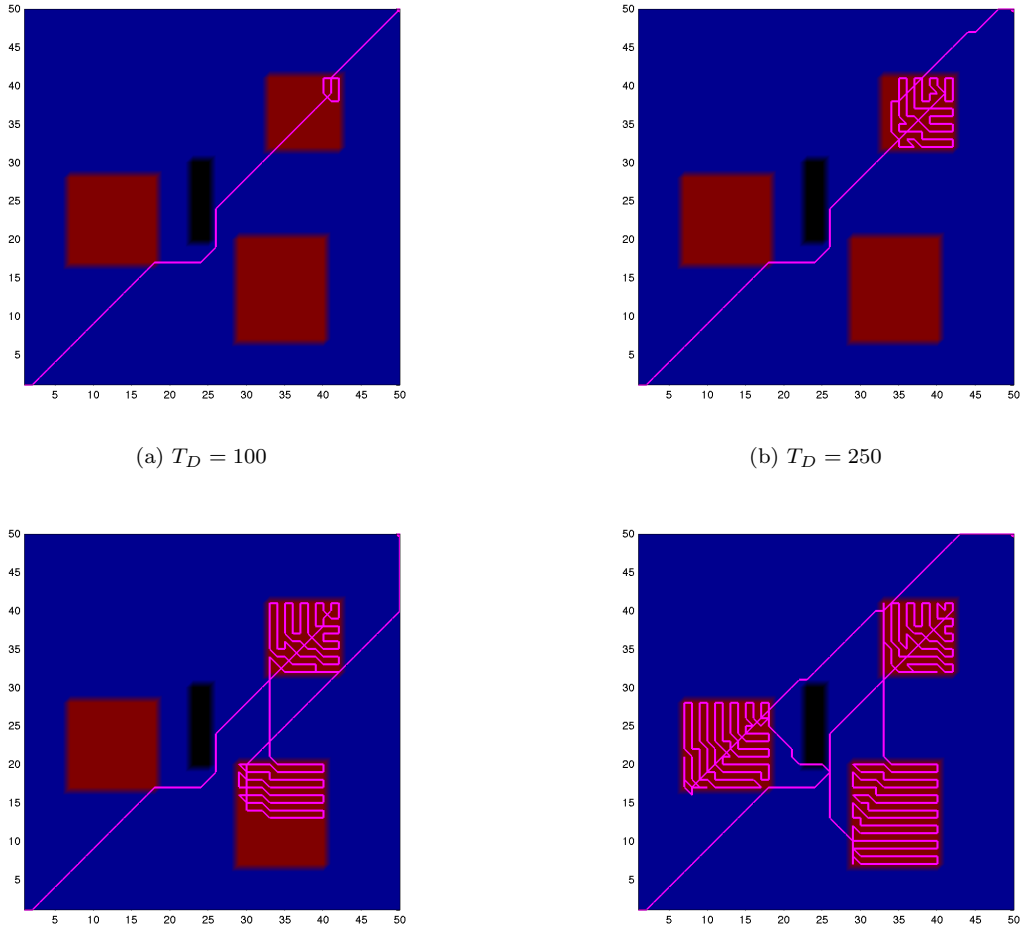


Figure 4.26: Simple environment with three information patches and one untraversable region (in black).

existing patches or making a random walk towards G. A more extensive discussion of this issue can be found in Section 5.2.3.



(a) $T_D = 100$

(b) $T_D = 250$

(c) $T_D = 500$

(d) $T_D = 1000$

Figure 4.27: Paths generated by IFA through large environment with deterministic obstacle.

4.4.3 Stochastic obstacles

The final big environment simulation places stochastic obstacles as well as the deterministic obstacle from the previous simulation. When the robot approached a stochastic arc it was randomly determined to be traversable or not based on the prior probability of traversability. As can be seen from Figure 4.29, the IFA still had no difficulty accommodating the stochastic arcs. As can

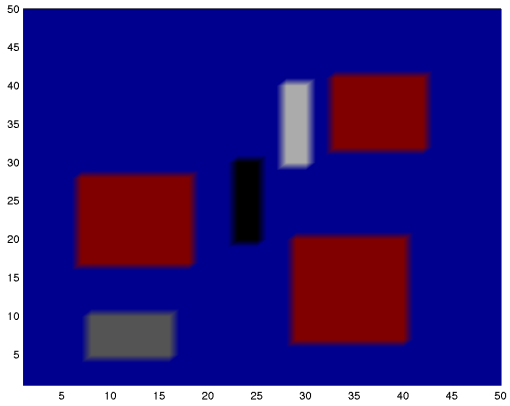


Figure 4.28: Simple environment with three information patches,
one untraversable region and two stochastically traversable regions.
Dark grey region has a probability of being traversable of 0.60. Light grey region is 0.75.

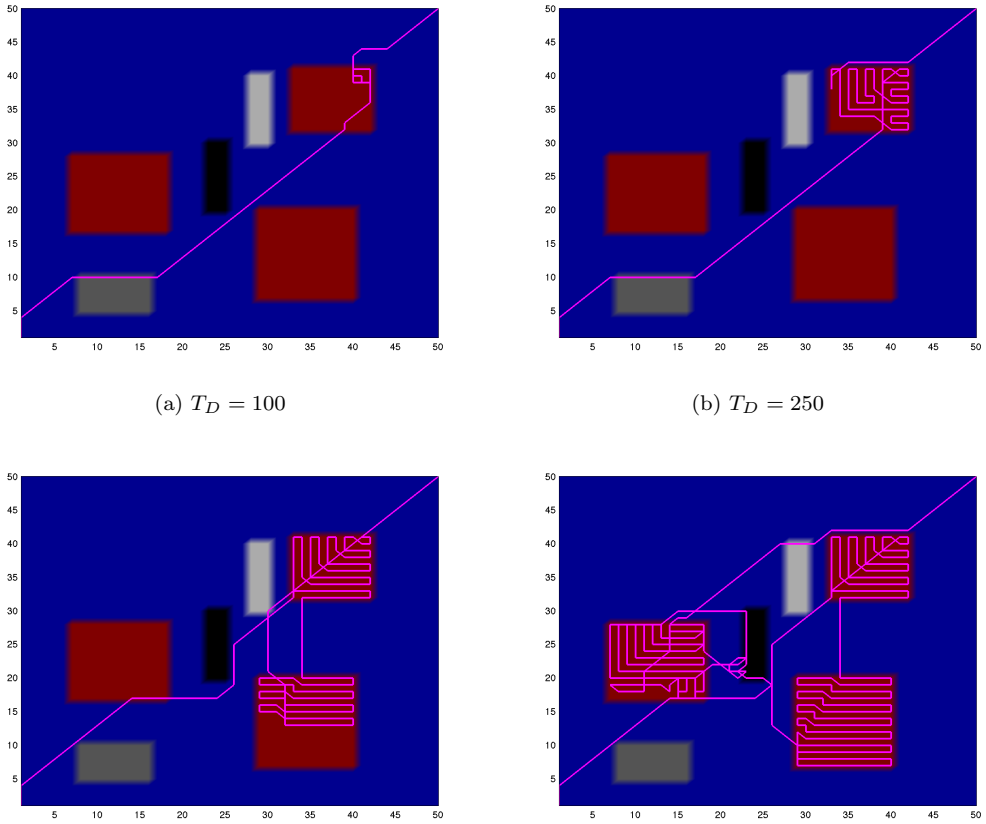
especially be seen in Figure 4.29d, the IFA would avoid a stochastic region that was determined to be untraversable, but cross over stochastic arcs that were deemed traversable.

4.4.4 Ocean-like environments

Robots are being frequently used to gather scientific data from the ocean. A high-profile example is the use of an autonomous underwater vehicle (AUV), the *Bluefin 21*, to search for Malaysian Airlines flight 370 in the Indian Ocean. The Monterey Bay Aquarium Research Institute (MBARI) has been using AUVs to survey Monterey Bay for close to 15 years [27, 78, 88].

In some ways, the ocean is a simpler environment for path planning. In many areas, the only real "obstacles" are the shoreline and the ocean floor. An AUV can safely operate by maintaining maximum depth and staying away from particular areas. As a result, most research AUVs do not operate with forward-scanning sonar (as opposed to the bottom-scanning sonar that mapping AUVs use) or otherwise invest resources into object avoidance.

On the other hand, the ocean is far more dynamic than most terrestrial environments. Scientists are frequently interested in sampling phenomena that are highly dynamic both spatially and temporally. For example, one type of feature that oceanographers are often interested in is



(a) $T_D = 100$ (b) $T_D = 250$

(c) $T_D = 500$ (d) $T_D = 1000$

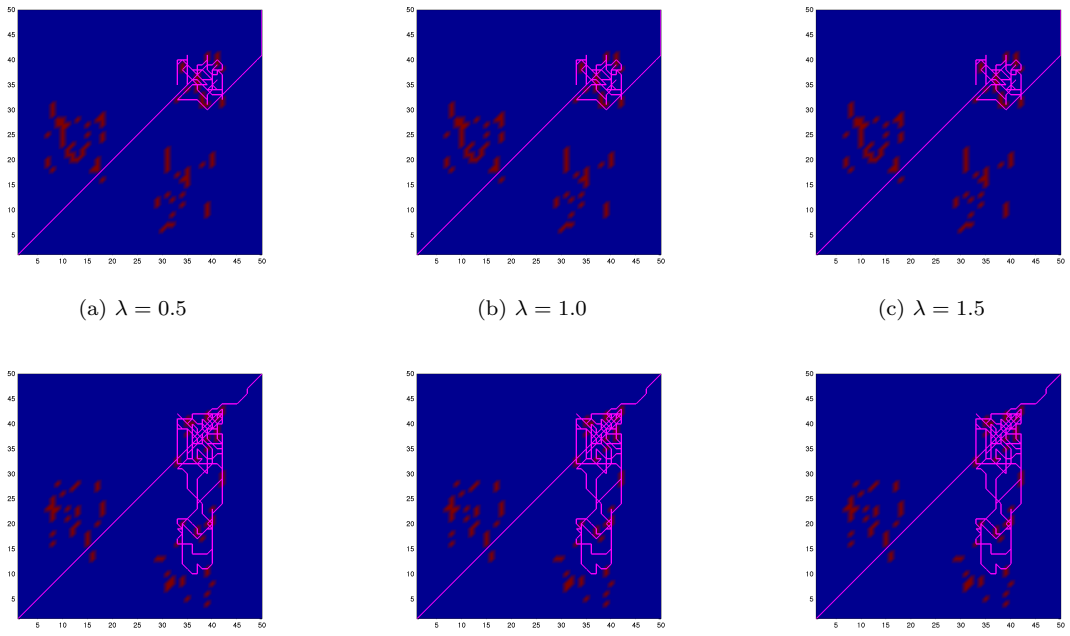
Figure 4.29: Paths generated by IFA through large environment with both deterministic and stochastic obstacles.

something called a *front* [7]. A front is essentially a boundary between different types of water. One of the most famous is the front between the Gulf Stream, which contains rapidly moving warm water, and the North Atlantic Ocean, which contains slowly moving cold water. Fronts are interesting to oceanographers because they are often areas where a large diversity of organisms converges (e.g., [43, 74]).

Belkin and O'Reilly [8] have developed a widely used algorithm for identifying fronts from satellite imagery of sea surface temperature (SST). From the time a satellite image is captured and processed until an AUV can arrive on station to being sampling a front, several hours will have passed. Because of the dynamic nature of ocean features, the front may have moved several

kilometers, or disappeared entirely. The strength of the front — the sharpness of the distinction between the water masses on either side — can also rapidly change over time. As a result, while there may be prior probabilities about where the front is, the environment model has to account for the fact that these priors are going to diffuse over time. The longer it takes for the AUV to arrive on station, the more spread out the probability patch will need to be in order to reasonably define the necessary search area.

To test the IFA under these conditions, a simulation based on the simple simulation in Section 4.4.1 was developed. The primary difference was that at random time intervals, information "diffused" from a given node to its adjacent nodes with a random probability. Figure 4.30 shows the paths generated for three different values of λ when $T_D = 250$ and $T_D = 500$.



(a) $\lambda = 0.5$ (b) $\lambda = 1.0$ (c) $\lambda = 1.5$

(d) $\lambda = 0.5$ (e) $\lambda = 1.0$ (f) $\lambda = 1.5$

Figure 4.30: Results from an ocean-like simulation with dynamic information movement.
a, b, and c: $T_D = 250$; d, e, and f: $T_D = 500$.
Red areas show final configuration of information. Pink line is the robot's path.

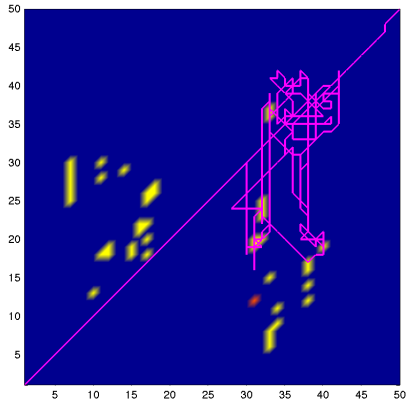
The IFA exhibits the same general behavior as was seen in Figures 4.25b and 4.25c. When

$T_D = 250$, as in Figure 4.25b, the IFA concentrates its search in the patch in the upper right quadrant of the environment, closest to the goal. When it has more time ($T_D = 500$) it proceeds to search the path in the lower right quadrant when it has exhausted the upper right.

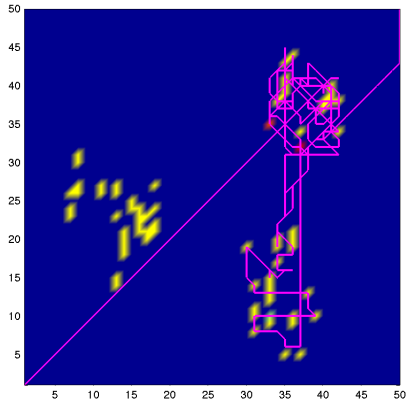
The primary difference is that the foraging behavior of the IFA within a patch was substantially less systematic than it had been with static information. This is entirely a function of the random diffusion of information over time. The IFA is making some attempt to be systematic, but every time information moves it stops and replans. Information may diffuse back into an already harvested node. Since the IFA is not explicitly keeping track of already visited nodes, it may go back to one of these nodes if it has a higher mass. It may also be the case that the next node on the IFA's intended path no longer has information, and thus it needs to find a new node to harvest. The overall effect is that the path, while still roughly contained within the patch, has more randomness in its movements.

In these particular simulations, the value of λ has no effect on the final path. This is due to the fact that every information containing node in the environment has the same $I(N) = 1$, which means that no matter what the value of λ is, $T_E(N)$ will drive the decisions about where to navigate. To better understand the effect of λ on paths planned through environments with dynamic information, a similar environment was created except that instead of $I(N)$ always being set to 1, it was set to a random value derived from the normal distribution with $\mu = 1$ and $\sigma = 0.25$.

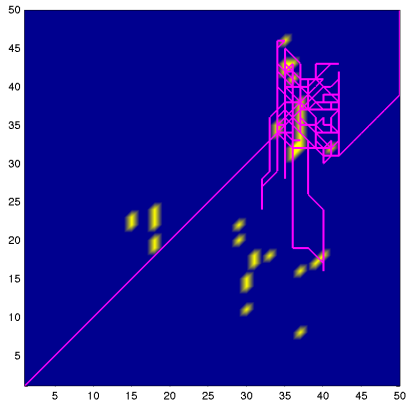
Figure 4.31 shows the paths generated by the IFA through this environment for different values of λ. One thing to note is that the final configuration of the information in the environment is different across the four paths. This is because the paths take different numbers of steps, noted below each figure, meaning there are different numbers of perturbations of the information. Also note that the two paths generated for the values of $\lambda > 1$ are both identical. These paths are more concentrated in the patch at the upper right of the environment. This is likely due to the fact that when $\lambda > 1$, $T_E(N)$ becomes more important than $I(N)$. Thus, the IFA is biased to examine nodes



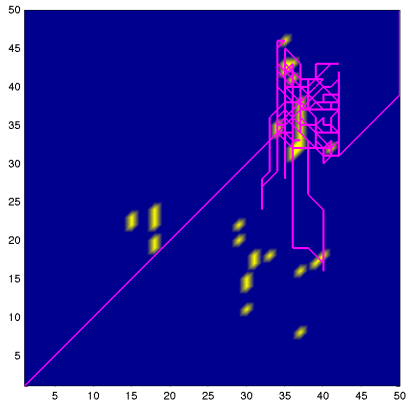
(a) $\lambda = 0.5$; 356 steps; Info gathered = 79.25



(b) $\lambda = 1.0$; 373 steps; Info gathered = 101.00



(c) $\lambda = 1.5$; 351 steps; Info gathered = 80.23



(d) $\lambda = 5.0$; 351 steps; Info gathered = 80.23

Figure 4.31: Results from an ocean-like simulation with random initial information and dynamic information movement. $T_D = 500$.
Non-blue areas show final configuration of information. Pink line is the robot's path.

that are nearby, so it will not move to a patch farther away until there is almost no information left in its current patch.

In the cases where $\lambda \leq 0$, as λ decreases the IFA is more influenced by $I(N)$, and thus more willing to travel to a different patch to explore if it senses nodes there with higher $I(N)$. It can especially be seen in Figure 4.31a where there are many more paths between the patches on the upper right and the bottom right than for the other values of λ.

All this extra traveling comes at a cost. As can be seen from the "Info gathered" statistics associated with each figure, the time spent moving around reduced the overall amount of time that could be spent harvesting information. Thus, the condition where $\lambda = 0.5$ was the worst performing condition within this dynamic environment. Likewise, when the IFA restricted itself to the patch in the upper right corner, it failed to move to a patch containing nodes with larger values of $I(N)$. At least within this specific simulation, $\lambda = 1.0$ seemed to strike the proper balance between minimizing travel time by staying within a patch, yet moving on to a new patch when the opportunities there appeared to be better.

4.4.5 Conclusions from the large environment simulations

The large environment tests make clear that the IFA is able to easily handle graphs that are larger and significantly more complex than the ones used in previous simulations. The IFA is able to adjust dynamically to information that is not known a priori, namely whether a given stochastic arc will ultimately be traversable or not. All of this happens relatively quickly (on the order of seconds inside of Matlab on a 2.8GHz i5 Macintosh computer). The ocean-like simulations show that the IFA can also operate in environments with very different types of information models. That is to say, the success of the IFA does not depend on the specific information model relevant to a particular environment. Together, these demonstrates the power of the IFA to scale up to complex domains, the flexibility of the IFA to be applied to many different types of foraging problems, and the utility of having a computationally tractable approach to the information gathering problem.

Chapter 5

Discussion

The Information Foraging Algorithm (IFA) was designed to enable a robot to maximize the amount of scientific information it gathered under deadline constraints while navigating through an uncertain environment. It was designed to guarantee that (within the limits of the robot's knowledge) the robot will meet whatever deadlines or horizon constraints are imposed upon it.

5.1 IFA Summary

The IFA works by generating a graph representation of the environment, often a grid, based on an initial map it receives reflecting stochastic knowledge of the environment generated by some external source such as post-processed satellite imagery or other remote sensing data. The user can use various graph-based planning algorthims (e.g., Djikstra's algorithm, A*, D*, etc.) in order to find lowest cost paths through the environment connecting the start and goal nodes to the information containing nodes of the graph.

Then the IFA generates a metric, Mass (see Section 3.2), for each node in the graph to reflect how desirable it is for the robot to visit that node and harvest its information. This desirability is based on the expected information value of the node and the remaining exploration time of visiting the node (defined in Sec. 3.2). Based on this, the IFA takes a greedy approach and directs the robot to navigate to the node in the environment with the largest mass.

If nothing in the environment changes, the robot will continue to navigate towards this

highest-mass node. If, however, it detects something is different in the environment from what it is expecting in its map, such as an arc cost having changed from uncertain to certain or the information content of a node having changed due to new data, the robot will stop and re-plan.

Because the IFA is fundamentally performing a series of minimum path graph searches, the planning and re-planning happen relatively quickly. The drawback to using graph-based methods, of course, is that finding an optimal solution to any sort of maximization problem is NP-hard. This necessarily means that the IFA is not finding optimal solutions. Rather, it is attempting to find relatively good, computationally tractable approximations to the optimal solutions.

The results from Chapter 4 clearly demonstrate that the IFA is achieving this goal. While it does not always collect the same amount of information as the optimal path through the environment would generate, it collects over 80% of the optimal amount of information in the average case, and collects more than 50% of the optimal amount in more than 98% of the paths it generates, even as environments get more complicated with more information containing nodes. This demonstrates that the IFA is at least a viable framework to use when constructing a physical robot to go out and perform scientific surveys of environments.

5.2 Future Directions

Obviously there are still improvements that could be made to the IFA. Since most of the environments it has thus far been tested in are simplified versions of real-world environments, it would be useful to get the algorithm to perform as well as possible before evaluating it in more real-world scenarios.

5.2.1 Planning ahead

Currently, the IFA takes a greedy approach and only plans for the next node it is going to visit and explore. It determines this next node based on the Mass metric. The problem with the greedy approach is exemplified in Section 4.3.2. In this simulation, the presence of the bottleneck

on the route to the goal, where every path to the goal had to go through one node, meant that the bottleneck node had a better opportunity cost than any other node. This meant the IFA selected it for exploration before it selected other nodes, even though any route through the environment would have to go back through that node eventually. This ended up wasting time that could have been used for exploration, causing that entire simulation to generate sub-optimal paths.

In this particular case, looking two nodes ahead instead of one would have avoided this issue. Thus, one of the first steps in improving the IFA should be giving it the ability to look ahead n steps, where n can be defined by the user. The trade-off, of course, is that looking ahead is what makes the maximization problem NP-hard in the first place, so n must be kept small enough that the problem is tractable. Essentially, this is a receding horizon control approach to the IFA.

This difficulty becomes especially pronounced when considering imposing a grid on the low-resolution satellite map of the real world. Frequently there are likely to be many nodes of the grid within each information containing blob. Thus n will have to be large for any significant benefit of looking ahead to be realized, which would make planning more computationally difficult. This leads to the second major thread of future research.

5.2.2 Dependent Nodes

Along with being greedy in its planning, the IFA also treats each node as independent of its surrounding nodes. In real world environments, though, this would rarely be true. If a given point in the environment is likely to contain a given feature, then, in general, points closer to it are also more likely to contain that feature. The opposite is true if a given point does not contain the feature of interest — adjacent points are also less likely to contain the feature.

Thus, one ongoing thread of improvement to the IFA involves exploring ways to have the Mass of node N be dependent on its characteristics, but also on those of its surrounding nodes. Establishing a mechanism for capturing these relationships also allows the the possibility of collapsing multiple information-containing nodes into a single node in the planning graph that represents the

entire patch. This will make planning more efficient by reducing the number of nodes and edges that need to be considered in the planning graph. It would also make look-ahead planning more effective by allowing the IFA to look ahead based on entire patches rather than just individual nodes.

The simulations presented previously (see Ch. 4) demonstrate the feasibility of planning based on patches, as that is essentially what they were doing. In this case, that was a decision driven by the need to be able to calculate optimal paths through the simulation graphs in a reasonable amount of time. However, if the number of nodes in the planning graph can be reduced substantially enough such that the number of look ahead steps can approach the number of information containing nodes, then the solutions the IFA will be able to generate will be closer to the optimal solutions.

5.2.3 Excess Time

If T_D is significantly more than the time it takes to harvest all the information in an environment (see Figures 4.25d, 4.27d, and 4.29d), there is an open question as to what the robot should do with the extra time it has. One possibility is to simply continue executing the IFA as before. When a robot has gathered all the information there is to gather, every node N in the map has $M(N) = 0$. This is because the only nodes that are given a non-zero mass are the ones where $I(N) > 0$ (see lines 19-21 of Algorithm 3.1). Thus, every node in the graph has equal mass. In the case of the simulations presented here, the heuristic used to choose among nodes with equal mass is to choose the closest node. If there are multiple closest nodes, it chooses randomly. There is nothing the prevents some other heuristic from being used. The obvious alternative is to choose randomly from among all the tied nodes.

The main alternative is to use a domain-specific search heuristic. In a domain with diffuse information it may make sense to conduct further explorations at the fringes of previously explored patches. This could be accomplished by having the robot enter a separate control state once the IFA is finished. It could also be accomplished by having an external function that modifies $I(N)$ of the fringe nodes after all the patches have been explored and re-invoking the IFA. Other domains

might be best served by a random walk towards the goal, or by simply sitting still until either new information appears or bingo time arrives. Further research would help clarify what types of heuristics are best for particular types of domains.

5.2.4 Fully Stochastic Domains

As currently structured, the IFA requires that every node eligible to be visited have at least one deterministic path from it to the goal, T_D, meaning there must be a substantial number of deterministic arcs. However, there exist real-world domains where few or none of the arcs are deterministic, causing large segments of the domain to be avoided by the IFA.

First, it must be accepted that there is no mechanism by which a path can be found in a fully stochastic domain that will guarantee deadline satisfaction. Thus, a viable approach to path planning in this circumstance has to take into account either a minimum acceptable likelihood for satisfying the deadline, or a tolerance factor by which it is acceptable to exceed the deadline time, or some combination of the two. How exactly these two factors need to be balanced for a given mission domain will determine what mechanisms are viable for path planning.

The easiest mechanism for using the IFA in a fully stochastic domain is to define downward the definition of "deterministic." Instead of requiring $P_t(a, b) = 1.0$ for the arc between a and b to be deterministic, the requirement could change to $P_t(a, b) > p$ where p is chosen to guarantee a large percentage of nodes have deterministic paths to the goal. There are several problems that make this approach undesirable. First and foremost is the lack of understanding for the cumulative probability across a path. If the "deterministic" threshold is set to 0.9, a path where every arc has $P_t = 0.91$ will be deemed to be deterministic in exactly the same fashion as a path with one arc where $P_t = 0.90$ and the rest of its arcs having $P_t = 1$. The environment may also be structured such that lowering the threshold enough to guarantee a deterministic path from every node to the goal will lower it below what is deemed an acceptable likelihood of satisfying the deadline. Lowering the threshold too much also has the potential to render the distinction between stochastic and "deterministic"

arcs moot.

A second mechanism would identify for each node the path with this highest cumulative traversal probability. Assuming the traversal probabilities of each path are all independent, the cumulative traversal probability from node a to b and from b to c is $P_t(a,c) = P_t(a,b) \times P_t(b,c)$. The two-step path with the highest cumulative $P_t(x,y)$ will be the one that minimizes $P_t(x,y)^{-1}$, and this scales for paths with more than two steps. Since $P_t(a,c)^{-1} = (P_t(a,b) \times P_t(b,c))^{-1} = P_t(a,b)^{-1} \times P_t(b,c)^{-1}$, setting the "cost" of each arc to $P_t(x,y)^{-1}$ allows standard graph-search techniques to find the path with the highest traversal probability.

Unfortunately, this mechanism does not take into account the estimated time to traverse any of the arcs along this path. The path may be so long that there is no chance of satisfying the deadline. The ideal mechanism would be one that maximizes either the probability of satisfying the deadline, the probability of arriving within the tolerance factor of the deadline, or both. Such a mechanism will have to account for both P_t and the estimated traversal time of each arc along each path. Without having a stochastic representation of the arc traversal times, this integration is difficult to accomplish. There are methods that have promise (e.g., [24]), but these are computationally intractable for graphs that do not meet some very specific conditions. Further research is needed to determine if robotic operating environments can be represented in a manner where these types of algorithms can be applied.

If the arc traversal times can also be represented stochastically, then the possibility of using the types of methods described in Section 2.3 becomes more realistic. In particular, a graph in which all of the arc traversal costs are stochastic is close to a structure that can be represented as any of a number of Markov-based statistical models, such as a Markov chain or Markov Decision Process (see, for example, [33, 34, 56, 59, 91]). Further research is required to understand how traversal probability and stochastic arc traversal times would be integrated to generate a proper cost representation for paths.

5.3 Conclusion

In conclusion, this dissertation presents a computationally feasible method for robotic path planning that maximizes the amount of information the robot will gather, the Information Foraging Algorithm (IFA). While the solutions paths generated are necessarily sub-optimal, the typical performance of the IFA is to generate paths that gather at least 50%, and almost always at least 75%, of the amount of information that could be gathered along the optimal path. The benefit is that the IFA is computationally simple and efficient when compared with the NP-hard problem of generating optimal maximization paths through a graph. Thus, the IFA represents a feasible approach to solving what in the future is likely to be a rapidly growing area of robotic utilization.

Bibliography

[1] Ecomapper auv. [Online] Available http://www.ysisystems.com/systemsdetail.php?EcoMapper-1.

[2] Liquid robotics pac x challenge. [Online] Available: http://liquidr.com/resdown/resources/case-studies/pacx.html.

[3] Jesús M. Almendros-Jiménez, José A. Piedra, and Manuel Cantón. Ontology-based modelling of ocean satellite images. In Miltiadis D. Lytras, Patricia Ordonez De Pablos, Adrian Ziderman, Alan Roulstone, Hermann Maurer, and Jonathan B. Imber, editors, *Knowledge Management, Information Systems, E-Learning, and Sustainability Research*, volume 111 of *Communications in Computer and Information Science*, pages 8–12. Springer Berlin Heidelberg, 2010.

[4] Diego Alvarez, Juan C Alvarez, and Rafael C Gonzalez. Online motion planning using laplace potential fields. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 3347–3352. IEEE, 2003.

[5] B.W. Andrews, K.M. Passino, and T.A. Waite. Foraging theory for decision-making system design: task-type choice. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 5, pages 4740 – 4745 Vol.5, dec. 2004.

[6] Sudipto Banerjee, Alan E Gelfand, and Bradley P Carlin. *Hierarchical modeling and analysis for spatial data*. Crc Press, 2004.

[7] Igor M. Belkin, Peter C. Cornillon, and Kenneth Sherman. Fronts in large marine ecosystems. *Progress In Oceanography*, 81(1-4):223 – 236, 2009. Comparative Marine Ecosystem Structure and Function: Descriptors and Characteristics.

[8] Igor M. Belkin and John E. O'Reilly. An algorithm for oceanic front detection in chlorophyll and sst satellite imagery. *Journal of Marine Systems*, 78(3):319 – 326, 2009. Special Issue on Observational Studies of Oceanic Fronts.

[9] James G Bellingham and Kanna Rajan. Robotics in remote and hostile environments. *Science*, 318(5853):1098–1102, 2007.

[10] JG Bellingham, B Hobson, MA Godin, B Kieft, J Erikson, R McEwen, C Kecy, Y Zhang, T Hoover, and E Mellinger. A small, long-range auv with flexible speed and payload. In *Ocean Sciences Meeting, Abstract MT15A*, volume 14, 2010.

[11] John Bellingham, Yoshiaki Kuwata, and Jonathan How. *Stable Receding Horizon Trajectory Control for Complex Environments*. American Institute of Aeronautics and Astronautics, 2014/05/27 2003.

[12] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

[13] Jeffrey J. Biesiadecki, P. Chris Leger, and Mark W. Maimone. Tradeoffs between directed and autonomous driving on the mars exploration rovers. *The International Journal of Robotics Research*, 26(1):91–104, 2007.

[14] J.J. Biesiadecki and M.W. Maimone. The mars exploration rover surface mobility flight software driving ambition. In *Aerospace Conference, 2006 IEEE*, page 15 pp., 0-0 2006.

[15] Robert Bogue. Mars curiosity: sensors on the red planet. *Sensor Review*, 32(3):187–193, 2012.

[16] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environments. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 572–577 vol.1, May 1990.

[17] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *AAAI/IAAI*, pages 11–18, 1998.

[18] Wolfram Burgard, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun. Experiences with an interactive museum tour-guide robot. *Artificial intelligence*, 114(1):3–55, 1999.

[19] Jackie Chappell, Zoe P. Demery, Veronica Arriola-Rios, and Aaron Sloman. How to build an information gathering and processing system: Lessons from naturally and artificially intelligent systems. *Behavioural Processes*, 89(2):179 – 186, 2012. ¡ce:title¿Comparative cognition: Function and mechanism in lab and field.¡/ce:title¿ ¡ce:subtitle¿A tribute to the contributions of Alex Kacelnik¡/ce:subtitle¿.

[20] E.L. Charnov. Optimal foraging, the marginal value theorem. *Theoretical population biology*, 9(2):129–136, 1976.

[21] A. Chaudhry, K. Misovec, and R. D'Andrea. Low observability path planning for an unmanned air vehicle using mixed integer linear programming. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 4, pages 3823 – 3829 Vol.4, dec. 2004.

[22] Jin-Liang Chen, Jing-Sin Liu, and Wan-Chi Lee. A recursive algorithm of obstacles clustering for reducing complexity of collision detection in 2d environment. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 3795 – 3800 vol.4, 2001.

[23] H. Choset, I. Konukseven, and J. Burdick. Mobile robot navigation: issues in implementating the generalized voronoi graph in the plane. In *Multisensor Fusion and Integration for Intelligent Systems, 1996. IEEE/SICE/RSJ International Conference on*, pages 241 –248, dec 1996.

[24] Gehan A. Corea and Vidyadhar G. Kulkarni. Shortest paths in stochastic networks with arc lengths having discrete distributions. *Networks*, 23(3):175–183, 1993.

[25] N.A. Cruz and A.C. Matos. Adaptive sampling of thermoclines with autonomous underwater vehicles. In *OCEANS 2010*, pages 1 –6, sept. 2010.

[26] J. Das, J. Harvey, F. Py, H. Vathsangam, R. Graham, K. Rajan, and G. S. Sukhatme. Multi-stage Bayesian Regression for Adaptive Sampling of Marine Phenomena. In *Workshop on Environmental Sensing, Robotics Science and Systems*, Sydney, Australia, 2012.

[27] Jnaneshwar Das, Thom Maughan, Mike McCann, Mike Godin, Tom O'Reilly, Monique Messié, Fred Bahr, Kevin Gomes, Frédéric Py, Jim Bellingham, Gaurav S. Sukhatme, and Kanna Rajan. Towards Mixed-initiative, Multi-robot Field Experiments: Design, Deployment and Lessons Learned. In *Proc. IROS*, 2011.

[28] G.N. Desouza and A.C. Kak. Vision for mobile robot navigation: a survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(2):237 –267, feb 2002.

[29] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[30] N. Fairfield and C. Urmson. Traffic light mapping and detection. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5421–5426, May 2011.

[31] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2), 2006.

[32] DB Fissel, JR Marko, and H Melling. Upward looking ice profiler sonar instruments for ice thickness and topography measurements. In *OCEANS'04. MTTS/IEEE TECHNO-OCEAN'04*, volume 3, pages 1638–1643. IEEE, 2004.

[33] A.F. Foka and P.E. Trahanias. Predictive autonomous robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 490 – 495 vol.1, 2002.

[34] Amalia Foka and Panos Trahanias. Real-time hierarchical pomdps for autonomous robot navigation. *Robotics and Autonomous Systems*, 55(7):561 – 571, 2007.

[35] H Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4):583–599, 1969.

[36] Matthias O. Franz, Bernhard Schölkopf, Hanspeter A. Mallot, and Heinrich H. Bülthoff. Learning view graphs for robot navigation. *Autonomous Robots*, 5:111–125, 1998. 10.1023/A:1008821210922.

[37] Liping Fu and Larry R Rilett. Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research Part B: Methodological*, 32(7):499–516, 1998.

[38] Marco Gori and Alberto Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(1):76–86, 1992.

[39] J. Gottlieb, R. Graham, T. Maughan, F. Py, G. Elkaim, and K. Rajan. An Experimental Momentum-based Front Detection for Autonomous Underwater Vehicles. In *IEEE International Conference on Robotics and Automation*, St. Paul, MN, 2012.

[40] R. Graham, F. Py, J. Das, D. Lucas, T. Maughan, and K. Rajan. Exploring Space-Time Tradeoffs in Autonomous Sampling for Marine Robotics. In *Intnl. Symp. on Experimental Robotics (ISER)*, Quebec City, Canada, 2012.

[41] JohnP. Grotzinger, Joy Crisp, AshwinR. Vasavada, RobertC. Anderson, CharlesJ. Baker, Robert Barry, DavidF. Blake, Pamela Conrad, KennethS. Edgett, Bobak Ferdowski, Ralf Gellert, JohnB. Gilbert, Matt Golombek, Javier Gómez-Elvira, DonaldM. Hassler, Louise Jandura, Maxim Litvak, Paul Mahaffy, Justin Maki, Michael Meyer, MichaelC. Malin, Igor Mitrofanov, JohnJ. Simmonds, David Vaniman, RichardV. Welch, and RogerC. Wiens. Mars science laboratory mission and science investigation. *Space Science Reviews*, 170(1-4):5–56, 2012.

[42] Raia Hadsell, J. Andrew Bagnell, Daniel Huber, and Martial Hebert. Space-carving kernels for accurate rough terrain estimation. *Int. J. Rob. Res.*, 29:981–996, July 2010.

[43] J.C. Haney and P.A. McGillivary. Midshelf fronts in the south atlantic bight and their influence on seabird distribution and seasonal abundance. *Bio. Oceanography*, 3:401–430, 1985.

[44] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[45] Roger Hine, Scott Willcox, Graham Hine, and Tim Richardson. The wave glider: A wave-powered autonomous marine vehicle. In *OCEANS 2009, MTS/IEEE Biloxi-Marine Technology for Our Future: Global and Local Challenges*, pages 1–6. IEEE, 2009.

[46] M.V. Jakuba, D. Steinberg, J.C. Kinsey, D.R. Yoerger, R. Camilli, O. Pizarro, and S.B. Williams. Toward automatic classification of chemical sensor data from autonomous underwater vehicles. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4722–4727, Sept 2011.

[47] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schröder, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, Oliver Pink, Christian Frese, and Christoph Stiller. Team annieway's autonomous system for the 2007 darpa urban challenge. *Journal of Field Robotics*, 25(9):615–639, 2008.

[48] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[49] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In IngemarJ. Cox and GordonT. Wilfong, editors, *Autonomous Robot Vehicles*, pages 396–404. Springer New York, 1990.

[50] Jinsuck Kim, N.M. Amato, and Sooyong Lee. An integrated mobile robot path (re)planner and localizer for personal robots. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 3789 – 3794 vol.4, 2001.

[51] James C Kinsey, Dana R Yoerger, Michael V Jakuba, Rich Camilli, Charles R Fisher, and Christopher R German. Assessing the deepwater horizon oil spill with the sentry autonomous underwater vehicle. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 261–267. IEEE, 2011.

[52] W Kirkwood, DW Caress, H Thomas, R McEwen, F Shane, R Henthorn, and P McGill. Results from mbari's integrated mapping system. In *OCEANS, 2005. Proceedings of MTS/IEEE*, pages 563–570. IEEE, 2005.

[53] WJ Kirkwood, DW Caress, H Thomas, M Sibenac, R McEwen, F Shane, R Henthorn, and P McGill. Mapping payload development for mbari's dorado-class auvs. In *OCEANS'04. MTTS/IEEE TECHNO-OCEAN'04*, volume 3, pages 1580–1585. IEEE, 2004.

[54] S. Koenig and M. Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on*, volume 1, pages 968 – 975 vol.1, 2002.

[55] Sven Koenig, Richard Goodwin, and Reid Simmons. Robot navigation with markov models: A framework for path planning and learning with limited computational resources. In Leo Dorst, Michiel van Lambalgen, and Frans Voorbraak, editors, *Reasoning with Uncertainty in Robotics*, volume 1093 of *Lecture Notes in Computer Science*, pages 322–337. Springer Berlin / Heidelberg, 1996.

[56] Sven Koenig and Reid G. Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122. MIT Press, 1998.

[57] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.

[58] T. Kubota, Y. Kuroda, Y. Kunii, and T. Yoshimitsu. Path planning for newly developed microrover. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 3710 – 3715 vol.4, 2001.

[59] S. Kumar, S. Celorrio, F. Py, D. Khemani, and K. Rajan. Optimizing Hidden Markov Models for Ocean Feature Detection. In *Proceedings 24th Intnl. Florida AI Research Society (FLAIRS) conference*, Palm Beach, FL, 2011.

[60] C. Kunz, C. Murphy, R. Camilli, H. Singh, J. Bailey, R. Eustice, M. Jakuba, K. Nakamura, C. Roman, T. Sato, R.A. Sohn, and C. Willis. Deep sea underwater robotic exploration in the ice-covered arctic ocean with auvs. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3654–3660, Sept 2008.

[61] W Hi Kwon, AM Bruckstein, and T Kailath. Stabilizing state-feedback design via the moving horizon method. *International Journal of Control*, 37(3):631–643, 1983.

[62] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[63] ZX Li and TD Bui. Robot path planning using fluid model. *Journal of Intelligent and Robotic Systems*, 21(1):29–50, 1998.

[64] Jonathan Edau Loh. Speed map for autonomous rovers over rough terrain. 2012.

[65] Jonathan Edau Loh, Gabriel Hugh Elkaim, and Renwick E Curry. Roughness map for autonomous rovers.

[66] Justin Manley and Scott Willcox. The wave glider: A persistent platform for ocean science. In *OCEANS 2010 IEEE-Sydney*, pages 1–5. IEEE, 2010.

[67] S.C. Martin, L.L. Whitcomb, D. Yoerger, and H. Singh. A mission controller for high level control of autonomous and semi-autonomous underwater vehicles. In *OCEANS 2006*, pages 1–6, Sept 2006.

[68] C. McGann, F. Py, K. Rajan, J. P. Ryan, and R. Henthorn. Adaptive Control for Autonomous Underwater Vehicles. In *AAAI*, Chicago, IL, 2008.

[69] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. A Deliberative Architecture for AUV Control. In *Intnl. Conf. on Robotics and Automation (ICRA)*, Pasadena, May 2008.

[70] M. McNaughton, C. Urmson, J.M. Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4889–4895, May 2011.

[71] Pitu B. Mirchandani. Shortest distance and reliability of probabilistic networks. *Computers & Operations Research*, 3(4):347 – 355, 1976.

[72] K. Nagatani, Y. Iwai, and Y. Tanaka. Sensor based navigation for car-like mobile robots using generalized voronoi graph. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 2, pages 1017 –1022 vol.2, 2001.

[73] Ioannis Emiris NUA, Elias Tsigaridas, and George Tzoumas NUA. Voronoi diagram of ellipses: cgal-based implementation. 2008.

[74] D.B. Olson, G.L. Hitchcock, A.J. Mariano, C.J. Ashjan, G. Peng, R.W. Nero, and G.P. Podesta. Life on the edge: marine life and fronts. *Oceanography*, 7:52–60, 1994.

[75] Theodore P. Pavlic and Kevin M. Passino. Foraging theory for autonomous vehicle speed choice. *Engineering Applications of Artificial Intelligence*, 22(3):482 – 489, 2009.

[76] Theodore P Pavlic and Kevin M Passino. Generalizing foraging theory for analysis and design. *The International Journal of Robotics Research*, 30(5):505–523, 2011.

[77] Mihail Pivtoraiko, Thomas M. Howard, Issa A.D. Nesnas, and Alonzo Kelly. Field experiments in rover navigation via model-based trajectory generation and nonholonomic motion planning in state lattices. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2008.

[78] T.K. Podder, M. Sibenac, H. Thomas, W.J. Kirkwood, and J.G. Bellingham. Reliability growth of autonomous underwater vehicle-dorado. In *OCEANS '04. MTTS/IEEE TECHNO-OCEAN '04*, volume 2, pages 856 – 862 Vol.2, November 2004.

[79] K. Rajan and F. Py. T-REX: Partitioned Inference for AUV Mission Control. In G. N. Roberts and R. Sutton, editors, *Further Advances in Unmanned Marine Vehicles*. The Institution of Engineering and Technology (IET), August 2012.

[80] K. Rajan, F. Py, and J. Berreiro. Towards Deliberative Control in Marine Robotics. In M. Seto, editor, *Autonomy in Marine Robots*. Springer Verlag, 2012. Accepted: to be published Fall 2012.

[81] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning representations by back-propagating errors*. MIT Press, Cambridge, MA, USA, 1988.

[82] Alireza Sahraei, Mohammad Manzuri, Mohammad Razvan, Masoud Tajfard, and Saman Khoshbakht. Real-time trajectory generation for mobile robots. In Roberto Basili and Maria Pazienza, editors, *AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*, volume 4733 of *Lecture Notes in Computer Science*, pages 459–470. Springer Berlin / Heidelberg, 2007.

[83] Gildardo Sánchez and Jean-Claude Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In Raymond Jarvis and Alexander Zelinsky, editors, *Robotics Research*, volume 6 of *Springer Tracts in Advanced Robotics*, pages 403–417. Springer Berlin / Heidelberg, 2003.

[84] T. Schouwenaars, J. How, and E. Feron. Receding horizon path planning with implicit safety guarantees. In *American Control Conference, 2004. Proceedings of the 2004*, volume 6, pages 5576 –5581 vol.6, 30 2004-july 2 2004.

[85] Tom Schouwenaars, Éric Féron, and Jonathan How. Safe receding horizon path planning for autonomous vehicles. In *PROCEEDINGS OF THE ANNUAL ALLERTON CONFERENCE ON COMMUNICATION CONTROL AND COMPUTING*, volume 40, pages 295–304. The University; 1998, 2002.

[86] Young-Woo Seo, Chris Urmson, and David Wettergreen. Exploiting publicly available cartographic resources for aerial image analysis. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 109–118, New York, NY, USA, 2012. ACM.

[87] Pierre Sermanet, Raia Hadsell, Marco Scoffier, Matt Grimes, Jan Ben, Ayse Erkan, Chris Crudele, Urs Miller, and Yann LeCun. A multirange architecture for collision-free off-road robot navigation. *Journal of Field Robotics*, 26(1):52–87, 2009.

[88] M. Sibenac, W.J. Kirkwood, R. McEwen, F. Shane, R. Henthorn, D. Gashler, and H. Thomas. Modular auv for routine deep water science operations. In *OCEANS '02 MTS/IEEE*, volume 1, pages 167 – 172 vol.1, October 2002.

[89] R. Simmons, E. Krotkov, L. Chrisman, F. Cozman, R. Goodwin, M. Hebert, L. Katragadda, S. Koenig, G. Krishnaswamy, Y. Shinoda, W. Whittaker, and P. Klarer. Experience with rover navigation for lunar-like terrains. In *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*, volume 1, pages 441 –446 vol.1, aug 1995.

[90] Reid Simmons, Richard Goodwin, Karen Zita Haigh, Sven Koenig, and Joseph O'Sullivan. A layered architecture for office delivery robots. In *Proceedings of the first international conference on Autonomous agents*, AGENTS '97, pages 245–252, New York, NY, USA, 1997. ACM.

[91] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *In Proceedings of IJCAI-95*, pages 1080–1087. IJCAI, Inc, 1995.

[92] Ryan N. Smith, Mac Schwager, Stephen L. Smith, Burton H. Jones, Daniela Rus, and Gaurav S. Sukhatme. Persistent ocean monitoring with underwater gliders: Adapting sampling resolution. *Journal of Field Robotics*, 28(5):714–741, 2011.

[93] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[94] A. Stentz et al. The focussed dˆ* algorithm for real-time replanning. In *International Joint Conference on Artificial Intelligence*, volume 14, pages 1652–1659. LAWRENCE ERLBAUM ASSOCIATES LTD, 1995.

[95] A. Stentz and M. Hebert. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots*, 2(2):127–145, 1995.

[96] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles Rosenberg, Nicholas Roy, Jamieson Schulte, et al. Minerva: A second-generation museum tour-guide robot. In *Robotics and automation, 1999. Proceedings. 1999 IEEE international conference on*, volume 3. IEEE, 1999.

[97] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.

[98] Chris Urmson, J Andrew Bagnell, Christopher R Baker, Martial Hebert, Alonzo Kelly, Raj Rajkumar, Paul E Rybski, Sebastian Scherer, Reid Simmons, Sanjiv Singh, et al. Tartan racing: A multi-modal approach to the darpa urban challenge. 2007.

[99] Christopher Urmson. *Navigation Regimes for Off-Road Autonomy*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2005.

[100] CaoAn Wang and Francis Chin. Finding the constrained delaunay triangulation and constrained voronoi diagram of a simple polygon in linear-time. In Paul Spirakis, editor, *Algorithms — ESA '95*, volume 979 of *Lecture Notes in Computer Science*, pages 280–294. Springer Berlin Heidelberg, 1995.

[101] Yunfeng Wang and Gregory S Chirikjian. A new potential field method for robot path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 977–982. IEEE, 2000.

[102] Ji wung Choi, R. Curry, and G. Elkaim. Path planning based on bezier curve for autonomous ground vehicles. In *World Congress on Engineering and Computer Science 2008, WCECS '08. Advances in Electrical and Electronics Engineering - IAENG Special Edition of the*, pages 158 –166, oct. 2008.

[103] Ji wung Choi, R.E. Curry, and G.H. Elkaim. Curvature-continuous trajectory generation with corridor constraint for autonomous ground vehicles. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 7166 –7171, dec. 2010.

[104] Ji wung Choi, Renwick E. Curry, and Gabriel Hugh Elkaim. Collision free real-time motion planning for omnidirectional vehicles, 2009.

[105] DR Yoerger, AM Bradley, SC Martin, and LL Whitcomb. The sentry autonomous underwater vehicle: field trial results and future capabilities. In *AGU Fall Meeting Abstracts*, volume 1, page 1674, 2006.