2018-06-01

Probabilistic Programming for Theory of Mind for Autonomous Decision Making

Iris Rubi Seaman
Brigham Young University

Probabilistic Programming for Theory of Mind for Autonomous Decision Making

Iris Rubi Seaman

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

David Wingate, Chair
Christophe Giraud-Carrier
Ryan Farrell

Department of Computer Science

Brigham Young University

ABSTRACT

Probabilistic Programming for Theory of Mind for
Autonomous Decision Making

Iris Rubi Seaman
Department of Computer Science, BYU
Master of Science

As autonomous agents (such as unmanned aerial vehicles, or UAVs) become more ubiquitous, they are being used for increasingly complex tasks. Eventually, they will have to reason about the mental state of other agents, including those agents' beliefs, desires and goals – so-called *Theory of Mind* – and make decisions based on that reasoning. We describe increasingly complex theory of mind models of a UAV pursuing an intruder, and show that (1) there is a natural Bayesian formulation to reasoning about the uncertainty inherent in our estimate of another agent's mental state, and that (2) probabilistic programming is a natural way to describe models that involve one agent reasoning about another agent, where the target agent uses complex primitives such as path planners and saliency maps to make decisions. We propose a nested self-normalized importance sampling inference algorithm for probabilistic programs, and show that it can be used with planning-as-inference to simultaneously reason about other agents' plans and craft counter-plans. We demonstrate that more complex models lead to improved performance, and that nested modeling manifests a wide variety of rational agent behavior.

Keywords: probabilistic programming, autonomous decision making, Theory of Mind

ACKNOWLEDGMENTS

I'd like to formally give my full gratitude to my advisor David Wingate, for his role in shaping my future. He not only provided me with an opportunity to learn, but a place to thrive. I'm grateful for the late nights he would sacrifice to work on papers with me, and for the personal interest in my dreams. I believe that everything happens for a reason, and I truly believe that David came to BYU was so he could change my life.

I'd like to thank my husband Eric Seaman for his selflessness and support in my pursuits and dreams. I'm grateful for everything he is to me. He never makes me feel that my dreams are too ambitious or too great.

I'd like to thank my mom, Vilma Martinez, for her example of a strong and independent woman. She inspired me to also believe that I could accomplish anything I desired. She always told me that "¡Querer es poder!"

I'd like to thank Vikash K. Mansinghka for the opportunity he provided for me at MIT, and the personal mentorship from Marco Cusumano-Towner. Never have I grown so much and quickly than at my time there. Only after coming back from MIT did I feel that I could do a lot of my research independently.

I'd like to thank Dennis Ng, for being the open door into graduate school at BYU. He motivated me to keep working on my application.

I'd like to thank Christophe Giraud-Carrier for being the first professor to expose me to research as an undergraduate. He was kind and always supportive of what I wanted to do. He always created an atmosphere that made me feel completely comfortable to express anything I had in mind.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

As autonomous agents, such as unmanned aerial vehicles (UAVs), and autonomous driving vehicles, become more ubiquitous, they are being used for increasingly complex tasks. Researchers and consumers aspire to more than just agents that perceive and respond to their environments, and instead desire agents who use their observations to reason and execute actions that lead to further reasoning to complete a goal. *Theory of Mind* takes this concept even further by describing an agent's ability to model the beliefs, intents, and desires of other intentional agents. Humans accomplish such reasoning every day; whether it's responding to a question, a facial expression, or gesture, we observe, reason, and then act on that reasoning. For this reason, this ability is indispensable if we hope to one day create agents capable of empathy, "reading between the lines," and interacting with humans as peers.

However, an agent that can model the beliefs, intents, and desires of other agents is more easily described than implemented. Since the mental state of another agent can never be known perfectly, agents require the ability to explicitly reason about a distribution over beliefs, goal, limitations, and plans of another agent in order to make decisions. Only then can agent make rational decisions under uncertainty to complete goals and integrate into society more easily.

1.2 Pursuit and Interception Problem

We scope this thesis into a ***pursuit and interception problem***, where we aim to simulate agents that can apply Theory of Mind to reason and make decisions to accomplish goals such as: (1) pursuing and intercepting another agent or (2) reaching a goal undetected by another agent.

Although there are many scenarios where this problem manifests itself, here we describe a few possible applications.

Consider the following scenario: An intruder (the **runner**) is now on the run in the city and is actively trying to avoid detection while reaching his safe base at some unknown location in the city. A UAV (the **chaser**) must locate and intercept this runner before the runner successfully escapes.

There are many different ways a chaser could approach this problem of detecting the runner. A naive chaser could employ a simple search pattern, scanning the city in, for example, a grid-like path. But even a moderately sophisticated runner would try to construct an escape route that would avoid detection from a grid-search.

A more sophisticated chaser could leverage knowledge that not all escape routes are equally likely – perhaps the runner would avoid large open spaces, or stick to small alleyways – or perhaps the chaser could leverage knowledge of likely locations of a getaway vehicle.

An even more sophisticated chaser, that applies Theory of Mind, would reason about the runner reasoning about the chaser, and plan accordingly – even if the runner might also be reasoning about the fact that the chaser is reasoning about him. Although each of these cases are suitable options for the chaser, one might outshine the others.

Consider the alternative scenario where an autonomous car is responsible for driving an innocent civilian to a safe base without being detected by a *chaser* where detection may result in exposing the civilian to danger. A naive autonomous car could simply plan a path to its goal and drive there; however, the probability of detection would be relatively high compared to a more sophisticated autonomous driving car. A more sophisticated car could

plan using its inferred beliefs of the chaser's location and plan, and design its own plan which will increase the probability of keeping the civilian safe on its way to its safe location.

Overall, applications for this *pursuing and interception problem* include a variety of different scenarios that are not only limited to (i) security enforcement with the use of UAVs to pursue and intercept intruders trespassing on private properties and (ii) autonomous driving cars pursuing vehicles fleeing from a crime scene, but also (iii) military forces inferring invading target locations based on a soldier's movements, and lastly, (iv) inferring safe escape routes from opposing military forces in pursuit.

In this thesis we aim to show that by comparing different levels of complexity in chaser models, we can get better detection rates by increasing complexity. Specifically, we show that the better detection rates are produced by the most complex models that simulate Theory of Mind.

1.3 Bayesian Model

We contribute a case study involving two agents, a *chaser* and a *runner*. We begin with the basic idea that the chaser seeks to intercept the runner and the runner seeks to reach his goal without detection. However, the runner's intended start location, goal location, and likely path to the goal are initially unknown to the chaser. Additionally, the runner does not know the true location of the chaser; instead, the runner maintains beliefs about the chaser's likely location, and plans around those beliefs. Both agents reason about each other, and about how they reason about reasoning.

We show in Figure 1.1 a simple Bayesian model of the chaser-runner problem. We show that in order for the chaser to intercept the runner's plan, the chaser must be able to have a representation of the map which it can use to infer probable plans and goals of the runner. In an ideal case, an agent would perceive pixels, transform the data into 3D points, and use those points to generate a map for navigation and planning. However for this thesis, we assume those steps have already been taken and a map has been provided for the agents.

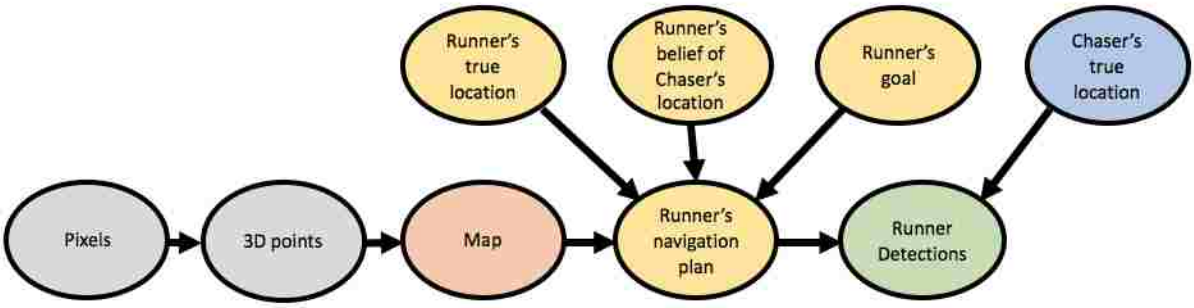

Figure 1.1: Bayesian model of the chaser-runner problem. A map is generated into polygons from pixels. The runner's plan is determined by the runner's location, his belief of the chaser's location, and his goal. Runner detections are based on the runner's navigation plan and the agent's true location.

We show, further that in order to learn the runner's navigation plan, we must use priors to describe the runner's location and goal. Once we estimate the runner's navigation plan, we can use the chaser's current location to help detect the runner.

1.4 Our Contribution

In this thesis, our goal is to take a step towards a framework that could actually be deployed on a real UAV. While we have remained in a 2D setting, we have built models with a variety of realistic components, including path planners, trajectory optimizers and visibility graphs.. We demonstrate that *probabilistic programming* is a natural tool that addresses two key issues with this: first, because probabilistic programming mixes arbitrarily complex deterministic code with random primitives, it is able to easily model both the complex robotics primitives, but also elements of uncertainty; and second, probabilistic programming is flexible enough to allow *nested inference*, which we use to model agents reasoning about other agents.

We also illustrate how our models exhibit a wide variety of rational behavior with a single, unified inference algorithm based on self-normalized importance sampling. Our experiments show that better models matter, and that the complexity of Theory of Mind is justified by improved decision making.

Chapter 2

Related Work

In this chapter, we discuss related works on Theory of Mind while attempting to clarify what the theory behind Theory of Mind is. To demonstrate by example, we discuss an example of a simple experiment of how cognitive scientists determine whether children demonstrate Theory of Mind, and what that entails for the child. We discuss work done to simulate Theory of Mind in agents, and how it requires nested inference to reason about reasoning. In addition we discuss background on planning as inference and autonomous decision making.

2.1 Theory of Mind

Theory of Mind describes an agent's ability to model the intents, beliefs or desires of others. Frith and Frith [12] explain that representing Theory of Mind is difficult because the agent's mental state is hidden and can only be approximated using observations. The process trying to represent the mental state of an agent produces high levels of uncertainty, since identical mental states may lead to different actions and identical observations may represent different mental states. Consider Baron-Cohen et al. [5]'s Theory of Mind experiment in which a child is shown two dolls (referred to as Sally and Anne) that are placed in a room holding a basket and a box respectively (see original cartoon of experiment in **??**). In this experiment, it is then observed that Sally "places" a marble into her basket and then leaves the room for a walk. While Sally is outside the room, Anne removes the marble from the basket and places

it into the box. The child is then told that Sally desires to play with the marble and is asked, "Where will Sally look for the marble?".

Although through full observation, it is obvious that the marble is now in the box, the child must be able to model the mental state of Sally to successfully pass this test. Since Sally was not present and was not told that the marble was removed from the basket and placed in the box, the child would have to demonstrate that they are aware that the other person, in this case Sally, has a different mental state from their own and from the "true" state of the world. For this case, they should determine that Sally will most likely look for the marble where she had last placed it, in the basket. Therefore, if the child responds that Sally will look in the basket, the child passes the test and has successfully modeled the mental state of Sally. If the child says that Sally will look in the box, then the child fails the test and is incapable of modeling the mental state of others. Through experiments such as these and a series of other experiments, it has been shown that human children typically demonstrate Theory of Mind during their early years of development, generally between the ages of three and six [9, 31].

2.2 Models of Theory of Mind

Fully-developed Theory of Mind, of necessity, requires the possibility of nested beliefs. Zettlemoyer et al. [35] address multi-agent filtering in environments with many agents and infinitely nested beliefs. Baker et al. [3] present a computational framework based on action understanding as inverse planning, while Koller et al. [15] present an inference algorithm for stochastic programs which extends easily to recursion. More recently, Stuhlmüller and Goodman [28] emphasized the value of probabilistic programs in modeling the reasoning of other agents.

Baker and Tenenbaum [2] illustrate a scenario where they successfully model the beliefs, intentions, and desires of an agent while traveling on a campus map looking for a particular food truck. This is attempted with an implementation of a theory-based Bayesian

(TBB) framework, which models the structured knowledge of Theory of Mind. Although they are able to model Theory of Mind through time with new and past observations, this implementation does not demonstrate coordination with other agents or autonomous decision making.

Frith argues that Theory of Mind can be approximately represented using probabilistic programming and demonstrates basic examples of nested conditioning with the probabilistic programming language, Church. Baker and Tenenbaum [2] expand this idea by proposing the framework Bayesian Theory of Mind, in which traditional methods of cognitive reasoning are applied to Bayesian models. Basic mental states are represented using partially observable Markov decision processes (POMDPs) to minimize loss and maximize rewards [14]; this allows reasoning through observations, assuming that the agent is not acting fully rational.

2.3 Agents as Probabilistic Programs

The flexibility of modeling agents with probabilistic programs is infinite, which is a benefit to the designer of such models. As with design, approaches vary. One recognized approach of modeling agents with probabilistic programs is by the implementation of agents that compute rational policies. This allows for the agent to be modeled in a very structured design that uses environments, sets of actions, transition functions, and utility functions. The decision rule is to take an action that maximizes utility. Usually, these agents are designed to take a state as input and return an action. This approach can be expanded to more complicated states, actions, and so forth. However, there is an alternative to computing the optimal action for any problem, and we do this by treating the act of choosing an action as an inference problem.

One simple way of switching from a policy to an inference approach is to instead sample random actions from a uniform distribution and condition on the preferred outcome. We can then infer from our observed consequences what action caused our preferred outcome. This is known as *planning as inference* [8], and is the approach we implement due to the

complexity of the chaser-runner problem and the uncertainty in decision making for our agents.

2.3.1 Collaborative Work

The development of Theory of Mind in machines leads naturally to interaction with their human counterparts. Awais and Henrich [1] and Fern et al. [11], and Nguyen et al. [21] investigate collaborative projects between humans and robots in which the robot must determine the human's (unobservable) goal. In a complementary line of research, Sadigh et al. [26] explore the idea of active information, in which the agent's own behaviors become a tool for identifying a human's internal state.

An example of a coordination game by Schelling [27] is discussed in Stuhlmüller and Goodman [28] centered on *reasoning about reasoning* through nested conditional inference. Their example introduces a scenario of two agents, Alice and Bob, who desire to meet at some location. These agents share common knowledge that there are two possible meeting locations. However, one location is slightly more popular than the other. Stuhlmüller and Goodman [28] formalize this game as a conditional distribution where one agent conditions its location on its partner choosing the same location. Since each of their agents begin with a bias of the other agent choosing the more popular place with higher probability, each are more likely to go the more popular place. This is specifically demonstrated as an instance of *planning as inference*, where they transform the problem from maximizing the expected utility to maximizing the likelihood. In their experiments, the proposed approach demonstrates convergence of the most popular meeting location.

This particular experiment is designed for the agent to make a single final decision of where it must go to increase its chances of meeting the other agent at the same location. However, this example lacks experiments needed to demonstrate how nested conditioning can apply to more convoluted and realistic models. An example of a more complicated model

would be one where agents constantly need to make new decisions conditioned on commonly known prior knowledge and a collection of past and new observations.

We demonstrate in this thesis that our agents solve the same problem as the maximizing utility agent, but do so by planning as inference. We create probabilistic programs and use inference algorithms to sample from the posterior distribution, and then use those outcomes to help the agents make rational decisions to complete their goal. With this implementation, agents are given the ability to make decisions using statistics as opposed to following pre-computed policies. In addition, we note that we unaware of previous work that has been done in designing models that include path planning and field of view (for detection) calculations, which ultimately makes approaching this problem difficult and our methodology novel.

Chapter 3

Primitives for the Chaser-Runner Model

The setup of the chaser-runner problem requires different components in order to simulate. We require a model that can "realistically" describe agents in this scenario before we can introduce any probabilistic programming, or inference. We require an environment that allows agents to accomplish their respective goals by having them plan paths and having some method of simulating sight and detection. Therefore, we now describe different primitive components that are combined together to construct our final model.

3.1 Environment

To search for and intercept the runner, the chaser requires a representation of the world with which it can reason about starting locations, goals, plans, movement and visibility. Our experiment is a stylized model designed around a known, fixed map of the city of Bremen, Germany [7]. This map was generated by flattening 3D point cloud data and converting into polygons (shown in Figure 3.1). This polygonal map enables simple, efficient calculations for both the chaser and runner. As explained previously, our work assumes the map is given to the agent in polygon form.

3.2 Starting Location and Destination Goal

In order to learn the runner's true navigation plan, the chaser begins with a set of priors describing the runner's possible location and goal. In this work, we use a discrete set of candidate start and goal locations. Figure 3.2 (left) shows the flattened polygon map of

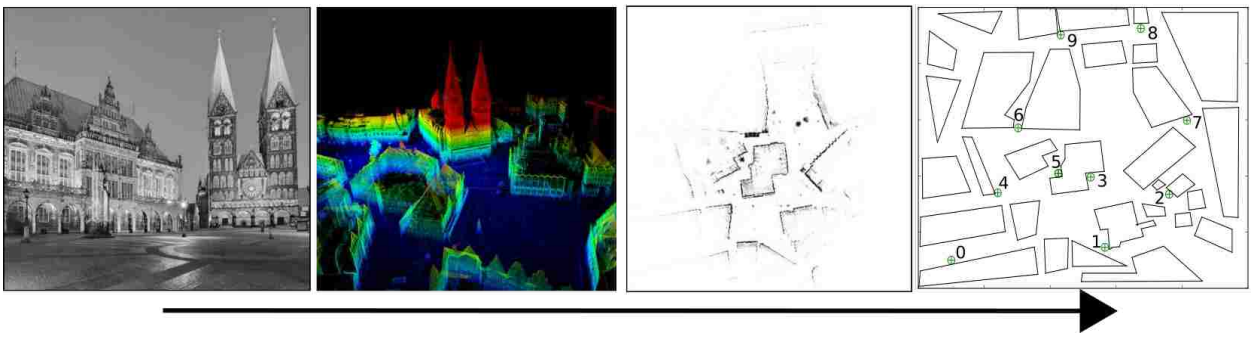

Figure 3.1: Our world is described using the city of Bremen, Germany. We generate a coarse polygon representation from point cloud data.

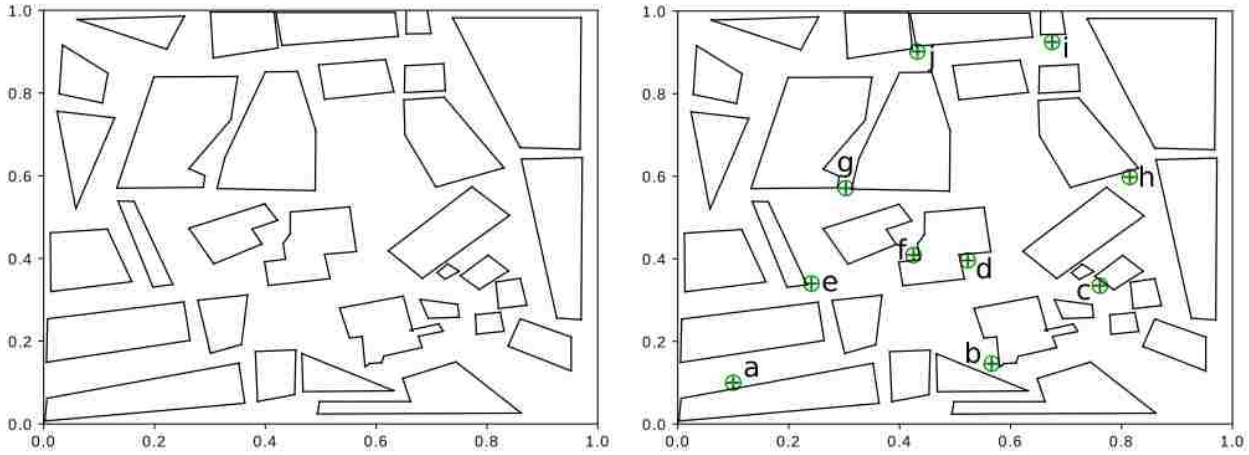

Figure 3.2: Left: the Bremen-Map layout with 31 obstacles of various shapes and sizes. Right: Marked in green circle-crosses, we show possible start and goal locations for the Bremen-Map.

the city of Bremen, Germany, and on the right we show the discrete locations on the map. These locations were chosen randomly in a uniform fashion across the map. A postulated navigation plan is estimated based on these priors, and the chaser plans a path to a likely goal location for the runner, with the intent of detecting the runner before he arrives at the goal; over time, we expect the chaser's posterior distribution to more accurately reflect the runner's whereabouts and objectives.

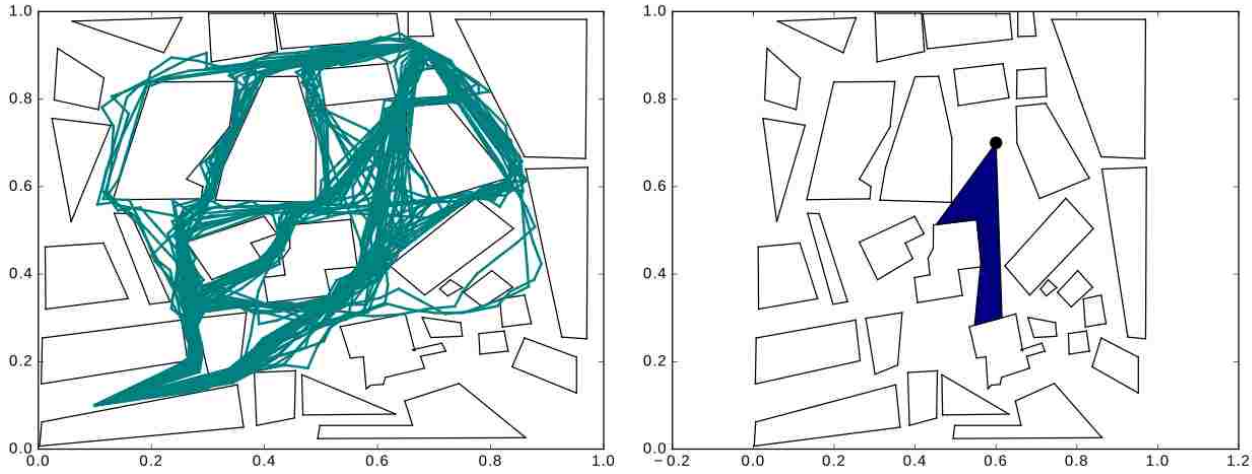

Figure 3.3: On the left: Visual distribution of paths a runner may take modeled with RRT. On the right: a 45° isovist, or range of sight, of the chaser. The isovist is properly blocked by buildings.

3.3 Path Planning

As a means to describe an agent planning to a goal, we needed a path planner that could explain all the random choices needed to make a plan which we could embed into our models. Also, we needed to consider the fact that this path planner needed to be computationally fast for inference, and needed to describe movement along the path in a uniform manner. Therefore, we next describe how we designed the path planner for our models.

3.3.1 Rapidly-Expanding Random Tree

We model paths using a Rapidly Exploring Random Tree (RRT) [16], a randomized path planning algorithm designed to handle nonholonomic constraints and high degrees of freedom. We leverage the random nature of the RRT to describe an entire distribution over possible paths: each generated RRT path can be viewed as a sample from the distribution of possible paths taken by a runner (see Figure 3.3, left). RRTs naturally consider short paths as well as long paths to the goal location. To foreshadow a bit, note that because we will be performing inference over RRTs conditioned on not being detected, the runner will naturally tend to use

paths that minimize the chance of detection, which are often, but not always, the shortest and most direct.

3.3.2 Trajectory Optimization

Our RRTs are further refined with a trajectory optimizer that eliminates unrealistic bumps and wiggles. Note that we will often condition this distribution over (optimized) paths on various criteria, resulting in a simple planning-as-inference [30] mechanism. To keep the randomness in path planning for our models, we do not fully "optimize" the paths produced by the RRT. This creates paths that have some variation while still maintaining the integrity of a random path generated. We refer to these paths as *semi-optmizined* in the Goal Inference section of this thesis, and later show what kinds of paths are produced by this optmizer. In addition, we discuss in that section our motion model, i.e. how we "walk" the agent along the path.

3.4 Visibility and Detection

Detection of the runner by the chaser is accomplished with the use of an isovist, a polygon representation of the chaser's current range of sight [6, 20]. Given a map, chaser location, and runner location, the isovist allows us to determine the likelihood that the runner was detected. Although an isovist usually uses a 360 degree view to describe all possible points of sight to the chaser, we bound the range of sight with a prespecified degree of vision of 45 degrees, and add direction to the chaser's sight as seen in Figure 3.3 (right). We note that isovist in general capture the full range of sight without taking into account distance. Our models take this into account by limiting the range, or distance in which an isovist is used. This is done to make the models more realistic and more computationally efficient.

Chapter 4

Probabilistic Programming for Theory of Mind

Embedding Theory of Mind into our Bayesian model allows the agent to reason about reasoning. The runner naturally would most likely choose paths to his goal location that minimize detection from the chaser. Therefore, in order for the chaser to make an educated guess about the runner's most likely intended path, the chaser must perform inference over paths the runner is likely to take. However, the runner's choice of paths is influenced by the runner's own belief of the chaser's location. Since the chaser is reasoning about the runner's intended path while the runner is reasoning about the chaser's location to choose a path, we represent our model using nested conditional inference which we will discuss in section 4.2.

4.1 Probabilistic Programs

RRTs, nested inference and isovists all involve complex, deterministic calculations and data structures mixed with random primitives. To both represent our generative model cleanly and to perform inference in it, we turn to the tools of probabilistic programming.

Probabilistic programming is a recent generalization of graphical models that blends *Bayesian probability* with *computer science*: modelers specify a stochastic process using syntax that resembles a modern programming language, and allows them to define distributions using recursion, libraries, or data structures. Example languages include Venture [17], Anglican [29], and Church [13].

Importantly for our purposes, these languages allow programmers to freely mix deterministic and stochastic elements, resulting in tremendous modeling flexibility. In

a probabilistic programming framework, it is relatively easy to (for example) describe distributions over RRTs and isovists, or even distributions that involve optimization problems as a subcomponent of the distribution.

4.1.1 Python Probabilistic Programming Language

For the experiments outlined in this paper, we implemented a simple, custom probabilistic programming framework in Python based on the lightweight transformational compilation technique [33]. This language supports basic Markov chain Monte Carlo (MCMC), black-box variational inference [25, 32], and self-normalized importance sampling; because the host language is Python, it is able to integrate seamlessly with any extant Python library. For our models, we rely only on numpy and other standard python packages. Although there are several existing probabilistic programming languages available for the public, we chose to continue the use of python since our primitives such as the RRT planner and isovist calculations are more efficiently written in this language. [1] The challenge is inference, which we now address.

4.2 Inference in Probabilistic Programming

Inference can be viewed as reasoning about the posterior distribution over execution traces conditioned on a particular program output, and is difficult because of the flexibility probabilistic programming languages present: in principle, an inference algorithm must behave reasonably for any program a user wishes to write. Sample-based MCMC algorithms are the state-of-the-art method, due to their simplicity, universality, and compositionality [13, 18, 23], although notable other approaches include variational message passing [19] and sequential Monte-Carlo [34].

Recall that a probabilistic program is an executable, generative function that, when run unconditionally, yields a sample from a prior distribution. Running probabilistic programs

[1] We note that Pyro, a probabilistic programming language written in python and supported by PyTorch was released late last year in 2017, but we chose to remain using the tools we already had been working with.

Algorithm 1 do_random_choice(choice_class, params)

```
1: name = params.get_name()
2: if cond_data_db.has_key(name) then
3:     new_val = cond_data_db[name]
4: else
5:     new_val = choice_class.sample(params)
6: end if
7: likelihood = choice_class.score(new_val, params)
8: cur_trace_score+ = likelihood
9: trace[name] = new_val
10: return new_val
```

forward can be quite fast, and is limited only by the native speed of the interpreter for the language.

Normally, we run MCMC algorithms such as Metropolis-Hastings on probabilistic programs by treating a forward run of the generative probabilistic program as a random sample. With Metropolis-Hastings, we sample K proposed traces sequentially and with each new sample, we either accept the proposal or reject it using an acceptance probability:

$$x' \sim p(x'|x),\ \alpha = \frac{P(x')p(x|x')}{P(x)p(x'|x)} \tag{4.1}$$

where x is a currently accepted state, or trace sampled from the probabilistic program, x' is the proposed sampled state, and $\frac{P(x')}{P(x)}$ is the probability ratio between the previously accepted state and the proposed state. We accept the proposed state if:

$$r = \min(1, \alpha),\ u \sim U(0,1),\ x_{new} = \left\{ \frac{x'\ if\ u < r}{x\ if\ \geq r} \right\} \tag{4.2}$$

The score of a trace is computed using the joint probability distribution of the generative probabilistic program as shown in Algorithm 1. As each random choice in the probabilistic program is performed, i.e. **do_random_choice(.)** is called, the trace score is updated.

However, since reasoning about execution traces can be computationally intensive [33], and since nested inference is particularly difficult to code well using MCMC methods, we use

self-normalized importance sampling, which has several attractive practical properties in this application including that it has the advantage computational efficiency. Most importantly, self-normalized importance sampling helps us solve the nested inference problem.

4.2.1 Self-Normalized Importance Sampling

Here, we briefly describe the theory of self-normalized importance sampling (see [22], Sec. 9.2; this implementation was also inspired by [10]); in the next section, we describe how it can be implemented as a probabilistic programming inference algorithm. Self-normalized importance sampling is a generic technique useful when either the nominal or importance distribution can only be computed up to a normalizing constant. Given a probabilistic program describing a joint distribution $p(x, o)$ over latent variables x, such as goal locations, and observations o, such as the start location and the lack of observing other agents, our goal is to compute the conditional expectation $E_{p(x|o)}[f(x)]$ for some function $f(x)$ of the latent variables:

$$\begin{aligned} E_{p(x|o)}[f(x)] &= \int_x f(x)p(x|o) \\ &= \int_x f(x)\frac{p(o|x)}{p(o)}p(x) \\ &\approx \frac{1}{n}\sum_i f(x_i)\frac{p(o|x_i)}{p(o)}, \; x_i \sim p(x) \end{aligned} \tag{4.3}$$

This suggests the following computational procedure: draw samples from the unconditional probabilistic program $p(x)$, and weight them according to the data likelihood $p(o|x_i)/p(o)$. However, the marginal data likelihood term $p(o)$ is intractable to compute. If we were to estimate

$$p(o) = \int_x p(o|x)p(x) \approx 1/N \sum_i p(o|x_i), x_i \sim p(x) \tag{4.4}$$

and use the same set of samples in Eq. 4.4 as Eq. 4.3, a ratio estimator may be computed as:

$$E_{p(x|o)}[f(x)] = \sum_i \frac{f(x_i)p(o|x_i)}{\sum_i p(o|x_i)}, \quad x_i \sim p(x) \tag{4.5}$$

Algorithm 2 do_random_choice_snis(choice_class, params)

```
1: name = params.get_name()
2: if cond_data_db.has_key(name) then
3:     new_val = cond_data_db[name]
4:     likelihood = choice_class.score(new_val, params)
5:     cur_trace_score+ = likelihood
6: else
7:     new_val = choice_class.sample(params)
8: end if
9: trace[name] = new_val
10: return new_val
```

which is exactly the form of self-normalized importance sampling. This estimator is unbiased, with variance that decreases as 1/n; for a discussion of other technical properties, see [22].

The use of self-normalizing importance sampling as described previously leads to a particularly elegant inference algorithm for probabilistic programs: samples are drawn from the unconditional prior $p(x)$, which is accomplished simply by running the model forward, and each sampled execution trace is weighted according to how well it explains the observations $p(o|x)$.

Part of running our inference algorithm means running our model several times to collect a number of traces, as discussed previously in section 4.2. For each sample we collect, we run the probabilistic program; as it is executed, each of the random choices calls the updated function **do_random_choice_snis(.)**, shown in Algorithm 2, to get samples for each random variable in the probabilistic program. As shown in Algorithm 2, we only increment the score of a conditioned trace by the log likelihood of each conditioned random choice. A mild performance benefit of this approach is that the log density of *unconditioned* random choices never needs to be computed.

There are some initial results that suggest many nested inference scheme involving nonlinear functions is inherently biased [24]. We do not attempt to address that issue in this thesis; while more sophisticated inference algorithms are possible, and could in principle

improve results, we will see in the experimental section that even this simple inference algorithm is able to construct rich inferences about agent behavior.

Chapter 5

Goal Inference

5.1 Demonstration of Planning and Goal Inference

An important element to implementing Theory of Mind for planning, is to have the ability to perform goal inference for agents. By embedding this ability into agents, they will be able to predict future locations of other agents, thus giving them the ability to make informed decisions contingent on their own respective goals. For example, if one agent is actively trying to avoid another agent, that agent requires the ability to infer future goals of the other agent to actively plan to avoid intercepting its path on its way to its own goal location. For this purpose we now describe how simple generative models using RRT planners can be used to infer the goals of agents.

5.1.1 The Goal Inference Demonstration Environment

We begin our goal inference demonstration on the following map shown in Figure 5.1, with 6 main obstacles of various shapes and sizes. In the figure, we also show possible start and goal locations for agents in crossed green circle markers. These locations serve as a basis of where agents will generally begin their planning and where they will end, i.e. reach their goal. These locations were chosen to cover different surrounding areas of obstacles on the map. In this map, agents are allowed to travel from one of the set locations (seen in Figure 5.1) to another.

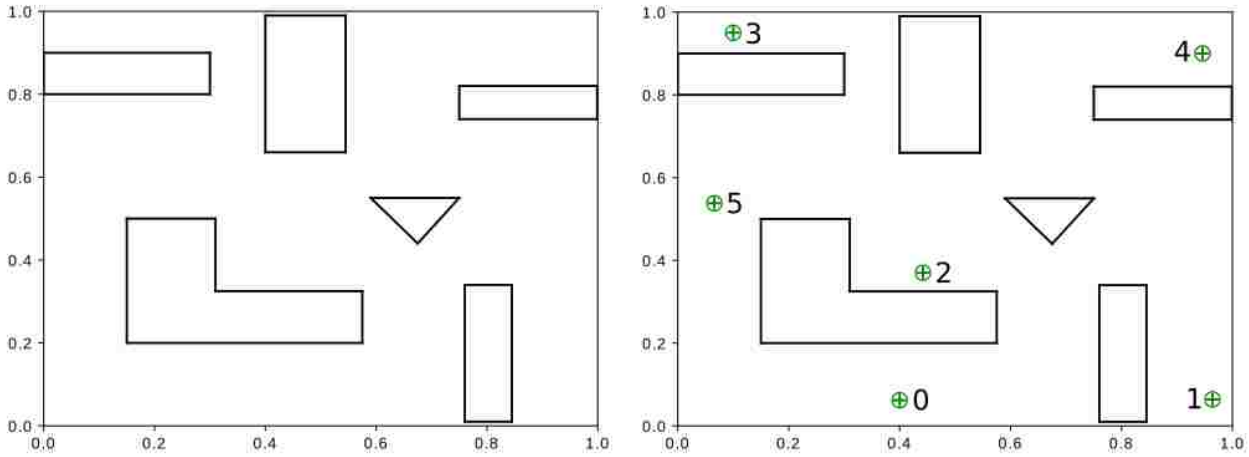

Figure 5.1: Left: Basic map layout with 6 basic obstacles of various shapes and sizes. Right: Marked in green circle-crosses, we show possible start and goal locations for this basic map.

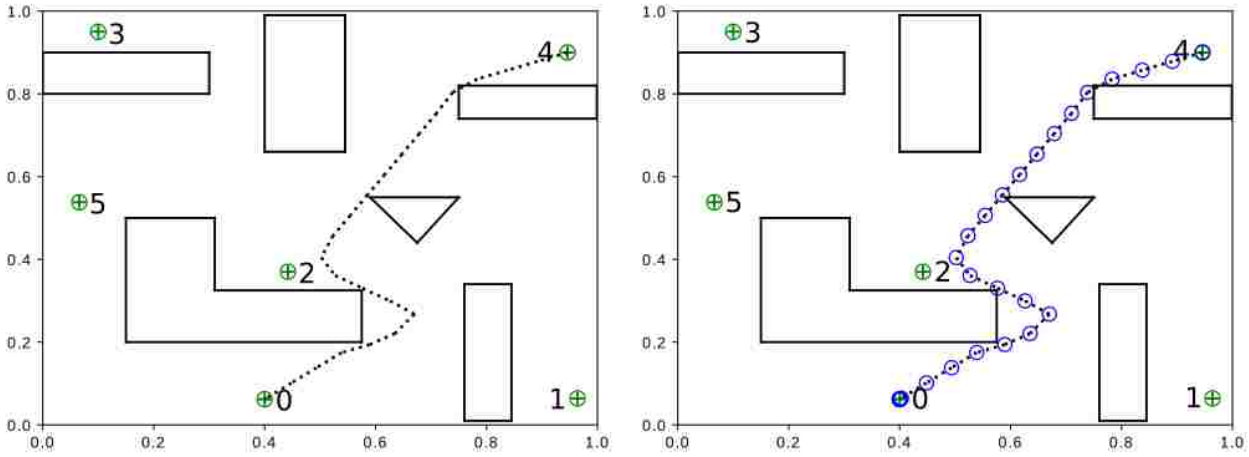

Figure 5.2: Left: We show an example of an agent's plan. Right: We show the time steps of how an agent traverses its plan in blue circles.

5.1.2 Agent Path Planning Demonstration

The path from one location to another may vary from and to each of the locations on the map shown in Figure 5.1. In Figure 5.2, we demonstrate what a path for an agent may look like. On the left side of the figure, we see the overall path of an agent represented in a dotted black path from location 4 to 0. On the right we show particular time steps, represented in blue circles. Each of these time steps are observations from the path. This means that at each of those times steps, we observe at which location along the path the agent will be. Each path has a set number of observations and we assume the agent will reach its goal location within those number of time steps. If the agent reaches its goal location before the end of the set number of time steps, it *hovers* on the goal location for the remaining time observations. Our path planner gives us the flexibility to control the speed in which the agent travels along the path. We show the same observation times in Figure 5.3 while the agent travels at different speeds from the same start and goal location. We see on the left-most plot, the agent travels slowly along the path, therefore revealing more time steps along the path compared to a faster agent, which we show on the right, where the rest of the observation times are shown on the goal location. For our experiments, we use a speed in between the two shown in Figure 5.3.

5.1.3 A Simple Planning Generative Model

Now that we have a path planner, we can move on to building a generative model which we can use to perform goal inference. The objective we want to accomplish when performing goal inference is to be able to infer which (goal) location an agent is traveling towards when we observe its past movements. To be able to make those kinds of inferences we need to begin by building a model that simulates an agent traveling from one location to another. To build such a model, we will need to make decisions on which variables in the model should be random variables. Since we will need to condition on both the start and goal locations of an agent's plan, both the start and goal locations need to be described with random variables.

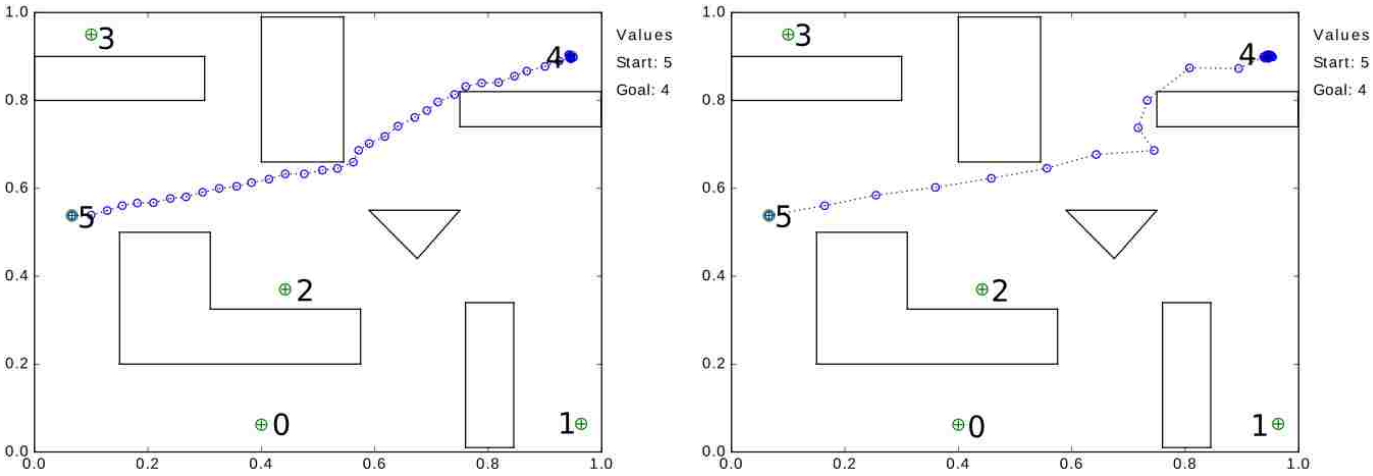

Figure 5.3: Each of the plots shown reveal a semi-optimized RRT path with observations at each blue circle along the path. However, each of the plots show different observations based on the speed in which we wish the agent to travel. On the left, we see observations were we observed the agent traveling slowly. On the right-most plot, we show the agent traveling very fast. Each of the paths contain the same number of observations, by having the agent over the goal location.

Algorithm 3 Basic Planning Model

1: Given: map
2: Variables: β = last time step
3: t $\sim$ uniform$(1, \beta)$
4: start $\sim$ Categorical(possible start locations)
5: goal $\sim$ Categorical(possible goal locations)
6: plan = optim_RRT(start, goal) + noise

In addition, we will need to condition on past observations of the agent, therefore each time step will need to be a random variable. Algorithm 3 shows an example of how we can set up a probabilistic program with this logic. Once the program has sampled the start and goal locations, we can use our path planner to get a coherent path with observed time steps. However to transform the latent variables of each time step into random variables, we must add noise to the original realizations. We do this using a normal distribution centered at the original realization at time t.

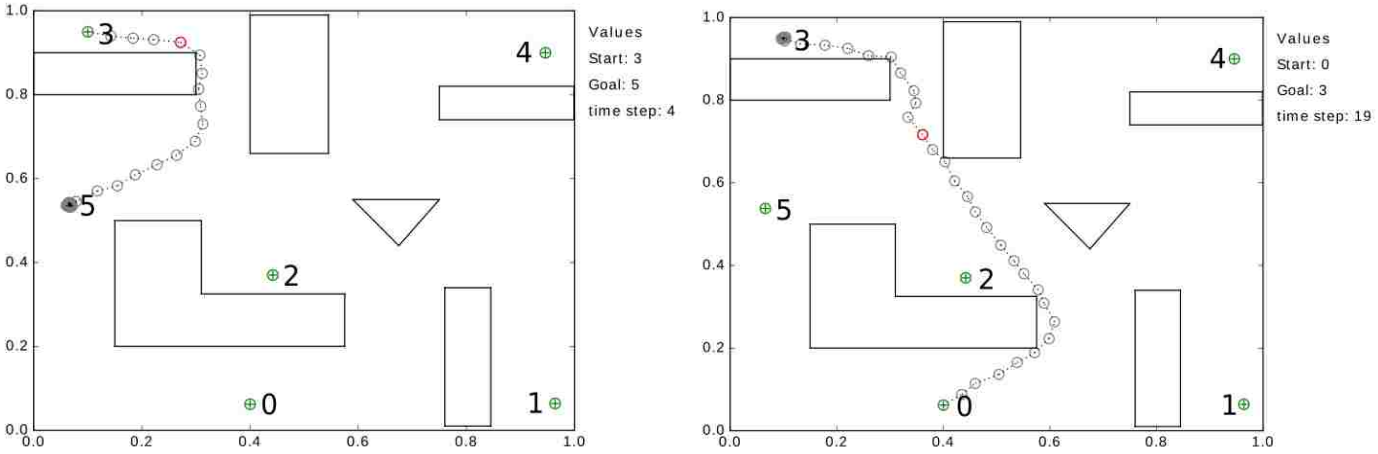

Figure 5.4: We show unconditioned forward traces of the basic planning model. The red circle on the path represents the time step i.e. the agent's location at that time step. On the left, we show a trace where the agent has planned path from locations 3 and 5, and its time step is at 4. The agent is traveling downward. On the right, we show another sampled trace from the prior where the agent is at time step 19 on a plan from location 0 to 3. The agent is traveling upward.

5.1.4 Basic Planning Model Experiments

To test if this model behaves as desired, we naturally run unconditioned forward samples using the probabilistic program. These are traces from the prior distribution. Figure 5.4 shows two samples which the basic planning model generated. On the left, we see that the time step is at 4 (depicted in red), and the agent is traveling downward the map. It starts at location 3 and plans to end at 5. On the right, the time step is at 19 (depicted in red), and the agent is traveling upward the map. It starts at location 0 and ends at 3. We expect the samples to vary in realizations for all random variables. This means that samples from the prior should produce a variety of possible instances for the scenario. We can have paths from and to any of the possible goal locations and have those paths vary as well. In addition, we can generate instances at varying time steps.

We can go further to see what kind of results we get when we condition the start, goal, and time steps. The first plot in Figure 5.5 shows 5 samples where we condition the start variable to be 2, the goal to be 4, and the time step to be 8. The second shows 25 samples

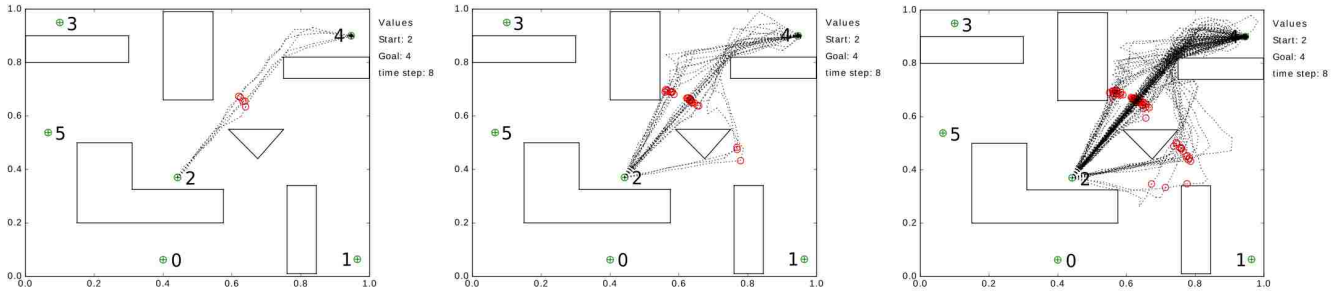

Figure 5.5: We show conditioned forward traces of the basic planning model. We condition the start, goal, and time random variables. The first figure shows 5 conditioned samples. The second shows 25, and the third shows 100 samples. We see how we get a distribution of paths when we do not optimize the path entirely.

and the third shows 100 samples with the same conditions. We see from these figures how we are able to get a nice distribution of paths from one location to another when we condition the start, goal, and time step in the model. This justifies why we do not optimize our paths fully and why refer to them as *semi-optimized*.

Now that we have seen that the generative model produces scenarios as desired, we can run inference on the model to determine goal locations based on past observations of agents. Note that the probabilistic program in Algorithm 3 has a a joint probability distribution of $p(\text{start}, \text{goal}, \text{step}_1, \text{step}_2, ...\text{step}_\beta)$ where *start* represents the discrete starting location of the agent and the *goal* is the discrete goal location. To find the probability density of the possible start and goal locations, we marginalize out all unknown future steps in the path given some observed steps as shown:

$$p(\text{goal}, \text{start}|\text{step}_1, ..., \text{step}_\text{k}) = \int_{\text{step}_{\text{k}+1}}^{\text{step}_\beta} p(\text{goal}, \text{start}, \text{step}_{\text{k}+1}, ..., \text{step}_\beta|\text{step}_1, ..., \text{step}_\text{k}) \quad (5.1)$$

Since Equation 5.1 is analytically intractable, we use approximate inference. For comparison, we implement Importance Sampling (IS) and Metropolis Hastings (MH), where MH samples proposals from the prior. Although both of these algorithms behave differently, we will see in this section that both produce similar posterior samples. Before allowing the agents to begin

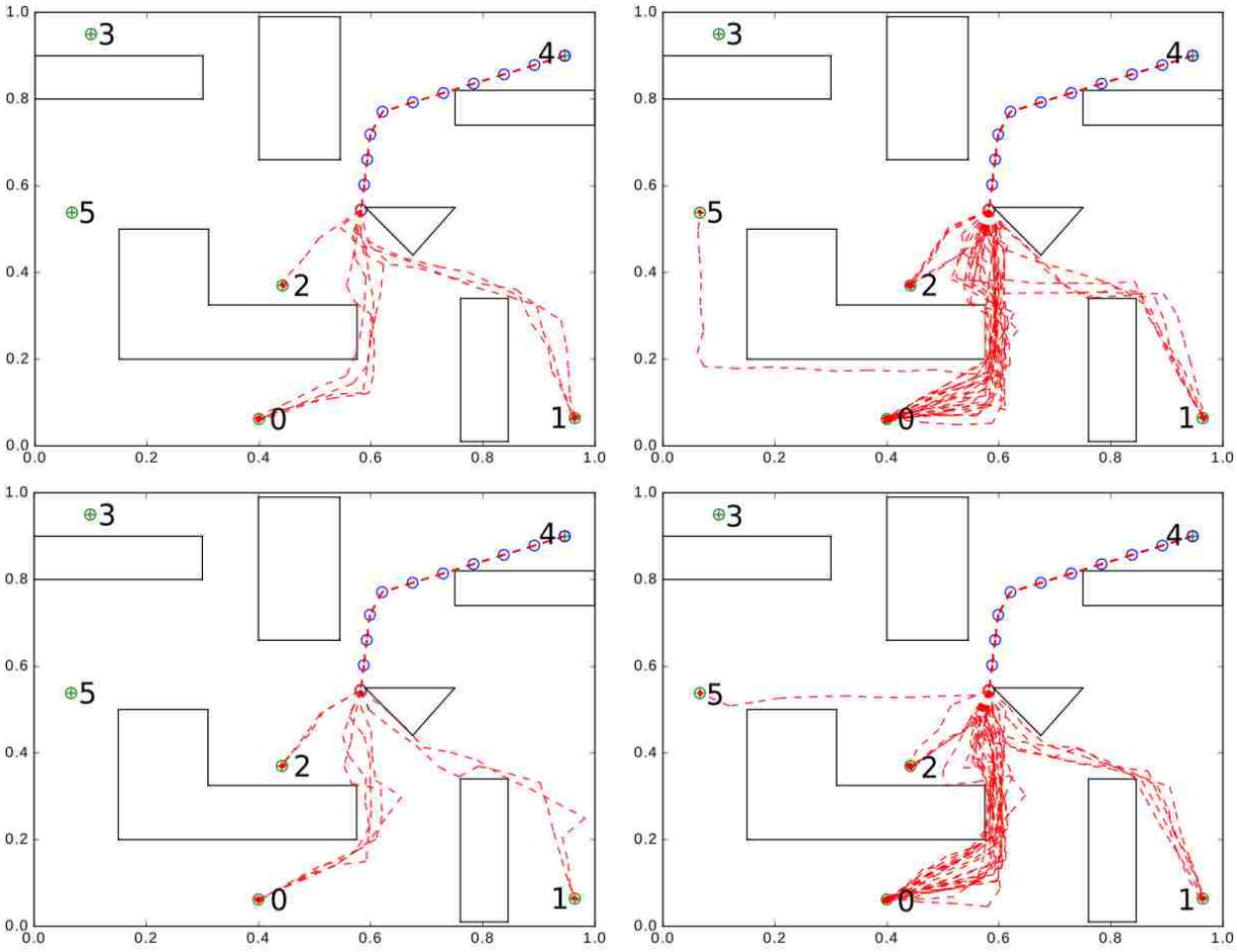

Figure 5.6: In each of these experiments we conditioned the start and goal location. Then we randomly generated a path between the two locations. Once we have the new random path, we conditioned the first 10 time steps. **Top Left:** Goal inference using Metropolis Hastings. 8 samples from the posterior using 32 particles with Metropolis Hastings. **Top Right:** we increase the posterior samples to 32 and the number of particles to 64. **Bottom Left:** Goal inference using Importance Sampling. 8 samples from the posterior using 32 particles with Importance Sampling. **Bottom Right:** Goal inference using Importance Sampling. 32 samples from the posterior using 64 particles.

Algorithm 4 Basic Goal Inference Simulation

```
Given: map, start, planned-path
Variables: β = last_time_step
for t = 1 : β do
    for i : t = 10 do
        condition(planned_path[i])
    end for
    inferred_goals = run_inference()
end for
```

playing games, i.e. running simulations of agents moving and simultaneously performing goal inference, we set up a scene where we can test our inference algorithms. We begin these experiments by randomly choosing a consistent starting and goal location, 4 and 0 respectively. Then we create a random semi-optimized path and use that path to condition the the first 10 time steps. We run inference algorithms to estimate the $p(\text{goal}|\text{start}, \text{step}_1, \text{step}_2, ..., \text{step}_{10})$.

The first algorithm we tested was Metropolis Hastings. We experimented by sampling 8 posterior samples using 32 particles where proposals are sampled from the prior. In the top-left plot in Figure 5.6, we see the results from running MH. The eight posterior samples are shown in red dashed lines, where again the blue circles indicate past observed locations. To further evaluate what kinds of samples we get from the posterior, we increase the number of samples and particles. On the right of that top-left plot, we show the results of running MH to sample 32 traces from the posterior using 64 particles. We conditioned on the same start and goal locations, and again the same first 10 time steps. Based on the past observations, we see that we get similar outcomes for the probabilities of the goal locations. We also see that the choices of goal locations are rational according to past observations. We ran the same experiments using Importance Sampling. We conditioned the same random values, and generated new random paths form which to condition on. Again we see that the outcomes produce rational goal locations based on the past time steps of agent movements.

We saw from setting up scenarios that our basic planning model can be used to perform goal inference when we observe the time steps of an agent. The next experiment we set up is to see how the inferred goals of an agent change as the agent moves. This means

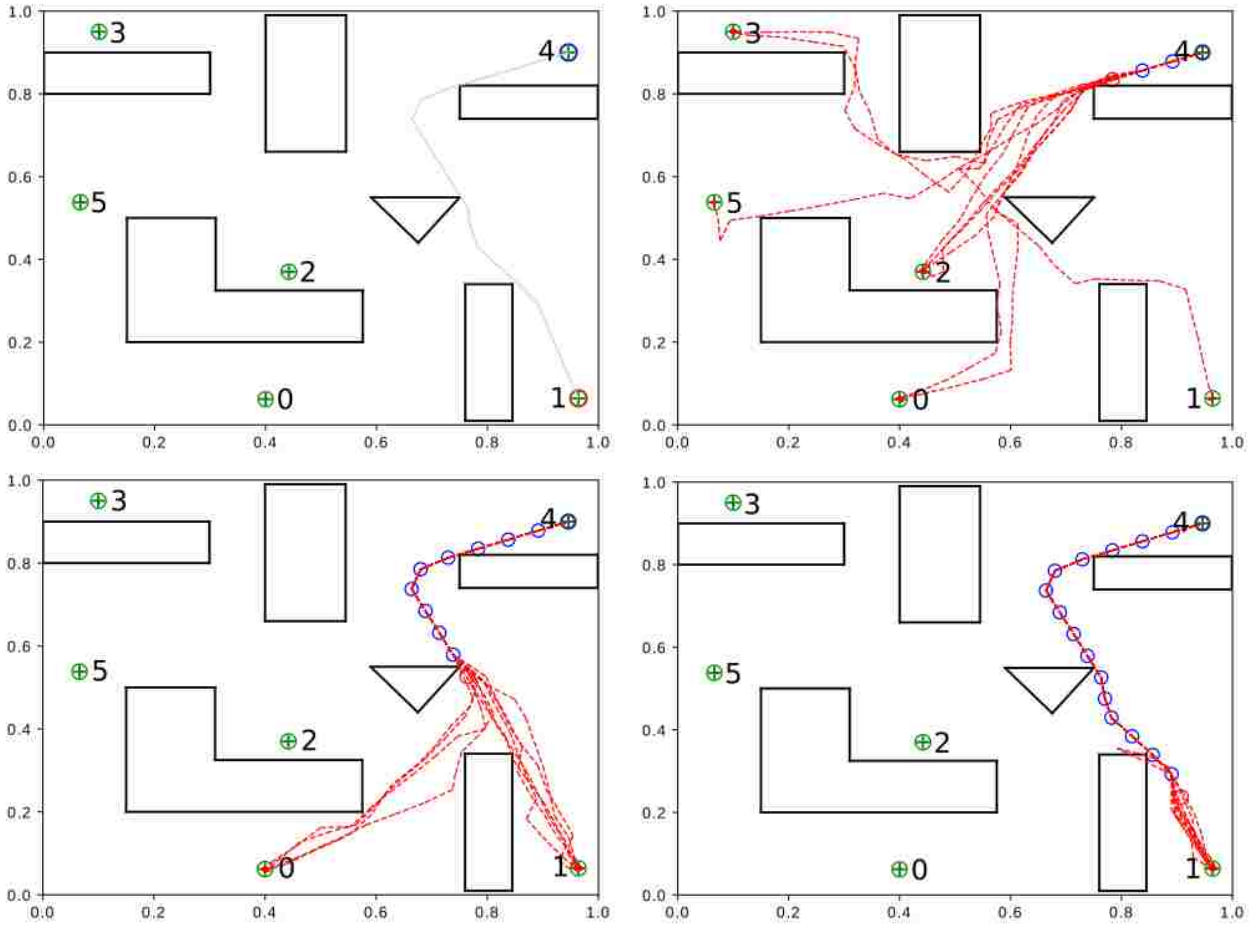

Figure 5.7: Top-Left: We show the pre-planned path of an agent from location 4 to 1. On the top-right, we show how all goal locations are possibilities based on current observations. On the bottom-left, we show how we converge to 2 possible goals, and on the bottom-right, we show how we converge to location 1 as the agent's final goal.

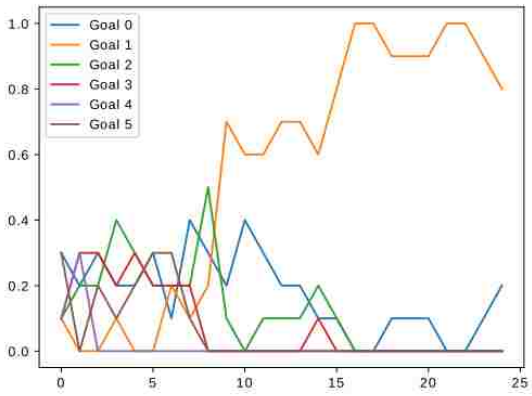

Figure 5.8: Left: We show the probabilities of each goal at each time step where the starting location was at 4 and the true goal is at location 1. This case reveals how the true goal location becomes more apparent with different amount of observations.

that we must perform goal inference at each new observations we gain from the agent. In this experiment we precomputed a random path from locations 4 to 1. Then we simulated the agent moving at each time step. At each time steps we conditioned the model to include new observations. We expected that as we gained new observations the inferred goal would converge to a single location. We set up the described simulation in Algorithm 4. Figure 5.7 shows the posterior samples for the rest of the path and the goal for the agent at different time steps. The plot on the top-left shows the pre-planned path in light grey. The start location is where the blue circle is located, ie. the agent's past observations. The goal location is at location 1, highlighted in red. To its left on the same Figure, we see inferences for the goal location at times step 3. We see that every goal is a possible candidate. On the bottom-left, at time step 10, we see that number of goal locations decreased to two, and at time step 16, the goal is very obvious and our inference algorithm captures that. To show how the probabilities of each goal change with more observations, we show Figure 5.8. This plot shows the probabilities for the previous simulation where the start location remains at 4 and the goal is at location 1. At the beginning of the plot we see how the probabilities for each goal remain low and quickly change as time progresses. By time step 10, the goal with

the highest probability is at location 1, with location 0 coming in second highest. However, as more observations are gathered, the probability of goal 1 continues to be higher.

We demonstrate with these experiments that goal inference can be performed on agents to predict their goal locations given movement observations. In addition, we demonstrate that we can infer more correct goals with more observations. Therefore we conclude this section by noting that when agents are given these models and inference methods, they are given the ability to predict (with high probability) the goals of other agents. Therefore, by designing agents to perform goal inference, we have agents that can infer the goals of other agents.

Chapter 6

The Chaser-Runner Model

In Chapter 5 we describe how we can perform goal inference on agents using simple models. However, in those scenarios, we assume the environment to be fully observable, meaning that we can observe all of the agents' movements through time as we attempt to infer their goal location. The reality is, however, that our chaser-runner problem is much more complex than the basic example previously described in Chapter 5. Therefore in this chapter, we discuss how we organize our nested models for the chaser-runner model and show how goal inference is performed in a more complex model. Additionally, we discuss the responsibilities of each level in the chaser-runner model and show how each level is designed and conditioned for inference.

6.1 Nested Modeling

Here we describe the centerpiece of this thesis, the *Chaser-Runner Model*. We will primarily describe this model from the perspective of the chaser; the runner model is similar, except that while the chaser wishes to maximize the probability of detection, the runner wishes to minimize it.

Our model has three levels: the **outermost model** describes the high level beliefs and planning of the chaser. The **middlemost model** describes what the chaser believes about the runner. The **innermost model** describes what the chaser believes about what the runner believes about the chaser. These three models work in tandem to create nuanced inferences about where the chaser believes the runner might be. Inference in the outermost model

Algorithm 5 Chaser's model [the outermost model]

```
Given: chaser_start, observations, t
t = ∼ categorical(possible time steps)
chaser_start ∼ categorical(possible start locations)
chaser_goal ∼ categorical(possible goal locations)
chaser_plan = optim_RRT( chaser_start, chaser_goal) + noise
runner_plan = run_inference( runner_model | chaser_start, chaser_goal, observations, t )
for i = 1 : T do
    if isovist(runner_plan[i], chaser_plan[i] ) == True then
        p = 0.999
    else
        p = 0.001
    end if
    runner_detected_{t_i} ∼ Bernoulli(p)
end for
```

reasons about the current location, planned path and goal of the runner, by marginalizing out the chaser's beliefs about the runner and his desire to avoid detection (the middlemost model). The middlemost model describes the runner's procedure to avoid detection: namely, by reasoning about where the chaser is (the innermost model) and planning a path to avoid detection.

6.1.1 Outermost Model

The outermost model, shown in Algorithm 5, describes the chaser trying to model what its path should be to increase the probability of detecting the runner. Because the chaser must model a path for itself to search for the runner, the start and goal locations are modeled as categoricals over a discrete set of candidate locations.

As part of the model, the chaser runs inference over a model of the runner and determines the most likely next location for the runner (note that this is nested inference, as *another* algorithm will be running inference over the outermost model). This nested inference conditions the middlemost model on a false detection, capturing the idea that the runner will plan to avoid detection.

Algorithm 6 Chaser's Runner model [the middlemost model]

```
Given: chaser_start, chaser_goal
t = ∼ categorical(possible time steps)
runner_start ∼ categorical(possible start locations)
runner_goal ∼ categorical(possible goal locations)
runner_plan = optim_RRT( runner_start, runner_goal) + noise
chaser_plan = innermost_model(chaser_start, chaser_goal)
for i = 1 : T do
   if  isovist(runner_plan[i], chaser_plan[i] ) == True  then
      p = 0.999
   else
      p = 0.001
   end if
   detected_{t_i} ∼ Bernoulli(p)
end for
```

The most likely next location is then used as the target for an optimized (but still random) RRT planner. The chaser samples a plan, and calculates whether or not the plan would result in detecting the runner. Thus, successful detection of the runner depends on correctly reasoning about the runner's location, goal, and which of many paths that the runner will take to navigate to his goal.

The outermost model is also conditioned on a sequence of observations. This represents what the chaser has seen so far; in our model, the simulation ends whenever the chaser spots the runner, so the observations will always be the absence of detections at previous time steps. These observations are handed to the nested inference to ensure that sampled runner's plans are consistent with those observations (we do not, for example, wish to sample paths for which we *would have* seen the runner, if in fact we did not).

Note that the chaser does not explicitly plan to intercept the runner; this is handled by conditioning the outermost model on successful detection. By conditioning on detection, the probabilistic programming language's inference algorithm automatically samples plans that result in detection with high probability, and is an example of planning-as-inference [30].

Algorithm 7 Chaser's Runner's Chaser model [the innermost model]

```
chaser_start ~ categorical(possible start locations)
chaser_goal ~ categorical(possible goal locations)
chaser_path = optim_RRT( chaser_start, chaser_goal )
return chaser_path + noise
```

6.1.2 Middlemost Model

The middlemost model, shown in Algorithm 6, describes the chaser's model of the runner. This model first reasons about the chaser via the innermost model, then constructs a plan and determines whether or not that plan will result in a detection.

Recall that the middlemost model is conditioned (by the outermost model) to result in a false detection. Samples drawn from the posterior distribution of this model will therefore describe paths that the chaser believes the runner will take to avoid detection, given where the the runner believes the chaser may be.

6.1.3 Innermost Model

The innermost model, shown in Algorithm 7, describes what the chaser believes the runner believes about the chaser. The chaser is modeled as having an arbitrary location, goal and path. The important part of this model is that it returns a sampled next location for the chaser; in expectation, this yields a distribution over where the chaser thinks the runner thinks the chaser will be on the next time step; the middlemost model then uses that information to plan to avoid detection.

This model is conditioned on the observations seen so far, thereby ruling out start/goal combinations that would result in paths where the runner would have been detected.

6.2 Conditioning the Models

Since the objective for the chaser is to detect the runner, we now briefly describe how to infer the future locations of the runner using nested inference. We first condition the outermost

model, Algorithm 1, with the given observations of the chaser's own locations and the runner's starting location. We also condition detections of the runner to be false, and the future detections of the runner to be true.

To determine if the runner will be detected in the future, we must first determine the runner's location in a future time step. To infer the location of the runner, we perform nested inference. This is where we condition the middlemost model, Algorithm 2, with the previously conditioned starting location for the runner, the sampled goal for the runner, and past detections of the runner to be false. Algorithm 2 uses Algorithm 3 to sample the next possible step of the chaser, which allows inference to be done over the next step of the chaser, which then the runner can use to plan around the future movement of the chaser.

Chapter 7

Chaser-Runner Experimental Setup and Results

Our experiments are designed to answer the key question: does a more sophisticated model of a runner enable a chaser to detect the runner more often? Our experiments are composed of three main categories: 1) *computational*, where we demonstrate that our inference algorithm empirically converges the same as using Metropolis Hasting, 2) *model flexibility*, where we demonstrate that rational behavior is manifested by the models depending on conditioning, and 3) *detection rates*, demonstrating that more complex models increase detection rates of the runner by implementing our full Theory of Mind chaser-runner model and thus justifying the complexity of nested inference.

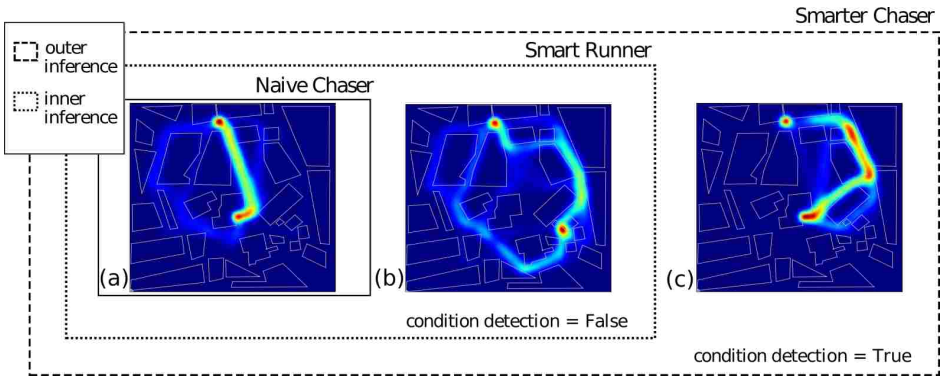

Figure 7.1: **The chaser-runner model** showing the innermost, middlemost, and outermost models. (a)-(c) show posterior distribution over paths after running importance sampling. See text for details.

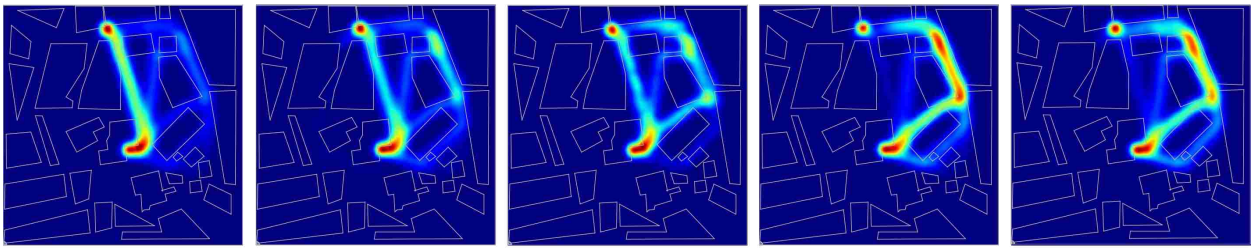

Figure 7.2: **Importance Sampling with Particle Counts of 16, 32, 64, and 128.** (a) - (d) shows expected occupancy from the posterior when running importance sampling. We determined that the distribution converged at 128 particles. As a comparison, we ran the model with an additional inference algorithm, Metropolis-Hastings, where proposals were sampled from the prior (e); the two are equivalent.

7.1 Computational Experiments

Our next experiment shows how nested inference converges empirically to rational behavior at each layer of the model. We begin the experiments by setting up inference to work with a worst case scenario of the runner having full knowledge of the chaser's whereabouts. Figure 7.1 (a) shows a heat map of paths of how the model's innermost model of the chaser naively plans from a conditioned start and goal location. Figure 7.1 (b) reveals how a runner, i.e. the middlemost model, uses the innermost model to plan and avoid detection while reaching the same goal as the chaser (which we explicitly condition to be the same). Lastly, Figure 7.1 (c) shows the outermost model converging to a plan that leads to higher detection probability of plans sampled from the middlemost model of the runner. We show an accumulation of 100 sampled paths from the posterior using importance sampling with 128 particles in Figure 7.1.

As a means to explain how we chose the particle count for our importance sampling algorithm, we show Figure 7.2 (a)-(d); a series of posterior distribution over paths from running outer inference on the chaser-runner model using importance sampling with varying particle counts, 16, 32, 64, and 128. As shown, we converge to the path of most future detections using 128 particles. As a comparison, we exchanged our outer inference algorithm with Metropolis-Hastings, where we sampled proposals from the prior. As shown in (e), the posterior distribution is similar to that produced when using importance sampling in (d).

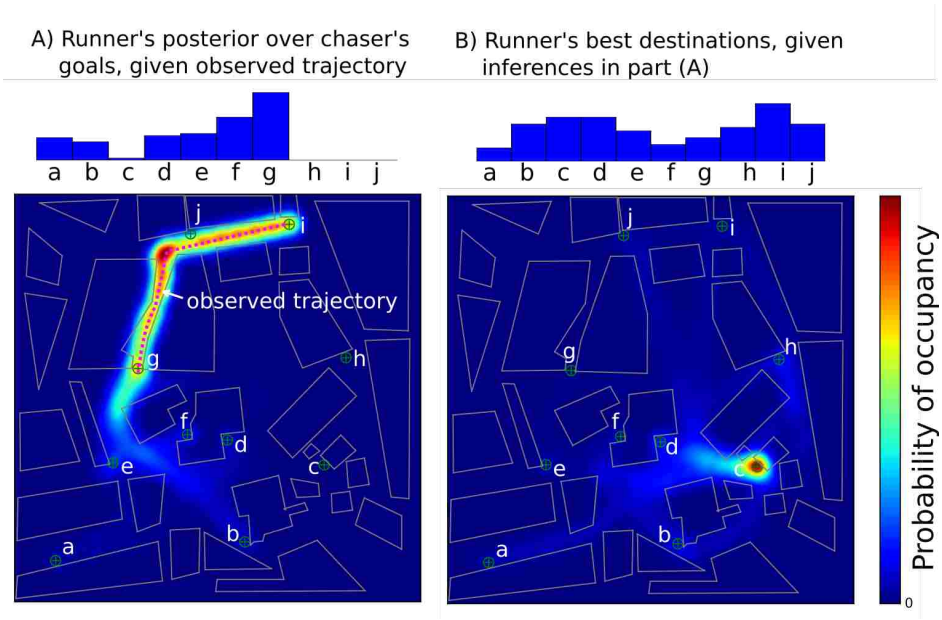

Figure 7.3: Here, we show that if the runner observes a partial chaser trajectory, it can infer the chaser's goals. An example is that location $\boldsymbol{i}$ on the map has low posterior probability, because if $\boldsymbol{i}$ were the chaser's destination, the observed path would be very unlikely. The runner then reasons about the safest possible destinations; location $\boldsymbol{i}$ has high probability, since the runner is convinced the chaser is not headed there.

7.2 Model Flexibility Experiments

The following experiments focus on the middlemost model, where we test the basic behavior of the chaser's model of the runner to infer goals and perform stealth path planning. The first experiment shows how the model performs goal inference in the middlemost level in order for the runner to avoid detection from the chaser (the innermost model). As the runner tries to minimize detection by conditioning $detection = false$, the runner infers the goals of the chaser and actively infers plans to the goals that the chaser is less likely to head to. The second experiment demonstrates how the middlemost model of the runner naturally manifests a stealth behavior when we simply condition the *detection* random variable.

7.2.1 Middlemost Model and Goal Inference

We setup this experiment by placing the chaser and runner in a simulation at time step 15, where the chaser is initially placed on the top right of the map (shown in Figure 7.3

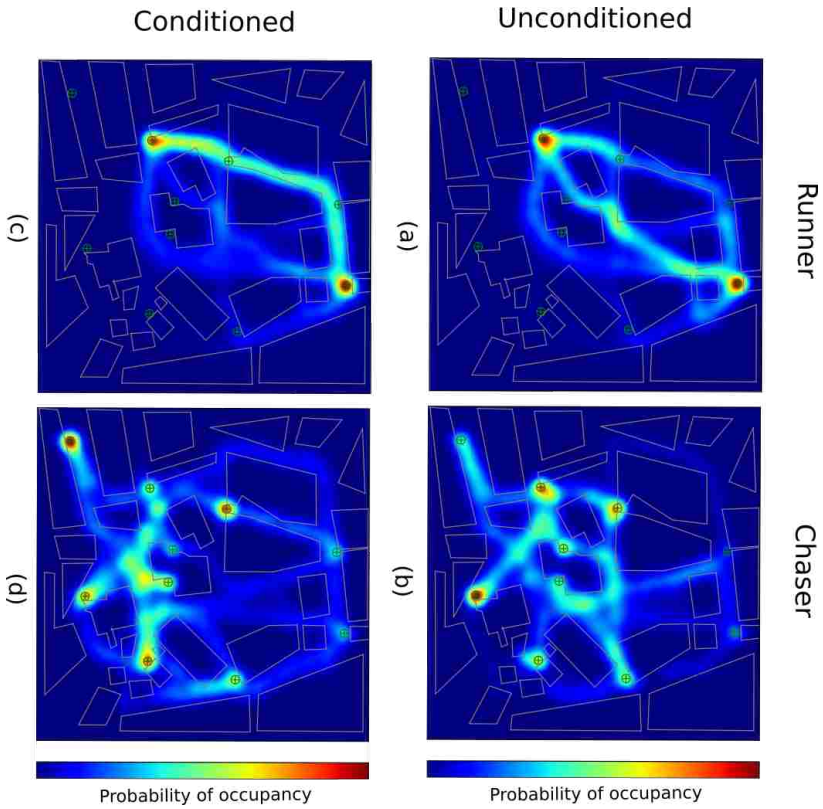

Figure 7.4: We demonstrate how the behavior of a stealth runner manifests itself from the models when we condition on detection=false. (a)-(b) show figures of a runner (a) and a chaser (b) where we ran inference over the middlemost model but did not condition the random variable *detection*. (c)-(d) show a runner (c) and a chaser (d) where we ran inference, but **conditioned** detection=false. Each figure shows an aggregate of 50 samples from the posterior using self-normalized importance sampling using 128 particles.

(A)) at location $\boldsymbol{i}$, and the runner is placed on the lower right side of the map (shown in Figure 7.3 (B)) at location $\boldsymbol{c}$. To run inference, we condition the starting locations for both agents, the observed chaser locations, and future detections of the runner to be false. Figure 7.3 (A) shows the posterior distribution over chaser goals (for 100 samples) given previous observations. Inferred goals for the chaser focus on the bottom left of the map while the goals for the runner focus on the top right of the map Figure 7.3 (B).

Since the runner infers that the top right goals (such as locations $\boldsymbol{h}$, $\boldsymbol{i}$, and $\boldsymbol{j}$) have zero probability for the chaser, the runner clearly heads toward the starting location of the chaser in order to avoid detection most often. This illustrates both the principle of rationality and counter-factual reasoning because if location $\boldsymbol{i}$ were the chaser's destination, then the observed path would have been very unlikely.

We note that normally when running inference over the full chaser-runner model, the goals for the agents would be conditioned, but we leave them unconditioned here to demonstrate that rational behavior can still be manifested by the models. In essence, we showed that the runner can perform goal inference on the chaser and choose appropriate goals to avoid detection.

7.2.2 Middlemost Model and Stealth Behavior

The next experiment we discuss demonstrates how the behavior of a stealth runner manifests itself from the models when we condition on *detection=false* in the middlemost level of the chaser-runner model. Figure 7.4 (a)-(b) show figures of a runner (a) and a chaser (b) where we run inference over the middlemost model, but do **not** condition the random variable *detection*. For 50 samples, we constrain the start and goal locations for the runner while the chaser has no conditioning at all. We show in Figure 7.4 (a) that when we do not condition *detection* in the model, the runner naively plans a path to its goal location. Figure 7.4 (c)-(d) show a runner (c) and a chaser (d) where we run inference, but now **condition** *detection=false*, thus demonstrating how the path of the runner changes in order to enforce detections to remain

false throughout time. Therefore, we show that our middlemost model naturally reveals the ability to plan stealth paths to goals in order to remain undetected by the chaser.

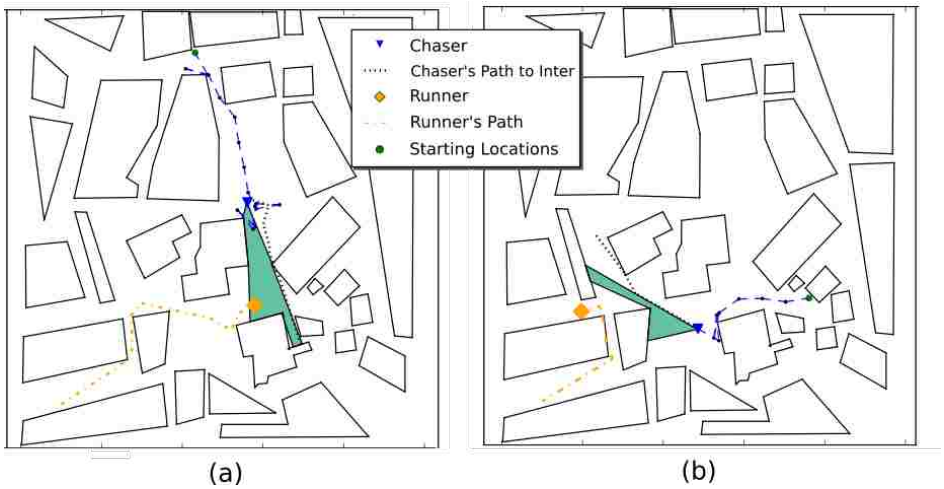

Figure 7.5: **Smart Chaser Simulations.** Examples of typical chase simulations using the smart chaser with a naive runner and a smarter runner. See text for details.

7.3 Detection Experiments

The following experiments compare several models of chasers and runners via detection rates. For these experiments, we run full simulations through time of these two agents interacting with one another. The runner, having an undisclosed destination, attempts to travel to its goal location with the additional goal of remaining undetected by the chaser, who is traveling as well in pursuit of the runner. The following sections compare models and show the detection rates for each case.

7.3.1 Experiment 1: Naive Runner, Smart Chaser

The first experiment illustrates how a relatively *smart chaser* can reliably head off and detect a *naive runner*. We define the naive runner to be a runner that simply plans a path from start to its goal location, without any particular reasoning about the chaser. We define the smart chaser the same as our nested runner model to be a chaser that has an accurate mental model of the naive runner – in other words, the smart chaser expects the naive runner to

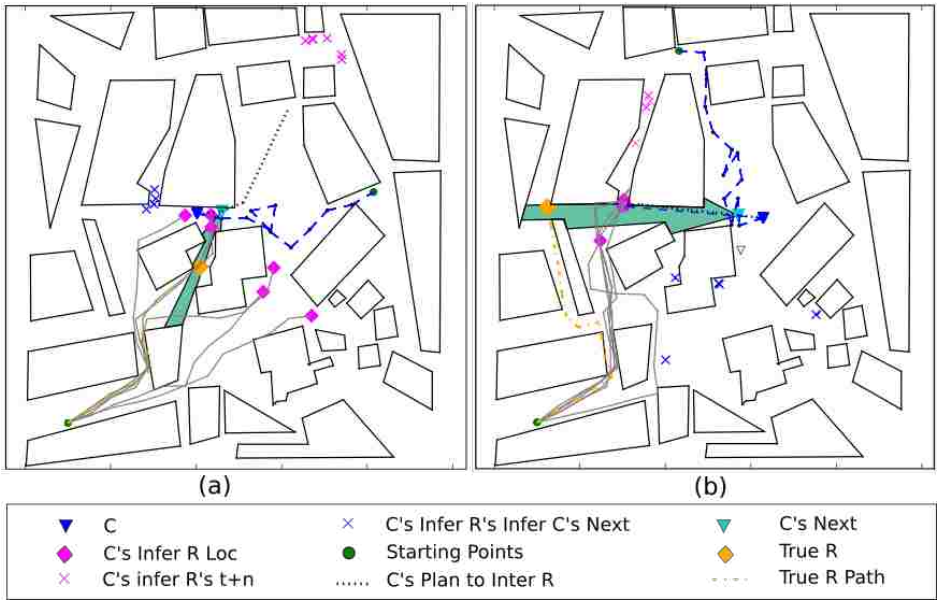

Figure 7.6: **Smartest Chaser Simulations.** On the left: a simulation of the smartest chaser against the naive runner. On the right: the chaser infers the runner's location to be more hidden, avoiding the center of the map. In this simulation, the chaser successfully detects the smarter runner. See text for details.

plan a path directly from start to goal. The smart chaser's plan is conditioned on detecting the naive runner, implying that it will sample plans that tend to head off the naive runner. Figure 7.5 (a) illustrates a prototypical result, where the blue upside down triangle describes the chaser, the yellow diamond describes the runner, and the teal polygon represents the field of view, or the isovist for the chaser. As the chaser travels down the map, it plans a path to intercept the runner (shown as a dotted black path) and detects the runner. In this experiment, the chaser's plans tend to keep crossing the center of the map where it reliably detects the runner. This experiment had a relatively high detection rates of ≈ 0.80 (28/35 simulations), shown in Table 7.1.

7.3.2 Experiment 2: Smarter Runner, Smart Chaser

For our second experiment, we increase the model complexity of the runner. The *smarter runner* is a runner with a mental model of a chaser that has a mental model of a simple

runner. The smarter runner expects the chaser to remain in the center of the map, as it is trying to head off a naive agent.

The smart chaser is the same as in Experiment 1. Figure 7.5 (b) illustrates a prototypical result. Because the smarter runner knows how to avoid the chaser, and because the chaser still has a mental model of a runner that goes straight for the goal, the runner avoids detection. In Figure 7.5 (b), the runner is seen swerving sharply left taking a longer path around the perimeter of the city to reach its goal. As a result, the chaser is unable to find the runner for the rest of the simulation. Our experiments revealed a lower detection rate of ≈ 0.36 (12/33 simulations), shown in Table 7.1.

7.3.3 Experiment 3: Naive Runner, Smartest Chaser

We now turn to the central experiments on the full Chaser-Runner model. In this model, the runner is the naive runner from Experiment 1, but the chaser is the full chaser described in Section 6.

Figure 7.6 (a) illustrates a prototypical result. This figure is an example of a successful detection. 'C' stands for Chaser; 'R' stands for runner. The blue triangle represents the chaser's true, current location; blue dashed lines and Xs represent the chaser's inferences about the runner's inferences about where the chaser will move next (this is the result of inference in the middlemost model). Magenta diamonds represent samples of where the chaser thinks the runner is; magenta Xs represent samples of where the chaser thinks the runner will move next. The chaser correctly predicts the runner's expected next location (blue circle) and plans a path to intercept it (black dashed line); the chaser's isovist is shown as a light green polygon.

Here, the multimodality of the model's inferences is apparent: the chaser predicts two possible modes where the runner could be (clusters of magenta triangles), but assigns more probability mass to the upper (correct) cluster; the result is that the chaser plans a path to

that location, which results in a detection. After running simulations, it had a detection rate of ≈ 0.88 (29/33 simulations), shown in Table 7.1.

7.3.4 Experiment 4: Smarter Runner, Smartest Chaser

This experiment tests our fully implemented Theory of Mind model against the smarter runner. Figure 7.6 (b) illustrates a prototypical result and an example of a successful detection. As seen, the chaser now predicts the runner will avoid highly visible areas of the map and travel through alley ways and around the city. In this scenario, we see that the model's inference become unimodel and the chaser is able to detect the runner. This experiment yielded a detection rate of ≈ 0.56 (19/34 simulations), shown in Table 7.1. Although these detection rates may seem low at first, recall that Experiment 2 yielded a low 0.36 detection rate. Our implementation of the full Theory of Mind model suggests that the chaser improves detection rates and is able to successfully model and infer the behavior of the smarter agent.

7.3.5 Detection Experiment Discussion

Table 7.1: Detection Rates for Types of Agents

	naive runner	smarter runner
smart chaser	0.80	0.36
smartest chaser	0.88	0.56

Table 7.1 collects the detection rates for each of the four experiments. In Experiment 1, the smart chaser has a correct model of the runner, and therefore has a relatively high detection rate of 0.80. When the naive runner competes against the smartest chaser (Experiment 3), the detection rate increases. When the smarter runner is introduced (Experiment 2), detection rates for the smart chaser decrease to 0.36. Lastly, when we simulate the smarter runner and the smartest chaser, the detection rates increase to 0.56.

This confirms a critical point: ***better models matter.*** When the runner reasons more deeply, he evades more effectively; when the chaser reasons more deeply, he intercepts

more effectively. Furthermore, a single, unified inference algorithm results in a wide variety of intuitive, rational behavior for both the runner and the chaser, suggesting that *model complexity* is more important than *inference algorithm complexity*. For example, there is no explicit notion that the runner might wish to hide in narrow alley ways, but this emerges simply by conditioning the model on not being detected.

Chapter 8

Conclusion

In the beginning of this thesis, we considered the question, "How do we give robots the ability to infer the mental state of another robot?", and more importantly, "How do we program a robot to reason about that mental state for decision making and planning?" This thesis demonstrates how we can do inference on complex models by using simple nested inference algorithms. We show that simple models of Theory of Mind can capture a variety of rich behaviors and that probabilistic programming is a natural way to describe those models. We demonstrate though a series of experiments that runner detections increase as we increase the complexity of the chaser model, therefore showing that more complex models produce improved behavior, and thus improved detection rates. Our ultimate goal is to implement these basic 2D ideas on actual UAVs; the extensive case study in this thesis suggests that such a goal is achievable in the near future.

8.1 Future Work

One of the virtues of a Bayesian approach is compositionality. While we assumed access to a high-level map, the same framework could be applied to a joint model that blends high-level reasoning with low-level perception. In such a model, inferences driven by theory of mind models could go beyond goals and paths, and could additionally infer (for example) the existence of objects or other agents seen by the runner, but not by the chaser. Such integrated models may additionally require inference metaprogramming; but how best to make such models computationally tractable is an open question. Barnes-Holmes et al. [4]

claims that levels of complexity must increase in order to understand informational states of the mind. However, as we increase the the amount of nested levels in our models, the computational time for running them increase exponentially. Therefore, we leave ameliorating computational complexity to future work.

References

[1] Muhammad Awais and Dominik Henrich. Human-Robot Collaboration by Intention Recognition Using Probabilistic State Machines. In *Robotics in Alpe-Adria-Danube Region (RAAD), 2010 Institute of Electrical and Electronics Engineers (IEEE) 19th International Workshop on*, pages 75–80. IEEE, 2010.

[2] Chris L Baker and Joshua B Tenenbaum. Modeling Human Plan Recognition Using Bayesian Theory of Mind. *Plan, Activity, and Intent Recognition: Theory and Practice*, pages 177–204, 2014.

[3] Chris L Baker, Rebecca Saxe, and Joshua B Tenenbaum. Action Understanding as Inverse Planning. *Cognition*, 113(3):329–349, 2009.

[4] Yvonne Barnes-Holmes, Louise McHugh, and Dermot Barnes-Holmes. Perspective-Taking and Theory of Mind: A Relational Frame Account. *The Behavior Analyst Today*, 5(1):15, 2004.

[5] Simon Baron-Cohen, Alan M Leslie, and Uta Frith. Does the Autistic Child Have A "Theory of Mind"? *Cognition*, 21(1):37–46, 1985.

[6] M L Benedikt. To Take Hold of Space: Isovists and Isovist Fields. *Environment and Planning B: Planning and Design*, 6(1):47–65, 1979. doi: 10.1068/b060047. URL `http://dx.doi.org/10.1068/b060047`.

[7] Dorit Borrman and Andreas Nuchter. Dataset Generated by Dorit Borrman and Andreas Nuchter of Jacobs University Bremen. `http://kos.informatik.uni-osnabrueck.de/3Dscans/`, 2017. Accessed: 2017.

[8] Matthew Botvinick and Marc Toussaint. Planning as Inference. *Trends in Cognitive Sciences*, 16(10):485 – 488, 2012. ISSN 1364-6613. doi: https://doi.org/10.1016/j.tics.2012.08.006. URL `http://www.sciencedirect.com/science/article/pii/S1364661312001957`.

[9] Nick Chater, Joshua B Tenenbaum, and Alan Yuille. Probabilistic Models of Cognition: Conceptual Foundations. *Trends in cognitive sciences*, 10(7):287–291, 2006.

[10] Marco F. Cusumano-Towner and Vikash K. Mansinghka. Encapsulating Models and Approximate Inference Programs in Probabilistic Modules. *Computing Research Repository (CoRR)*, abs/1612.04759, 2016. URL `http://arxiv.org/abs/1612.04759`.

[11] Alan Fern, Sriraam Natarajan, Kshitij Judah, and Prasad Tadepalli. A Decision-Theoretic Model of Assistance. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1879–1884, 2007.

[12] Chris Frith and Uta Frith. Theory of mind. *Current Biology*, 15(17):R644–R645, 2005.

[13] Noah Goodman, Vikash Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua Tenenbaum. Church: A Language for Generative Models. In *Uncertainty in Artificial Intelligence (UAI)*, 2008.

[14] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and Acting in Partially Observable Stochastic Domains. *Artificial intelligence*, 101(1):99–134, 1998.

[15] Daphne Koller, David McAllester, and Avi Pfeffer. Effective Bayesian Inference for Stochastic Programs. In *Association for the Advancement of Artificial Intelligence (AAAI)/ Innovative Applications of Artificial Intelligence (IAAI)*, pages 740–747, 1997.

[16] Steven M LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998.

[17] Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: A Higher-Order Probabilistic Programming Platform with Programmable Inference. *arXiv preprint arXiv:1404.0099*, 2014.

[18] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic Models with Unknown Objects. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1352–1359, 2005.

[19] T. Minka, J.M. Winn, J.P. Guiver, and D.A. Knowles. Infer.NET 2.4, 2010. Microsoft Research Cambridge.

[20] Vlad I Morariu, V Shiv Naga Prasad, and Larry S Davis. Human Activity Understanding Using Visibility Context. In *International Conference on Intelligent Robots and Systems (IEEE/RSJ) International Conference on Intelligent Robots and Systems (IROS) Workshop: From Sensors to Human Spatial Concepts (FS2HSC)*, 2007.

[21] Truong-Huy Dinh Nguyen, David Hsu, Wee-Sun Lee, Tze-Yun Leong, Leslie Pack Kaelbling, Tomas Lozano-Perez, and Andrew Haydn Grant. Capir: Collaborative Action Planning with Intention Recognition. *arXiv preprint arXiv:1206.5928*, 2012.

[22] Art B. Owen. *Monte Carlo Theory, Methods and Examples.* 2013.

[23] Avi Pfeffer. IBAL: A Probabilistic Rational Programming Language. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 733–740, 2001.

[24] Tom Rainforth, Robert Cornish, Hongseok Yang, and Frank Wood. On the Pitfalls of Nested Monte Carlo. In *Neural Information Processing Systems (NIPS) Workshop on Advances in Approximate Bayesian Inference*, 2016.

[25] Rajesh Ranganath, Sean Gerrish, and David M Blei. Black Box Variational Inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 814–822, 2014.

[26] Dorsa Sadigh, S Shankar Sastry, Sanjit A Seshia, and Anca Dragan. Information Gathering Actions Over Human Internal State. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 66–73. IEEE, 2016.

[27] T. C. Schelling. The Strategy Of Conflict. 1960.

[28] Andreas Stuhlmüller and Noah D Goodman. Reasoning About Reasoning by Nested Conditioning: Modeling Theory of Mind with Probabilistic Programs. *Cognitive Systems Research*, 28:80–99, 2014.

[29] David Tolpin, Jan Willem van de Meent, Hongseok Yang, and Frank Wood. Design and Implementation of Probabilistic Programming Language Anglican. *arXiv preprint arXiv:1608.05263*, 2016.

[30] Marc Toussaint, Stefan Harmeling, and Amos Storkey. Probabilistic Inference for Solving (PO)MDPs. Technical Report EDI-INF-RR-0934, University of Edinburgh, 2006.

[31] Henry M Wellman. The Child's Theory of Mind. 1990.

[32] David Wingate and Theophane Weber. Automated Variational Inference in Probabilistic Programming. *Computing Research Repository (CoRR)*, abs/1301.1299, 2013. URL `http://arxiv.org/abs/1301.1299`.

[33] David Wingate, Andreas Stuhlmueller, and Noah D. Goodman. Lightweight Implementations of Probabilistic Programming Languages via Transformational Compilation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[34] Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A New Approach to Probabilistic Programming Inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1024–1032, 2014.

[35] Luke Zettlemoyer, Brian Milch, and Leslie P Kaelbling. Multi-Agent Filtering with Infinitely Nested Beliefs. In *Advances in Neural Information Processing Systems*, pages 1905–1912, 2009.