MODELING AN INTELLIGENT INTRUDER IN A REGION MONITORED BY A WIRELESS SENSOR NETWORK

by

SRIRAM SRINIVASAN

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTERS IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2013

ACKNOWLEDGEMENTS

Firstly I would like to thank my supervising professor Dr. Matthew Wright without whom this thesis could not have been possible. I thank him for his constant support, guidance and inspiration which kept me going till the end. I am extremely grateful to Dr. Manfred Huber and Dr. Gregely Zaruba, my committee members who have given their invaluable time, and shown interest in my thesis.

I would further like to thank all my colleagues in the Information Security lab who have constantly helped me, and supported me at different times during my presence in this lab. It was always a pleasure working in the lab. I would like to thank all my roommates who have constantly helped and supported me throughout my masters. Their support, belief, and encouragement always gave me a moral boost and always kept me going.

Finally I would like the express my fullest gratitude to my beloved parents and brother for always believing in me and supporting me at all times.

November 22, 2013

ABSTRACT

MODELING AN INTELLIGENT INTRUDER IN A REGION MONITORED BY A WIRELESS SENSOR NETWORK

Sriram Srinivasan, M.S.

The University of Texas at Arlington, 2013

Supervising Professor: Dr. Matthew Wright

Monitoring to detect unauthorized border crossing is very important for protecting national security. To accomplish this by continuous physical monitoring by border patrol agents is impractical. Networks of low-cost wireless sensors have been identified as a useful tool in monitoring with minimal human intervention. However, ensuring the effectiveness of unattended monitoring against an intelligent intruder is difficult, since the intruder can probe the system for weaknesses. To better understand the capabilities of such an intruder, we propose a model for an intelligent intruder whose purpose is to find a detection-free path across the border by which he could cross back and forth without the risk of detection. Our intruder model is composed of four agents, each with a specific task: Explore, Exploit, Evade, and Policy. The first three agents follow the best course for achieving their named goals. The Policy agent is an intelligent agent that learns on different maps whether to pick Explore or Exploit or Evade given the intruders current knowledge about the map. We use Q-learning (QL) to train the Policy agent over various maps. In QL, the agent gets a positive reward for doing something right (e.g. moving to a detection-free zone closer

to the goal) and a negative reward for doing something wrong (e.g. getting caught by a sentry). We investigated different factors that affect the intruder's behavior, like the effect of different rewards, different sensor coverage of the region, and faster and more effective sentries. Our results show that after getting trained on enough number maps, the agent becomes better at finding a detection-free path across the border region. In particular, it reduces the capture rate and the number of steps required to find a detection-free path. Therefore, an intelligent intruder like this can be used to build and test different defense mechanisms.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Sensors are low-cost and low-power devices that posses capabilities to do sensing, limited processing, and wireless transmission of signals. These sensors are generally used in monitoring the physical world. In most cases a collection of such sensors are used to perform a specific task in the real world, and such a collection is called a wireless sensor network [1]. These networks can be used to identify and track movements and activities in a region. Hence, there is potential to leverage these networks in unmanned surveillance in border regions between countries and surrounding sensitive facilities such as nuclear and chemical plants, city water supplies, and military bases [2].

In this thesis, we use surveillance of the country borders as our primary motivating factor. Monitoring country borders for countries like the United States of America, whose border lasts for over 7000 miles is a very difficult task. The United States of America spends billions of dollars in securing its borders by constructing fences, video surveillance and other means. Monitoring such a vast border requires large amounts of man-power as well. Even after spending all those resources, it is extremely difficult to stop different unlawful groups like drug cartels and terrorists from breaching these borders. These groups keep looking for areas that are monitored poorly and capitalize on them. Their main goal for intrusion is to get their illegal goods and supplies into the country. To do so they rely on such weakly monitored areas where the risk of losing the goods or their men is low. Hence, there is a necessity for a better monitoring system. This could be achieved by the use of sensor networks.

Even though the use of such networks could be very useful, wireless sensor networks deployed in the real world are vulnerable to all sort of attacks. The attack could be both physical(like device tampering, jamming) or logical(like Sybil, eavesdropping[3], selective routing). These different attacks could be used to mitigate the capabilities of a network. But when we consider physical intrusion in a sensor network, there are 2 major factors that determine the ability to detect a physical intrusion :

- *Connectivity:* The ability to transmit a message from one node to another in a network is called connectivity. A network is completely connected if every node in that network is reachable.
- *Coverage:* The amount of physical region the sensors can cover is called the network coverage. A network is said to have full coverage if every part of a region is accessible by at least one sensor in that network.

Maintaining complete coverage and connectivity [4] is not a trivial task. There are many difficulties in achieving this in a vast region such as country borders [5]. One of the major challenges would be that, the amount of sensors needed to cover a vast region such as the U.S border (with sensors whose range is ten meters) would be very large. Additionally, it would be a bigger challenge to place them in the border at right intervals throughout to get complete coverage. The other major problem is that the battery life of a sensor is mostly about six months, even if we manage to precisely place the sensors every ten meter on all kinds of terrains, it would be extremely expensive to replace those every 6 months manually. Apart from this, node faults and other problems further affect the network and make it more difficult to maintain. Hence, getting complete coverage and connectivity is practically impossible. This means that there is most likely a path in this area that will take an intruder from one

country to the other without being detected by any sensor. We call such a path as a detection-free path.

A detection-free path may not be obvious when a member of the group enters the network and start exploring. But the sensors broadcast signals making it completely vulnerable to eavesdropping. Hence, an intruder could silently eavesdrop and at the least obtain information such as signal strength which could be used to precisely find the location of the sensor, after enough probing. And there is always a delay in capture of intruder after being sensed. Hence, he could gain a significantly large amount of information about the network and communicate with his members. His group members could use this information to further intrude and discover a completely detection-free path.

The most obvious form of defense against such an attack is to increase the coverage. But this could be very expensive considering the vastness of the region [6]. There are different approaches people have tried to improve detection [7, 8, 24]. But to improve intrusion detection, it is extremely important to have an intruder who would adapt, improve, and behave like a real-world human intruder. In this work we try to build an intelligent intruder which will adapt by learning on many maps using a technique called Q-learning.

Contributions: We try to model an intruder whose purpose is to find a detection-free path in a border monitored by sensors. We also try to minimize the number of times the intruder gets captured and number of steps it takes to find this path. We generalize and abstract the state space for our agent so that the agent can re-use the information learned on one map in another. We further analyze the intruder's behavior on maps with different sensor coverage, better sentries, and different costs for getting captured by a sentry. Our results show the improvement of the intruder

before and after learning. We could further use this intruder to build a more effective detection system.

CHAPTER 2

BACKGROUND

In this chapter we talk about a few topics which will be essential in understanding our attack model. We have modeled our attacker to use a concept of machine learning called reinforcement learning. The goal of our attacker is, physical intrusion in an area monitored by wireless sensor networks. Hence, we discuss in brief about wireless sensor networks and their security, machine learning, and reinforcement learning. We also talk about a few related works in this section.

2.1 Wireless sensor networks

With recent advancements in hardware technologies and wireless communication, people have been able to build small and low-power devices called wireless sensors. These sensors can measure a certain aspect of the environment around them. These sensors, upon detecting a change in the environment, can send a signal/data to their neighbors through wireless media. A collection of such sensors are used cooperatively to carry out a specific task. Such a collection or network is called a wireless sensor network (WSN)[9]. In a WSN, every sensor is called a node. Nodes by themselves do not analyze the data acquired, i.e. they can only sense the environment and send a message. This message is received by a key component of the network called the base station. The base station is like the backbone of the network. It acts as an interface between the user and the sensors. Once the message reaches the base station, the user can access this message and analyze the environment. These sensors can read different sets of data like temperature, sound, vibrations, movement

and so on. Hence the application of such a network can be found in various fields ranging from military to home applications[10]. For example in border monitoring, these sensors are placed randomly throughout the border and each sensor can detect movement in the environment. As the sensor detects movement around it, a sentry is made to inspect the area to capture the intruder.

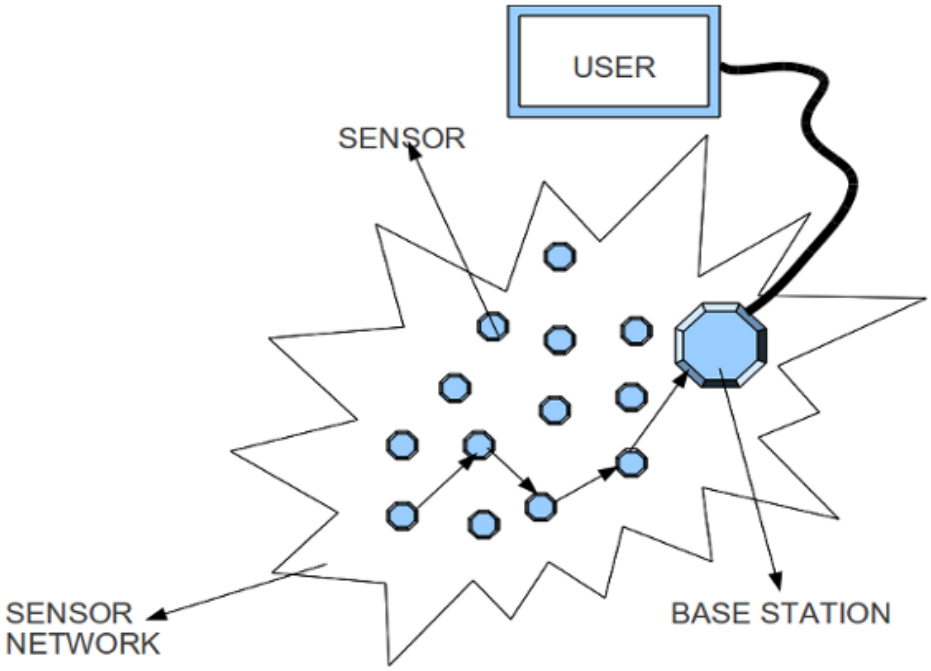

Figure 2.1. A wireless sensor network.

A WSN is generally completely distributed. Each sensor keeps sensing a small area around it. As it detects a change in the environment it prepares to send a signal to the base station. These sensors are deployed in a vast area. The wireless signal range of these sensors are very small when compared to the area they are deployed in. Hence, it is not possible for the sensor to send a signal directly to the base station. However, each sensor can pass messages to other nodes in its wireless range. This ability to pass messages is used to design different protocols to route

messages to the base station through different nodes in the network. This type of message passing is called multi-hop message passing. Once the base station receives this signal, it sends back an acknowledgment, and if the node does not receive this, the signal is retransmitted. Once the signal reaches the base station, the data can then be analyzed to make interpretations about the environment. Fig. 2.1 represents a typical WSN and its working. These sensor networks face a lot of security threats in general. The threats range from an attacker physically damaging it to an intruder silently listening to the network[1, 11]. The focus of our research is on using WSN for border monitoring and physical intruder detection. The goal of our attacker is to be able to cross the border without being detected or caught by a sentry. The type of attacks that would make most sense in aiding an attacker cross borders undetected are the ones where the intruder, captures and/or destroys nodes, or attacks the base station. Many intrusion detection systems have been developed to deal with different type of attacks. We have discussed a few of them in the related works section at the end of this chapter. However, our intruder model does not use any regular attack strategies but tries to find ways to circumnavigate across the network.

2.2 Reinforcement learning

Machine Learning(ML) is a set of algorithms that can learn and improve its performance via experience [12]. In general an ML algorithm has a hypothesis function h and tries to map the input to the output and adjust its hypothesized function to an ideal function f [13].

ML is classified into three forms of learning: supervised learning, unsupervised learning, and Reinforcement learning. In supervised learning, there is another system that is responsible for telling the agent what the right output is supposed to be, for any given input. In unsupervised learning, the system is fed with just the input and

the task of the agent to find a meaningful explanation to the data given. The third type of learning falls between the first and second, and is called reinforcement learning (RL)[14]. We model our intruder based on this method.

In RL, an agent tries to find the right action in an environment that would increase its cumulative reward. The environment is unknown to the agent at the start and it learns to behave as it gains experience in this world. Every RL problem can be formulated as a Markov decision process (MDP) with states representing the state of environment, and the transition of states depend on agents' actions [15].

An RL model consists of a set of states S, actions a, rewards r, and a policy π. Here, an agent tries to learn an optimal policy π^* that maximizes its cumulative payoff. To learn this policy the agent starts from a state S_t and either picks actions a_t randomly or picks them based on its current policy π. It ends in a state S_{t+1} to get a reward r_t. This reward is used to update the policy π. This process is repeated until policy π converges to a perfect policy π^*.

There is a model-free RL called Q-Learning [16]. Here each state-action pair is associated with a value called the Q-value. A table of such Q-values is used to arrive at an optimal policy. $Q(S_t, a_t)^\pi$ is the Q-value of the agent at state S_t, picking an action a_t, using the policy π. The Q-value is a function of reward and is updated after every action. These updates occurring at every iteration are called episodes.

$$Q(S_t, a_t) = Q(S_t, a_t) + \alpha * (r_{t+1} + \gamma * (maxQ(S_{t+1}, a) - Q(S_t, a_t))) \tag{2.1}$$

In the equation above, α is called the learning rate, and γ is called the discount factor [17, 18].

The value of γ represents how much the previous reward is supposed to contribute, and the value of α tells us how much effect this reward will have in the future.

The value for these two parameters ranges from 0 to 1. When $\alpha = 1$ the Q-value is set based on just the current reward, and if $\alpha = 0$, no learning happens. When $\gamma = 0$, past events are ignored, and if $\gamma = 1$, agent strives for longterm high reward. Hence, these parameter values have to be carefully chosen.

In Q-learning, there are generally two modes of operation: explore and exploit. When an agent explores, it tries to pick a random action and update the policy. When it exploits, it picks actions purely based on its policy, like an MDP. For complete convergence, an agent should have visited every state and picked up every action infinite number of times. But, for most practical purposes,(especially in a small world problem with fairly limited states) random exploration works well. There are different types of exploration technique used, the most popular ones being the Epsilon greedy, Boltzmann, and Metropolis criterion [19].

RL algorithms are very versatile and can be used in many different areas. A very useful property of RL is that it can perform on-line learning and keep improving its policy as it encounters more scenarios. The use of RL in an intruder will help the intruder understand the environment better as it is rewarded and punished based on its actions. The attacker could simply learn more by simulation rather than actually entering a network to learn. But, the most crucial part of using RL is to pick the right state variables that represent the environment precisely, as the states determine what the intruder learns.

2.3 Related work

There has been previous work done on building intruder detection systems. A lot of the work focuses on improving and enhancing node coverage. But improvement in node coverage is not a very good solution, given that the network is going to be deployed in country borders. Country borders are generally very vast, and most of

the deployment is done by dropping sensors from an aerial vehicle. This causes a lot of randomness and hence makes it very hard to improve node coverage. Some lines of research focus on using multiple sensors and co-ordinating them to detect intruders. Doing this can improve detection, but they also increase miss rate or false alarms. This is because a lot of co-ordination is required and each set of sensors will raise false alarms at different time periods thus decreasing its reliability. Some research has been done in building intelligent agents and using them for intrusion detection. But these mostly focus on indoor environments and this can not be compared with outdoor environments like country borders where the possibility of false a signal is much higher. We have discussed a few papers below that focus on intrusion detection or building efficient intruders to test the detection system.

In a research by Deng et al [20] the authors tried to find the base station's location by using time-correlation attack. The attacker tries to find the frequency in which the node sends and receives packets. The nodes closer to a base station are more likely to receive or send a packet. This helps the attacker find the location of the base station. Some defense mechanisms suggested for such an attack were to introduce redundancy and randomized routes to confuse the attacker.

In a paper by Saipulla et al [21], the authors discuss about how the sensor deployment strategy could affect the node coverage. They take Poisson point process to be their base model and show that using a more controlled approach on sensor deployment could increase the node coverage. They reduce the random offset below the sensing range of a sensor. They show in their results that, by using such strategies on single or multiple line based deployment, the node coverage could be improved. But in their results it is seen that, as the physical area being sensed increases, the probability of full node coverage decreases exponentially down to almost 0.

In a research by Liu et al on approximate coverage in WSN [4], the authors discuss the full coverage problem. They show that full coverage in a WSN is an NP hard problem. They come up with a heuristic approach to partial, yet maximal coverage. Their heuristic approach also focuses on increasing the network lifetime, by turning sensors on and off and using limited sensors at a given time.

In a paper by Yuan et al on intrusion detection system [22], the authors build a system that used unsupervised fuzzy adaptive resonance theory based neural network to learn about and detect intruders in unknown environments. They used a combination of WSN and a mobile robot. A hierarchical learning structure is used for the system, with the robot as the root. A Markov model is used to learn time series and time related changes in the environment. They show that their system performs better than a previous but similar approach [23]. In the paper that they compare their results to [23], they use unsupervised learning techniques to distinguish between good and faulty sensors, and also detect unusual patterns and try to improve security alerts. They measure performance by computing miss rate, false alarm, sensitivity, and specificity. Their implementation however, lacked an intelligent intruder and the intruder did not have any intention of staying undetected. Also, this system was deployed in an indoor environment where the sensor data is more predictable and reliable.

In a paper by Zhi Sun et al [24], the authors come up with a strategy for intrusion detection that uses multiple technologies. Their hybrid approach involves the use of multiple types of sensors: underground sensors, on-ground sensors, camera towers, aerial and ground autonomous robots. In this network, all of the above communicate and coordinate to detect an intrusion. The ground sensors send a signal to cameras when they detect an intrusion, this is followed by the movement of robots and it ends with the confirmation of an intrusion. In their approach, the ground sensors are much

higher in number than the cameras, hence, cameras could be overloaded with signals which further lead to an increase in miss rates and false alarms.

Dejmal et al have modeled an attacker based on RL, in vulnerability assessment of peer to peer networks [25]. Here, the authors focus on problem of proactive security in DoS attack, in P2P network. They then model their problem's solution using a gradient descent method in RL. They also show that their method outperforms a few other heuristic approaches.

Pranav et al used RL to model their attacker to study intrusion detection in a sensor monitored area [7]. The authors make their intruder agent move in a grid world with sensors and the sentry tries to catch the intruder. As the intruder moves in the map, it learns about the map and uses that to decide whether to head towards goal or to retreat. Even though the agent is learning about the map and using that information, if the same agent were placed in another map with sensors placed in different locations, the agent will have to relearn everything in the new map. Hence, the approach is applicable to a specific map only, at any given point in time.

In a research by Servin et al, the authors have used a multi-agent RL to build an intrusion detection system for a distributed sensor network [26]. Here, they use multiple agents to work towards a common network intrusion detection goal. They use two sets of agents called Sensor agents and Decision agents. Sensor agent tries to find the state of environment partially and the Decision agent learns to interpret Sensor agent's signals to find an abnormal state. Their results show that these agents detect intrusion more successfully, with increase in the amount of training.

CHAPTER 3

SYSTEM MODEL

In this chapter we explain our proposed model for a physical intruder and the way we have modeled the entire system.

3.1 Modeling world

Country borders where wireless sensors are deployed, are vast and have different terrain. But, for the sake of simplicity, we consider our world to be an nxm grid world. Every block/grid in this world can be located using an i, and j value in the x, and y axes. This entire grid world represents the border which will be monitored for intrusion. Sensors are placed randomly in this grid world. The sensing coverage of each sensor is just one grid, i.e, if a sensor is at location (i, j), it can sense the presence of an intruder in only that block. Even though the sensing range is one grid, we assume that it can successfully communicate information required to the base station. The number of sensors in a map can be varied based on the node coverage desired. We measure node coverage as,

$$Nodecoverage(\%) = \left(\frac{N_s}{N_g}\right) * 100 \tag{3.1}$$

$$P_0(sensor) = \left(\frac{N_s}{N_g}\right) \tag{3.2}$$

As our model is a grid world, to propagate through the world, the sentry and in-

Table 3.1. Parameters used to compute node coverage

N_s	Number of sensors in the grid world.
N_g	Number of grids in the world.

truder, both are allowed to pick only one of the four directions possible:up, down, right, or left. Fig. 3.1 represents a $6x6$ grid world with eight sensors, one sentry, and an intruder.

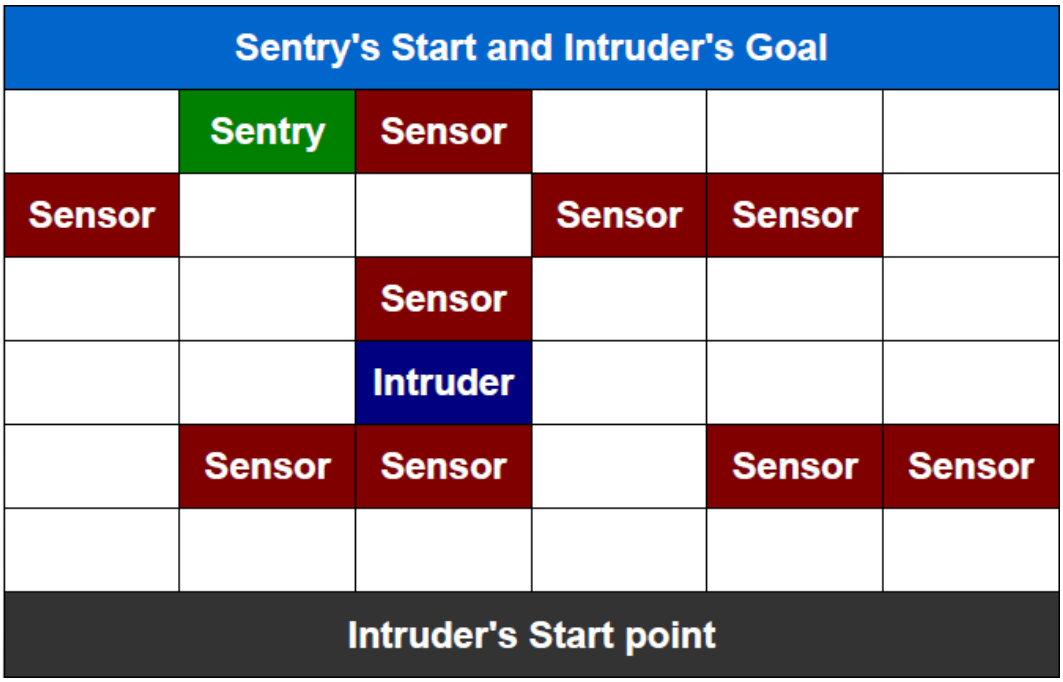

Figure 3.1. A simple grid world.

As described in the introduction section, the types of intruders we address here are mostly those that work in groups. Hence, there are multiple intruders with the same motive. In our model we assume that there is an infinite set of intruders in the group waiting at the start point. Hence, getting captured by the sentry is not the end of our simulation. Each intruder after getting captured by the sentry is assumed

to have already passed on the information he has gained about the map to the next intruder who would then continue exploring. Losing a man could be very expensive for a group with small number of people and very low for a group with large number of people. The details about the expense of losing an intruder for the group is explained later in this section as a function of the reward awarded when captured. Details on how the system is reset is explained in the sentry model.

3.2 Sentry Model

A sentry is similar to an officer on patrol duty at the border. The job of a sentry is to protect the place from unauthorized border crossing. Sentry monitors the network for activities. As a sensor gets triggered in the network, the sentry goes to the place and tries to find and capture the intruder. When a sensor is triggered in the network, our sentry gets to know the exact location of the sensor that was triggered. As shown in the Fig. 3.1, our sentry starts from the top of the map which is also the intruder's destination.

The sentry after receiving a signal from a sensor starts moving towards that sensor using the shortest path. The shortest path distance, is basically the Manhattan distance from its present location to the sensor's location. We assume that, the sensing radius of the sensor is less than the sentry's viewing radius. Based on this, our sentry can spot an intruder x-blocks away from his location. We also assume that the sentry is on some sort of a vehicle hence, can move through multiple blocks/grids in one unit time. For example a sentry could pick up, up, right, up when the intruder could do just up, or down, or right, or left. Once the sentry reaches the grid where sensor detected intrusion and did not find any intruder, it starts moving in random direction around the grid trying to find the intruder. If it spots the intruder at any point and if it can reach to it before intruder escapes, then we say that the intruder

was captured and the sentry goes back to its start with captured intruder. Just like in the real world, a captured intruder means one man caught and not the end of intrusion. In our model, after an intruder is captured, the intruder is reset to his start position and our sentry is sent back to his start position. During sentry's random walk or when moving towards a sensed region, if sentry receives another signal from a different sensor, the sentry starts moving towards that location. The sentry will keep looking for an intruder after reaching sensed area with a probability. We calculate the probability to be,

$$Probability\ of\ intruders\ existence\ :\ P_{IE} = \frac{No_{IS}}{(No_{IS} + T_{LS})} \tag{3.3}$$

Table 3.2. Parameters for calculating P_{IE}

No_{IS}	Number of times intruder was sensed.
T_{LS}	Number of steps since intruder was sensed.

This probability is set to zero when we see that the value of P_{IE} goes below a certain threshold β. Value of zero means that the sentry has to go back to its start position. Algorithm 1 explains the behavior of our sentry's code.

Algorithm 1 Sentry's response

Require: $P_{IE} \neq 0$

```
1: if !reached sensed node then
2:    %% Note:n depends on speed of sentry. %%
3:    take_step(n, sensedNode);
4: else
5:    %% Note:β is a threshold set by user. %%
6:    if P_IE > β then
7:       take_step(n, random);
8:    else
9:       go_to(base);
10:      if Reached base then
11:         P_IE ⇐ 0
12:      end if
13:   end if
14: end if
```

3.3 Modeling Intruder

Our intruder is a location aware attacker. The purpose of the intrusion is to find a detection free path, from one end to the other end of the border. We restrict our intruder to be able to move one grid in a unit time, unlike our sentry. The intruder picks either up, down, right, or left to move in the map. As the intruder moves, he can figure out his exact location in the map. As he enters a grid, he gets to know if that block has a sensor or not. The intruder has a memory and can keep track of where he has and has not been to. The intruder also keeps track of the grids in which it encountered sensors. Just like the sentry, the intruder can see x-blocks around

him and can spot a sentry within this radius. Being able to see the sentry gives the intruder an opportunity to escape after being spotted. However, the intruder is at a disadvantage, as the sentry is generally much faster than the intruder. The intruder at every point takes one step in some direction with a goal of being able to find a detection free path. He continues to move in the map till he finds one such path or gets caught. If he gets caught he is reset to his start location. But he has the knowledge that was gained so far. The intruder keeps track of the map by marking each grid with a $confidence$ value. This value is the probability of a sensor's existence in the grid. A no sensor block gets a $confidence$ value of zero, if the intruder has been to that block and a block with sensor gets a $confidence$ value of one, after the intruder has been there. The best possible route to reach to the other side is found by using a shortest path algorithm like, Dijkstra's algorithm. The way Dijkstra's algorithm has been applied is by considering each block as node and the edge between two nodes has a weight equal to the inverse confidence value of the block that the edge leads to. The inverse confidence value to a block is

$$inv(confidence) = (confidence - 1) * (1 - \epsilon) \tag{3.4}$$

$$-1 > inv(confidence) \leq 0$$

The value for ϵ in the above equation is set to 0.000001 in our simulation. The value $(1 - \epsilon)$ should be as close to one as possible to get minimum information loss in the shortest path value computation. The value can be small and large depending on the size of the map. For example, if the $(1 - \epsilon)$ is set to .9 the value of shortest path for path with ten grids will be $-(0.9^9) = -0.3874$ even though the true value should be close to -1, and if we set the $(1 - \epsilon)$ to be equal to 1, then we wont be able to distinguish between a detection free long path and a detection free short path. Hence, it is important to pick a value that would be close to 1 to get the right precision.

3.3.1 Modified Dijkstra's Algorithm :

A small modification is done to Dijkstra's algorithm to find the shortest path. Instead of adding the weights ($inv(confidence)$), we multiply the weights and make sure the resultant value is kept negative. We make sure that all the weights are always less than or equal to zero and greater than negative one. The best route is the route with smallest multiplicative weight. This means that the shortest path is actually a value that tells us the probability of existence of a sensor in the path. The probability of existence of a sensor in the path is the shortest path value plus 1. If this shortest path at any point is zero, that means there is no detection-free path that exists in this map. And if the shortest path is less than a threshold (a value close to negative one, we set this threshold as -0.9), then the path is said to be detection-free. If one of the grid in the shortest path has a sensor then the value of the shortest path becomes zero, as the weight gets multiplied with this zero. This makes it easy to find the non-existence of a detection-free path.

As we need to loop through the nodes just like in original Dijkstra's algorithm, the time and space complexity of this algorithm still remains the same. Algorithm 2 is the algorithm for our modified Dijkstra's. It can be noted that everything is the same as original Dijkstra's[1], except line number 14.

[1]http://en.wikipedia.org/wiki/Dijkstra's_algorithm

Algorithm 2 Modified Dijkstra's

```
1: for each vertex v in Graph do
2:     dist[v] ⇐ infinity ;
3:     previous[v] ⇐ undefined ;
4: end for
5: dist[source] ⇐ 0 ;
6: Q ⇐ the set of all nodes in Graph ;
7: while Q is not empty do
8:     u ⇐ vertex in Q with smallest distance in dist[] ;
9:     remove u from Q ;
10:    if dist[u] = infinity then
11:        break ;
12:    end if
13:    for each neighbor v of u do
14:        %% Make sure this value is negative after multiplication. And also make
       sure that all weights are negative or 0.%%
15:        alt ⇐ dist[u]*dist_between(u, v)*-1 ;
16:        if  alt < dist[v] then
17:            dist[v] ⇐ alt ;
18:            previous[v] ⇐ u ;
19:            decrease-key v in Q;
20:        end if
21:    end for
22: end while
23: return dist;
```

3.3.2 Agents

The main focus of our work is to build an intruder who would match up to a real world smart intruder (a human trying to cross border). In the process of achieving this, we have modeled the attacker to be able to learn and adapt based on his knowledge. Our intruder model consists of four parts. There are three specialized agents:Explore, Exploit and Evade. There is one intelligent agent that uses Q-learning to learn the environment and best possible response:Policy.

3.3.2.1 Explore agent

As an intruder enters a map, he knows nothing about the map. Hence, the confidence value for every grid is the same($P_0(sensor)$). Therefore it is essential for the intruder to explore the map in the right way. To be able to find a detection-free path, it is essential for the intruder to have explored enough in the map. Hence, the job of the explore agent is to be able to pick the best direction for exploration that would help the intruder expand the knowledge he has about the map. Picking the right direction for exploration is slightly tricky. The confidences that the intruder maintains tells the intruder the probability of a sensor's existence in that block. Choosing an action to explore a grid will not just affect the system's current state, but also on how fast or slow a detection-free path will be found. Hence, the intruder should not just choose a direction based on its adjacent grid's exploration level, but also consider a greater area before deciding the direction. The direction for exploration has to be picked in such a way that the intruder gains more knowledge about the map in less time. Hence, an area with high or low confidence is not the right pick for exploration. And another thing to keep in mind is that it should not run into a known sensor while exploring.

To take care of exploration we introduce a parameter called Exp_{cd}(explore confidence of a direction) that will help us decide which direction to explore in. This value is calculated by:

$$AC_c = \frac{\sum_{reachable\_direction} C_{blk}}{number(reachable\_direction)} * 8 \tag{3.5}$$

$$C_{cd} = \sum_{M_d<n} \frac{|C_{blk} - P_0(sensor)|}{M_d} \tag{3.6}$$

Table 3.3. Parameters used to compute explore values

M_d	Manhattan distance.
AC_c	Average confidence around current block.
C_d	Sensor existence in the grid next to current grid in direction d.
C_{cd}	Weighted average confidence in particular direction d.
C_{blk}	Confidence of the block.
n	The number of grids from current block.
$reachable\_direction$	Directions in which agent can move and end up in a different grid in one step.

$$Exp_{cd} = C_{cd} + C_d \tag{3.7}$$

$$C_d = \left\{ \begin{array}{ll} max\_value & \text{confidence of grid immediately next to it is 1} \\ 0 & \text{otherwise} \end{array} \right\} \tag{3.8}$$

n is a value that is used to determine what radius we want to look before making a decision. Algorithm 3 explains the working of an explore agent.

Algorithm 3 Explore agent

1: $AC_c = \frac{\sum_{reachable\_direction} C_{blk}}{number(reachable\_direction)}$;

2: for all four direction $C_{cd} \Leftarrow \sum_{M_d<n} \frac{|C_{blk} - P_0(sensor)|}{M_d}$;

3: for all four direciton$Exp_{cd} \Leftarrow C_{cd} + C_d$;

4: direction_to_move $\Leftarrow$ min(all four direction's Exp_{cd});

3.3.2.2 Exploit agent

This is a simple agent. Once intruder has explored enough, he needs to move forward towards the goal and be able to cross border using a detection-free path. The exploit agent is specialized to do this. Based on the confidences known, the exploit pick the direction that is most detection free. The main component of this agent is to find the shortest path from the current position. The way in which this is computed is the same as discussed earlier in modified Dijkstra's section. At every stage this agent finds shortest path(Sp_{ex}) that would lead to the goal. We introduce another parameter called $Expt_{cd}$(exploit confidence of a direction) to find exploitation level that will be used by policy agent(discussed later in this section).

$$Expt_{cd} = integer\left(\frac{\log(|Sp_{ex}|)}{\log(|inv(P_0(sensor))|)}\right) \tag{3.9}$$

Algorithm 4 explains the working of an exploit agent.

Algorithm 4 Exploit agent

1: $Sp_{ex} \Leftarrow$ (Double Path_length,Array Shortest_Path);

2: $Sp_{ex} \Leftarrow$ ModifiedDijkstra's(map, curr_grid, goal);%%3 parameters for Dijkstra's: map, start point and destination.%%

3: $Expt_{cd} \Leftarrow integer\left(\frac{\log(|Sp_{ex}.Path\_length|)}{\log(|inv(P_0(sensor))|)}\right)$;

4: next_grid $\Leftarrow$ pop(Sp_{ex}.Shortest_Path);

5: next_move $\Leftarrow$ direction_towards(next_grid);

6: **if** Sp_{ex}.Path_length == 0 **then** %% the best path has sensor %%

7: next_move $\Leftarrow$ up; %% move up %%

8: **end if**

9: move(next_move) ;

3.3.2.3 Evade agent

Sometimes when the intruder has hit sensors way too many times, it is best to retreat. This agent is designed to do exactly that. The goal of this agent is to find a path that will take the intruder safely back to the start point. The purpose of this agent might be different from that of exploit, but the functionality is exactly the same as the exploit. The only difference between evade and exploit agent is that the goal state for the evade is the start point. Algorithm 5 explains how the evade agent works.

Algorithm 5 Evade agent

1: Sp_{ev} ⇐ (Double Path_length,Array Shortest_Path) ;
2: Sp_{ev} ⇐ ModifiedDijkstra's(map,curr_grid, start);%%3 parameters for Dijkstra's: map, start point and destination.%%
3: next_grid ⇐ pop(Sp_{ev}.Shortest_Path);
4: next_move ⇐ direction_towards(next_grid);
5: **if** Sp_{ev}.Path_length == 0 **then**%% the best path has sensor %%
6: next_move ⇐ down;%% move down %%
7: **end if**
8: move(next_move);

3.3.2.4 Policy agent

This is one of the key component of our intruder model. This is a learning agent and at any given point in time is responsible to decide which agent to deploy (Explore, Evade or Exploit). This decision is taken based on the current state of the intruder. We have modeled this agent using Q-learning. The agent trains himself over different maps, and learns the best agent to deploy, given its current state.

As mentioned Explore, Exploit, and Evade are the 3 actions that the policy agent can pick. This is like the brain of the intruder. To be able to learn, the policy agent should be able to understand the environment. For example, the policy agent should know if the intruder just entered the map and has no idea about the map, or if it has hit a sensor and sentry is nearing, etc. To make the agent understand the exact state of the intruder, we need to choose the right parameters to be a part of the agent's state. As the location of the sensors keep changing in every map, having the x and y location of the intruder as a part of its state would not be very useful.

Hence, we need a slightly more abstract way of representing the state so that we can re-use the information gained in one map on another. We have modeled the following to represent the state of our intruder:

- *Minimum Exploration Level:* This parameter tells the agent about the area that is least explored around it. This value is the same as the minimum of Exp_{cd} in the explore agent. Having this as a part of its state will give the agent an idea on whether exploring at this moment is a good option or not. This state value is discretized from zero to four.
- *Distance from goal to intruder:* This value is same as $Expt_{cd}$ computed by the exploit agent. This helps the agent understand how good the current best detection free path is. As long as there is a detection free path, the physical distance from the goal does not matter. Hence a distance based on confidences makes sense to be a part of the state.
- *Average Exploration Level:* This helps the agent understand how well explored the entire area is. This value is the average confidence of, all the blocks around current grid. This along with the first parameter will make it easier to understand the exploration level of the place around the intruder. This state value is discretized from zero to seven.
- *Sentry's presence:* This indicates the policy agent if the intruder was spotted or not by the sentry. Every time the agent is caught this value is true hence making the agent understand that entering a state with this value set is a bad thing.
- *Time since last damage:* This value is an indicator of how long it has been since the intruder ran into a sensor. This helps in making the decision of whether to evade or not. The value is reset to zero after hundred time detection-free steps.

- *Damage times:* This value is a count of how many times intruder was detected by sensors so far. The larger this value is, more likely that intruder is going to get caught. The upper limit to this number is ten. If the agent gets caught more than ten times, the policy will still see it as ten. This value could be bigger or smaller depending on the size of the map. As we increase this number the states in which agent can be in increases by a factor equal to the combination of all the other state variables. This will increase the time to take to learn as well. In our simulation on 10x10 map, the average number of times agent runs into sensor was less than ten. Hence we limit this number to ten.

With the above mentioned parameters the agent understands state of the intruder. As the intruder moves, the agent learns by getting rewards from the environment. The reward is both positive and negative. A positive reward indicates that, the action that is took was a good response given the state and a negative reward indicates the opposite. Following are the sets of rewards we use in our model:

- *-1:* for picking any step. This ensures that the agent tries to get to the goal as quickly as possible.
- *-50 :* for getting caught by the sentry. This value could be increased and decreased based on how expensive it is to lose an intruder.
- *+100 :* for finding a detection free path.
- *+20 :* for improving a path, ie. for taking an action that reduced the exploit's shortest path value. This will help the agent learn little faster.
- *-5 :* for picking an evade action. This helps the agent learn slightly faster by making the agent avoid certain scenarios where the agent's best action keeps the intruder stuck between two states after partially learning the Q-values.

These rewards are used to compute a Q-value for the current state. The Q-value for a state action pair is calculated by:

$$Q_{t+1}(S_t, A_t) = (1 - \alpha) * Q_t(S_t, A_t) + \alpha * (R_{t+1} + \gamma * (max(Q_t(S_{t+1}, A))) \quad (3.10)$$

Table 3.4. Parameters used to compute Q-value

Q_t	Q-value at time t.
S_t	State at time t.
A_t	Action taken at time t.
α	Learning rate.
γ	Discount factor.
R_{t+1}	Reward obtained at time t+1.
N_{vis}	Number of times you visited this state.

$$\alpha = 1/N_{vis},\ \gamma = 1 \quad (3.11)$$

The agent uses softmax/Boltzmann exploration as its explore exploit policy. The equation for softmax is :

$$P(A_t|S_t) = \frac{e^{Q(S_t,A_t)/T}}{\sum_{A=0}^{n} e^{Q(S_t,A)/T}} \quad (3.12)$$

Table 3.5. Parameters used to compute Softmax equation and Temperature

$P(A_t\|S_t)$	Probability of picking an action at time t given the state at time t.
T(Temperature)	The value that determines whether to explore or exploit.
δ	A constant to determine how quickly T should start dropping.
$visit(S_t)$	Number of times agent has visited state S_t so far.
κ	A large number.

High T means exploration and low value means exploitation. T is computed using the equation below :

$$T = e^{-\delta * visit(S_t)} * \kappa + 1 \tag{3.13}$$

Algorithm 6 and 7 explains the working of an policy agent in learning and evaluation mode.

Algorithm 6 Policy agent: Learning Mode

1: $i \Leftarrow 0$
2: **while** $i < N_l$ && not found detection-free path **do**
3: $S_t \Leftarrow$ state($Expt_{cd}$,Exp_{cd},AC_c,damage_count,time_since_damage,sentry_visible);
4: $T \Leftarrow e^{-\delta * visit(S_t)} * \kappa + 1$;
5: $P(A_t|S_t) \Leftarrow \frac{e^{Q(S_t,A_t)/T}}{\sum_{A=0}^{n} e^{Q(S_t,A)/T}}$;
6: action $\Leftarrow$ random();
7: $A_t \Leftarrow$ first action $< P(A_t|S_t)$;
8: $R_{t+1} \Leftarrow$ Take_action(A_t);
9: Compute $Expt_{cd}$,Exp_{cd},AC_c,damage_count,time_since_damage,sentry_visible;
10: $S_{t+1} \Leftarrow$ state($Expt_{cd}$,Exp_{cd},AC_c,damage_count,time_since_damage,sentry_visible);
11: $Q(S_t, A_t) \Leftarrow Q(S_t, A_t) + \alpha * (R_t + 1 + \gamma * (maxQ(S_{t+1}, A)Q(S_t, A_t)))$;
12: $S_t \Leftarrow S_{t+1}$
13: $i++$;
14: $visit(S_{t+1})++$;
15: **end while**

The agent is trained on multiple maps by picking random action from different states using softmax(Boltzmann exploration) and the Q-value is updated. As the

Algorithm 7 Policy agent: Evaluation Mode

1: **while** $i < N_e$ && not found detection-free path **do**

2: $S_t \Leftarrow$ state($Expt_{cd}$,Exp_{cd},AC_c,damage_count,time_since_damage,sentry_visible);

3: $T \Leftarrow 1$;

4: $P(A_t|S_t) \Leftarrow \frac{e^{Q(S_t,A_t)/T}}{\sum_{A=0}^{n} e^{Q(S_t,A)/T}}$;

5: action $\Leftarrow$ random();

6: $A_t \Leftarrow$ action $< P(A_t|S_t)$;

7: Take_action(A_t);

8: Compute $Expt_{cd}$,Exp_{cd},AC_c,damage_count,time_since_damage,sentry_visible;

9: $S_t \Leftarrow$ state($Expt_{cd}$,Exp_{cd},AC_c,damage_count,time_since_damage,sentry_visible);

10: $i++$;

11: **end while**

agent trains on more and more maps, he starts picking actions which will lead the agent in finding a detection free path along with minimizing the risk of getting caught.

As this is a learning agent, we run it in two modes:training and evaluation. In training mode we try action based on the Q-value and T of the current state. And the rewards obtained are used in updating the Q-values. But in evaluation mode we choose actions strictly based on Q-value. This is done by setting the temperature value (T) in the Boltzmann's equation to one. We do not update the Q-values in this mode.

3.4 Simulation

The simulation starts with the intruder at the bottom of the map in a random location. The sentry is at the top. The sensors are spread randomly throughout the grid world. The number of sensors and grid size can be varied. The confidence value

of all the grids are initialized with a value equal to the $P_0(sensor)$. The intruder takes one step into the map.

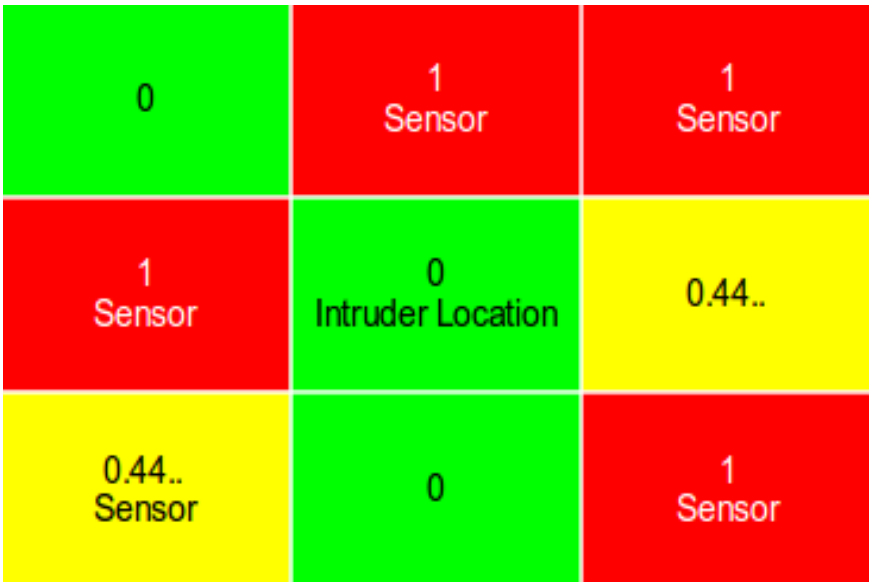

Figure 3.2. A grid world with confidence value.

The policy chooses which agent to deploy in order to find the direction to move. The agent that is picked decides the best direction to move. After a step is taken, the Policy agent is rewarded based on the outcome of the step. And after the step the confidence of a grid is reduced to zero if the grid does not have a sensor and it is increased to one if there was a sensor. When a sensor detects an intruder, the sentry starts looking for the intruder by taking a step. If sentry catches the intruder, the intruder is reset to start from bottom again. The knowledge of the new intruder is equal to the knowledge that the captured intruder had (ie. the Q-values and the information about the map is not reset, only the state of the agent is reset along with sentry's location). This goes on till the intruder finds a detection free path, or if it is found that there is no detection free path, or when maximum number of steps is

reached(this is an upper limit set in order to ensure that the agent does not take a lot of time on a single map). The upper limit on number of steps on evaluation mode is set based the maximum number of steps it took for agent to find a detection free path on a hundred maps by picking random actions. The upper limit on learning mode is set to the average number of steps instead of maximum. The Q-value of policy is updated after every step only in training mode. Fig. 3.2 shows confidence of few grids. Red indicates a sensor, yellow indicated unexplored zone (may or may not be a sensor), red indicates encountered sensors.

CHAPTER 4

RESULTS AND DISCUSSION

We ran different simulations with different parameter settings to analyze the impact of a trained intruder. Different parameters in the simulation change how accurately and quickly the agent learns. To test and understand the agent, we tried different parameter settings on 10X10 maps. Table 4.1 shows the amount of time and memory consumed by different sized maps. On each map, it takes a certain number of steps before the agent finds a detection free path(Figure 4.1). For the agent to learn all the Q-values, it has to learn on thousands of maps. Considering the amount of time it would take to finish, based on the amount of time it takes for a step obtained from Table 4.1 and the average number of steps required to find a detection free path in a map obtained from Figure 4.1, we restrict the number of maps the agent learns on, to 250,000 maps. And to see the progress after every 50,000 maps, we run the agent in evaluation mode for a hundred maps. We noted that to finish one such simulation on a 10X10 it takes about 24 to 48 hours which is close the the value computed using the table. Even though restricting the number of maps to learn on will restrict the learning of Q-values, we noted that the difference in the capture count and the number of steps it takes to find a detection free path does not significantly improve in most cases after 250,000 maps on a 10X10 map.

Table 4.1. Time and Space required for different map size

Map Size	Time to take 1000 steps(hrs)	Memory required (GB)
5X5	0.000168	0.231
10X10	0.000609	0.231
20X20	0.00185	0.403
30X30	0.00720	0.404
40X40	0.0214	0.50
50X50	0.0521	0.63
60X60	0.108	0.71

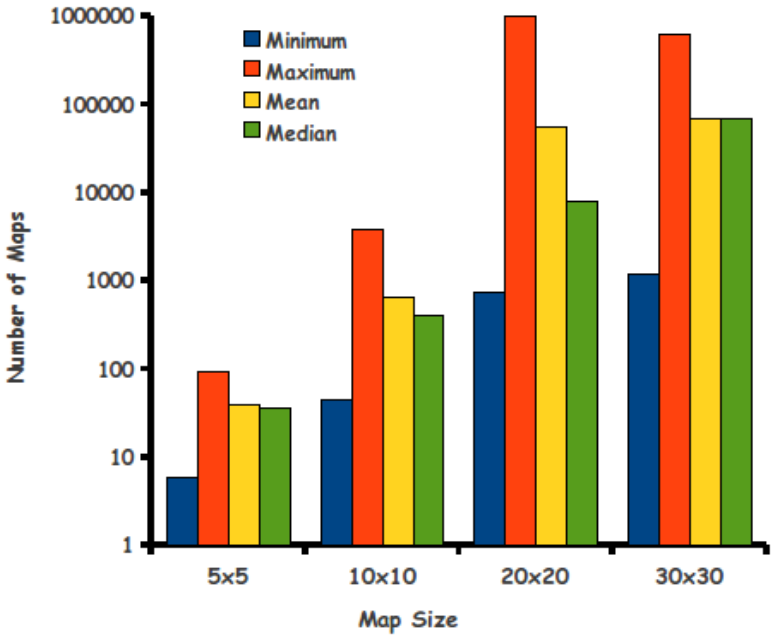

Figure 4.1. Average number of steps to find detection free path with 20% sensor coverage.

Based on our intuition about the problem and behavior of our agent, we considered the following factors as the most significant in deciding the effectiveness of learning.

1. *Sensor Coverage:* Rate of capture is intuitively a function of sensor coverage. A low coverage means, the probability of agent getting sensed is low and hence, the probability of getting captured by sentry also reduces. Similarly, high density means large probability of getting captured by the sentry and low probability of existence of a detection free path. Hence intuitively we can see that the effect of learning is expected to be low for both extreme cases. This means that that we would expect significant improvement of the intruder's behavior after learning on maps with average sensor coverage. To see this, we ran our agents on maps with sensor coverage ranging from 10-60%.
2. *Effectiveness of Sentry:* Effectiveness of a sentry if determined by how effectively the sentry can capture an intruder after detection. This is controlled by two factors in the sentry, first the number of grids he can see(vision), and secondly the number of grids he can move in one unit time(speed). Similar to sensor coverage rate of capture is also a function of the effectiveness of a sentry. A very effective sentry guarantees capture of the intruder, and a weak sentry can never capture the intruder. Hence both these extremes don't look like an interesting place to learn a lot for our agent, and similar to sensor coverage, a setting with an average sentry is where we would expect learning to have maximum effect. To see this, we tried running simulations with three different sets of vision and speed values.
3. *Reward:* The reward that the learner obtains determines what the agent learns, and how quickly he learns. To test the effect of a different value of reward for capture on learning, we used a low and a high penalty(-50 and -150) for capture. Other than penalty for capture, we also tried a low and high penalty for doing evade action(-1 and -5) for one setting. The reason for this is explained later in this section.

Table 4.2 shows the different parameters we tried.

Table 4.2. Parameter combinations

Sensor Coverage	Penalty For Capture	Effectiveness of Sentry(Speed, Vision)
10	-50	2, 4
20%	-150	4, 6
30%		6, 8
40%		
50%		
60%		

We see that it takes a lot of time to run the simulation. Hence, we restrict the number of steps

- Learning mode: the average number of steps it takes to find a detection free path by taking random actions sampled over a hundred maps.
- Evaluation mode: the maximum number of steps it takes to find a detection free path by taking random actions sampled over a hundred maps.

The reason we restrict the number of steps in evaluation mode is because we have noticed in some cases that the agent can get stuck between two states. This is possible as our model represents the world in an abstract level, and the evaluation mode is run with incompletely learned Q-values.

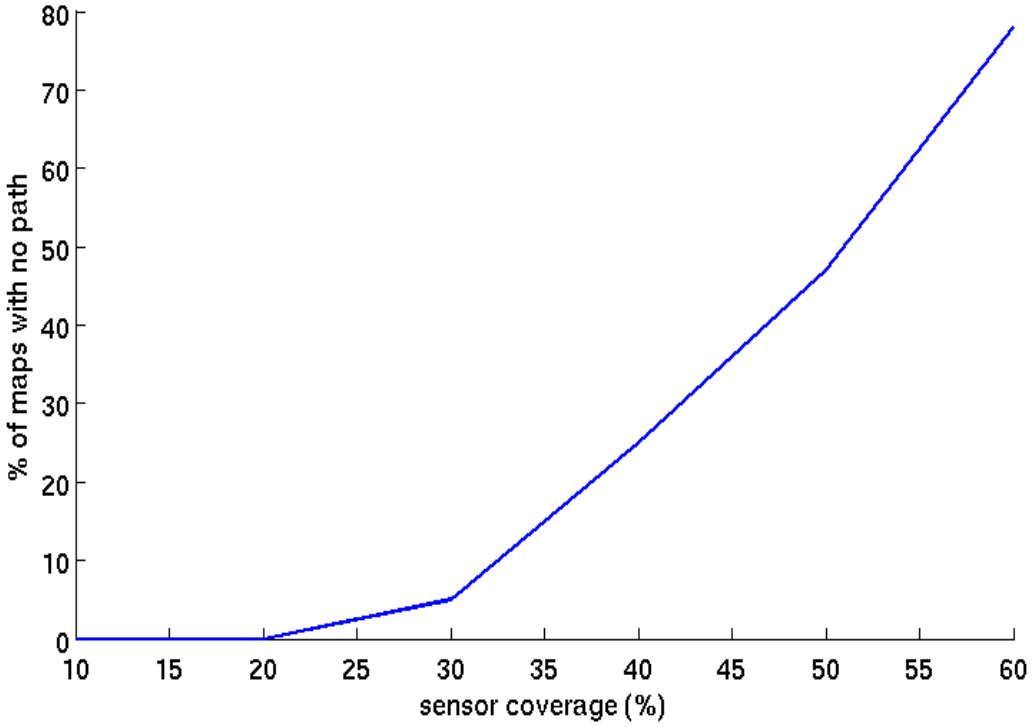

Figure 4.2. Percentage of maps in which there is no detection-free path.

Our primary focus is to reduce the capture rate and to reduce the time it takes to find a detection-free path in a map. Figure 4.2 shows the percentage of maps in which no detection-free path was found. This figure helps us understand the relation between sensor coverage and possibility of existence of a detection-free path. As expected, we see that as the sensor coverage increases, the probability of finding a path reduces. Before running the simulation to see the effect of learning in agents, we ran a simulation to see if our specialized agents do a better job compared to just randomly moving in the map. We noted a significant difference in capture count and number of steps to find the detection-free path. The average number of capture count was two for a certain setting when we used our agents randomly and for the same setting, picking up, down, right, or left randomly yielded an average capture count of

thirteen. And for the number of steps in a map, it was average 750 if we used agents and 4500 if we did not. After confirming the effectiveness of our specialized agents, we ran our actual simulation on 10x10. We did not find any significant difference in using a higher negative reward for getting captured over smaller negative reward. Hence, we show only graphs for simulations with lower negative reward for getting captured. Figure 4.3-4.4, we show the effect of learning on capture count and number of steps it takes to find a detection-free path, respectively, for setting with weak sentry (speed 2X, speed 4X). We see that the capture count does not increase significantly for increasing sensor coverage while performing random action, and the actual capture count is on lower side(on an average 1.5 for 30% sensor coverage). We also notice an improvement between random and learning on greater than 50,000 maps. This improvement is observed in both the capture count and number of steps. The capture count goes down from 1.5 on an average in 30% coverage when doing random actions to 0.7 after learning over 50,000 maps for the same. And number of steps goes down from 1000 on an average for 30% sensor coverage and performing random actions to 100 after learning on 50,000 maps. But the improvement in capture count and number of steps is not significant between learning on 50,000 and 250,000 maps. In Figure 4.5, 4.6, 4.7, 4.8 we show the capture count and number of steps to find detection-free path for setting with average(speed 4X, vision 6X) and very strong(speed 6X, vision 8x) sentry. We notice similar pattern as we did for weak sentry setting. But when compared to the weak sentry, we see that the number of times the intruder gets captured increases significantly(on an average to 3.7 in average and 4.4 in very strong sentry for 30% sensor coverage). As the number of capture count is higher than weak sentry setting, we see that the difference between random actions and learning over 50,000 maps to be higher as well. For a setting with average sentry and 30% sensor coverage, we see the capture count drop down from 3.7 on an average by

doing random actions to 1.8 on an average after learning on 50,000 maps, and we see the number of steps to drop down from 1200 on an average by performing random to 100 on an average after learning on 50,000 maps. The numbers look similar for the setting with very strong sentry as well. (Note: the plot of 3 lines in all figures is offset slightly to ensure clear visibility of the confidence intervals).

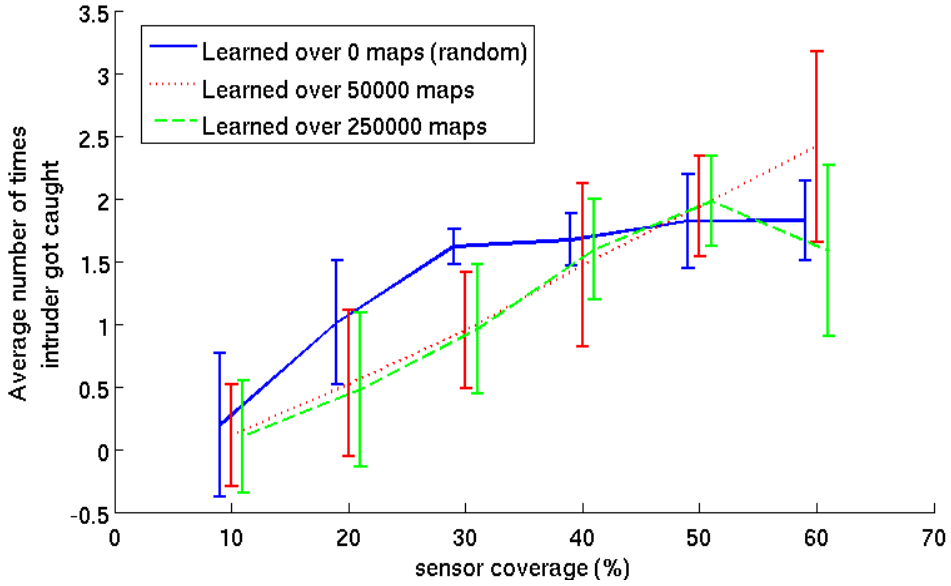

Figure 4.3. Capture Count for a setting with Sentry = (2 speed, 4 vision), reward = -5 for Evade and -50 for capture.

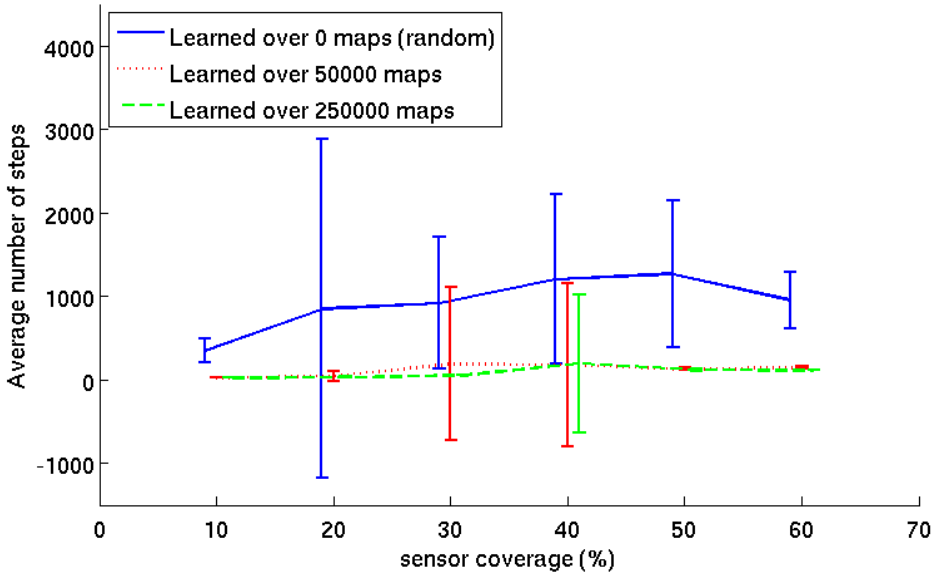

Figure 4.4. Number of Steps to find detection-free path for a setting with Sentry = (2 speed, 4 vision), reward = -5 for Evade and -50 for capture.

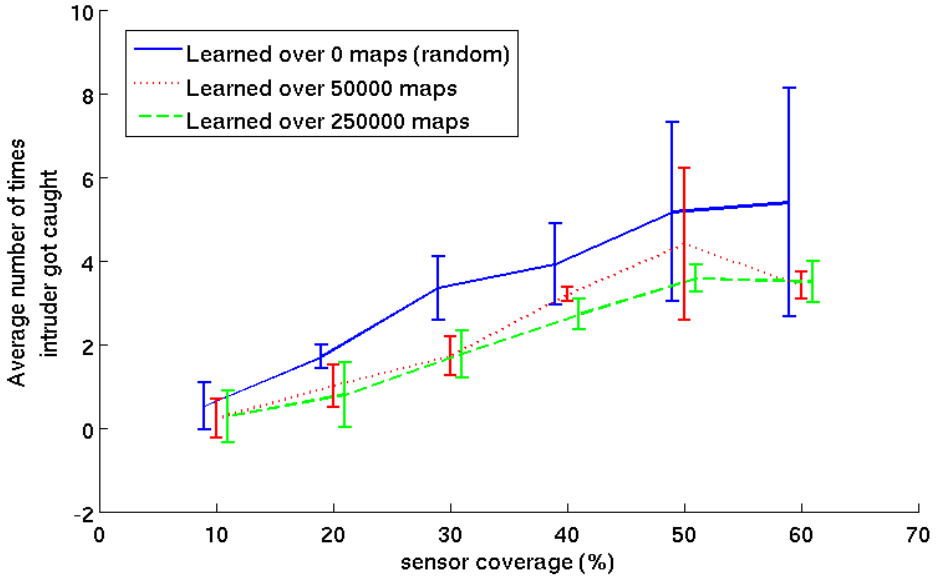

Figure 4.5. Capture Count for a setting with Sentry = (4 speed, 6 vision), reward = -5 for Evade and -50 for capture.

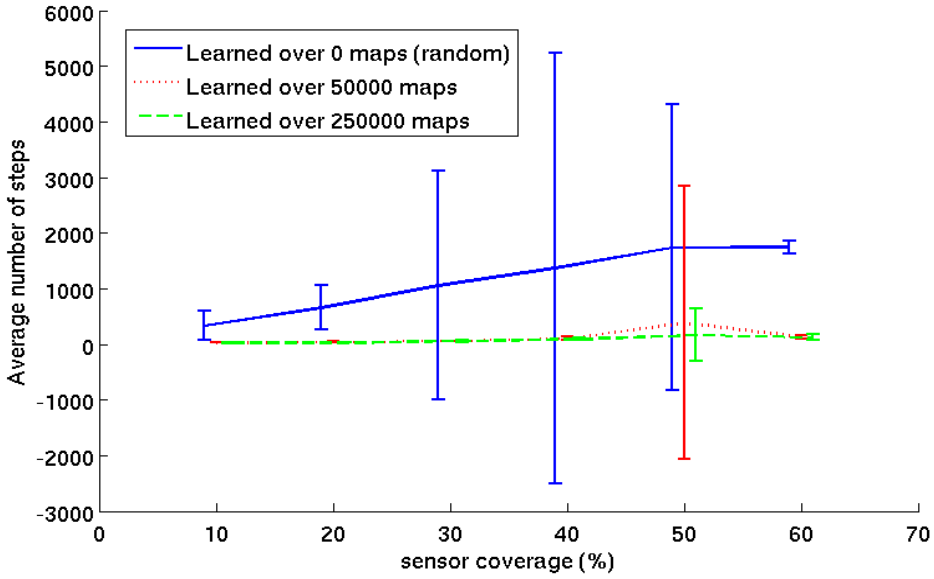

Figure 4.6. Number of Steps to find detection-free path for a setting with Sentry = (4 speed, 6 vision), reward = -5 for Evade and -50 for capture.

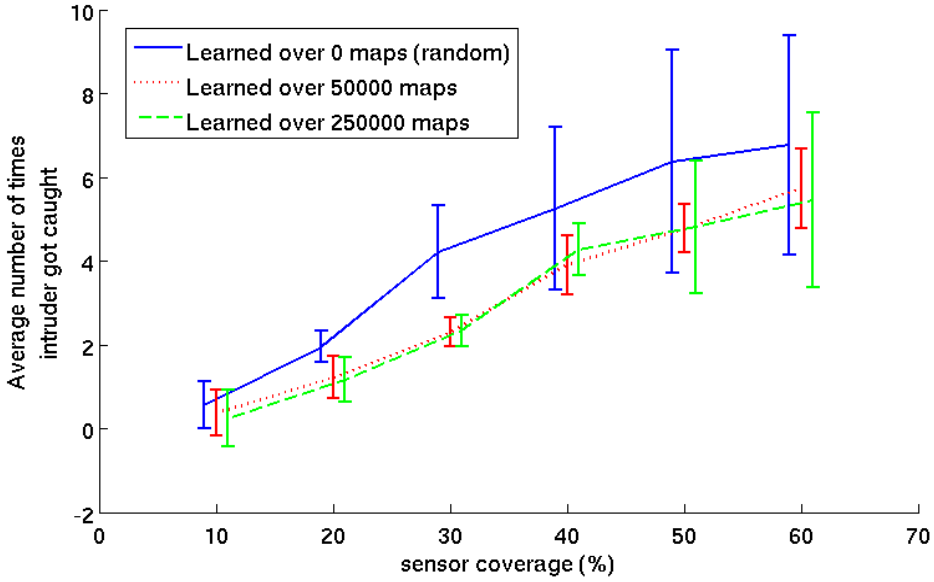

Figure 4.7. Capture Count for a setting with Sentry = (6 speed, 8 vision), reward = -5 for Evade and -50 for capture.

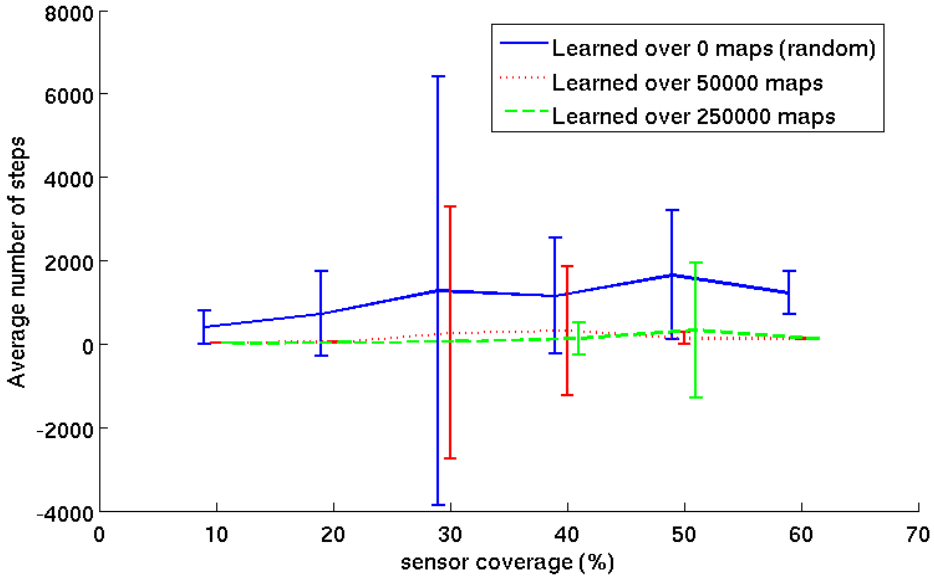

Figure 4.8. Number of Steps to find detection-free path for a setting with Sentry = (6 speed, 8 vision), reward = -5 for Evade and -50 for capture.

As expected we see that the difference in capture count between random and trained agent is maximum in the cases where the sensor coverage is 20-40% and sentry's speed and vision is four and six respectively. And the since we use a penalty of five for performing evade we notice that the learning agent does not perform well after 250,000 maps on sensor coverage 50-60%. This is because of the high sensor coverage. As sensor coverage is high we would have to evade more often and we would need to learn the Q-values completely in order to learn this. To show this we ran a simulation with a penalty of one for evade. Figure 4.9-4.10 shows the capture count and the number of steps it takes. We notice that for sensor coverage 50-60%, the agent performs better than the previous case. But the problem of learning a small penalty for evade is that it takes a long time to learn for the agent in low sensor coverage that evade actions is not a good option. Figure 4.11 shows the number of times the agent

ends up getting stuck between states. We notice that in 10% sensor coverage, the agent gets stuck more often as it learns. This is caused due to incomplete Q-values in the Q-table. To learn faster we tried implementing the lambda(Q)[27] learning. But for our problem, we realized that the overall amount of time it took to learn was almost the same as Q-learning. By changing the lambda value we could make it slightly faster, but in some cases the agent ended up learning the wrong action, and required more actions to unlearn what it had learned incorrectly.

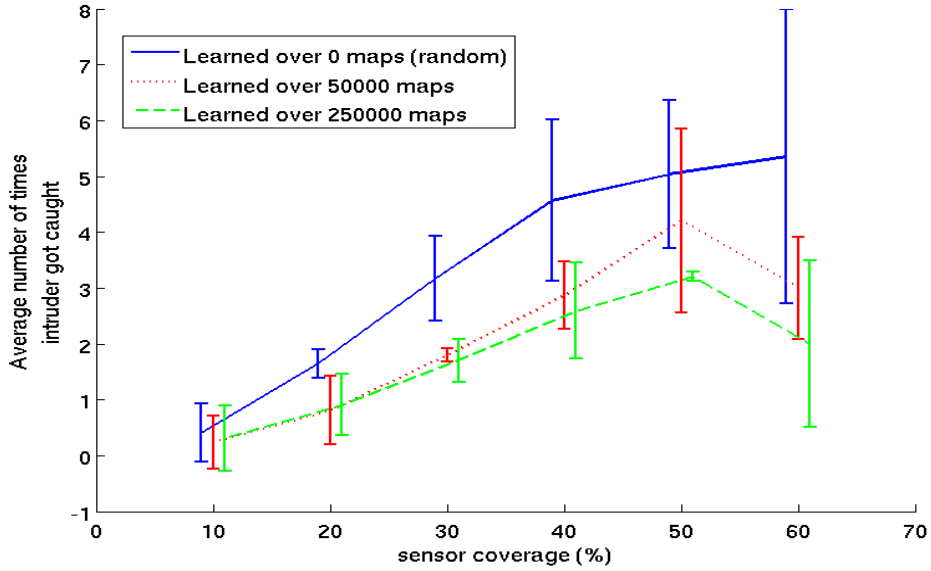

Figure 4.9. Capture Count for a setting with Sentry = (4 speed, 6 vision), reward = -1 for Evade and -50 for capture.

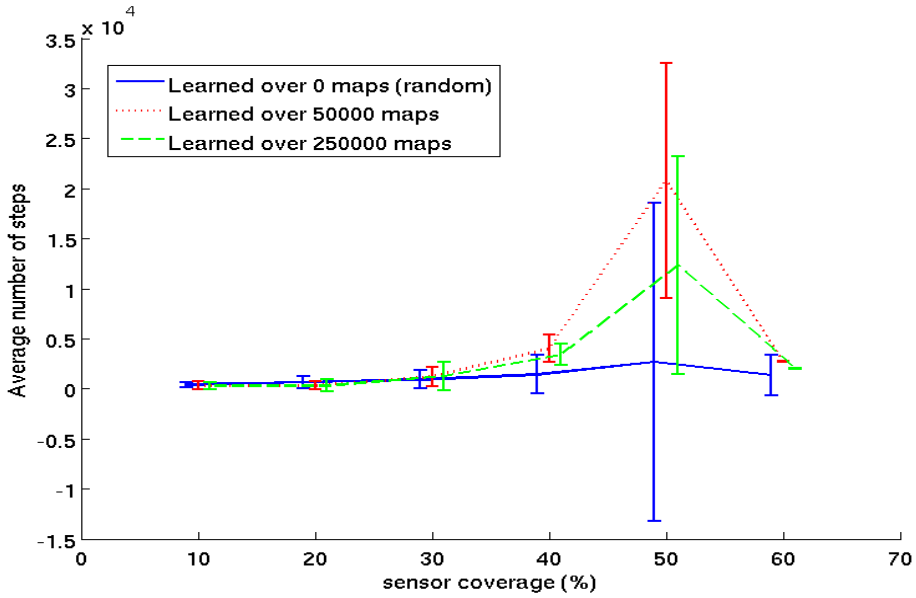

Figure 4.10. Number of Steps to find detection-free path for a setting with Sentry = (4 speed, 6 vision), reward = -1 for Evade and -50 for capture.

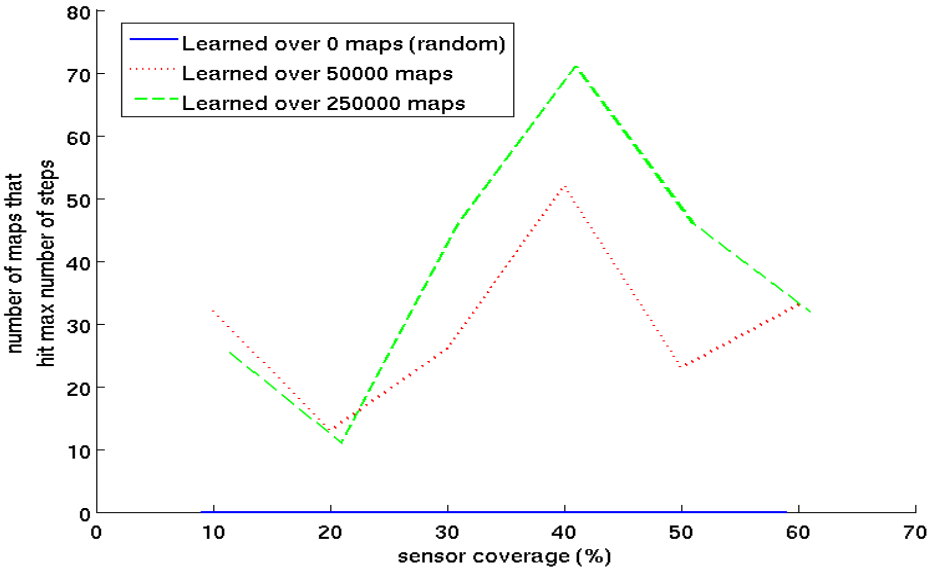

Figure 4.11. Number of maps in which agent got stuck between two states for a setting with Sentry = (4 speed, 6 vision), reward = -1 for Evade and -50 for capture.

Based on the results we found on 10X10 grid we ran the simulation on a 20X20 grid. we picked the following configurations for the simulation:

- Sensor coverage = 20%, capture penalty = -50, evade penalty = -5, sentry vision = 8 grids, sentry speed = 4 grids.
- Sensor coverage = 50%, capture penalty = -50, evade penalty = -5, sentry vision = 14 grids, sentry speed = 8 grids.

The results were as expected. It was similar to that on 10x10. For the first scenario, we noted that the capture count went down by half after learning and the number of steps it takes to reach the goal significantly went down as we had noted. For the second scenario, we were not able to finish the simulation as it was running for a really long time. But we noted that the number of steps went down from 130,000 with confidence interval 500,000 to 1500 with confidence interval of 500 after learning on 50,000 maps. But the capture count did not reduce significantly. It went down from 23 with confidence interval of 13 to 21 with confidence interval of 12. This is expected, as it requires lot more training as there is a necessity to evade more often on high density maps. We chose -5 for evade to speed up learning. But for this set up, it will end up not picking evade initially. Thus, not significantly reducing the capture count. Figure 4.12 and 4.13 show the capture count and number of steps graph for the first scenarios. We notice a drop in capture count from 7 on an average by performing random actions to 3.5 on an average after learning on 250,000 maps. The number of steps to find a detection-free path was around 22,000 on an average by doing random actions, and it dropped down to 2500 on an average after learning on 250,000 maps.

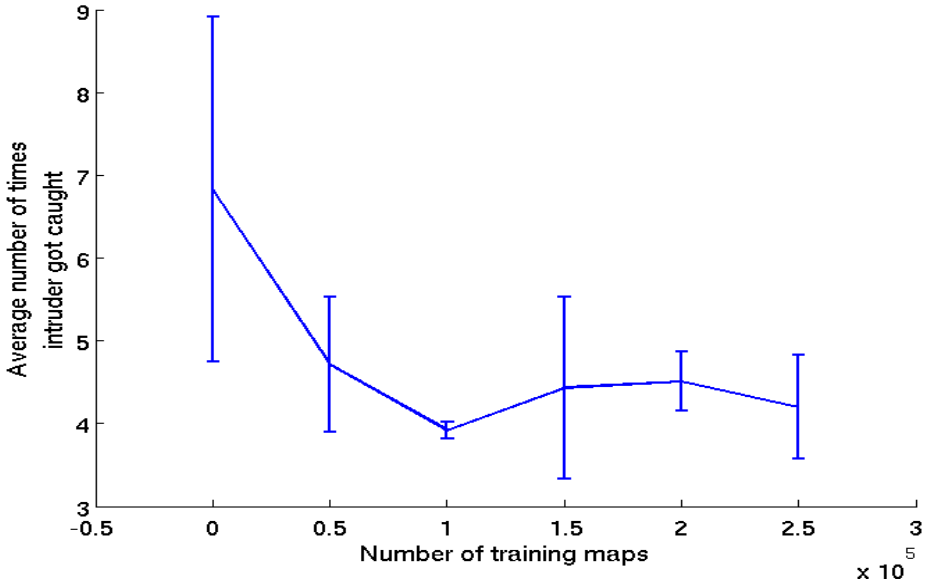

Figure 4.12. Capture Count for a setting on 20x20 with Sentry = (4 speed, 8 vision), reward = -5 for Evade and -50 for capture.

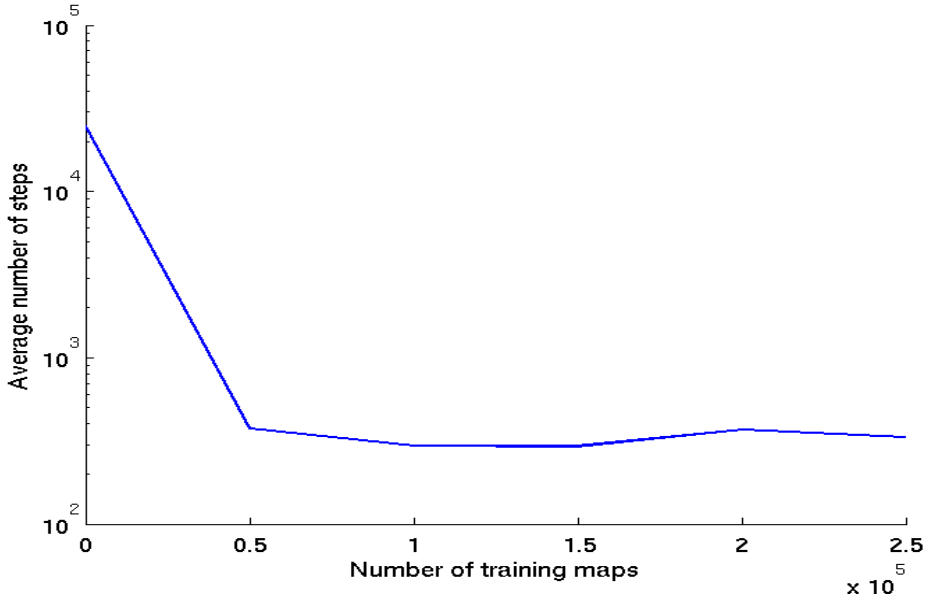

Figure 4.13. Number of Steps to find detection-free path on 20x20 with Sentry = (4 speed, 8 vision), reward = -5 for Evade and -50 for capture. The confidence interval for first point is 50,000 and rest less than 100.

CHAPTER 5

CONCLUSION

With growing use of sensors in monitoring areas it becomes extremely important to get a good model of attacker to find the right defense. We proposed a reinforcement learning based attacker who uses abstract state spaces to train over maps, and exploit the gained information in other maps. Using an attacker modeled through this method helped improving the intrusion in a map. The ability to find a detection-free path in a sensor monitored area improved, and the method also minimized the cost of finding such a path. It reduced the time it takes to find a path and also reduced the number of times the intruder would get captured. It was also noted that the maximum effect of learning is on the environment with average sentry and sensor coverage. The use of such model could help analyze an intrusion detection system better.

Future Scope : There are lot of possible future enhancement to the current system. In the current system, we send in signal from sensor as soon as the intruder triggers it. Instead we could give a delay in sending the signal. This will reduce the chance of the intruder being able to find the existence of the sensor. This will make it harder for the intruder to learn on maps.

The current sentry is not smart and it is assumes that the only way a sensor gets triggered is if a sensor is triggered by an intruder. But in real world, a sensor if often triggered by wild animals. Hence, we need to model a better sentry who learns the behavior of the environment and intruder to defend better. We could then

test the intruder with that sentry. We could use the current intruder model and test against the other detection system that exist currently.

REFERENCES

[1] K. Sharma and M. Ghose, "Wireless sensor networks: An overview on its security threats," *International Journal of Computers and Their Applications*, vol. 1, pp. 42–45, 2010.

[2] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, and L. Gu, "Energy-efficient surveillance system using wireless sensor networks," in *In Mobisys*. ACM Press, 2004, pp. 270–283.

[3] Formal Eavesdropping and its Computational Interpretation Martn Abadi. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.1076

[4] Y. Liu and W. Liang, "Approximate coverage in wireless sensor networks," in *LCN*, 2005, pp. 68–75.

[5] A. S. Tanenbaum, C. Gamage, and B. Crispo, "Taking sensor networks from the lab to the jungle." *IEEE Computer*, vol. 39, no. 8, pp. 98–100, 2006. [Online]. Available: http://dblp.uni-trier.de/db/journals/computer/computer39.html

[6] H. Zhang and J. Hou, "Maintaining Sensing Coverage and Connectivity in Large Sensor Networks," *Ad Hoc & Sensor Wireless Networks*, vol. 1, no. 1-2, 2005. [Online]. Available: http://www.oldcitypublishing.com/AHSWN/AHSWN 1.1-2 abstracts/Zhang abs.html

[7] P. Krishnamoorthy and M. Wright, "Towards modeling the behavior of physical intruders in a region monitored by a wireless sensor network," in *Proceedings of the 3rd ACM workshop on Artificial intelligence and security*, ser.

AISec '10. New York, NY, USA: ACM, 2010, pp. 1–7. [Online]. Available: http://doi.acm.org/10.1145/1866423.1866425

[8] M. Cardei and J. Wu, "Energy-efficient coverage problems in wireless ad-hoc sensor networks," *Comput. Commun.*, vol. 29, no. 4, pp. 413–420, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2004.12.025

[9] N. Xu, "A survey of sensor network applications," *IEEE Communications Magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[10] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *Computer networks*, vol. 38, no. 4, pp. 393–422, 2002.

[11] T. Zia and A. Zomaya, "Security issues in wireless sensor networks," in *ICSNC'06. International Conference on*. IEEE, 2006, pp. 40–40.

[12] T. Mitchell, *Machine Learning (Mcgraw-Hill International Edit)*, 1st ed. McGraw-Hill Education (ISE Editions), Oct. 1997.

[13] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[14] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, ser. Adaptative computation and machine learning series. A Bradford Book, 1998.

[15] S. P. M. Choi, D.-Y. Yeung, and N. L. Zhang, "Hidden-mode Markov decision processes for nonstationary sequential decision making," in *Sequence Learning*, 2001, pp. 264–287.

[16] C. J. C. H. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[17] E. Even-Dar and Y. Mansour, "Learning rates for q-learning," *J. Mach. Learn. Res.*, vol. 5, pp. 1–25, Dec. 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1005332.1005333

[18] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge,England, 1989.

[19] C. Andalora, "Q-learning and the impact of exploration policies," Nov. 2007.

[20] J. Deng, R. Han, and S. Mishra, "Countermeasures against traffic analysis attacks in wireless sensor networks," in *SecureComm 2005. First International Conference on.* IEEE, 2005, pp. 113–126.

[21] A. Saipulla, C. Westphal, B. Liu, and J. Wang, "Barrier coverage of line-based deployed wireless sensor networks," in *INFOCOM*, 2009, pp. 127–135.

[22] Y. Y. Li and L. E. Parker, "Intruder detection using a wireless sensor network with an intelligent mobile robot response," in *SOUTHEASTCON, 2008. IEEE.* IEEE, 2008, pp. 37–42.

[23] A. Kulakov and D. Davcev, "Tracking of unusual events in wireless sensor networks based on artificial neural-networks algorithms," in *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, vol. 2, 2005, pp. 534–539 Vol. 2.

[24] Z. Sun, P. Wang, M. C. Vuran, M. Al-Rodhaan, A. Al-Dhelaan, and I. F. Akyildiz, "Bordersense: Border patrol through advanced wireless sensor networks," *Ad Hoc Networks*, vol. 9, no. 3, pp. 468–477, 2011.

[25] S. Dejmal, A. Fern, and T. Nguyen, "Reinforcement learning for vulnerability assessment in peer-to-peer networks," in *Proceedings of the 20th national conference on Innovative applications of artificial intelligence - Volume 3*, ser. IAAI'08. AAAI Press, 2008, pp. 1655–1662. [Online]. Available: http://dl.acm.org/citation.cfm?id=1620138.1620146

[26] A. Servin and D. Kudenko, "Multi-agent reinforcement learning for intrusion detection: A case study and evaluation," in *ECAI*, 2008, pp. 873–874.

[27] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, ser. A Bradford book. Bradford Book, 1998.

BIOGRAPHICAL STATEMENT

Sriram Srinivasan was born in Bangalore, India in the year 1990. He received his Bachelors in engineering in Computer Science and engineering from Nagarjuna College of Engineering and Technology (a part of Visvesvaraiah Technological University). He worked as summer intern at Indian Institute of Sciences Bangalore in 2010 and 2011, and Amazon inc. in 2013. He has been a part of Information Security lab at University of Texas at Arlington since 2012. His research interest includes machine learning, algorithms and mathematics.