Technical University of Crete
School of Electrical and Computer Engineering

Coordinated Coverage in Sensor Networks via Reinforcement Learning

Georgios Kotzampasakis

Thesis Committee

Associate Professor Michail G. Lagoudakis (ECE)

Associate Professor Georgios Chalkiadakis (ECE)

Associate Professor Antonios Deligiannakis (ECE)

Chania, September 2018

Πολυτεχνείο Κρήτης
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Συντονισμός Κάλυψης σε Δίκτυα Αισθητήρων μέσω Ενισχυτικής Μάθησης

Γεώργιος Κοτζαμπασάκης

Εξεταστική Επιτροπή

Αναπληρωτής Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)

Αναπληρωτής Καθηγητής Γεώργιος Χαλκιαδάκης (ΗΜΜΥ)

Αναπληρωτής Καθηγητής Αντώνιος Δεληγιαννάκης (ΗΜΜΥ)

Χανιά, Σεπτέμβριος 2018

Abstract

Machine Learning is a fast developing and ever growing field in computer science. In addition to that, Sensor Networks are also a very promising field that has significant impact on a variety of applications. Given these facts, a multi-agent system (MAS) approach on wireless sensor networks (WSNs) comprising sensor-actuator nodes is very promising, as it has the potential to tackle the resource constraints inherent in these networks by efficiently coordinating the activities among the nodes. Furthermore, a very common issue in the field of sensor networks is the sensing coverage problem, which is the task of properly and sufficiently covering an area. In this thesis, we consider the coordinated sensing coverage problem and study the behavior and performance of the fully distributed Q-Learning algorithm for reinforcement learning using linear value function approximation. We use the Tossim platform to simulate our TinyOS application, which consists of different topologies of sensor networks with parametric sizes. Subsequently, we present the results of our simulation and display a number of graphs to visualize performance and learning outcomes on three specific topologies. We consider issues, such as successful convergence to optimal policies and maximization of local and global rewards. The implementation results are quite promising, since our algorithms exhibit high percentage of successful convergence to optimal policies.

Περίληψη

Η μηχανική μάθηση είναι ένα ταχύτατα και διαρκώς αναπτυσσόμενο πεδίο στην επιστήμη των υπολογιστών. Εκτός από αυτό, τα δίκτυα αισθητήρων είναι επίσης ένα πολλά υποσχόμενο πεδίο που έχει σημαντική επίδραση σε μία ποικιλία από εφαρμογές. Βάσει των παραπάνω, μία προσέγγιση πολυπρακτορικού συστήματος (MAS) σε ασύρματα δίκτυα αισθητήρων (WSNs) που περιλαμβάνει αισθητήρες-ενεργοποιητές κόμβους είναι πολλά υποσχόμενη, καθώς μπορεί δυνητικά να αντιμετωπίσει τους περιορισμούς σε πόρους που είναι έμφυτοι σε αυτά τα δίκτυα με το να συντονίζει αποδοτικά τις δραστηριότητες μεταξύ των κόμβων. Επιπλέον, ένα κοινό θέμα στο πεδίο των δικτύων αισθητήρων είναι το πρόβλημα της συντονισμένης κάλυψης, στο οποίο καλείται κάποιος να καλύψει κατάλληλα και επαρκώς μία περιοχή με αισθητήρες. Σε αυτή τη διπλωματική εργασία, εξετάζουμε το πρόβλημα της συντονισμένης κάλυψης των αισθητήρων και μελετάμε τη συμπεριφορά και την απόδοση του τελείως κατανεμημένου Q-Learning αλγορίθμου ενισχυτικής μάθησης χρησιμοποιώντας γραμμική προσέγγιση της συνάρτησης χρησιμότητας. Χρησιμοποιούμε την πλατφόρμα Tossim για να προσομοιώσουμε την TinyOS εφαρμογή μας, η οποία αποτελείται από διαφορετικές τοπολογίες δικτύου αισθητήρων με παραμετροποιημένο μέγεθος. Στη συνέχεια, παρουσιάζουμε τα αποτελέσματα της υλοποίησης μας και δείχνουμε έναν αριθμό από γραφήματα για να οπτικοποιήσουμε τις εκβάσεις της απόδοσης και της μάθησης σε τρεις συγκεκριμένες τοπολογίες. Λαμβάνουμε υπ’ όψιν θέματα, όπως επιτυχή σύγκλιση σε βέλτιστες πολιτικές και μεγιστοποίηση των τοπικών και καθολικών ανταμοιβών. Τα αποτελέσματα της υλοποίησης είναι αρκετά ενθαρρυντικά από την άποψη των υψηλών ποσοστών επιτυχών συγκλίσεων του αλγορίθμου μας σε βέλτιστες πολιτικές.

Acknowledgements

This thesis would be impossible to complete without the positive reinforcement from my advisor M. Lagoudakis, and his constant support.

I wish to thank him for his patience and his seemingly endless eagerness to help me with any problem throughout this thesis. Furthermore, I want him to know that he has my utmost respect and my everlasting gratitude.

I would also like to thank my family and friends for their involvement in my efforts to keep me motivated and productive during this thesis.

Table of Contents

CHAPTER 1. Introduction 15

1.1 Thesis Introduction 15

1.2 Thesis Contribution 16

1.3 Thesis Outline 16

CHAPTER 2. Background 17

2.1 Machine Learning 17

2.2 Reinforcement Learning 17

2.3 Markov Decision Processes 18

Definition: 18

Markov Property: 18

Problem: 19

Value Functions: 19

2.4 Q-Learning Algorithm 20

Algorithm: 20

Explore vs Exploit 21

Discount Factor 21

Initial Conditions (Q_0) 21

Q-Learning Properties 21

2.5 Sensors 22

2.6 Wireless Sensor Networks 22

2.7 TinyOS and Tossim 23

CHAPTER 3. Problem Statement 25

3.1 Operating Environment 25

3.2 State-Action Spaces 25

Local agent states: 26

Local agent Actions: 26

Global state of the MAS: 27

Memory requirements of the algorithm: 27

3.3 Related Work 27

CHAPTER 4. Our Approach 29

4.1 Solving the simpler problem 29

4.2 Fully Distributed Q-Learning with Linear Approximation 29

Choosing Basis Functions φ: 30

Arranging Basis Functions into Blocks: 33

4.3 Exploration Rate and Learning Rate 33

4.4 Reward Function 37

4.5 Initial States and Necessary Information 38

4.6 Process of the Simulation 38
4.7 Moving to larger grids with more Motes and random Topology 39
CHAPTER 5. Results 41
CHAPTER 6. Conclusions 59
6.1 Discussion 59
6.2 Future Work 59
6.3 Lessons 60
Bibliography 61

Table of Figures

Figure 1. 10 × 10 grid 25
Figure 2. Example of 10 × 10 grid 26
Figure 3. 6 × 10 grid 29
Figure 4. Example of 10 × 10 grid 32
Figure 5. Exploration rate using sigmoid function 34
Figure 6. Linear Learning Rate function 35
Figure 7. Sigmoid Learning Rate function 36
Figure 8. Exponential Learning Rate function 37
Figure 9. Snapshot of 20×20 grid 40
Figure 10. Optimal policy for 6 × 10 map 41
Figure 11. Global Reward over time (sigmoid learning rate) 42
Figure 12. L1 Norm for 6×10 with sigmoid learning rate 43
Figure 13. Global Reward over time (linear learning rate) 44
Figure 14. L1 Norm for 6×10 with linear learning rate 45
Figure 15. Global Reward over time (exponential learning rate) 46
Figure 16. L1 Norm for 6×10 with exponential learning rate 47
Figure 17. Optimal policy for 10 × 10 grid 48
Figure 18. Global Reward over time for 10×10 map (sigmoid learning rate) 49
Figure 19. L1 Norm for 10×10 map with sigmoid learning rate 50
Figure 20. Global reward over time for 10×10 map (linear learning rate) 51
Figure 21. L1 Norm for 10×10 map with linear learning rate 52
Figure 22. Global reward over time for 10×10 map (exponential learning rate) 53
Figure 23. L1 Norm for 10 × 10 map with exponential learning rate 54
Figure 24. Highest global reward policy for 20×20 55
Figure 25. Global reward over time for 20×20 map (sigmoid learning rate) 56
Figure 26 Global reward over time for 20×20 (linear learning rate) 57
Figure 27. Global reward over time for 20×20 (exponential learning rate) 58

CHAPTER 1. Introduction

1.1 Thesis Introduction

Consider a Multi-Agent System (MAS), which consists of a number of autonomous agents, each of which has its own states and its own actions. Modern sensor networks are examples of such MAS; each sensor is in fact an agent able to sense and cover some area around it (states), whose size may be a function of the energy consumed for sensing (action). These agents must cooperate with one another in order to maximize the sensing coverage of a wide area while, at the same time, minimizing the overall energy consumption. We are interested in studying the effectiveness of the fully distributed Q-Learning algorithm for this task, which includes also the ideas of reinforcement leaning we care to apply.

From a given agent's point of view, the MAS case differs from the single agent case, in the sense that the environment dynamics can be influenced by other agents. In addition to the uncertainty or stochastic nature that may be inherent in the environment, other agents can affect the environment in unpredictable ways due to their actions.

In a distributed learning and decision making system, the system behavior is influenced by the whole team of simultaneously and independently acting agents. Thus, the dynamics of the environment are likely to change more frequently than in the single-agent case. [1] As a learning method that does not need any prior model of the environment and can perform online learning, reinforcement learning (RL) is well suited for cooperative MAS, where agents usually have little, or in many cases none at all, information about each other. Reinforcement learning is also a robust and natural method for agents to learn how to coordinate their action choices.

An important problem addressed in the Wireless Sensor Networks (WSN) literature is the sensing coverage problem [2], [3], [4]. In this problem, the task of the sensor network is to properly cover an area in order to make sure that all important events which occur in that area can be accurately detected by at least one sensor.

A distributed approach to the sensing coverage problem is attractive for several reasons. First, sensing entities are usually spatially distributed, thus forming distributed systems using a decentralized approach is more natural. Second, sensor networks can be very large, i.e. containing hundreds or thousands of nodes; hence a distributed approach would always be more scalable than a centralized one. Finally, a distributed approach is compatible with the resource-constrained nature of sensor nodes. Many of the sensors are usually small devices with limited memory and computational capabilities and are energy constrained, since there are battery-powered. Therefore, a distributed approach to performing computation, i.e. using distributed algorithms, and limiting the amount and distance of communication are necessary design parameters in order to achieve an efficient, energy-aware and scalable solution. Furthermore, the restricted communication bandwidth and range in WSNs would exclude a centralized approach.

Given the above observations, we consider a monitor application of a field represented by an $m \times n$ grid. In fact, three different discrete grids are used for our simulations in order to give substance to our problem. Then, we use the Tossim platform to simulate our TinyOS application in order to experiment on the coordinated sensing coverage problem using RL, as mentioned earlier.

1.2 Thesis Contribution

The way we tackle our problem involves the following steps. Firstly, we focused on two small grids to initially simplify and understand our problem. We applied a linear approximation of the value function in the Q-Learning algorithm to overcome memory requirements restrictions of the problem's space state. After concluding our simulations on the first two grids, we moved on to a larger grid in order to test our algorithm in a harder and more sophisticated challenge.

The results on the first and more trivial problem were, as ought to be, great. The algorithm converged to the optimal policy every time. On the second grid, the results were also nearly perfect, with very high successful convergence percentages to the optimal policy. While in the last one, the upward trend of the global reward graph showed encouraging results, when it came to larger and more difficult grids.

All in all, introducing reinforcement learning in the field WSNs proves to be a very promising and fertile idea for a plethora of applications and is definitely worth the time and effort to continue experimenting and researching towards this aspect.

1.3 Thesis Outline

In Chapter 2 we include all the theoretical background needed for this thesis. We present an overview of the Machine Learning field, Markov Decision Processes and Sensor Networks. Furthermore, we display basic information about the Q-Learning algorithm. In Chapter 3 we state the problem and everything you need to know about it. Continuing with Chapter 4, we present our approach of the problem thoroughly and the core ideas behind it. In Chapter 5 we present the results of our simulations and analyze the outcomes. Finally, Chapter 6 concludes our work and lists ideas for future improvements.

CHAPTER 2. Background

2.1 Machine Learning

Machine learning [5] is an application of Artificial Intelligence that provides systems with the ability to automatically "learn" from and improve their performance with experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use them to learn for themselves.

The process of learning starts with observations or data with the purpose of finding patterns in order to make better future decisions based on the information that was initially given. The main goal is to allow computer systems to learn on their own, without human intervention or assistance. Machine learning is applied on a variety of computing tasks, where writing and programming explicit algorithms with high performance is very difficult or, in many cases, infeasible; example applications include email filtering, detection of network intruders, and computer vision.

Machine learning is closely related to computational statistics, which also focuses on prediction-making through the use of computers. It has strong ties to mathematical optimization, which delivers methods, theory and application domains to the field. Machine learning is sometimes confused with data mining, where the latter subfield focuses more on exploratory data analysis in order to discover structure in the data and is widely known as "unsupervised learning".

2.2 Reinforcement Learning

Reinforcement learning (RL) [6] is an area of machine learning that has to do with how software agents learn to take actions in an environment in order to maximize a reward that corresponds to a specific principle of acting. The problem, because of the fact that it is quite generalized, is studied in many other concepts, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems and many more.

Reinforcement learning, in the context of artificial intelligence, is a type of dynamic programming. Moreover, RL is an approach to machine learning that is inspired by behavioral psychology. It looks a lot like the way a child is taught to perform an activity. The main difference between RL and other machine learning algorithms is that in RL the agent is not programmed in advance in order to perform a task, but it is left with no supervision or human assistance in order to address the problem on its own through trial and error.

In reinforcement learning, the environment is typically formulated as a Markov Decision Process (MDP), as many RL algorithms for this context utilize dynamic programming techniques. RL contrasts with the classical dynamic programming methods in that RL does not require knowledge of an exact mathematical model of the MDP and it targets large MDPs where exact methods become infeasible. RL differs from standard supervised learning in the aspect that correct input/output pairs do not have to be displayed, and sub-optimal actions need not be explicitly corrected. Instead the focus is on performance, which requires to find a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

2.3 Markov Decision Processes

A Markov Decision Process (MDP) [7] is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations, where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning. MDPs are used in many fields, including robotics, automatic control, economics and manufacturing.

At each time step, the process is in some state ***s***, and then decision maker may select any action ***a***, which is available in state ***s***. The process responds at the next step by randomly moving into a new state ***s'***, and giving the decision maker a corresponding reward $\mathbf{R_a(s, s')}$. The probability by which the process shifts into its new state ***s'*** is influenced by the selected action. Specifically, it is derived by the state transition function $\mathbf{P_a(s, s')}$. Thus, the next state ***s'*** depends on the current state ***s*** and then decision maker's action ***a***. But given ***s*** and ***a***, it is conditionally independent of all previous states and actions; that means, the state transition of an MDP satisfies the Markov property.

Definition:

A Markov Decision Process is a 5-tuple $(\mathbf{S, A, P_a, R_a}, \gamma)$, where

- **S** is a finite set of states
- **A** is a finite set of actions (alternatively, $\mathbf{A_s}$ is the set of actions available in **s**)
- $\mathbf{P_a(s, s') = Pr(s_{t+1} = s' \mid s_t = s, a_t = a)}$ is the probability that action **a** in state **s** at time **t** will lead to state **s'** at time **t+1**
- $\mathbf{R_a(s, s')}$ is the immediate reward (or expected immediate reward) received after transition from state **s** to state **s'**, due to action **a**.
- $\gamma \notin \mathbf{[0, 1]}$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

Markov Property:

- The next state is independent from past record
- The reward is independent from past record

Problem:

The core problem of MDPs is to find a “policy” for the decision maker: a function **π** that specifies the action **π(s)** that the decision maker will choose when in state **s**. Once a Markov decision process is combined with a policy in this way, this fixes the action for each state and subsequently resulting combination behaves like a Markov chain (since the action chosen in state $\mathbf{a_t}$ is completely determined by **π(s)** and $\mathbf{Pr(s_{t+1} = s' \mid s_t = s, a_t = a)}$ reduces to $\mathbf{Pr(s_{t+1} = s' \mid s_t = s)}$, a Markov transition matrix).

The primary aim is to choose a policy **π** that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon:

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$$

where **γ** is the discount factor, satisfies $0 \le \gamma < 1$ and typically is close to 1.

Value Functions:

- **State Value Function V**

$$V^{\pi}(s) = E_{a_t \sim \pi;\, s_t \sim P;\, r_t \sim R}\Big(\sum_{t=0}^{h} \gamma^t r_t | s_0 = s\Big)$$

$$\begin{tikzcd} s \arrow[r, "\pi(s)", "r_0"'] & s_1 \arrow[r, "\pi(s_1)", "r_1"'] & s_2 \arrow[r, "\pi(s_2)", "r_2"'] & s_3 & \cdots & s_{h-1} \arrow[r, "\pi(s_{h-1})", "r_{h-1}"'] & s_h \end{tikzcd}$$

From state **s** following policy **π(s)** we move to state $\mathbf{s_1}$ receiving reward $\mathbf{r_0}$.
From state $\mathbf{s_1}$ following policy $\mathbf{\pi(s_1)}$ we move to state $\mathbf{s_2}$ receiving reward $\mathbf{r_1}$.
Inductively, we can see that from state $\mathbf{s_{h-1}}$ following policy $\mathbf{\pi(s_{h-1})}$ we move to state $\mathbf{s_h}$ receiving reward $\mathbf{r_{h-1}}$.

- **State- Action Value Function Q**

$$Q^{\pi}(s,a) = E_{a_t \sim \pi;\, s_t \sim P;\, r_t \sim R}\Big(\sum_{t=0}^{h} \gamma^t r_t | s_0 = s, a_0 = a\Big)$$

$$\begin{tikzcd} s \arrow[r, "a", "r_0"'] & s_1 \arrow[r, "\pi(s_1)", "r_1"'] & s_2 \arrow[r, "\pi(s_2)", "r_2"'] & s_3 & \cdots & s_{h-1} \arrow[r, "\pi(s_{h-1})", "r_{h-1}"'] & s_h \end{tikzcd}$$

From state **s** taking action **a** we move to state $\mathbf{s_1}$ receiving reward $\mathbf{r_0}$.

From state $\mathbf{s_1}$ and following policy $\boldsymbol{\pi}$ thereafter, similarly to our previous example, we can deduct that from state $\mathbf{s_{h-1}}$ we move to state $\mathbf{s_h}$ receiving reward $\mathbf{r_{h-1}}$.

2.4 Q-Learning Algorithm

Q-learning [8] is a RL algorithm used widely in machine learning. The primary aim of Q-Learning is to conclude to a policy, which tells a given agent which action to follow under what conditions. It does not need any prior model of the environment and it can perform online learning. Furthermore, it can handle problems with stochastic transitions and rewards, without requiring adaptations.

For any finite Markov Decision Process (FMDP), Q-learning finds a policy that is optimal in the sense that it maximizes the expected value of the total reward over all successive steps, starting from the current state. Q-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy.

Algorithm:

The weight for a step from a state $\mathbf{\Delta t}$ steps into the future is calculated as $\gamma^{\Delta t}$. γ (the discount factor) is a number between 0 and 1 ($0 \le \gamma < 1$) and has the property to evaluate the rewards which are received earlier higher than those that are received later. $\boldsymbol{\gamma}$ may also be interpreted as the probability to succeed at every step $\mathbf{\Delta t}$. The algorithm, therefore, has a function that calculates the quality of a state-action combination:

$$\mathbf{Q: S \times A \rightarrow R}$$

Before learning starts, **Q** is initialized to a possibly arbitrary constant value (selected by the programmer). Then, at each time **t** the agent chooses an action $\mathbf{a_t}$, receives a reward $\mathbf{r_t}$, enters a new state $\mathbf{s_{t+1}}$ (that can possibly depend on both the previous state $\mathbf{s_t}$ and the selected action $\mathbf{a_t}$), and **Q** is updated. The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information:

$$\boldsymbol{Q^{new}(s_t, a_t) \leftarrow (1 - a) * Q^{old}(s_t, a_t) + a * (r_t + \gamma * maxQ(s_{t+1}, a'))}$$

where $\mathbf{r_t}$ is the reward observed for the current state $\mathbf{s_t}$, and $\boldsymbol{\alpha}$ is the learning rate ($0<\alpha\le1$). An episode of the algorithm ends when state $\mathbf{s_{t+1}}$ is a final or *terminal* state. However, Q-Learning can also learn in non-episodic tasks. If the discount factor is lower than 1, the action values are finite, even if the problem can contain infinite loops in the state space.

For all final states $\mathbf{s_f}$, $\mathbf{Q(s_f, a)}$ is never updated , but is set to reward value $\mathbf{r}$ observed for state $\mathbf{s_f}$. In most cases $\mathbf{Q(s_f, a)}$ can be taken equal to zero.

Explore vs Exploit

The update of the Q-Values does not cater for actions that are never selected. Exploitation selects the best known action, or in other words, the greedy actions at all times. Exploration selects random actions every now and then, in order to improve the estimates of all the Q values in the Q-array, so that better actions may be found. The balance between exploitation and exploration is dependent on the accuracy of the Q-value estimation and the level of stochastic behavior in the environment.

Discount Factor

The discount factor γ determines the significance of future rewards. A factor of 0 will make a given agent short-sighted by only concerning itself with current rewards, while a factor approaching 1 will make it to work hard for a long-term high reward. If the discount factor becomes equal to 1 or if it goes beyond 1, the values of the action may deflect.

For $\boldsymbol{\gamma=1}$, without a final state, or if the agent never gets to one, all environment records become infinitely long, and utilities with additive, undiscounted rewards generally become infinite.

Initial Conditions (Q_0)

Since Q-learning is a repetitious algorithm, it implicitly requires a starting condition, before the first update happens. High initial values, also referred to as "optimistic initial conditions", can encourage exploration: no matter what action is chosen, the update rule will cause it to have lower values than the other alternative, thus increasing their selection probability. The first reward **r** can be used to reset the initial conditions. According to this idea, the first time an action is selected the reward is used to set the value of **Q**.

Q-Learning Properties

- **Advantages**
  - It elaborates every sample directly
  - It has minimum update cost per sample
  - It sets no restrictions on the sample's collection (off policy)
- **Disadvantages**
  - It requires a huge number of samples
  - It requires cautious handling of the exploration rate
  - The usage of each sample is minimum
  - The order of the sample appearance affects the outcome
  - It often displays unstableness with approximating representations

2.5 Sensors

A sensor is a device that detects and responds to some type of input from the physical environment [9]. The specific input could be light, heat, motion, moisture, pressure, or any one of a great number of other environmental measurements. The output is generally a signal that is converted to human-readable display at the sensor location or transmitted electronically over a network for reading or further processing. A sensor is always used with other electronics, whether as simple as a light or as complex as a computer. Sensors are employed in everyday objects, such as touch-sensitive elevator buttons (tactile sensor) and lamps which dim or brighten by touching the base, besides innumerable applications of which most people are never aware. Applications include manufacturing and machinery, airplanes and aerospace, cars, medicine, robotics and many other aspects of our day-to-day life. A sensor's sensitivity indicates how much the sensor's output changes when the input quantity being measured changes. Sensors are frequently designed to have a small effect on what is measured. Making the sensor smaller usually improves this and could introduce other advantages.

2.6 Wireless Sensor Networks

A Wireless Sensor Network (WSN) can be defined as a network of devices that can communicate the information gathered from a monitored field through wireless links. The data is forwarded through multiple nodes, and with a gateway, the data is connected to other networks such as wireless Ethernet [10]. These are a lot like wireless ad hoc networks in the aspect that they rely on wireless connectivity and spontaneous formation of networks, so that sensor data can be transported wirelessly. WSNs are spatially distributed autonomous sensors to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to cooperatively pass their data through the network to a main location. The more modern networks are bi-directional, also enabling control of sensor activity.

The WSN consists of "nodes", from a few to several hundreds or even thousands, where each node is connected to one (or sometimes several) sensors. Each such sensor network node has typically several parts: a radio transceiver with an internal antenna or connection to an external antenna, a microcontroller, an electronic circuit for interfacing with the sensors and an energy source, usually a battery or an embedded form of energy harvesting.

There are many types of WSNs depending on the environment they are deployed in. Some of them are terrestrial, underground, underwater, multimedia and finally mobile WSNs.

The cost of sensor nodes is similarly variable, ranging from a few to hundreds of dollars, depending on the complexity of the individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as communications bandwidth, computational speed, energy, and memory. The topology of WSNs may vary from a simple star network to an advanced multi-hop wireless mesh network.

2.7 TinyOS and Tossim

TinyOS and Tossim are the tools we used to write our implementation code and simulate the sensor network environment we needed in order to test our algorithms behavior and performance.

TinyOS [12] is an “operating system” designed for low-power wireless embedded systems. Fundamentally, it is a work scheduler and a collection of drivers for microcontrollers and other ICs commonly used in wireless embedded platforms. TinyOS is written in nesC, a dialect of C.

Tossim [13] simulates entire TinyOS applications. It works by replacing components with simulation implementations. Tossim is a library: you must write a program that configures a simulation and runs it. Tossim supports two programming interfaces: Python and C++. Python allows you to interact with a running simulation dynamically, like a powerful debugger. However, since the interpretator can be a performance bottleneck when obtaining results, Tossim also has a C++ interface. Usually, transforming code from one to the other is very simple.

CHAPTER 3. Problem Statement

3.1 Operating Environment

We consider a monitoring application of a field represented by an N × M grid containing a number of sensors. We present here the 10 × 10 grid shown in Figure 1, containing five sensors as an example to depict our problem.

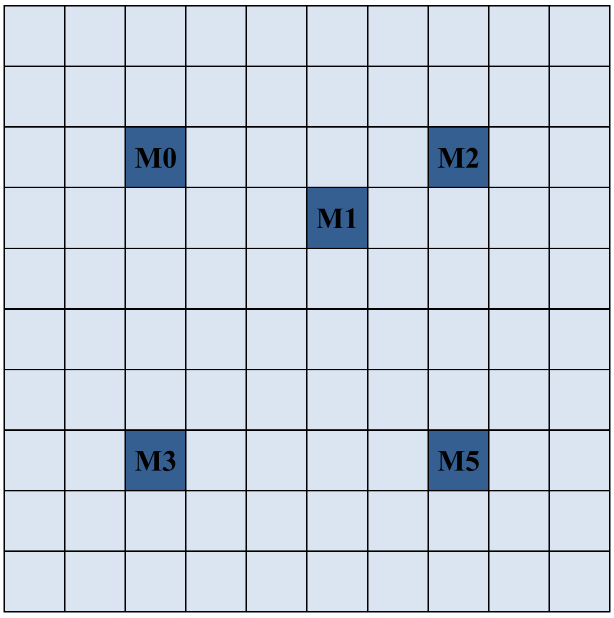


Figure 1. 10 × 10 grid

This field contains a group of agents on five motes with sensing capabilities labeled from M0 to M5 that are randomly deployed (the motes are fixed thereafter). The objective is for the agents to sense the maximum amount of area in an energy-efficient way, i.e. achieve the best level of coverage while minimizing the energy consumption resulting from the motes' sensing. We consider a deterministic environment with deterministic state transitions and rewards.

3.2 State-Action Spaces

The sensing area of an agent i, denoted as Area^i, refers to the 5 × 5 grid square centered on the agent i.

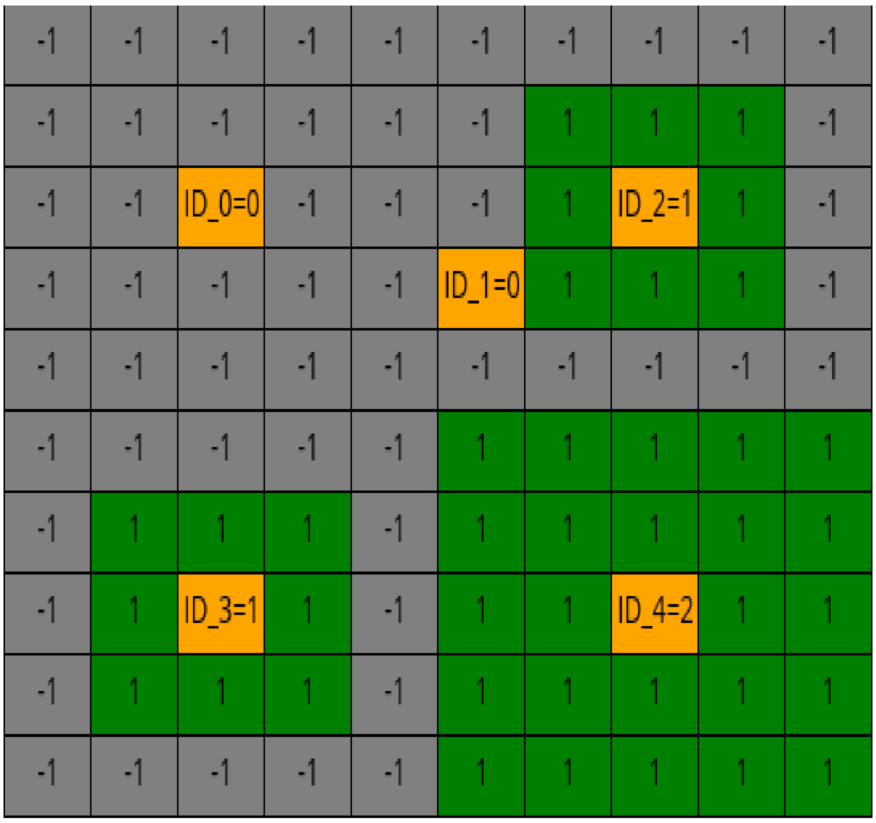
Local agent states:

Each mote i senses its area. We define the status of a cell as binary: {sensed, not sensed by a mote}.However, for practical reasons sensed corresponds to the value of +1, while not sensed corresponds to the value of –1. A cell is considered sensed, if it is covered by at least one mote. The local state s^i of an agent is the concatenation of the sensing status of the twenty-five (25) cells in its area. Therefore, there are 2^{25} possible states for each agent.

Local agent Actions:

Each mote has the ability to take one of the following three actions in any state it lands in. The action space A^i is:

- Action 0: Turn OFF its sensor, as shown by M0 and M1 in Figure 2 below.
- Action 1: Turn on its sensor in LOW mode. In this mode the sensor senses nine (9) cells around itself, as shown by M2 and M3 in Figure 2 below.
- Action 2: Turn on its sensor in HIGH mode. In this mode the sensor senses twenty-five (25) cells around itself, as shown by M4 in Figure 2 below.



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	1	1	-1
-1	-1	ID_0=0	-1	-1	-1	1	ID_2=1	1	-1
-1	-1	-1	-1	-1	ID_1=0	1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	1	1	1	1	1
-1	1	1	1	-1	1	1	1	1	1
-1	1	ID_3=1	1	-1	1	1	ID_4=2	1	1
-1	1	1	1	-1	1	1	1	1	1
-1	-1	-1	-1	-1	1	1	1	1	1

Figure 2. Example of 10 × 10 grid

In Figure 2 above you can see a snapshot of the 10 × 10 grid.

Yellow cells are centers of motes. You can also see their ID and their selected action.

Cells with grey color and the "–1" value are not sensed by any of the motes.

Cells with green color and the "+1" value are sensed by exactly one mote.

Global state of the MAS:

The global state seen by the five agent MAS, is the concatenation of the sensing status of all the cells in the field. Thus, there are 2^{100} possible global states.

Memory requirements of the algorithm:

Expression: $S^i \times A^i$, Actual Values: $2^{25} \times 3 = 100{,}663{,}296$ values

3.3 Related Work

There are several papers published that present the concept of using reinforcement learning in the field of wireless sensor networks for a variety of purposes, such as cooperative communication, coordinated coverage, routing and rate control.

The whole idea of this thesis is based on a paper presented at the IEEE 14th International Conference in Singapore in 2006 [11]. As shown here, the concept was to introduce reinforcement learning algorithms in sensor networks in order to achieve better sensing coverage. A number of distributed value function algorithms were presented and tested out in terms of policy convergence and energy consumption. However, the focus remained only on the 10 × 10 grid.

Another paper, where one of the authors was also an author on the previous paper mentioned, was presented at the 3rd International Conference on Intelligent Sensors held in Melbourne, Australia in 2007 [12]. The context of the paper was a lot similar to the previous one. It presented two of the distributed value functions included in the previous paper and it contained a newly developed algorithm. The performance of these three algorithms was compared in terms of convergence and energy consumption, in higher and lower sensor node densities.

Another example of reinforcement learning used in sensor networks many years ago, is the paper of Michael Littman and Justin Boyan in 1993 [13] . The paper introduced the idea of using reinforcement learning for better networking routing. More specifically, they present a learning algorithm for routing packets efficiently in an irregularly-connected communication network with unpredictable usage patterns.

Finally, a more recently published article in 2015 [14] provides an extensive review on the application of reinforcement leaning to WNSs. Furthermore, it presents how most schemes in wireless sensor networks have been approached using the traditional and enhanced reinforcement learning models and algorithms. In addition to that, it displays performance enhancements brought by RL algorithms and problems still not addressed regarding the application of RL in WSNs.

CHAPTER 4. Our Approach

4.1 Solving the simpler problem

Our idea was not only to solve the 10 × 10 problem shown earlier as an example, but to create a parameterized algorithm as well, which we could use later for any random topology and a bigger number of nodes in order to find good or optimal coordinated coverage policies. However, in order to do that, we thought we should first take a step back, towards solving an even simpler problem, which would have an obvious and definitely optimal solution. So we came up with the 6 × 10 map using three motes, which is shown in Figure 3 below. It is basically the same layout as the 10 x 10 map shown before, without the last two motes. By solving, we mean converging to an optimal policy as often as possible.

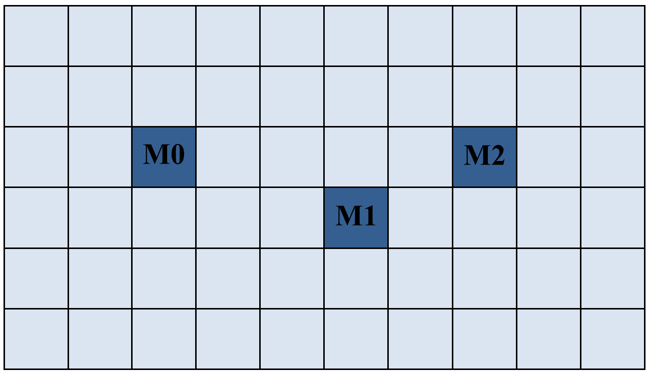


Figure 3. 6 × 10 grid

In this scenario, the optimal solution would be for M0 and M2 to choose action 2, which is to function on High mode and for M1 to choose action 0, which is to turn off. This is obvious to see, since if M1 remains turned off, then M0 and M2 have each exactly twenty-five cells to sense. This setting returns the maximum of reward for the given topology, which we will discuss about later.

4.2 Fully Distributed Q-Learning with Linear Approximation

Given the problem's memory requirements in order to store the Q-values, using an approximation to depict the problem seemed like the only way to go, since as it was displayed earlier, the total number of Q-values amounts to more than 10^8. Thus, we decided to use Q-Learning with Value Function Approximation and a Linear Approximation Architecture. Below, the two main equations that describe the idea of the algorithm are shown:

$$Q(s, a; w) = \sum_{i=1}^{k} w_i \varphi_i\,(s, a) = \varphi(s, a)^T w$$

Update:

$$w_i \ \leftarrow w_i + a\ \varphi_i(s, a)[r + \ \gamma max\varphi(s', a')^T w - \ \varphi(s, a)^T w]$$

φ_i : are basis functions which give an approximation/abstraction of the state

w_i: are weight values for the corresponding basis functions

r: is the reward received

α: is the learning rate

γ: is the discount factor

Choosing Basis Functions φ:

A critical part for a linear approximation, in order to be successful, is choosing accurate enough basis functions. However, this is no trivial task, because you can't know in advance which functions are better than others or if some functions are good enough until you actually test them out. Following this concept, we also performed a lot of experimenting and testing until we came up with our final set of basis functions. Basically, our course towards finding the best set of basis functions can be divided into three stages.

In the first stage, our set of basis functions was the following. We used a set that was consisted of eight basis functions. The first three correspond to the percentage of the sensed cells in the 1 × 1, 3 × 3, and 5 × 5 outer rings of the agent's area accordingly. The next four represented the percentage of sensed cells in every 3×3 quartile of the agent's 5 × 5 area. The last one is a constant that always has the value of '1'. It is needed as a convenient numerical trick for an additive constant (the corresponding weight), just in case we want to shift the entire curve of the approximated function up or down. In order to make it easier to grasp:

- φ_0= value of '0' in case mote *i* is turned OFF or value of '1' if mote *i* is ON
- φ_1= percentage of sensed cells in the 3 × 3 perimeter of agent's *i* 5×5 area
- φ_2= percentage of sensed cells in the 5 × 5 perimeter of agent's *i* 5×5 area
- φ_3 = percentage of sensed cells in the upper left quartile of agent's *i* 5×5 area
- φ_4= percentage of sensed cells in the upper right quartile of agent's *i* 5×5 area
- φ_5= percentage of sensed cells in the bottom left quartile of agent's *i* 5×5 area
- φ_6= percentage of sensed cells in the bottom right quartile of agent's *i* 5×5 area
- φ_7= '1'

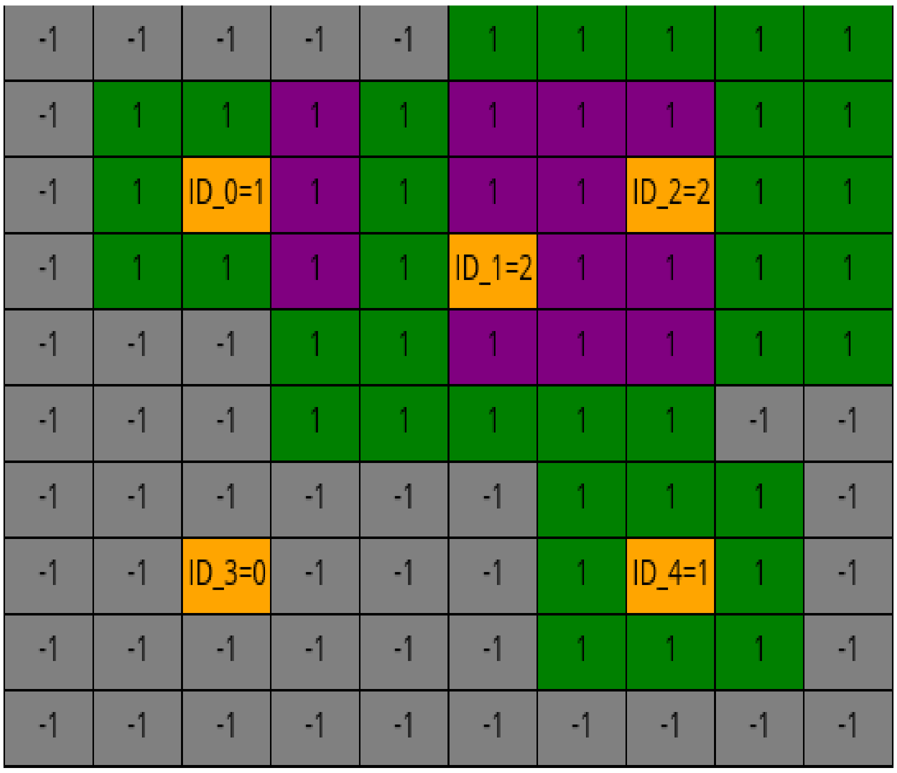
However, the results were not that good in terms of percentages of convergence to the optimal policy. As, we later found out the approximation of the agent's state was not accurate enough using this set of seven basis functions.

In the second stage, we tried the following set of basis functions. The value of each cell of the twenty-five cells that compose the 5 × 5 area, which constitutes the state of agent *i,* corresponds to one basis function φ. By value, we refer to the sensed and non-sensed status of the cell (+1, –1). That means we have a set of 25 basis functions, which of course is different for every agent considering its own state. Again, we used an extra basis function that was constant as mentioned previously. And in order to make it more vivid and easier to visualize:

- φ_0 = value of (0,0) element of agent's *i* 5×5 sensing area
- φ_2 = value of (0,1) element of agent's i 5×5 sensing area
- φ_3 = value of (0,2) element of agent's i 5×5 sensing area
- .
- .
- .
- φ_{23} = value of (4,3) element of agent's i 5×5 sensing area
- φ_{24} = value of (4,4) element of agent's i 5×5 sensing area
- φ_{25} = '1'

Once again, the results were not good enough in terms of percentages convergence to the optimal policy.

In the third and final stage, we simply combined the previous sets of basis functions to a total of 33 (only one constant is needed) and then our results were finally the desired ones. Of course, this combination naturally made a lot of sense, since in general, the more basis functions one uses, the more accurate the approximation of the state becomes, hence the results are better, as long as these basis functions are not linearly dependent with each other.



-1	-1	-1	-1	-1	1	1	1	1	1
-1	1	1	1	1	1	1	1	1	1
-1	1	ID_0=1	1	1	1	1	ID_2=2	1	1
-1	1	1	1	1	ID_1=2	1	1	1	1
-1	-1	-1	1	1	1	1	1	1	1
-1	-1	-1	1	1	1	1	1	-1	-1
-1	-1	-1	-1	-1	-1	1	1	1	-1
-1	-1	ID_3=0	-1	-1	-1	1	ID_4=1	1	-1
-1	-1	-1	-1	-1	-1	1	1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure 4. Example of 10 × 10 grid

Let's consider the snapshot of 10 × 10 grid shown in Figure 4 above. Firstly, we should mention that the cells in purple color are cells that are sensed by more than two motes. There other two colors that have already been explained. Now, if we focus on Mote 0, which is the upper right mote with id 0, and take a look at the 5 × 5 area which is his current state, then the values of the corresponding basis function's vector would be the following:

φ_0	-1	φ_{17}	1
φ_1	-1	φ_{18}	1
φ_2	-1	φ_{19}	1
φ_3	-1	φ_{20}	-1
φ_4	-1	φ_{21}	-1
φ_5	-1	φ_{22}	-1
φ_6	1	φ_{23}	1
φ_7	1	φ_{24}	1
φ_8	1	φ_{25}	1/1
φ_9	1	φ_{26}	8/8
φ_{10}	-1	φ_{27}	5/16
φ_{11}	1	φ_{28}	4/9
φ_{12}	1	φ_{29}	6/9
φ_{13}	1	φ_{30}	4/9
φ_{14}	1	φ_{31}	8/9
φ_{15}	-1	φ_{32}	1
φ_{16}	1		

φ_0-φ_{24} : correspond to those of the state

φ_{25}-φ_{27} : correspond to the rings

φ_{28}-φ_{31} : correspond to the quartiles

φ_{32} : corresponds to the constant

Arranging Basis Functions into Blocks:

We have seen where our basis functions are taking their values from. However, these basis functions values are solely dependent on the state of each agent so far. As we saw earlier in the equations of Q-learning, the Q-values must also be dependent on the action agent *i* chooses. So, we also have to build a dependency from the action. We made that possible by shifting the basis functions into different blocks depending on the action value. More specifically, we used a bigger vector than the basic ones (state-dependent) just shown above, that had the size of the total number of state basis functions times 3 (one block/copy of basis functions for each of the three actions) plus one for our constant basis function.

To elaborate more, if the action of the agent *i* is 0, then the state basis functions would be stored in the first 32 slots of the array (positions 0-31) and the rest would be filled with zeros (except from the one that holds the value of our constant basis function). Respectively, if the agent *i* chooses action 1, then the state basis functions would be stored in slots 32-63 and then again the rest would be filled with zeros, except the constant. Finally, if the agent *i* chooses action 2, then the state basis functions are stored in slots 64-96.

So to sum up, in the first stage, where we had 8 basis functions (7+1 for the constant), the total size of the vector was $7\times3+1 = 22$. In the second state, where we had 26 basis functions (25+1 for the constant) the total size of the vector was $25\times3+1=76$. While, in our last stage, where the basis functions took their final 32+1 form, the total size of the vector was $32\times3+1 = 97$.

4.3 Exploration Rate and Learning Rate

Another issue that needs to be addressed regarding a simulation of reinforcement learning is how to go about setting the Exploration and Learning Rates. In our implementation, we use a sigmoid function for exploration, simply because it suited our needs better, the smoothness of it. In Figure 5, below you can see the sigmoid function that the exploration rate follows. On the upper right of the graph, you can see the expression of the function for the 10000 iterations. On the x axis are the iterations and on the y axis is the value of the exploration rate.

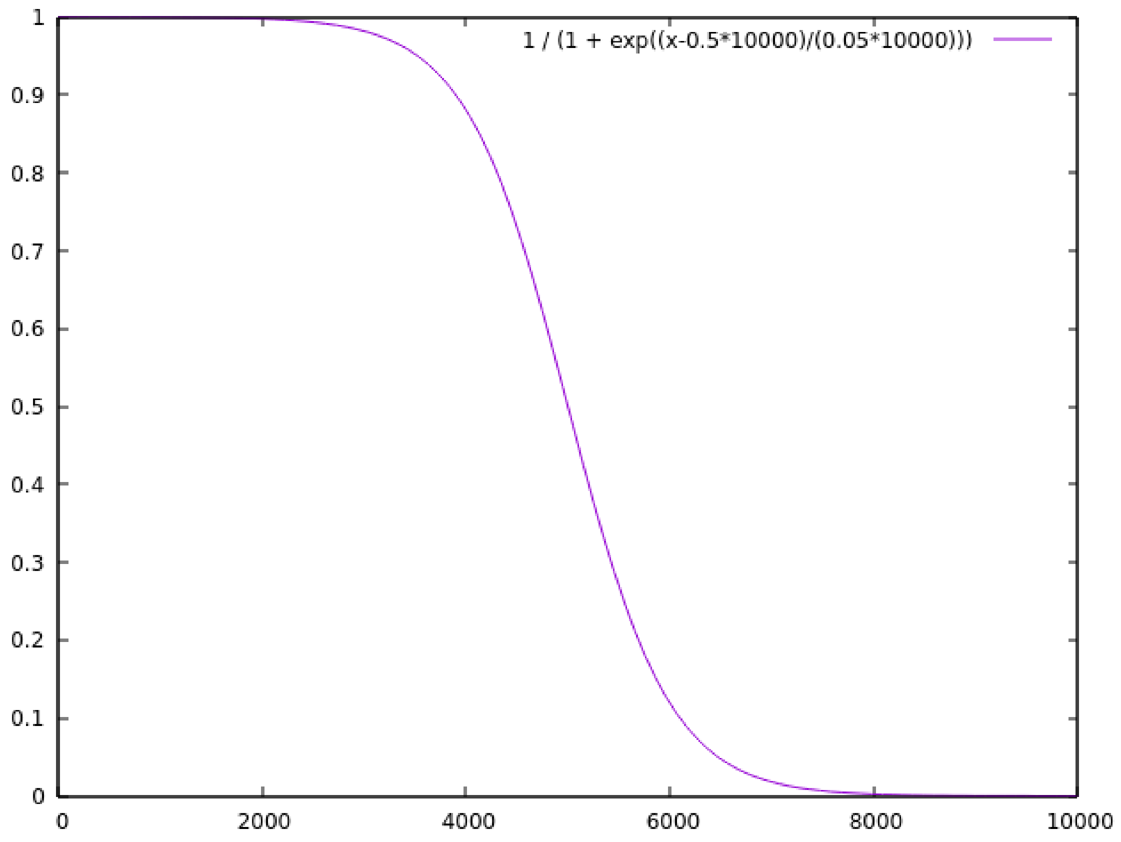


Figure 5. Exploration rate using sigmoid function

This means that in the earlier stages of the simulation our agents are more likely to choose an action randomly. This aids our cause, since there isn't yet enough information for a proper selection of action. So, the encouragement of exploring more options makes a lot of sense. Now, the more we get close to the end of the simulation (10000 iterations), the less we want our agents to choose randomly. So, as seen from the graph above, at around 8000 iterations the probability of choosing randomly tends to zero. At that point, in most of our simulations our agents have already converged towards a policy.

Regarding the learning rate, which is also included in the update equation shown before, its accurate and cautious selection is very important in order to have good results. Therefore, we experimented a lot with before concluding to a decision. The learning rate determines to what extent newly acquired information overrides old information in the Q-function. A value of zero allows the agent to learn nothing, exclusively exploiting prior knowledge, while a value of one forces the agent to concern itself only with the most recent information (ignoring prior knowledge to explore possibilities). Similarly to exploration rate, at first we want the learning rate to take a bigger value, so that the agent "learns" faster and gradually lower it. In any case, the extreme values of 0 and 1 are avoided.

We considered three different functions for learning rate (one at a time), a linear, a sigmoid and an exponential. The three graphs follow:

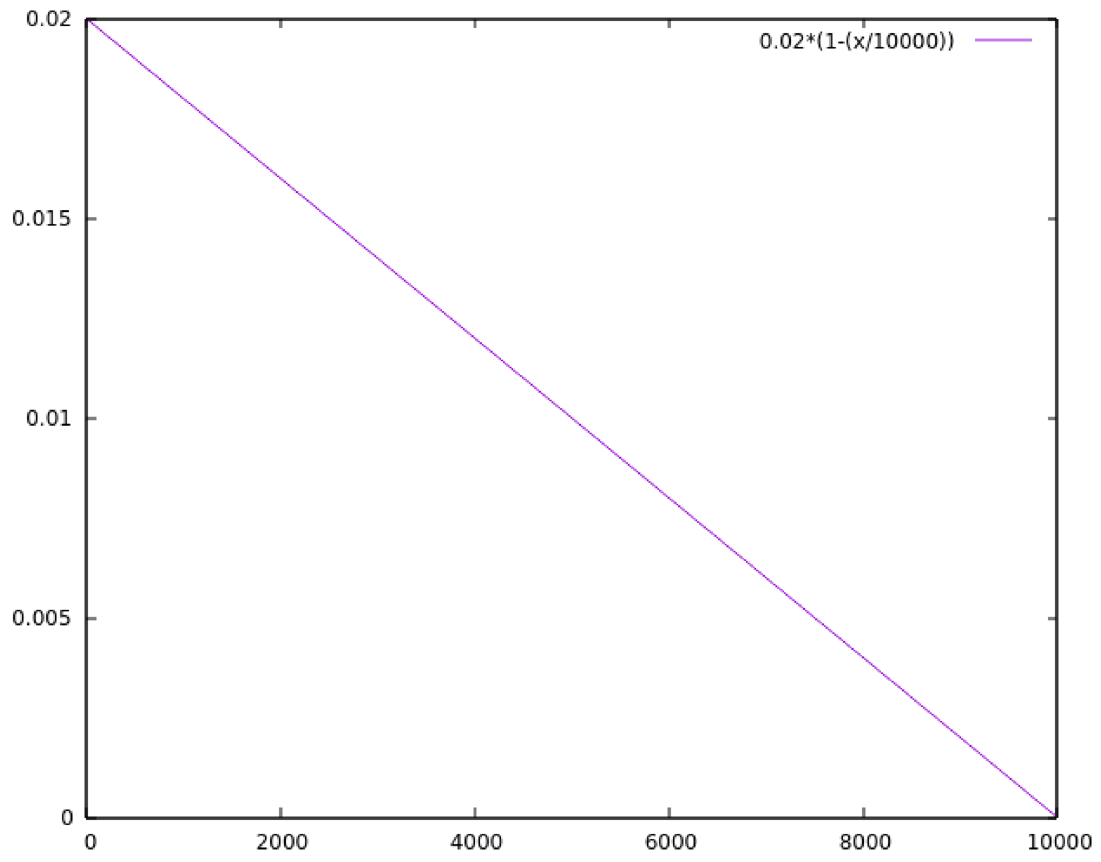


Figure 6. Linear Learning Rate function

In Figure 6 above you can see the linear function that the learning rate follows. On the upper right corner you can see the expression for the 10000 iterations. On the x axis lie the iterations and on the y axis the value of the learning rate.

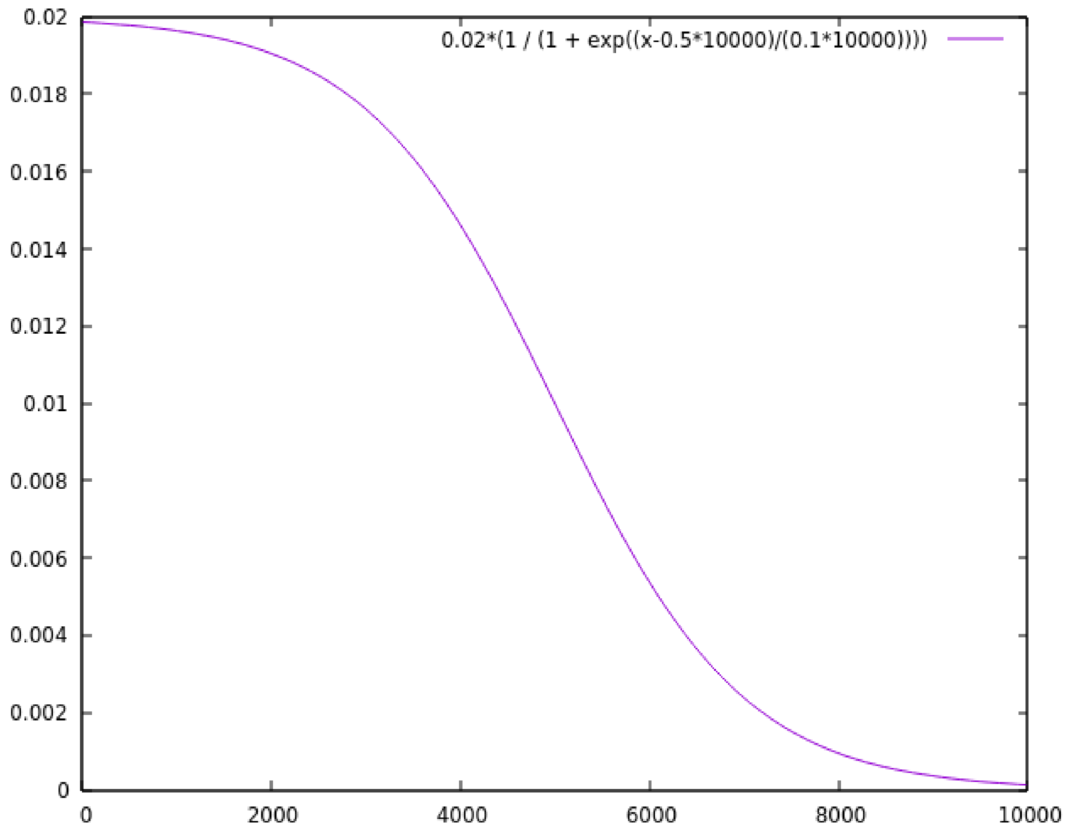


Figure 7. Sigmoid Learning Rate function

In Figure 7 above you can see the linear function that the learning rate follows. On the upper right corner you can see the expression for the 10000 iterations. On the x axis lie the iterations and on the y axis the value of the learning rate. You can notice as well, that the inclination of this sigmoid function compared to the one shown previously for the exploration rate, is less steep. This is because we want the learning rate to decrease more slowly compared to the exploration rate.

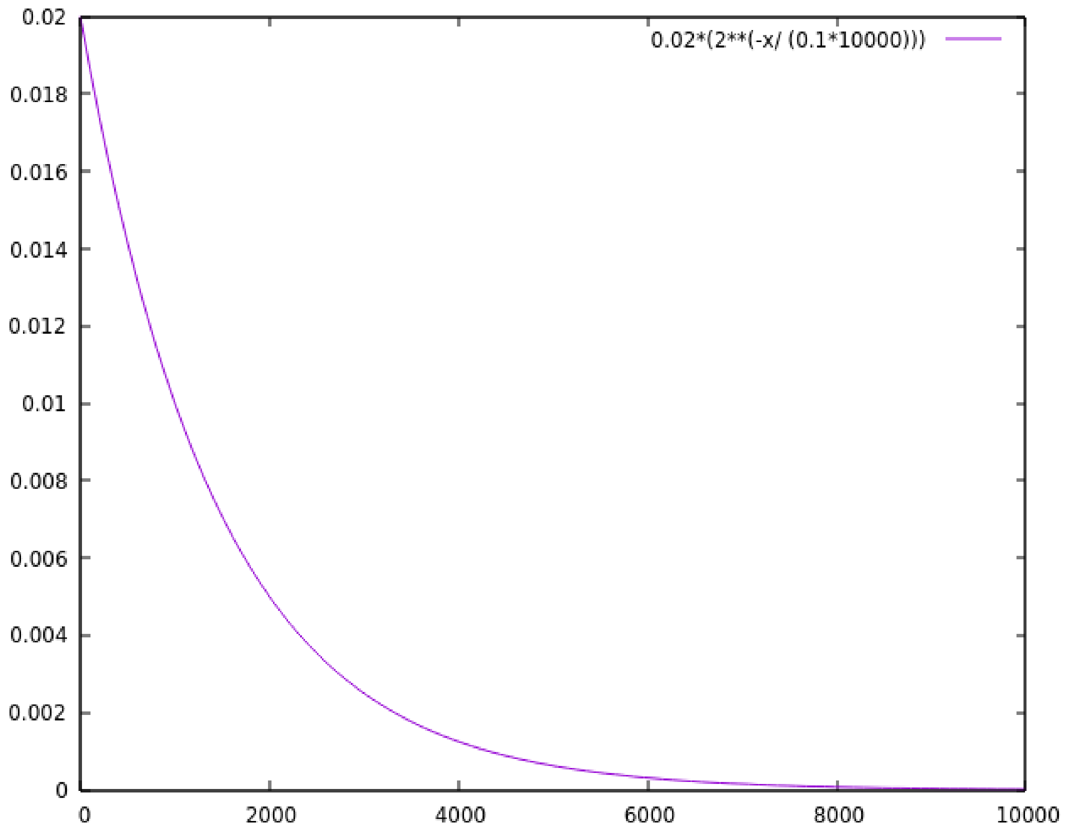


Figure 8. Exponential Learning Rate function

In Figure 8 above you can see the exponential function that the learning rate follows. On the upper right corner you can see the expression for the 10000 iterations. On the x axis lie the iterations and on the y axis the value of the learning rate.

You will notice that in all three graphs shown above the starting value of the learning rate is 0.02. This value has been accrued after extensive experimentation and is the maximum that can take as an initial value, otherwise the weight values, and therefore the Q-values, become unstably large and diverge. For similar reasons, when using the exponential function the value of learning rate isn't allowed to drop below 0.01, because the agent practically learns nothing. We will also discuss the impact of each different function in the results section.

4.4 Reward Function

The agents use only information that is locally available to them to choose an action. The reward for agent i, denoted as $\mathbf{r}^{\mathbf{i}}$, is a function of its state $\mathbf{s}^{\mathbf{i}}$ and is based solely in our problem: covering the largest area while minimizing the energy consumption. It can be defined by:

$$r^i(s^i) = G^i(s^i) - C^i$$

Where:

- $G^i(s^i)$ corresponds to the gain of covering the cells in the area of agent i. This gain is a linear function of the number of cells that illustrates the concept that the more cells are covered, the better the solution is :
  $G^i(s^i)$= number_of_cells_covered(Area^i) × GAIN
- C^i represents a cost (energy consumption) resulting from the previous action of agent i. We used :

$$C^i = \begin{cases} 0, & \text{if action=0;} \\ \text{COST\_LOW}, & \text{if action=1;} \\ \text{COST\_HIGH}, & \text{if action=2;} \end{cases}$$

Where COST_LOW < COST_HIGH, since the low sensing mode consumes less energy than the high mode. In our simulations GAIN is fixed at 0.2, COST_LOW at 0.8, and COST_HIGH at 3.0, following the choices in the literature. By changing the COST_LOW and COST_HIGH settings we can change the equilibria of the problem, however, we did not experiment towards that direction.

Moreover, we can extend this logic to the global map of the simulation and in the same way we can compute a global reward for the policy of the agents which allows us to quantify the global value of the policy. This is the main way for us to compare policies, as we will see in the results.

4.5 Initial States and Necessary Information

Before referring to the actual process of the simulation and how it progresses, we should first mention the starting conditions and some of the information we require to have prior to the beginning of the simulation.

Firstly, we need to know the topology of the map and the number of the motes taking part. We also assume we know the centers of the motes' position. And lastly, we need the dimensions of the grid. These parameters can be easily changed. However, they are set at the beginning of the simulation and are held fixed throughout each simulation. The initial state of the simulation, regardless of the map topology or the number of nodes, is for all motes to be turned off.

It is also worth mentioning that the discount factor γ is fixed at 0.99 and the weight factors of the basis functions are initialized to 1.

4.6 Process of the Simulation

At first, there is the initialization of the radio and the motes. Then, the routing phase follows which is a way for the motes to synchronize their active time. More specifically, a timer is called for every mote which determines when a given mote will wake up to sense and to take a new action.

Our simulation, as mentioned before, is consisted of 10000 repetitions per mote. This practically means that each mote chooses 10000 actions totally until the end of the simulation. However, each of these cycles is divided into two separate rounds.

In the first round, every mote chooses an action based on the current state. As we said before, this action can either be random, following the exploration rate function, or is selected greedily as the best action, corresponding to the maximum Q-Value. However, the same condition applies to all motes, meaning that they either all choose random or they all choose greedy. Then, the global state changes depending on the actions chosen. Each mote receives its own reward, signaling the end of the first round.

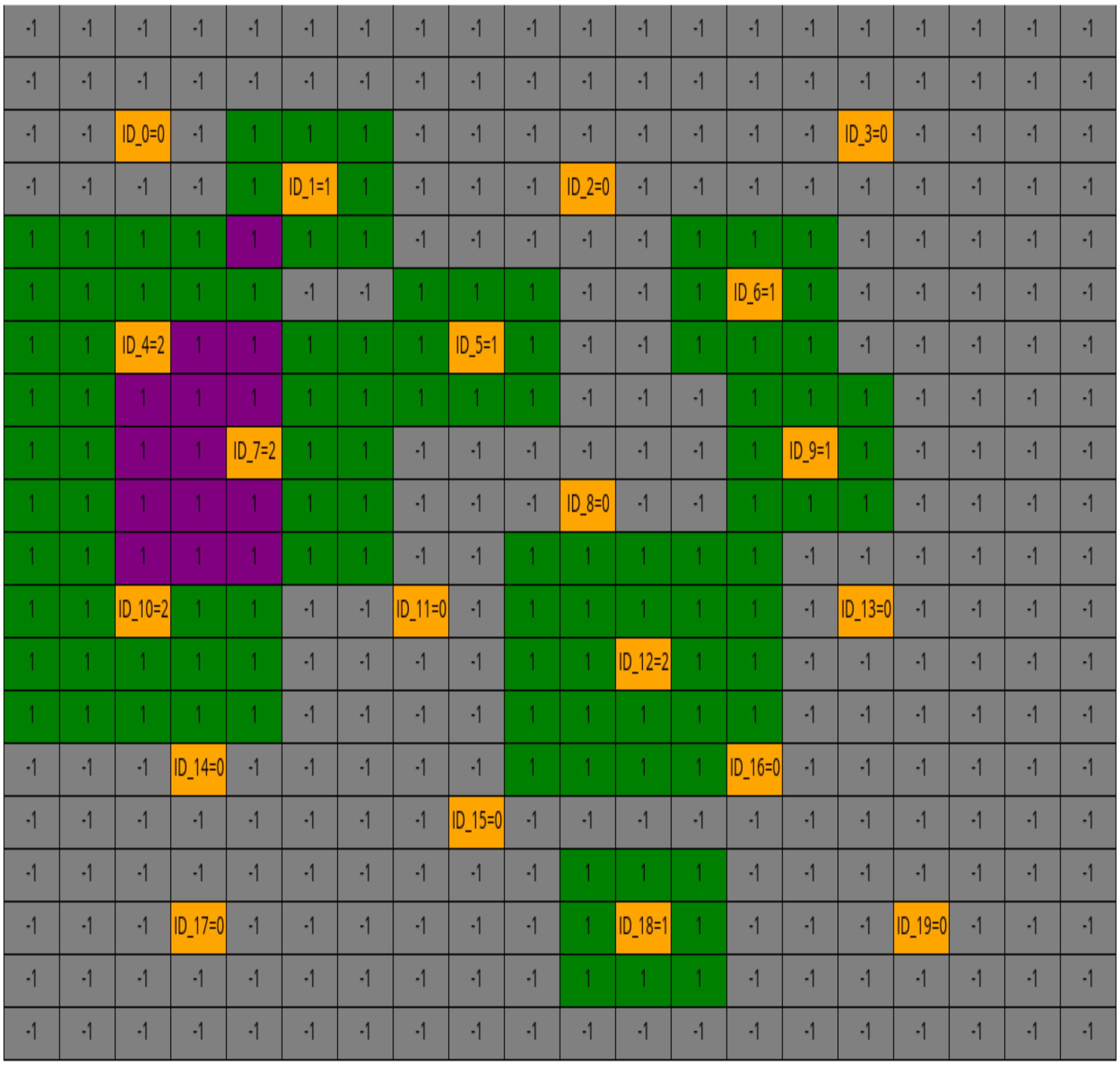
In the second round, the learning update takes place. Every node computes the necessary values, such as the maximum Q-Value of the next state, which is now the current state since the global state has already changed, and updates the weight factors of the basis functions. This concludes the second round.

The simulation continues in the same way, following these two rounds in an alternating manner thereafter.

4.7 Moving to larger grids with more Motes and random Topology

As stated before, after solving the 10×10 map problem with five motes, the idea was to try the fully distributed Q-Learning algorithm on larger grids with more motes. However, in more complicated topologies with a larger number of nodes, it is not possible to know the optimal policy beforehand (if there is, in fact, one) and there is also no guarantee that convergence will lead to optimal policies. Nevertheless, we can compare the global reward of the resulting policies to the initial policies to figure out that learning indeed improves performance.

In our third and larger map, we chose a 20×20 map with twenty motes and we spread them randomly around the grid. Figure 9 shows a snapshot of the map and provides a general idea of the topology.



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	ID_0=0	-1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	ID_3=0	-1	-1	-1	-1
-1	-1	-1	-1	1	ID_1=1	1	-1	-1	-1	ID_2=0	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1	-1	-1	1	1	1	-1	-1	-1	-1	-1
1	1	1	1	1	-1	-1	1	1	1	-1	-1	1	ID_6=1	1	-1	-1	-1	-1	-1
1	1	ID_4=2	1	1	1	1	1	ID_5=1	1	-1	-1	1	1	1	-1	-1	-1	-1	-1
1	1	1	1	1	1	1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	-1
1	1	1	1	ID_7=2	1	1	-1	-1	-1	-1	-1	-1	1	ID_9=1	1	-1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	-1	ID_8=0	-1	-1	1	1	1	-1	-1	-1	-1
1	1	1	1	1	1	1	-1	-1	1	1	1	1	1	-1	-1	-1	-1	-1	-1
1	1	ID_10=2	1	1	-1	-1	ID_11=0	-1	1	1	1	1	1	-1	ID_13=0	-1	-1	-1	-1
1	1	1	1	1	-1	-1	-1	-1	1	1	ID_12=2	1	1	-1	-1	-1	-1	-1	-1
1	1	1	1	1	-1	-1	-1	-1	1	1	1	1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	ID_14=0	-1	-1	-1	-1	-1	1	1	1	1	ID_16=0	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	ID_15=0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	ID_17=0	-1	-1	-1	-1	-1	-1	1	ID_18=1	1	-1	-1	-1	ID_19=0	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure 9. Snapshot of 20×20 grid

CHAPTER 5. Results

As done in the previous chapter, we will begin with the results of the small topologies first and gradually move towards the large one. Table 1 holds the results of the percentages of convergence to optimal policies for the grids 6 × 10 and 10 × 10, when using our algorithm with the different sizes of the basis functions vector and the different learning rate schedules.

		6 × 10	10 × 10
Basis F : 22	Linear	76 %	69 %
Basis F : 22	Sigmoid	28 %	33 %
Basis F : 22	Exponential	15 %	15 %
Basis F : 76	Linear	76%	75 %
Basis F : 76	Sigmoid	20 %	17 %
Basis F : 76	Exponential	51 %	35 %
Basis F : 97	Linear	**100 %**	**98 %**
Basis F : 97	Sigmoid	74 %	67%
Basis F : 97	Exponential	**100 %**	**96%**

Table 1. Percentages of successful convergence to optimal policies

As one can see in the table above, the use of the set of 97 basis functions produces significantly better results than the reduced ones. Moreover, you can notice that generally the linear and exponential learning rate schedules are performing a lot better than the sigmoid one, with the exception of the first case, where the sigmoid schedule is slightly better than the exponential one.

Subsequently, we will present some graphs from the results of our simulations using the complete set of basis functions, namely the set of 97 basis functions. Regarding, our smallest grid, the 6 × 10 grid, we present results on global reward performance and convergence of the learned weights. The optimal policy for the 6 × 10 map, as mentioned before, is 2-0-2, which you can see in Figure 10 shown below.

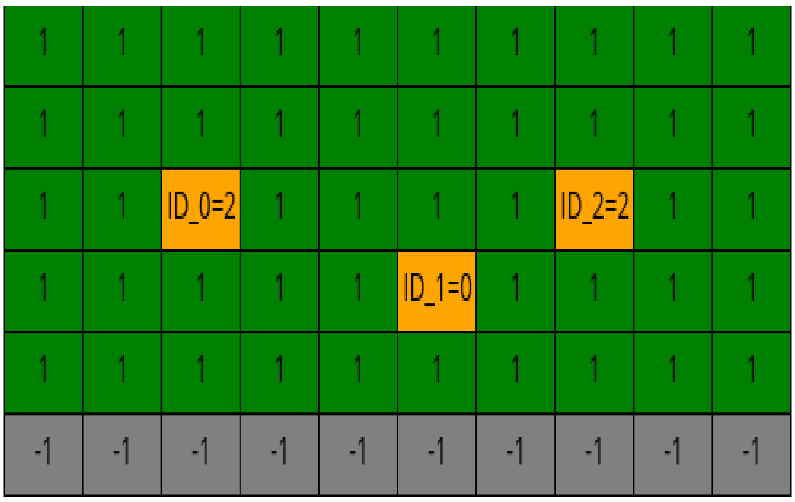


Figure 10. Optimal policy for 6 × 10 map

Using sigmoid learning rate for 6×10 grid

Our algorithm converges to the optimal policy 74% of the time, as shown already in Table 1. The graph of the global reward over time applying the sigmoid learning rate is shown in Figure 11 below.

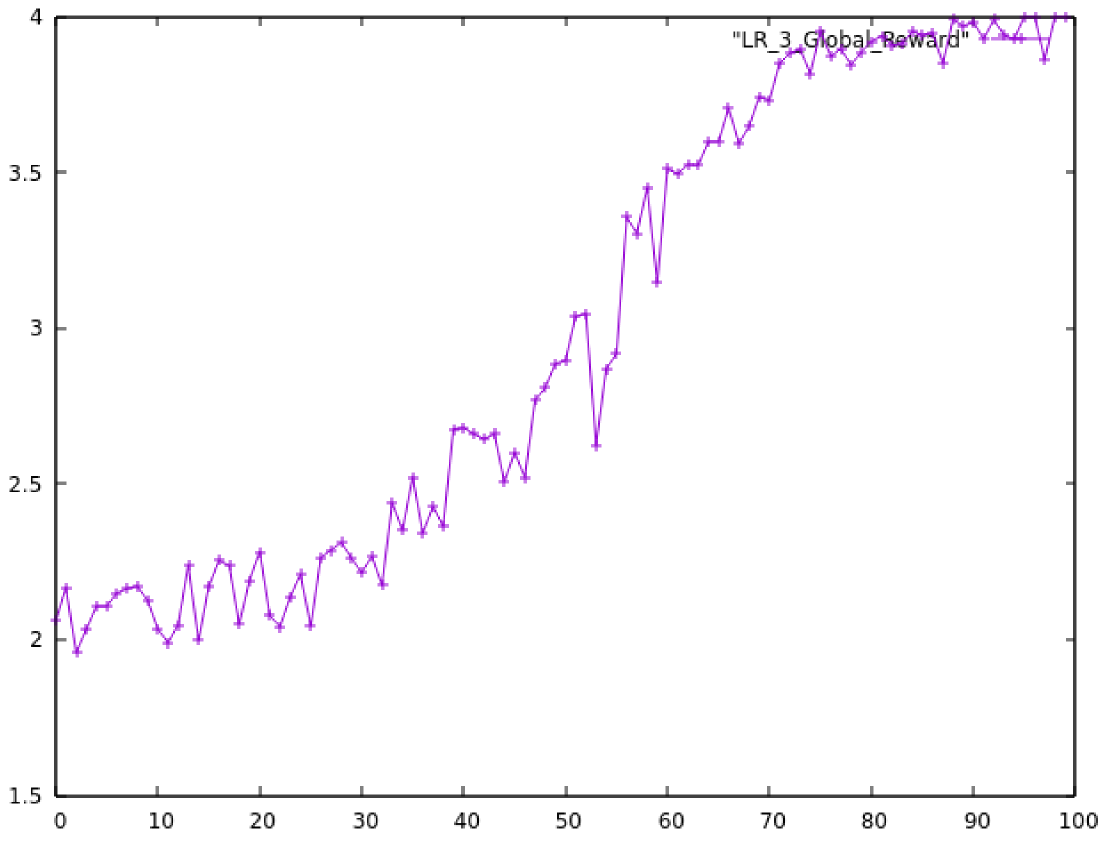


Figure 11. Global Reward over time (sigmoid learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see, the maximum global reward achievable is 4. This graph corresponds to a total of 10000 iterations. However, it has been smoothed by taking the average value per 100 iterations for the whole run, thus 100 values, because the initial graph using all the values was too spiky. You can notice the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information.

We also computed the L_1 norm of the weight factors for every node, in order to check for convergence of the weights to specific values. The L_1 norm is the summation of the absolute values of the weight factors. In Figure 12 below you can see the combined graph of the L_1 norm for the three nodes.

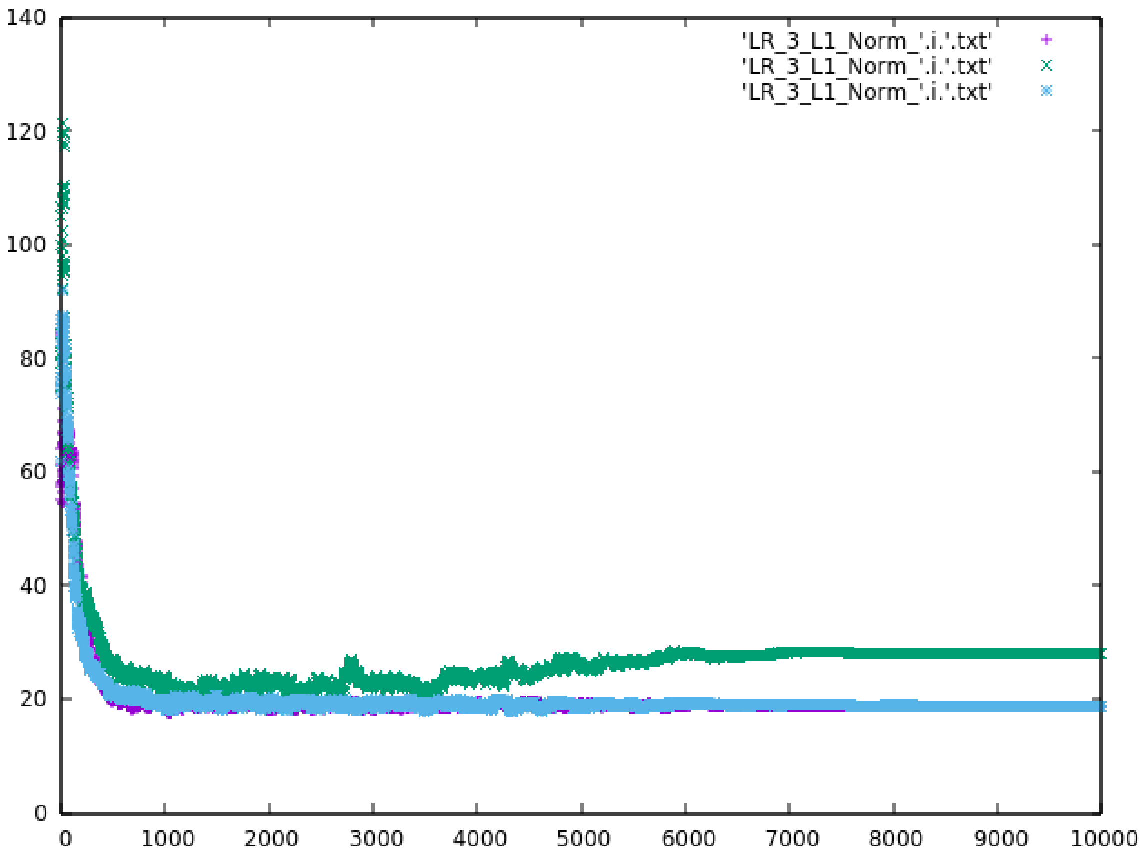


Figure 12. L1 Norm for 6×10 with sigmoid learning rate

On the y axis lies the value of L_1 norm, while on the x axis are the iterations. The purple graph corresponds to Mote with id 0, the green graph corresponds to Mote with id 1 and the light blue graph corresponds to Mote with id 2. You can notice that Mote 0 and Mote 2 are practically converging to the same value, which makes a lot of sense, since the conditions of their positions are a lot alike.

Using linear learning rate for 6×10 grid

Our algorithm converges to the optimal policy 100% of the times in this case, as shown already in Table 1. The graph of the global reward over time applying linear learning rate schedule is shown in Figure 13.

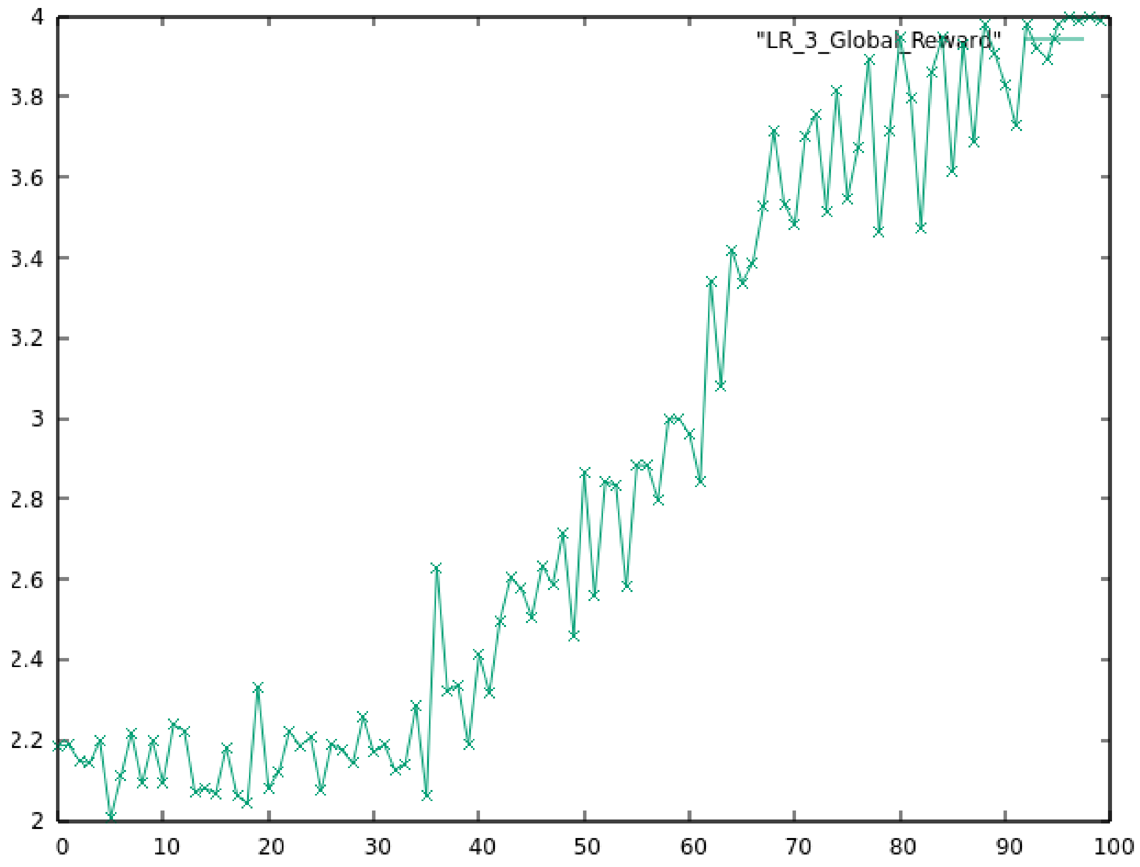


Figure 13. Global Reward over time (linear learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see the maximum global reward achievable is 4. Similarly to the sigmoid one, this graph corresponds to a total of 10000 iterations. However, it has been smoothed by taking the average value per 100 iterations for the whole run, thus 100 values, because the initial graph using all the values was too spiky. You can notice the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information.

We also computed the L_1 norm of the weight factors for every node, in order to see whether or not we had convergence. In Figure 14 below you can see the combined graph of the L_1 norm for the three nodes.

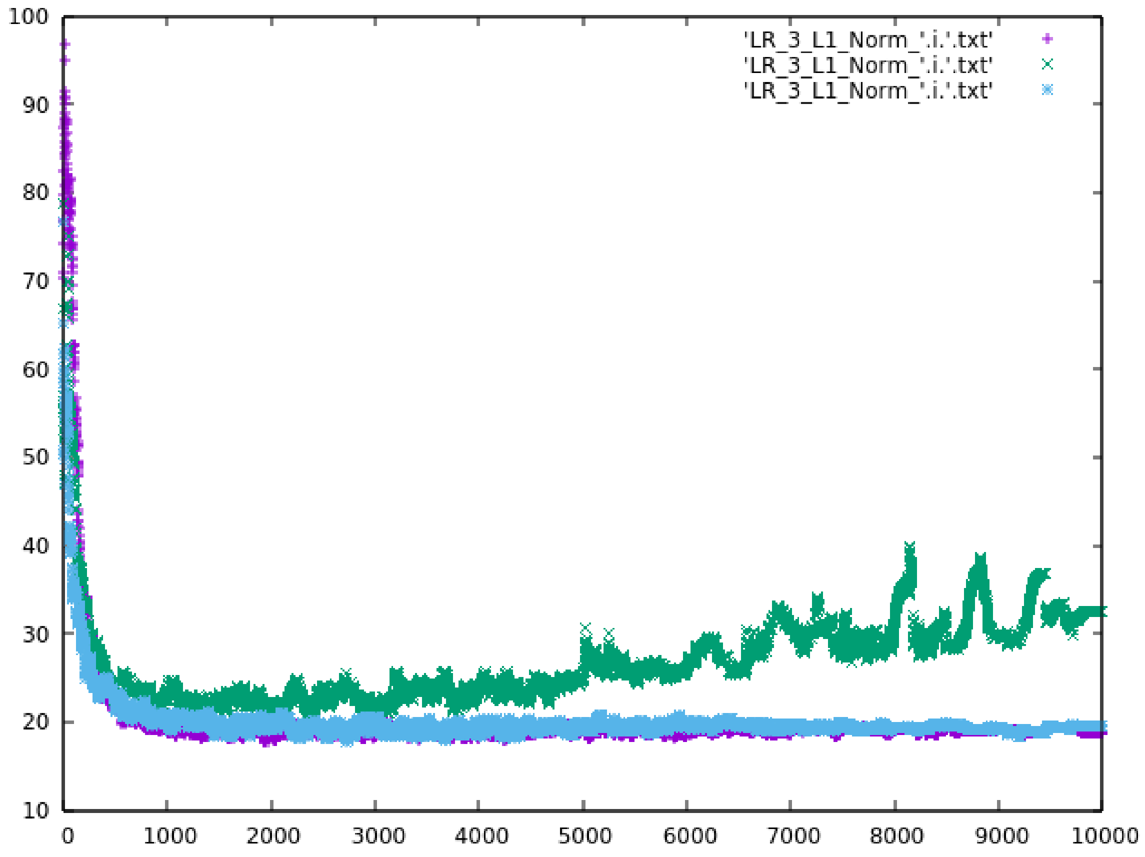


Figure 14. L1 Norm for 6×10 with linear learning rate

On the y axis lies the value of L_1 norm, while on x axis are the iterations. The purple graph corresponds to Mote with id 0, the green graph corresponds to Mote with id 1 and the light blue graph corresponds to Mote with id 2. Similarly to the sigmoid graph for L1 norm, you can notice that Mote 0 and Mote 2 are practically converging towards the same value, which makes a lot of sense, since their conditions are a lot alike.

Using exponential learning rate for 6×10 grid

Our algorithm converges to the optimal policy 100% of the time, as we saw already in the Table 1 above. The graph of the global reward over time using the exponential learning rate schedule is shown in Figure 15 below.

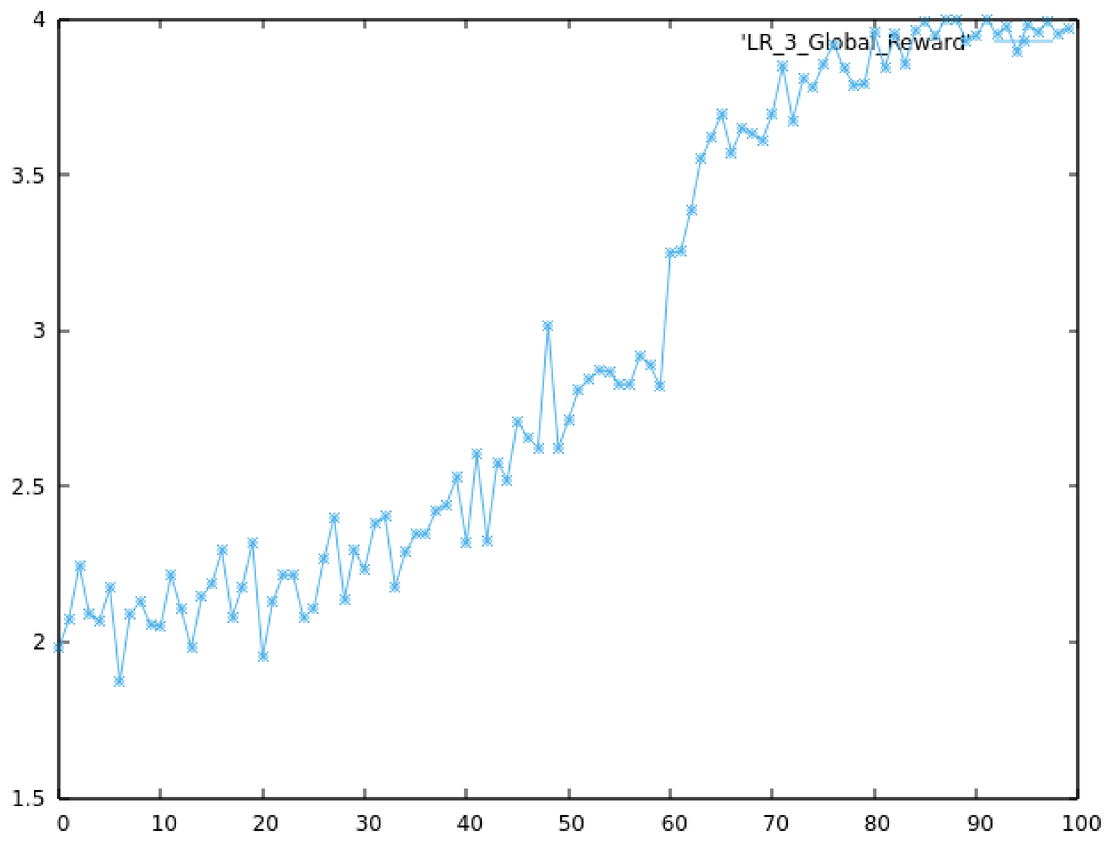


Figure 15. Global Reward over time (exponential learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see the maximum global reward achievable is 4. Similarly to the sigmoid one, this graph corresponds to a total of 10000 iterations. However, it has been smoothed once again by taking the average value per 100 iterations for the whole run, thus 100 values, because the initial graph using all the values was too spiky. You can notice the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information.

We also computed the L_1 norm of the weight factors for every node, in order to check for convergence of the weights to specific values. In Figure 14 below you can see the combined graph of the L_1 norm for the three nodes.

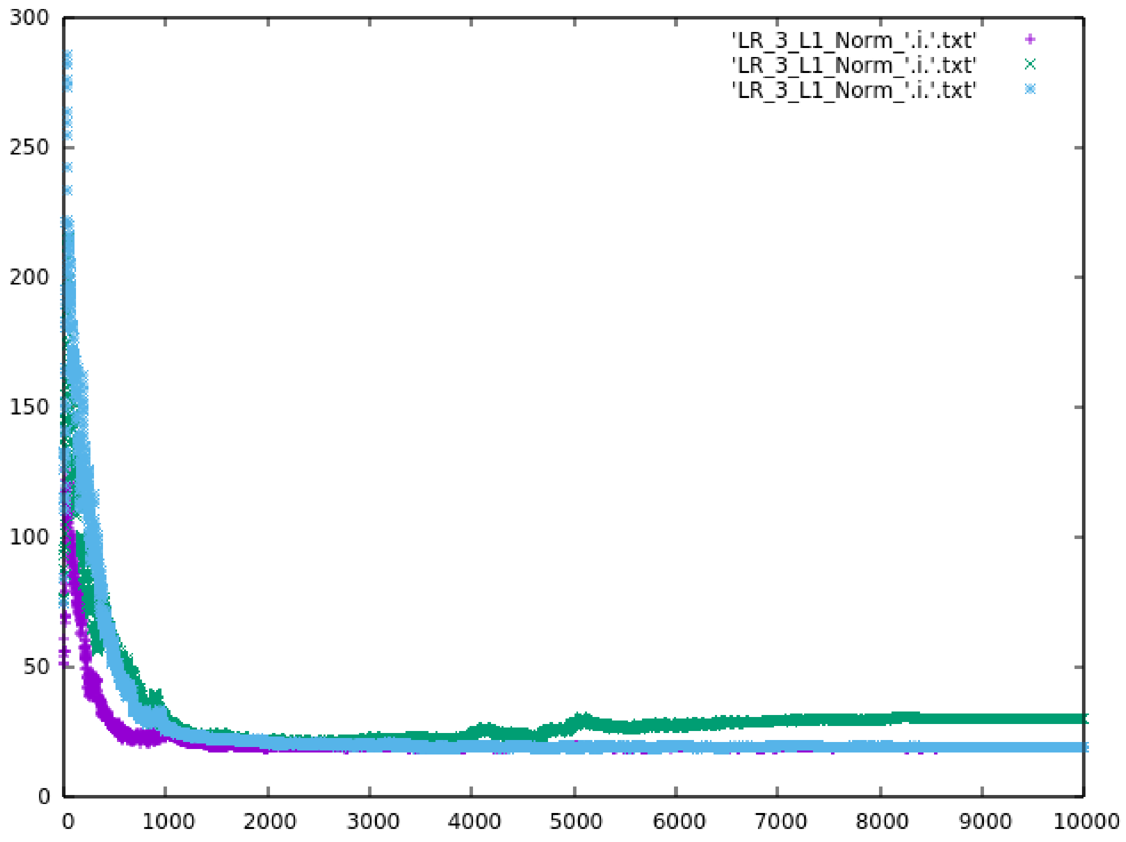
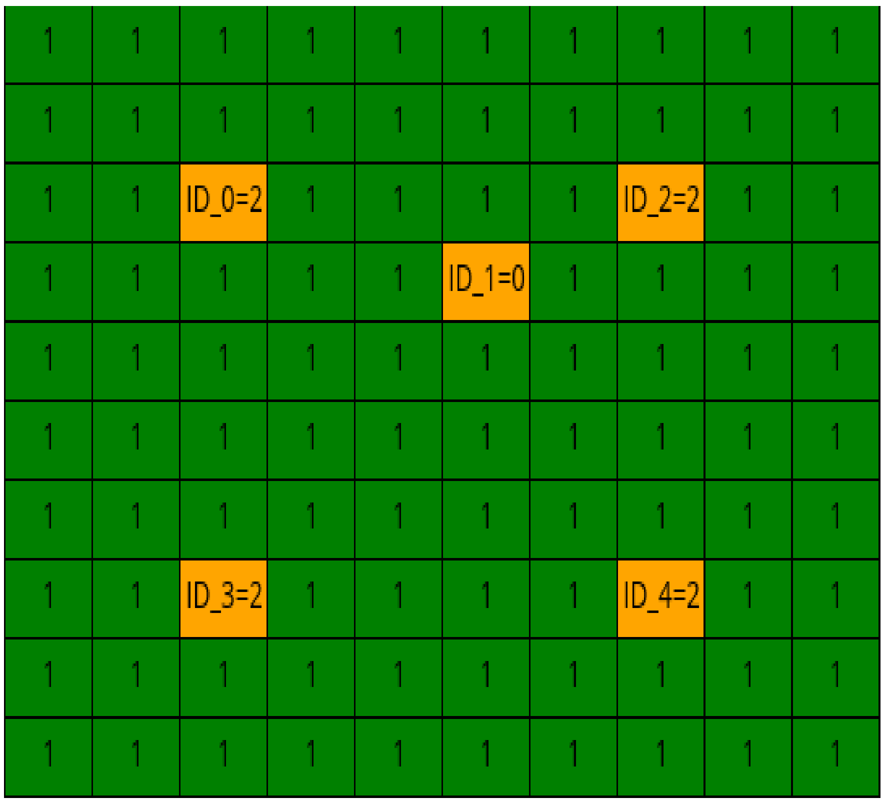


Figure 16. L1 Norm for 6×10 with exponential learning rate

On the y axis lies the value of L_1 norm, while on x axis are the iterations. The purple graph corresponds to Mote with id 0, the green graph corresponds to Mote with id 1 and the light blue graph corresponds to Mote with id 2. Similarly to the sigmoid graph for the L1 norm, you can notice that Mote 0 and Mote 2 are practically converging towards the same value, which makes a lot of sense, since their conditions are a lot alike. Moreover, you can see that the values of the L_1 norm are quite higher than in the previous graphs. This is because of the exponential nature of the function. The weights in the first stages of the simulations, where the learning rate is higher, are taking high values and thus the L_1 norm is also higher as a result.

Moving on to the 10x10 map the results are the following. The optimal policy for this map is 2-0-2-2-2. This means everyone, but Mote 1, which is turned off, is operating on High mode, as you can see in Figure 17. This policy returns the maximum global reward which is 8.



1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	ID_0=2	1	1	1	1	ID_2=2	1	1
1	1	1	1	1	ID_1=0	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	ID_3=2	1	1	1	1	ID_4=2	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Figure 17. Optimal policy for 10 × 10 grid

As you can see with this policy, every cell is sensed exactly by one mote. Mote 1 is not needed at all, since every cell can be covered by the other motes and therefore is turned OFF.

Using sigmoid learning rate for 10×10 grid

Our algorithm converges to the optimal policy 67% of the times. The graph of the global reward over time is shown in Figure 18 below.

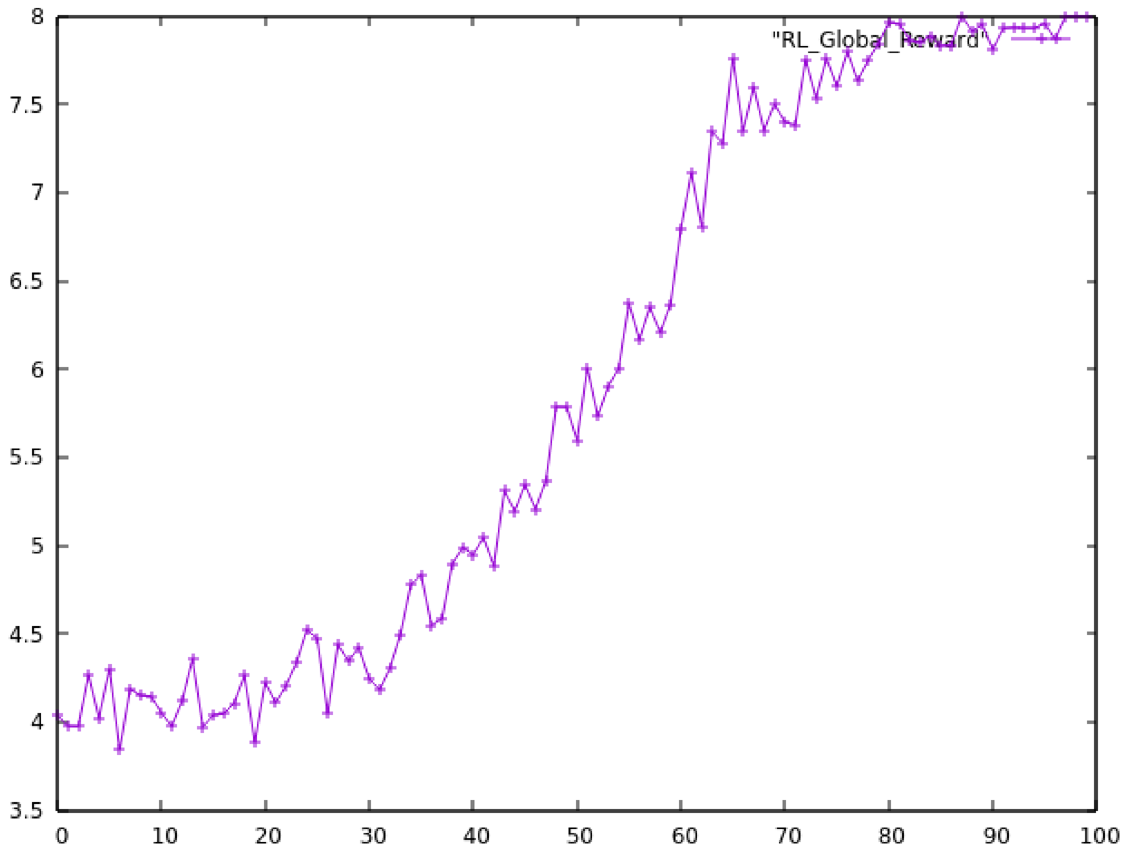


Figure 18. Global Reward over time for 10×10 map (sigmoid learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see the maximum global reward available is 8 and is achieved at the end of the iterations. Similarly to the sigmoid one, this graph corresponds to a total of 10000 iterations. However, it has been smoothed once again by taking the average value per 100 iterations for the whole run, thus 100 values, because the initial graph using all the values was too spiky. You can notice the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information.

We also computed the L_1 norm of the weight factors for every node, in order to check for convergence of the weights to specific values. In Figure 19 below you can see the combined graph of the L_1 norm for the five nodes.

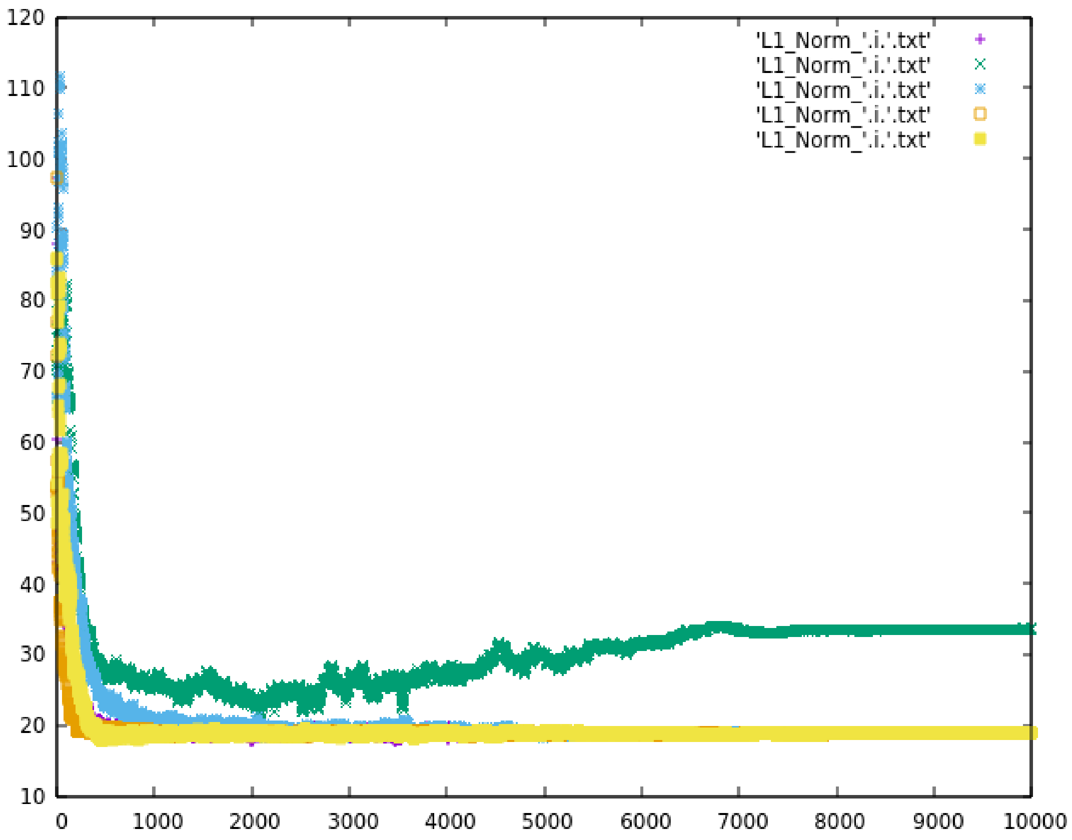


Figure 19. L1 Norm for 10×10 map with sigmoid learning rate

On the y axis lies the value of L_1 norm, while on the x axis are the iterations. The purple graph corresponds to Mote with id 0, the green graph corresponds to Mote with id 1, the light blue graph corresponds to Mote with id 2, the orange graph corresponds to Mote with id 3 and the yellow graph corresponds to Mote with id 4. You will also notice that the weight factor's behavior for Mote 1 is a lot different than the rest of the motes, which converge towards the same range of values due to the symmetry of their positions.

Using linear learning rate for 10×10 grid

Our algorithm converges to the optimal policy 98% of the times, as shown already in Table 1. The graph of the global reward over time is shown in Figure 20.

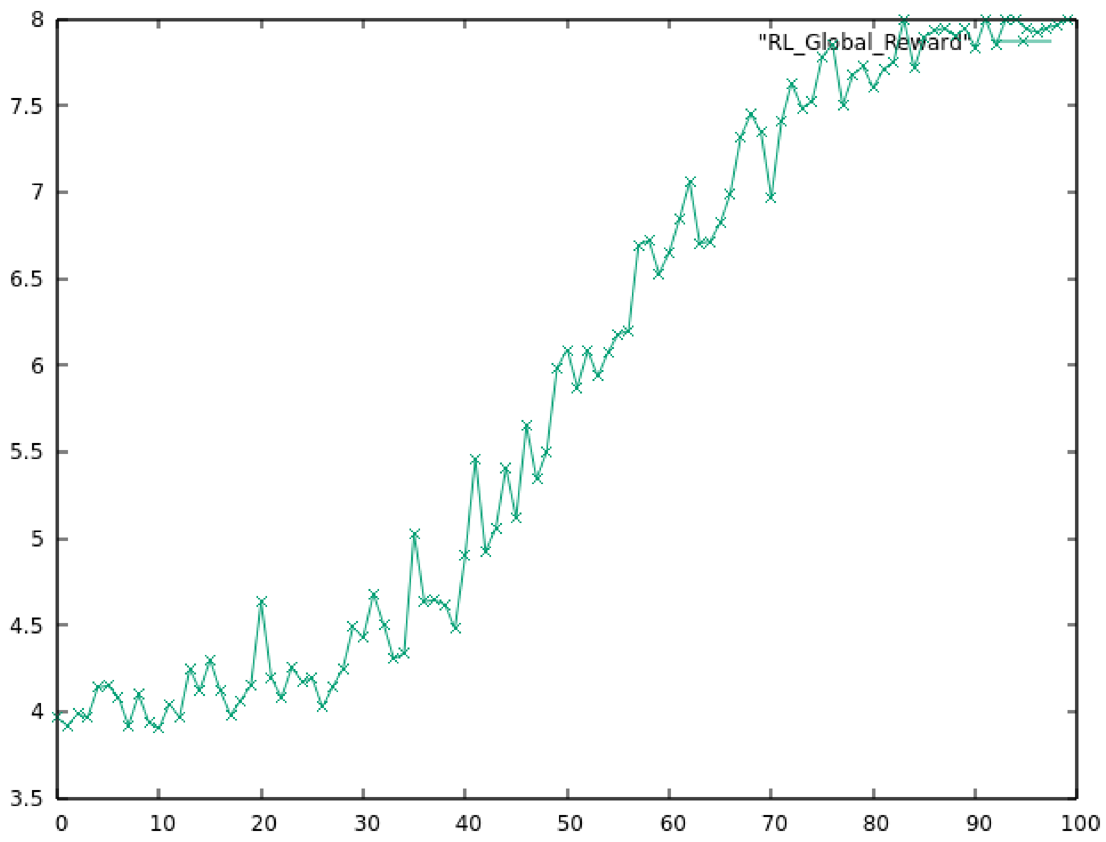


Figure 20. Global reward over time for 10×10 map (linear learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see the maximum global reward available is 8 and is achieved by the end of the iterations. Similarly to the previous graphs, this graph as well corresponds to a total of 10000 iterations. However, it has been smoothed once again by taking the average value per 100 iterations for the whole run, thus 100 values, because the initial graph using all the values was too spiky. You can clearly see the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information.

We also computed the L_1 norm as done previously so far. In Figure 19 below you can see the combined graph of the L_1 norm for the five nodes.

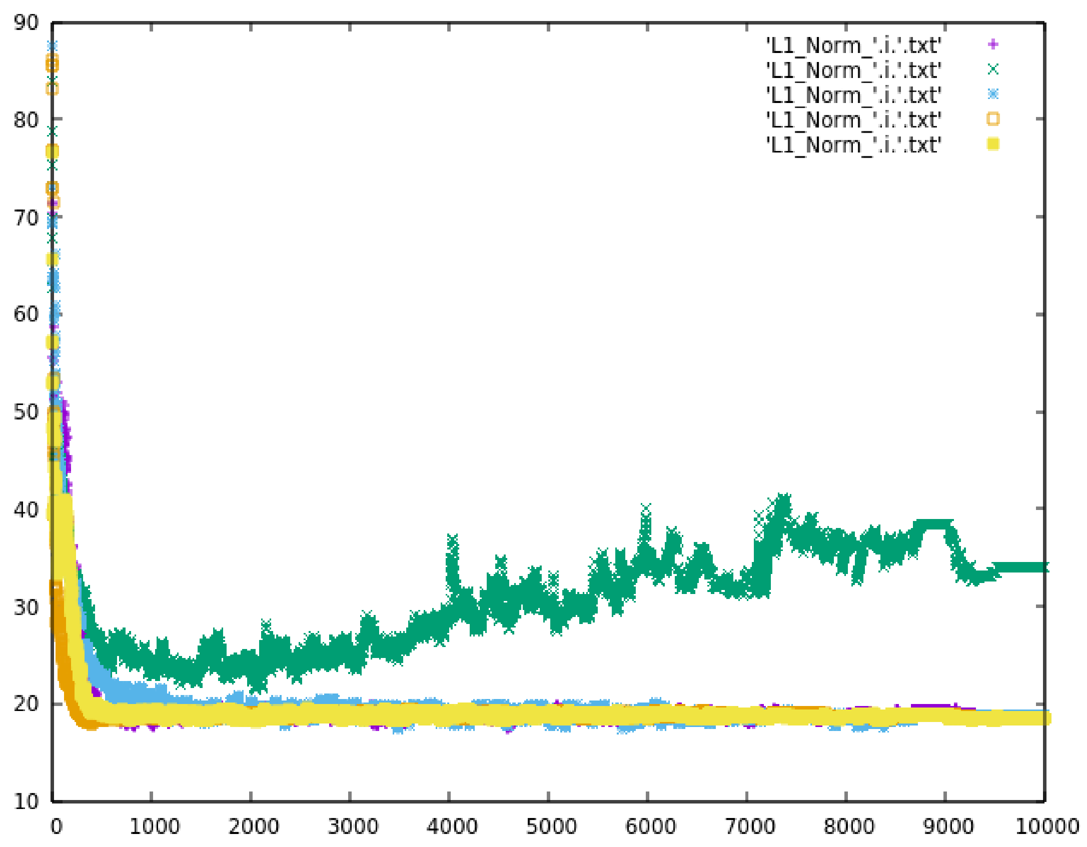


Figure 21. L1 Norm for 10×10 map with linear learning rate

On the y axis lies the value of L_1 norm, while on x axis are the iterations. The purple graph corresponds to Mote with id 0, the green graph corresponds to Mote with id 1, the light blue graph corresponds to Mote with id 2, the orange graph corresponds to Mote with id 3 and the yellow graph corresponds to Mote with id 4. You will also notice that the weight factor's behavior for Mote 1 is a lot different than the rest of the motes, which converge towards the same range of values due to the symmetry of their positions.

Using exponential learning rate for 10×10 grid

Our algorithm converges to the optimal policy 96% of the times, as mentioned in Table 1. The graph of the global reward over time is shown in Figure 22 below.

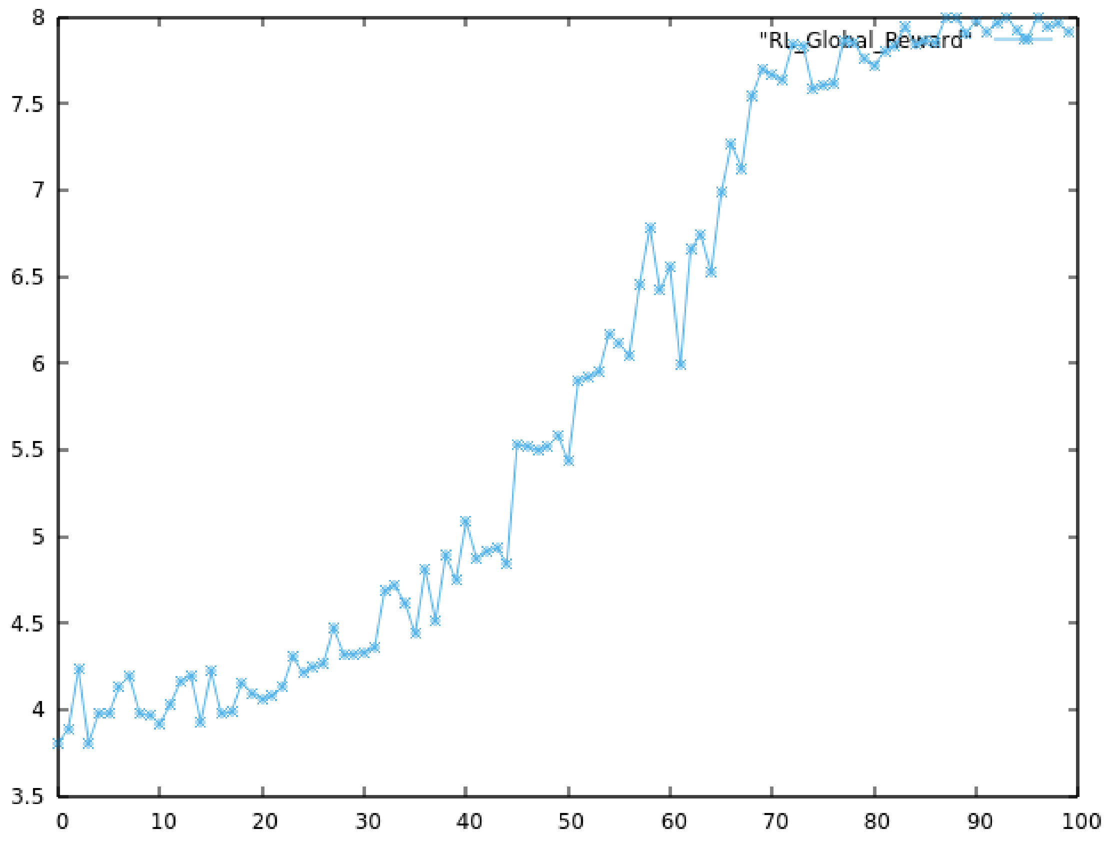


Figure 22. Global reward over time for 10×10 map (exponential learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see the maximum global reward available is 8 and is achieved by the end of the iterations. Similarly to the previous graphs, this graph as well corresponds to a total of 10000 iterations. However, it has been smoothed once again by taking the average value per 100 iterations for the whole run, thus 100 values, because the initial graph using all the values was too spiky. You can clearly see the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information.

We also computed the L_1 norm as done previously so far. In Figure 23 below you can see the combined graph of the L_1 norm for the five nodes.

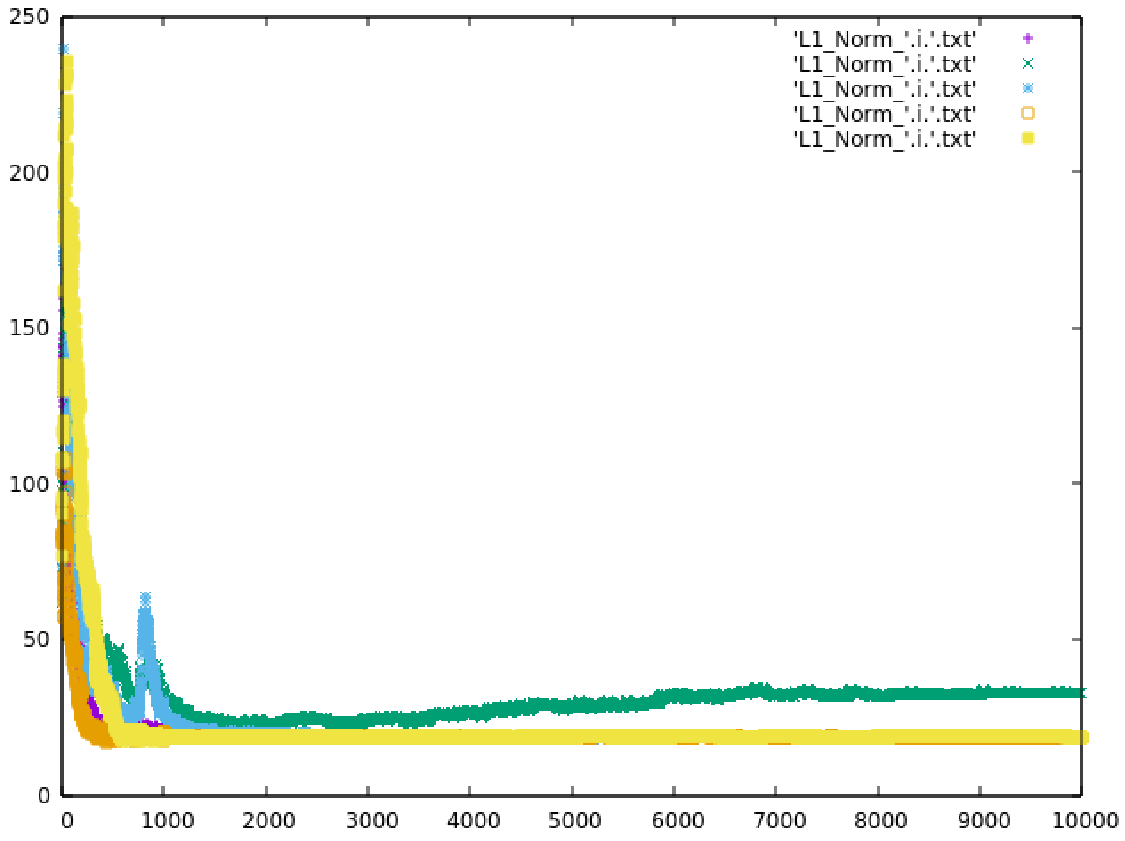
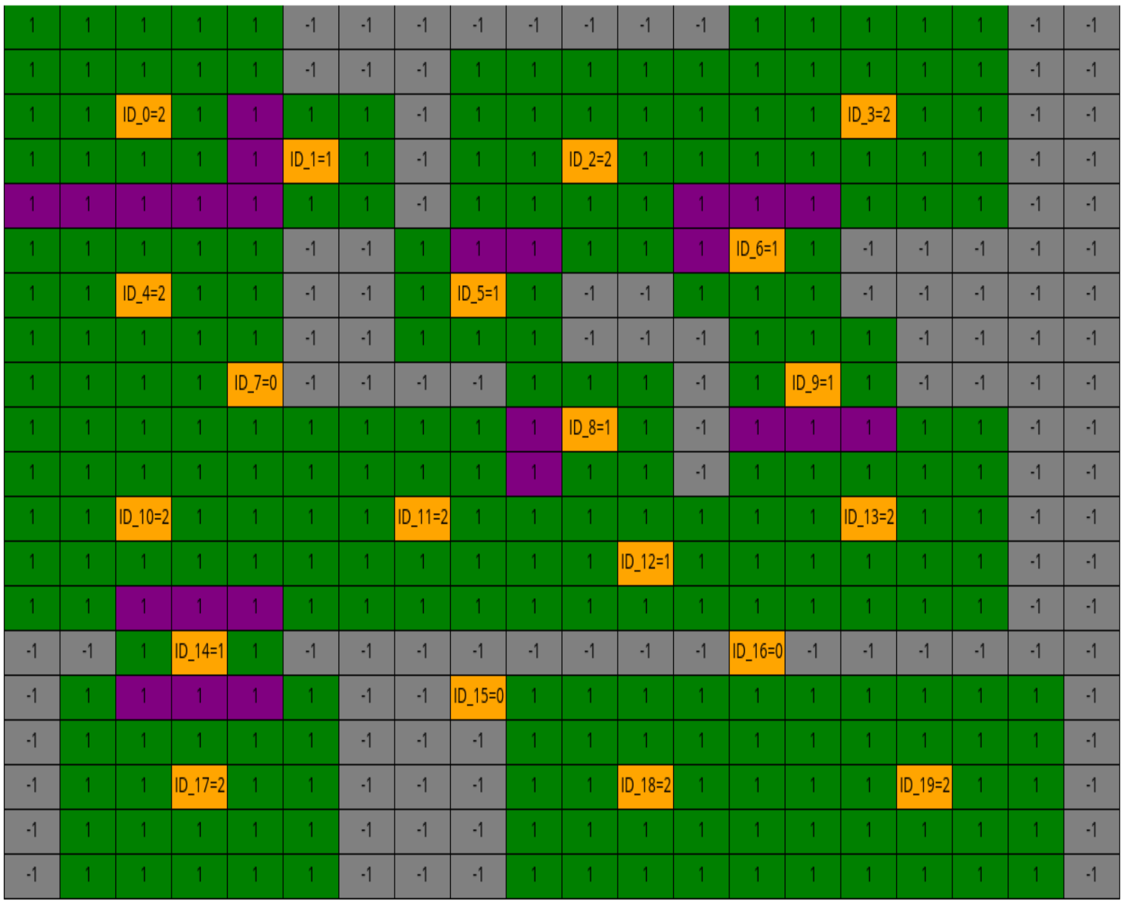


Figure 23. L1 Norm for 10 × 10 map with exponential learning rate

On the y axis lies the value of L_1 norm, while on x axis are the iterations. The purple graph corresponds to Mote with id 0, the green graph corresponds to Mote with id 1, the light blue graph corresponds to Mote with id 2, the orange graph corresponds to Mote with id 3 and the yellow graph corresponds to Mote with id 4. You will also notice that the weight factor's behavior for Mote 1 is a lot different than the rest of the motes, which converge towards the same range of values due to the symmetry of their positions.

Moving on to the final map, the 20×20 grid, we have the following results. Firstly, a typical policy convergence for all three learning rate schedules is the one depicted in Figure 24 below, achieving a global reward of 22.4. As we came to understand from the simulations we ran on this grid, there are several different policies that have very similar global reward.



1	1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	1	1	-1	-1
1	1	1	1	1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	-1	-1
1	1	ID_0=2	1	1	1	1	-1	1	1	1	1	1	1	1	ID_3=2	1	1	-1	-1
1	1	1	1	1	ID_1=1	1	-1	1	1	ID_2=2	1	1	1	1	1	1	1	-1	-1
1	1	1	1	1	1	1	-1	1	1	1	1	1	1	1	1	1	1	-1	-1
1	1	1	1	1	-1	-1	1	1	1	1	1	1	ID_6=1	1	-1	-1	-1	-1	-1
1	1	ID_4=2	1	1	-1	-1	1	ID_5=1	1	-1	-1	1	1	1	-1	-1	-1	-1	-1
1	1	1	1	1	-1	-1	1	1	1	-1	-1	-1	1	1	1	-1	-1	-1	-1
1	1	1	1	ID_7=0	-1	-1	-1	-1	1	1	1	-1	1	ID_9=1	1	-1	-1	-1	-1
1	1	1	1	1	1	1	1	1	1	ID_8=1	1	-1	1	1	1	1	1	-1	-1
1	1	1	1	1	1	1	1	1	1	1	1	-1	1	1	1	1	1	-1	-1
1	1	ID_10=2	1	1	1	1	ID_11=2	1	1	1	1	1	1	1	ID_13=2	1	1	-1	-1
1	1	1	1	1	1	1	1	1	1	1	ID_12=1	1	1	1	1	1	1	-1	-1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	-1
-1	-1	1	ID_14=1	1	-1	-1	-1	-1	-1	-1	-1	-1	ID_16=0	-1	-1	-1	-1	-1	-1
-1	1	1	1	1	1	-1	-1	ID_15=0	1	1	1	1	1	1	1	1	1	1	-1
-1	1	1	1	1	1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	-1
-1	1	1	ID_17=2	1	1	-1	-1	-1	1	1	ID_18=2	1	1	1	1	ID_19=2	1	1	-1
-1	1	1	1	1	1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	-1
-1	1	1	1	1	1	-1	-1	-1	1	1	1	1	1	1	1	1	1	1	-1

Figure 24. Highest global reward policy for 20×20

As mentioned earlier, we can't know the optimal policy for this grid, but we present the one policy our algorithm converged to most of the time, with one of the highest global reward accumulated. We display the global reward results for the three learning rate schedules.

Using sigmoid learning rate for 20×20 grid

The graph of the global reward over time is shown in Figure 25 below.

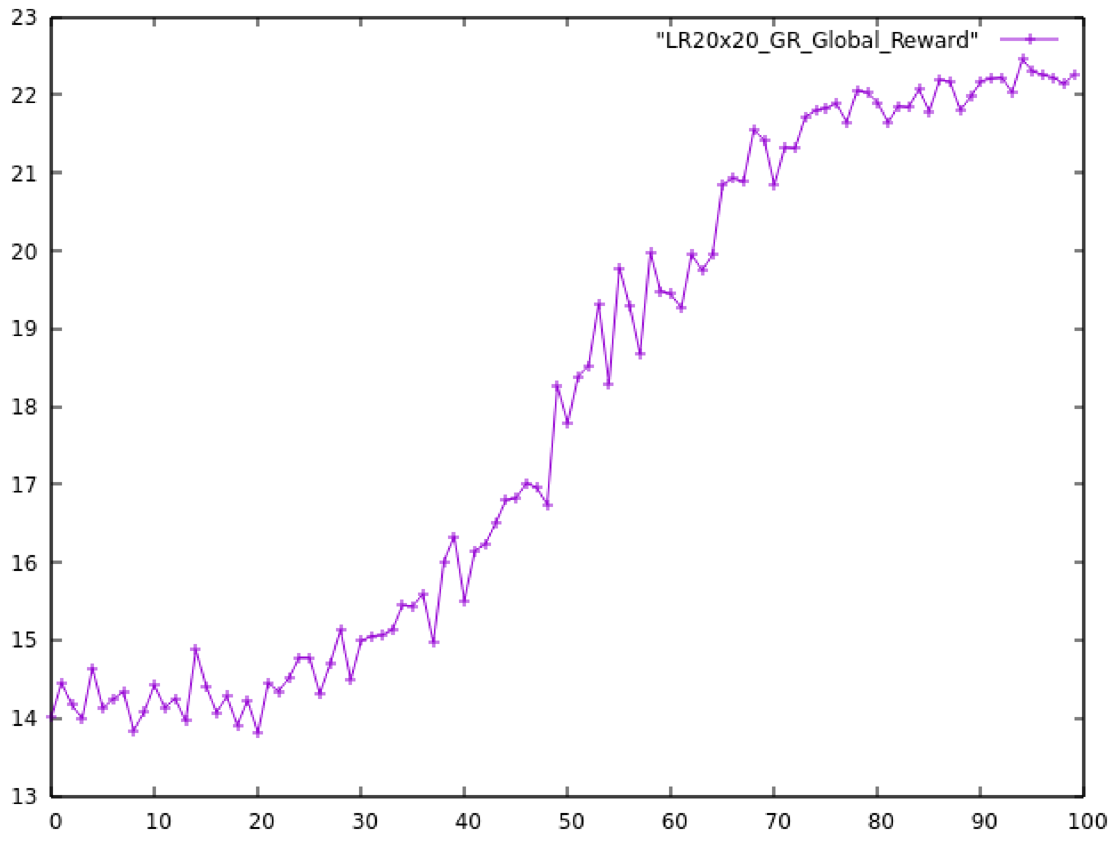


Figure 25. Global reward over time for 20×20 map (sigmoid learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see the global reward achieved by the end of the simulation is 22.4. Similarly to the previous graphs, this graph as well corresponds to a total of 10000 iterations. However, it has been smoothed once again by taking the average value per 100 iterations for the whole run, thus 100 values, because the initial graph using all the values was too spiky. You can clearly see the upward trend of the graph as the number of iterations increases and the agents choose more and more greedy having gathered more information.

Using linear learning rate for 20×20 grid

The graph of the global reward over time is shown in Figure 26 below.

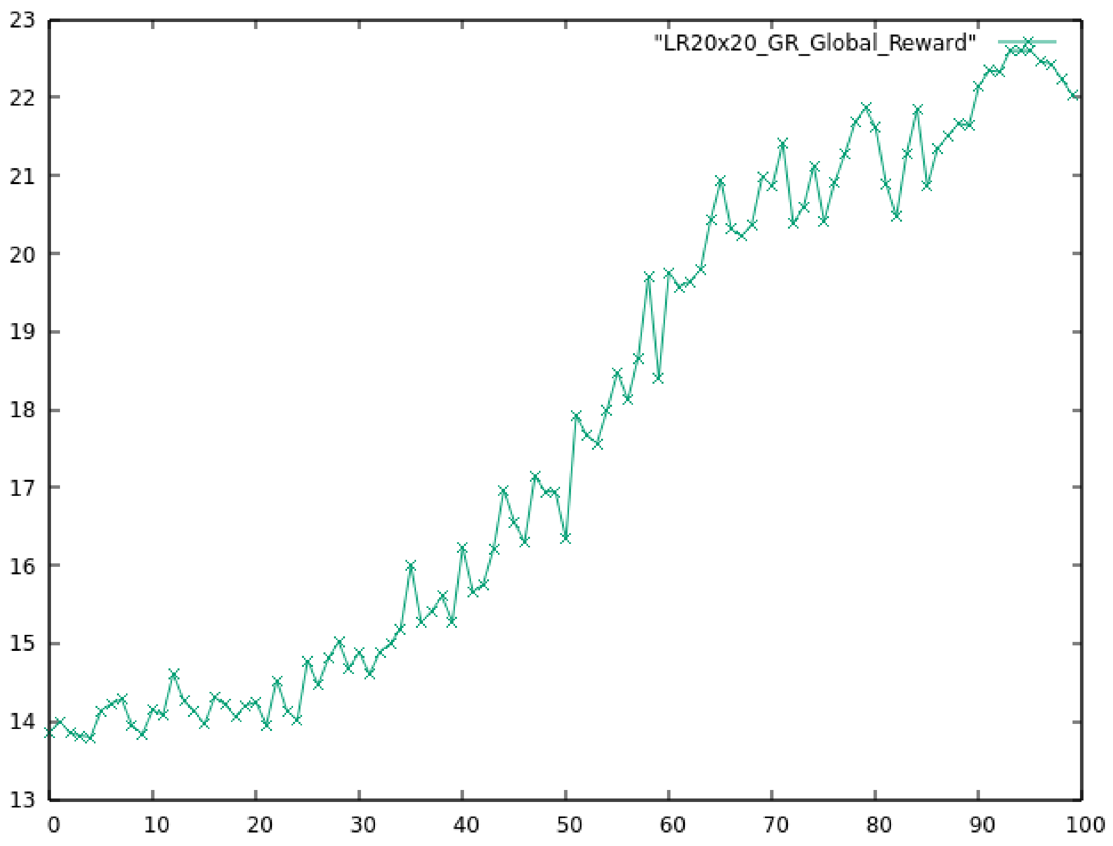


Figure 26 Global reward over time for 20×20 (linear learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. As you can see the global reward achieved by the end of the simulation is 22.4. However, due to the smoothening, mentioned before for the other graphs as well, the final global reward drops to 22. You can clearly see the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information.

Using exponential learning rate for 20×20 grid

The graph of the global reward over time is shown in Figure 27 below.

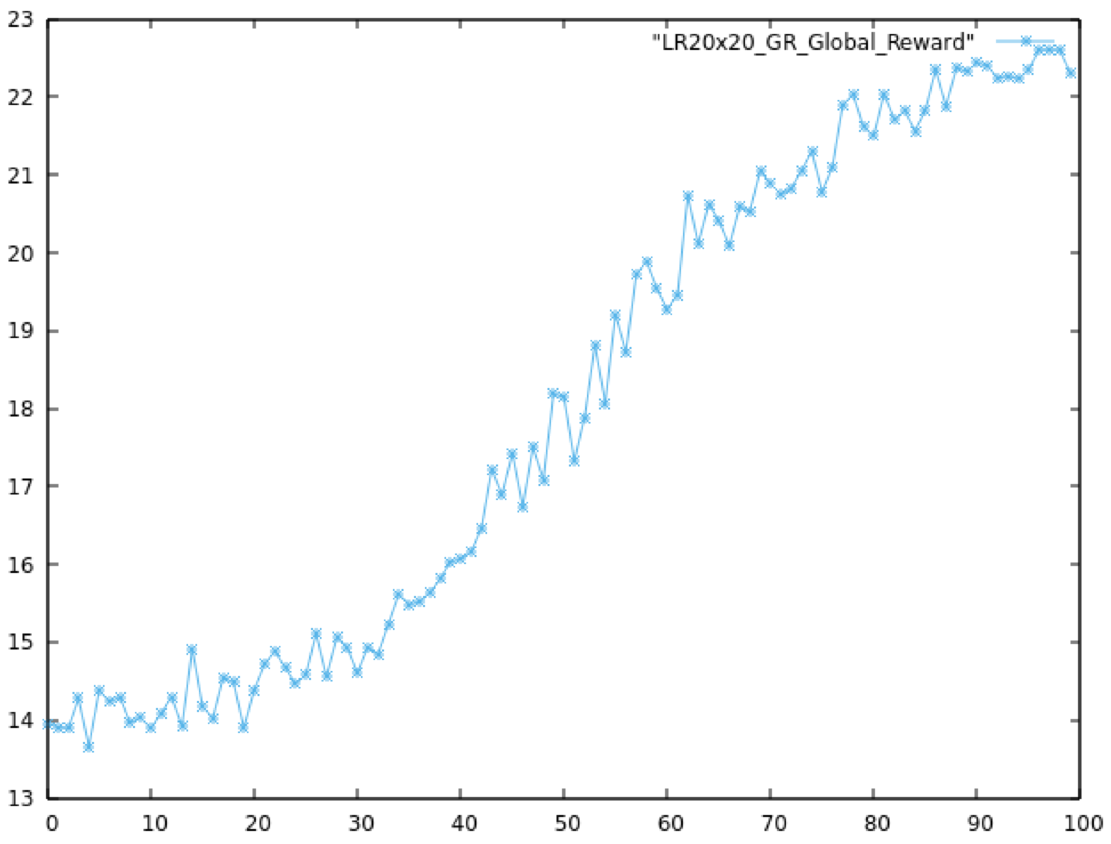


Figure 27. Global reward over time for 20×20 (exponential learning rate)

On the y axis is the value of the global reward and on the x axis are the iterations parts. You can clearly notice the upward trend of the graph as the number of iterations increases and the agents choose more and more greedily having gathered more information. As you can notice, the global reward schemas for the three functions are quite similar. However, as with the previous grids in this one as well, exponential and linear learning rate present a better solution for our algorithm. In other words, using exponential or linear learning rate we have higher probability of reaching a policy with higher global reward.

Using our algorithm, as shown in the graphs above, the global reward value achieved for this grid is a value between 22 and 23. If we used a naïve policy, for the purpose of comparison, the global reward value would be around eight. The naïve policy we mentioned is the one in which all the sensors choose action 2 regardless of their conditions, in order to maximize the number of cells covered.

Finally, I would like to point out that all of the succession converging percentages came from a lot of repetitions of the experiment in order to have a well-tested and as accurate as possible outcome. More specifically, for the two first grids we had 100 repetitions of the simulation for each of the cases. In the last grid, we had 40 repetitions of the simulation for each of the learning rates, due to fact that it was a lot more time consuming than in the previous smaller grids.

CHAPTER 6. Conclusions

6.1 Discussion

Our algorithm implemented for usage in our problem statement delivered a reasonable outcome. In the first two grids, we achieved convergence to the optimal policy in the vast majority of the simulation repetitions. In the last and largest grid, we managed to have a more than decent policy with an upward global rate function, that produced a quite higher global reward, when compared to the naïve policy.

Moreover, given the fact that our agents do not communicate throughout the simulation phase, the fact that the energy consumption for computation cost is small, and the fact that the memory requirements are restricted by using an approximation of the state, our algorithm is viable to be used on actual motes with the known limitations.

In fact, the whole idea of introducing reinforcement learning in the field of sensor networks proves to be very smart, especially for certain problems where the requisite outcome is not known in advance. In addition to that, reinforcement learning can work wonders in situations where the environment is altering or often unpredictable. Such a situation can often be found in sensor network applications.

However, it needs to be stated that every RL application or every sensor network application is very problem dependent. That means that its solution or its configuration would most of the times be very unique and specific, in order to address exactly the conditions and limitations of the problem.

6.2 Future Work

In our implementation of the algorithm the agents are not communicating with each other. Even though this does not hinder the results of the simulation, and in fact aids the cause of keeping energy consumption levels down, in bigger and more difficult problems this will prove unrealistic. To be honest, we had tried another algorithm in which the agents communicated with each other exchanging valuable information in order to cooperate for a better total outcome. However, the results were not as good, despite the fact that the agents were exchanging local information and thus they were consuming more energy.

It would be interesting to perceive the problem as a multi-agent system where agents communicate with each other and exchange important information and reshape this algorithm in order to receive better results, but still keep energy consumption to a viable level.

Moreover, there are still plenty of learning algorithms that could also be applied to our problem statement and it could prove very intriguing to test these out too.

Finally, the difficulty of the problem at hand can always be increased in order to evaluate smarter and more sophisticated learning algorithms were they applied.

6.3 Lessons

The amount of work needed to complete this thesis made it possible to understand the kinds of problems involved in large project management. First of all, there was a deeper understanding of how important it is to have proper organization and code infrastructure when working on a big project. Furthermore, in order to overcome this level of complexity you need to come up with a good plan and reserve yourself with a lot of patience.

Secondly, error correction provided me with a good idea of how debugging features work and an overall insight into what kinds of problem can be caused by what kind of errors.

Moreover, I acquired more experience on working with C++ and enriched my cognitive understanding about this language. In addition to that, I got in touch with Python for the first time and had the chance to appreciate the simplicity and power of the language.

What is often said in programming is that there has to be a flow of ideas that can be easily translated into code. This project helped me practice exactly this behavior, where an understanding of the problem caused a flurry of ideas to come forth and be translated into code.

All in all, it was a character molding experience that helped me build confidence in solving any kind of problem given the necessary concentration and effort.

Bibliography

[1]. P. Stone and M. M. Veloso, "Multiagent systems: A survey from a machine learning perspective," Autonomous Robots, vol. 8, no. 3, pp. 345-383, 2000 (Placeholder1)

[2]. S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in INFOCOM, 2001, pp. 1380-1387.

[3]. M. Cardei and J. Wu, Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems. CRC Press, 2004, ch. 19, Coverage in wireless sensor networks.

[4] H. Zhang and J. C. Hou, "Maintaining sensing coverage and connectivity in large sensor networks," Wireless Ad Hoc and Sensor Networks: An International Journal, vol. 1, no. 1-2, pp. 89-123, January 2005.

[5]. https://en.wikipedia.org/wiki/Machine_learning_

[6] https://en.wikipedia.org/wiki/Reinforcement_learning

[7] . https://en.wikipedia.org/wiki/Markov_decision_process

[8]. https://en.wikipedia.org/wiki/Q-learning

[9]. https://en.wikipedia.org/wiki/Sensor

[10] https://en.wikipedia.org/wiki/Wireless_sensor_network

[11] Renaud J.C, Tham C. K Coordinated sensing coverage in sensor networks using distributed reinforcement learning https://doi.org/10.1109/ICON.2006.302580

[12] M. W. M. Seah, C. Tham, V. Srinivasan and A. Xin, "Achieving Coverage through Distributed Reinforcement Learning in Wireless Sensor Networks," *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, Melbourne, Qld., 2007,

[13] Michael Littman and Justin Boyan. 1993. A Distributed Reinforcement Learning Scheme for Network Routing. Technical Report. Carnegie Mellon Univ., Pittsburgh, PA, USA

[14] Yau, KL.A., Goh, H.G., Chieng, D. et al. Computing (2015) 97: 1045. https://doi.org/10.1007/s00607-014-0438-1

[15] https://en.wikipedia.org/wiki/TinyOS

[16] http://tinyos.stanford.edu/tinyos-wiki/index.php/TOSSIM