University College Cork Computer Science Department

Distributed Learning of Pedestrian Behaviour

Student Name:	James Giller
Student ID:	109466711
Supervisor:	Dr Ken Brown
Second Reader:	Dr Steve Prestwich

B. Sc. Final Year Project Report

2013

Declaration of Originality.

In signing this declaration, you are confirming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:-

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;

- with respect to my own work: none of it has been submitted at any educational institution contributing in any way towards an educational award;

- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Name: James Giller

Signed: ________________________________

Date: 12/04/2013

Contents

I. Abstract

Wireless Sensor Network technology is a flexible, powerful means of linking the physical world with the digital world. Networks of small computers equipped with sensors, called nodes, can be used to gather data about the environment in which they are deployed to drive applications in areas that include military, healthcare, and smart building environments. The traditional network architecture in which sensor nodes upload data to a static data sink computer is often prohibitively expensive. An alternative architecture involving the use of mobile consumer devices as data collectors has been proposed in Wireless Sensor Network literature. This report investigates the application of machine learning to the problem of opportunistic data collection by pedestrians carrying such mobile devices. Pedestrians may visit the area in which a Wireless Sensor Network is deployed for their own purposes. The movements of pedestrians around the network are uncontrolled; however, they can be modeled as repeating patterns with variance. The goal of machine learning in this case is to discover these underlying patterns and exploit them to predict when a wireless sensor node should attempt to upload data to a passer-by. A reinforcement learning algorithm has been designed and shown to be effective in learning patterns with both an hourly and daily period. This algorithm has also been compared with ideas already found in the literature for similar scenarios involving controlled mobile data collectors.

II. Introduction

A. *Wireless Sensor Networks*

Wireless Sensor Networks (WSNs) are networks of sensor nodes [1] used to monitor aspects of the environment in which they are deployed. Nodes are small computers that can gather data about the environment using one or more sensors; perform some basic processing; then either store the data locally, or transmit to other devices using a short-range radio transceiver. The components of an individual sensor node are illustrated in Figure II.1.

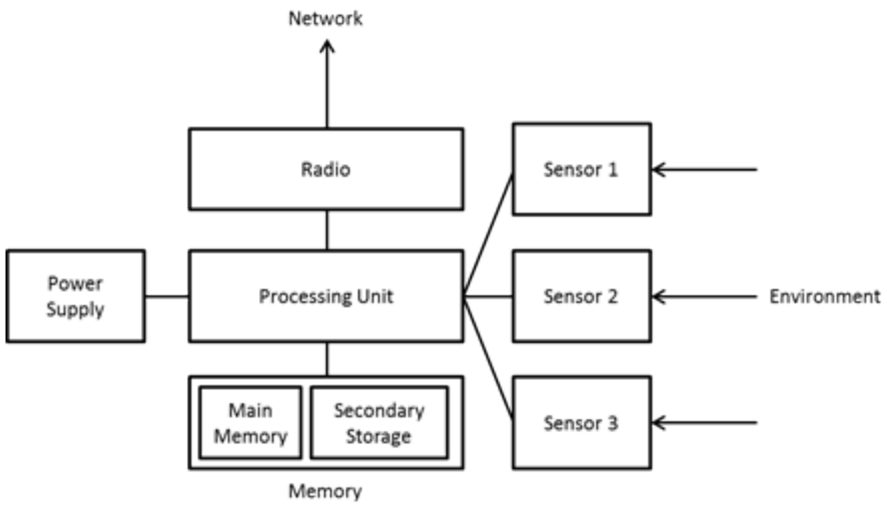


Figure II.1: Block diagram of the typical components of an individual sensor node. A RISC based microcontroller is often used as a processing unit, whereas external Flash memory is a common secondary storage medium.

Many WSN applications do not require individual nodes to be performing a task at all times. A *duty cycle* is the fraction of time a node, or one of its components spends in a high power, active state. The power supply available to most sensor nodes is a non-rechargeable battery. To extend their operational lifetime, nodes must have a very low average duty cycle, which can be accomplished by for example the processor entering a sleep state, or individual peripheral components being switched off. Actions such as sensing, calculating and communicating cost different amounts of energy. In general, using the radio is the most expensive operation, so the time spent communicating should also be minimised.

Sensor nodes are constrained with the minimum computational resources required for their special purpose. Long-term data storage and advanced processing is carried out by one or more powerful, general purpose computers, called *data sinks*. Nodes are configured to cooperate to upload data gathered within the WSN to a data sink. In many networks, messages are forwarded from node to node towards a data sink located nearby, in what is known as *multi-hop communication* (see Figure II.2.)

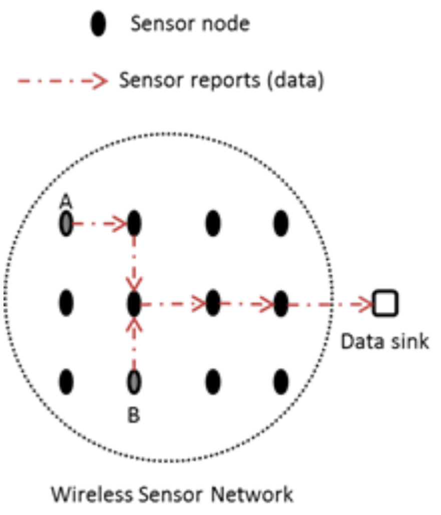


Figure II.2: Multi-hop communications in a WSN with a traditional static data sink architecture. Sensor nodes *A* & *B* must forward data to neighbouring nodes in order to upload them to a remote, static data sink.

WSNs can be used strictly for information gathering, or to enable digital applications that interact with or control objects in the real world. Areas of application include military, healthcare, and smart building environments [1].
The deployment and maintenance of dedicated data sink computers can be prohibitively expensive. They are inflexible, having to remain close to the WSN and support the same communication protocols. Further, sensor nodes close to data sinks can work too hard forwarding messages in multi-hop communication, reducing their lifetime.

A solution to this, especially suited to business and domestic environments, is to exploit mobile consumer devices such as smartphones and tablet computers as *data collectors*. Wireless sensor nodes can transfer data to these devices when their owners travel through an area where a network is deployed (e.g. Figure II.3.) These consumer devices can serve as data sinks or as *relays*, sending collected data to its final destination over the Internet. The reuse of existing devices is more efficient than deploying dedicated machines. The large number of potential data collectors can increase network reliability, eliminate costly multi-hop communication, and make WSN applications more robust to changes in network structure [2].

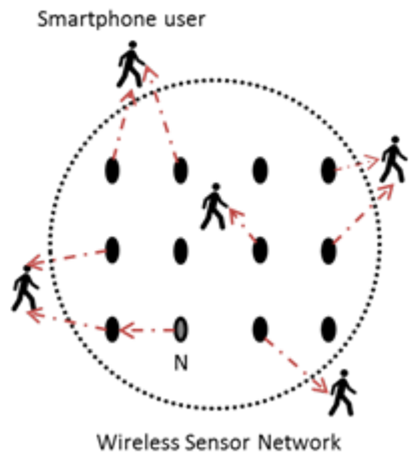


Figure II.3: Data collection by smartphone users travelling through a Wireless Sensor Network. Node *N* can still forward its data to a neighbour, but how should it make a choice?

B. *Project Objective*

There are several difficulties that must be overcome for this approach based on mobile data collectors to be effective. The number of user applications running on modern mobile consumer devices will reduce the time and resources available to spend on communicating with wireless sensor nodes. Similarly, the amount of data that different mobile devices are willing to accept is likely to be highly variable.

More important than these technical difficulties, the availability of individuals with mobile devices that can be contacted within the WSN is uncertain, and often time-dependent. Furthermore, some nodes in the network may be visited less frequently than others or not at all. The present study investigates the use of machine learning at an individual sensor node, to perform efficient data transfer to a mobile data collector in the presence of uncertainty. A machine learning algorithm has been designed based on the work of Shah and Kumar [3] to adapt the behaviour of a wireless sensor node to mobility patterns of pedestrians that pass it by. The objective is to enable the node to learn to become active at the times when a high level of human activity is expected, and sleep at other times to conserve energy.

III. Background

A. Related Work

This project builds on the work of two recent studies related to data collection in WSNs by mobile devices. The inspiration for this project is provided by the work of Brown, Sreenan, and Wu in [2]. The authors consider the use of mobile consumer devices to collect WSN data *opportunistically*; that is while the device owners have visited the area for purposes other than data collection. A mechanism in which sensor reports are propagated towards frequently visited nodes is proposed. This mechanism, known as Data Pre-Forwarding (DPF), relies on the correct timing of rounds of communication so that data is forwarded to these "hot" nodes just in time to be collected by a visitor. Learning about the movement patterns of visitors at different locations in the network and distributing this knowledge is a vital step in DPF. This present study describes a machine learning algorithm that can be used to model people's movements around a single node. How this algorithm might be extended to distribute models throughout the network will be discussed in Section X.

The algorithm that has been designed is based on the Distributed Independent Reinforcement Learning algorithm (DIRL) created by Shah and Mohan [3]. It is an algorithm designed for nodes in *sparse* wireless sensor networks, where the distance between any pair of nodes is too large for them to directly communicate. DIRL has been applied in the scenario of data collection by a mobile device, creating a framework for Resource-Aware Data Accumulation (RADA) [4]. Due to the restriction of sparse WSNs, the authors consider a *controlled* mobile device that is programmed to visit each sensor node according to some routine. It is the job of sensor nodes to discover the device when it comes within range for data transfer. In the network scenario of RADA, a mobile data collector periodically broadcasts messages to alert sensor nodes to its presence. A sensor node can discover a mobile data collector when its radio is on and the collector comes within range for its broadcasts to be received. DIRL is used to learn a duty cycle policy for a node's radio to perform timely, and energy efficient discovery. A few details of DIRL and RADA are discussed in Section V on algorithm design, only for the purpose of comparison with the design choices made in this present study.

The present study assumes a *dense* WSN, in which pairs of nodes can communicate with each other. In contrast to DIRL, the ability of nodes to communicate with each other is central to the idea of Data Pre-Forwarding, and is only possible in the context of dense WSNs. DPF also describes a mobile data collector of a different nature than in DIRL and RADA. DPF is concerned with opportunistic data collection by mobile consumer devices whose owners may visit an area for potentially different purposes. Therefore, unlike the situation with a controlled mobile data collector, not every node is guaranteed to be visited according to a schedule, and some nodes may not be visited at all.

B. Reinforcement Learning & Q-Learning

DIRL is based on Watkins' Q-Learning [5]. Q-Learning is a form of reinforcement learning that exploits dynamic programming to reduce computational demands, and does not require an internal model of the world. This is very suitable for implementation on wireless sensor nodes that have limited resources. Reinforcement learning is a technique used in the field of Artificial Intelligence. It is common to refer to the subject in such techniques (i.e. the learner, which in the present case would be a sensor node) as a *rational agent*; an entity whose actions are based on a rational decision process [6]. Reinforcement learning is an online learning technique, meaning that an agent must learn which actions to make in its environment on-the-fly, without ever being exposed to examples of correct, or incorrect behaviour.

A reinforcement learning agent chooses an action to perform at any time based on previously learned *utility values*. The utility of an action is the accumulation of *rewards* over time. An agent receives a reward based on the result of performing an action; the reward can be positive or negative.

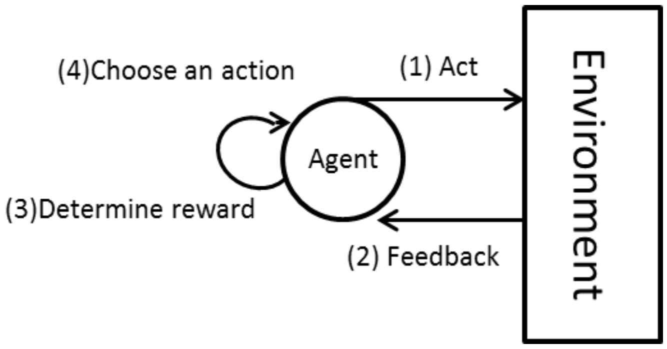


Figure III.1: An abstraction of the decision process of a reinforcement learning agent. The agent determines a reward based on feedback from the environment. Rewards accumulated over time determine the utility of an action.

Reinforcement learning algorithms are differentiated by how they calculate rewards and utilities, and how they choose an action to perform (steps 3 and 4 in Figure III.1.) Rewards are calculated from the result of performing an action, which can be a *change in the state of the environment.* A reward essentially measures how good it is to transition from one state to another (e.g. in a game of chess: how good the board is for a player after moving a piece.) A function that maps perceived states to rewards is called a reward function. A reinforcement learning algorithm must define variables that characterise the state of the world.

Q-Learning defines a function Q for updating the utility of an action based on the latest reward. This function is presented below in Equation III-1, referenced from [3].

$$Q(s, A) = (1\text{-}\alpha) * Q(s, \mathrm{A}) + \alpha * (r + \gamma \mathrm{e}(s'))$$

Equation III-1: The function used in Q-Learning for updating the utility of performing an action in a certain state.

In the function Q, s is the original state in which the agent perfomed the action A, and s' is the resulting state. The updated utility of performing A while in s is a linear combination of the old utility and the newly observed reward r.

There are two tunable parameters to this function: the *learning rate* α, and the *discount value* γ. The learning rate is used to tune the importance of the previously learned utility. It can assume values in the range (0;1]. A learning rate of 1 causes the function to disregard previously learned information in favour of the latest immediate reward, whereas a value close to 0 leads to very slow updates.

The discount value applies to the expected future reward for transitioning to the state s'. The expected future reward is defined as

$$\mathrm{e}(s') = \mathrm{Max}(\ Q(s', A')\ ).$$

Equation III-2: The expected future reward for arriving in a new state.

The discount can assume values in the range [0;1]. A discount of 1 gives equal weight to possible rewards in the future as to the actual reward just received, whereas a value of 0 ignores future rewards.

The core data structure in Q-Learning is a table (e.g. Table III.1) that maps state-action pairs to utility values calculated by the function Q. Whenever a Q-Learning agent must choose an action; it will look up this table for the best choice in the current state.

	S_1	S_2	…
A_1	$u_{A1,S1}$	$u_{A1,S2}$	…
A_2	$u_{A2,S1}$	$u_{A2,S2}$	…
…	…	…	…

Table III.1: An abstraction of the table used in Q-Learning to map state-action pairs (S_i, A_j) to utility values $u_{Aj,Si}$.

The best choice is often determined by the state-action pair with the highest utility value. This choice is known as *exploitation*, and is how a reinforcement learning agent accomplishes its goal of maximising reward over time.
Sometimes it is useful to try a different action, known as *exploration*. In the first few phases of a reinforcement learning algorithm, it is not known which choice of action will result in the highest reward. Without exploration, an agent can become fixed on a certain policy even if it becomes sub-optimal due to a change in the environment. As in human learning, the best results can be achieved through trial and error. Exploration can mean choosing an action uniformly at random, but different *policies* can also be used, as will be discussed later.

IV. Research Task Description

This section defines the problem, presented informally in Section II.B, as a reinforcement learning task. This definition encompasses five aspects: the rational agent performing the learning; the sensors available to the agent; the actuators available to the agent; the environment in which the agent exists; and measures of the agent's performance. Several attributes of the environment will also be discussed.

A. A Wireless Sensor Node as a Rational Agent

In the context of this reinforcement learning task, a wireless sensor node is a rational agent with the objective to continuously gather data about its environment (e.g. temperature, light levels) and transfer these data to a mobile data collector. One can consider other application specific requirements, but it is preferred to keep this task description general. Data collectors are smartphones and tablet computers carried by pedestrians travelling through the deployment area of a WSN. For ease of language, the terms pedestrian and data collector are interchangeable throughout this description.

The agent has limited storage capacity and operates on battery. Data can expire if storage is used up and new data is gathered, or, depending on the type of data, if it is past the time of usefullness. The agent itself will cease to be active when its battery is depleted. The rate of depletion depends on the actions taken by the agent over time.

The agent can have any kind of sensing apparatus required to perform application tasks. The type of sensors used and the nature of the WSN application will affect the data generation rate. The more frequently data is gathered by its sensors, the more often the agent will have to attempt data transfer.

The agent can try to complete its objective by communicating directly with a data collector. The actuator available for sending data is a short-range radio transceiver. The radio can also be used to send instructions, queries, and other messages. Messages can be shared with data collectors within a certain distance from the agent, known as the contact area (see Figure IV.1.)

A connection must be established between the agent and a data collector before any messages can be shared. The agent can broadcast special messages called *beacons* to alert nearby data collectors to its presence. Broadcasting a beacon is the first step in a handshaking protocol to establish a connection. An example of such a protocol can be found in [2]. The agent knows when a data collector is willing to establish a connection by receiving a reply to a beacon. A data collector that replies to a beacon is known as a *contact*. The acts of broadcasting a beacon and listening for a response can be referred to as *contact probing* [7]. Recent studies have shown that in WSN scenarios containing mobile data collectors, *sensor initiated contact probing* is the preferred approach [7].

B. The Environment of a Sensor Node Agent

The environment in which a sensor node agent exists is the physical area where it is deployed as part of a WSN. There is access to this area for pedestrians walking or travelling by vehicle. Pedestrians are assumed to be in the environment for purposes other than data collection. One can consider a single- or multi-agent environment. In either case, only sensor nodes will be considered agents, and the pedestrians moving through the environment will be considered objects. This decision eliminates the need to describe the objectives of each pedestrian in the environment.

A simple single-agent environment is illustrated in Figure IV.1. A pedestrian footpath passes through the contact area of a sensor node, the agent. Mobile devices carried by pedestrians have their radios on at all times, so it is possible for a pedestrian to receive beacons as long as they remain within the contact area. Devices carried by those ouside of the contact area are invisible to the agent and cannot be contacted.

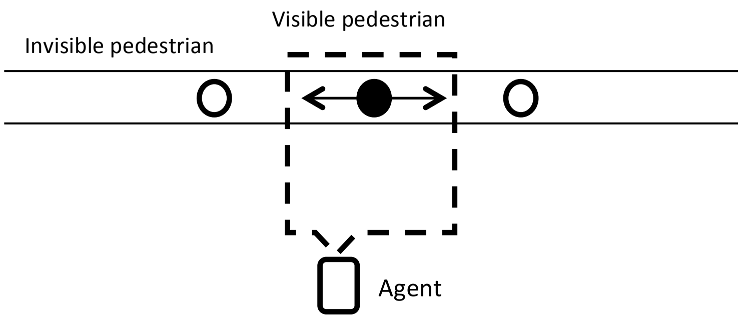


Figure IV.1: A single-agent environment consisting of a footpath that passes through the contact area of a wireless sensor node. Connections can be established with *visible* pedestrians within the contact area.

The movement of pedestrians along the footpath can be characterised by a *repeating pattern*. This pattern can be hourly for example, such that many people walk past the agent at five minutes past every hour. The agent should learn the utility of contact probing at different times according to this pattern. An online learning technique such as Q-Learning is suited to this dynamic environment. In a multi-agent environment, using an online learning technique preempts the configuration by hand of hundreds or even thousands of nodes that may constitute a WSN.

The agent's performance in this environment can be measured in many ways. The agent should discover times that experience sufficiently high pedestrian activity, with respect to the data generation rate. A small number of contacts can guarantee the agent an opportunity to transfer small amounts of data. It is also important that the node spend as little energy as possible while learning, reserving battery for carrying out its regular duties once it has determined the pattern. Similarly, the total time spent learning should be minimised according to the quality of service requirements of the WSN application. For example, there may be a deadline after which the node should deliver *x* reports per day. It is important for the reward function in a reinforcement learning algorithm to reflect the performance measure. The performance measure used to date in the present study is how accurately a pattern of pedestrian activity can be learned while also reducing energy expenditure.

C. Attributes of the Environment

Artificial Intelligence: a Modern Approach [6] describes several attributes that can be used to characterise the environment of a rational agent. This section discusses these attributes as they relate to the environment described in Section IV.B.

An environment is *fully observable* if aspects of the state of the environment that are relevant to an agent's choice of action are accessible at each point in time. If an agent cannot observe all relevant information, the environment is called *partially observable*. Arguments can be made for either case in this situation.

Since ideally the agent spends the majority of time in a sleep state, it cannot observe, for example, that a pedestrian passed through its contact area the instant before it awoke. This event that was missed might indicate the imminent arrival of more pedestrians, or the end of a rush hour. Whether one considers the act of sleeping at certain times to be part of the learning process or a result of learning is important. It is impossible for an agent to observe anything while asleep, but relevant information is accessible within its contact area when awake. A noisy radio signal may introduce problems to observability. The extreme case of an agent that broadcasts data into its contact area, unable to detect the presence or absence of pedestrians is an example of an *unobservable* environment.

As stated previously, pedestrians are objects within the environment in question. In the real world, people have free will but their movements often exhibit spatial locality [8]. The behaviour of pedestrians can be modelled stochastically as repeating patterns with variance. The environment is therefore *stochastic*.

Adapting the policy of a sensor node according to patterns of pedestrian behaviour is a *sequential decision problem*. The nature of patterns necessitates making decisions based on information gathered in the past.

The environment in which a sensor node agent exists is clearly *dynamic*. Other events besides the actions of the agent can bring about changes in the world.

V. Algorithm Design

This section describes an algorithm for the task defined in Section IV. This algorithm is based on Q-Learning and Distributed Independent Reinforcement Learning. The creators of DIRL have applied their algorithm to a similar task involving controlled mobile data collectors in sparse WSNs. It is prudent to apply some of the tried and tested techniques used in DIRL to the present study. There is also the opportunity to evaluate these ideas in a novel situation.

A. State Definition

As stated in Section III.B, a reinforcement learning algorithm must define variables that characterise the state of the environment. The choice of state variables is influenced by the period of the repeating pattern of pedestrian activity the algorithm should be able to learn. In general, the longer the period is the more state variables are required to characterise it, and the larger the utility table will be. The following patterns and state variables have been used in the present study (Table V.1.)

Pattern to learn	Repeating hourly	Repeating daily	Repeating weekly[1]
State variables	Minute of hour	Hour and minute	Day, hour and minute

Table V.1: Repeating patterns of different periods and their associated representations.

It is assumed that patterns of people's movement are accurate to within 5 minutes. This is the same assumption made by Google in their Calendar software, which schedules events to finish 5 minutes before the hour by default. This assumption is exploited to reduce the size of the utility table by aggregating times that are within the same 5 minute interval. In this way, the node perceives time as a *series of five minute intervals* in which pedestrians may or may not enter its contact area. This assumption forms the *bias* required for learning [9]. A state variable representing the time of day is also proposed in [4] for learning patterns arising from a schedule or timetable.

[1] Repeating patterns with a weekly period were not simulated due to limitations in the simulation software; however, those state variables shown in Table V.1 are recommended by this author and the authors of [4], and will be used in future work.

The means of aggregating states is general, and is defined as the weighted hamming distance between states. The weighted hamming distance between two states is the sum of the weighted differences between corresponding state variables. A state is considered similar to another state if the weighted hamming distance between them is less than a threshold [4].

The battery power remaining at the node and the number of contacts made so far are also observed as aspects of state, but this information is lost in the process of aggregation. These aspects of state are used when calculating rewards, but *do not characterise the times when decisions are made*. A result of this design has been observed in the experiment described in Section VII.A. If at any time of day the node should make a different decision based on its battery level, then this information can persist in the state space of the utility table.

B. Action Definition

The node will choose an action to perform based on the policy for its current state. A chosen action will be performed over an entire 5 minute interval. The actions that can be chosen are duty cycles that determine the frequency of contact probing. In the present study, the following duty cycles are utilised (Table V.2.)

Action	# Contact Probing Attempts
High Duty Cycle	50
Low Duty Cycle	25
Zero Duty Cycle	0

Table V.2: Radio duty cycles and the corresponding number of contact probing attempts.

The creators of DIRL do not consider a zero duty cycle in their framework for Resource-Aware Data Accumulation [4]. In fact, the zero duty cycle here replaces RADA's very low duty cycle, defined as being 10% of high duty cycle activity. The network scenario considered for RADA involves a controlled mobile data collector that is guaranteed to visit every node in the WSN under normal circumstances. This guarantee has to be met in the context of sparse WSNs because multi-hop forwarding is impossible.

Nodes in a sparse wireless sensor node have no other option for offloading their data, so they must take every opportunity to make contact with a mobile data collector. In RADA, the very low duty cycle should be performed when the probability of the data collector being in range is low but non-zero. The zero duty cycle does account for the possibility that a node may never be visited by a pedestrian. In this situation, the node will not spend any energy using its radio.

The low duty cycle is equivalent to performing contact probing once every 12 seconds. Several beacons may be broadcasted during a single contact probe for reliable detection by any pedestrians in the contact area. During a period of high pedestrian activity, this frequency should be adequate for making contacts and reduce the energy spent by the node. This duty cycle is suitable for groups of people passing through the contact area, which minimises the time that the contact area is empty.

An individual pedestrian might pass through the contact area quite quickly, depending on the communication radius of the node's radio. The high duty cycle performs a contact probe once every 5 seconds in order to make contacts in periods of low pedestrian activity, at the cost of greater energy consumption. This frequency of probing may only be effective for contacting pedestrians on foot. A higher frequency should be considered when contacting vehicles, due to the increased speed of movement and shorter time within the contact area.

The roles of the high and low duty cycle suggested in RADA have been reversed in the present study. In RADA, a high duty cycle should be used for timely detection of a mobile data collector at the times when it is expected to be within range. This rationale can be traced again to the importance of every single contact to a node in a sparse WSN. By using a low duty cycle in periods of high pedestrian activity and vice-versa, progress is made towards the goal of reducing energy expenditure as stated at the end of Section IV.B.

C. Reward Function

Rewards are calculated based on the number of contacts made, and the energy spent over the duration of an action. RADA defines the reward for performing a duty cycle action A as

$$r(A) = (nc * mp - 1) * es.$$

Equation V-1: The reward function used in RADA.

In Equation V-1, nc is the number of contacts made, es is the energy spent, and mp is a parameter used to ensure a positive reward for at least a single contact. Experiments in the present study found this reward function to be biased towards a high duty cycle, which costs more energy and has a greater number of expected contacts. A new reward function has been defined as

$$r(A) = ((\beta * \min(nc_{max}, nc)) - (\sigma * es).$$

Equation V-2: A more energy efficient reward function.

In Equation V-2, nc_{max} is the maximum number of contacts that can be rewarded, and β and σ are configuration parameters. Specifying a maximum number of contacts to consider removes the bias towards a high duty cycle and can help to eliminate noise caused by a single instance of exceptionally high activity. Using the energy spent as a subtrahend rather than a coefficient punishes high energy consumption, which is in keeping with the objective of energy efficiency. The calculated reward is passed as an argument to the function Q defined in Section III.B, in order to update the utility table (Table III.1.)

D. Exploration Policy

Two exploration policies have been used in the present study. The first is also the policy used in RADA. According to this policy, exploration will be performed with a probability q, otherwise exploitation will be performed. The probability of performing exploration decreases over time according to the function in Equation V-3. Exploration itself involves choosing an action uniformly at random, whereas exploitation involves choosing the action with the highest utility in the current state, breaking ties with a predefined priority ordering over actions.

$$q = q_{min} + \mathrm{Max}(\,0,\,(q_{max} - q_{min})\,*\,(nc_{max} - nc)\,/\,nc_{max})$$

Equation V-3: Probability to perform exploration in RADA.

The second policy is a form of *randomised exploitation*, in which the utility of an action determines the probability to perform that action. Because the duration a duty cycle will be performed is fixed, the duty cycles themselves are well defined and the energy consumption of radio operations can be modelled, therefore the minimum reward can be calculated from Equation V-2 as the energy consumed during a high duty cycle when no contacts are made (u_{min} in Equation V-4 below.) Thus the probability of choosing a candidate action A is calculated as

$$prob(A) = \frac{\mathrm{u}_A - \mathrm{u}_{min}}{\sum_{A_i}(\mathrm{u}_{A_i} - \mathrm{u}_{min})}.$$

Equation V-4: Probability of choosing an action in a randomised exploitation policy.

Under the second policy, exploitation is a stochastic process. The greater the utility attributed to an action, the more likely it will be chosen; however, if two actions are closely matched, then the action with the lower utility at the time is expected to have more opportunities to be chosen. During the early stages of learning, the probability of choosing each action will be roughly equal. As learning progresses, the higher utility actions will tend to be chosen more often, but some exploration will still be carried out.

Under the first policy, the actions chosen by exploitation during the early stages of learning will be largely predetermined by the priority ordering over the actions, until the utility values begin to diverge. This may give an advantage to certain actions in the long run, especially since exploration and exploitation are mutually exclusive. Experiments are yet to be carried out to compare the two policies; however, such experiments are recommended for future work.

E. Walkthrough of the Algorithm

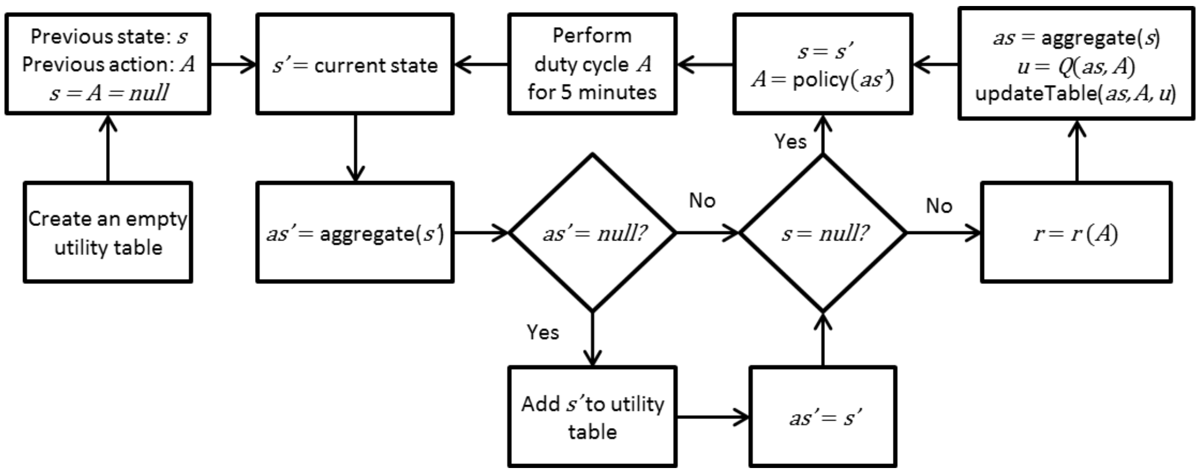


Figure V.1: Flowchart of the algorithm.

Executing the algorithm illustrated in Figure V.1, the utility table is initially empty and the previous state and action are assigned a null value. At the start of each iteration, the current state of the world s' is observed. If there are no previously observed states that are similar to s', then a new set of entries can be made in the utility table for s'. Otherwise, current state is mapped to the aggregation of states as' that would contain it (e.g. the 5 minute interval as in Section V.A.)

If possible, the reward for the action just performed *A* is calculated and used to update the utility table. Finally, a new action is chosen based on the policy for exploitation and exploration, and this action is the duty cycle that will determine the frequency of contact probing over the next 5 minutes.

VI. Experimentation

A. Purpose of Experiments

To evaluate the algorithm presented in Section V, several experiments were carried out in simulation, to determine the capabilities and limitations of the algorithm, and to investigate the most important aspects that contribute to effective learning. The experiments discussed in this report are listed in Table VI.1, categorised under these two objectives. The simulation software itself is outlined in Sections VIII and IX.

Categories of Experiment	
Evaluative	**Investigative**
Learning hourly patterns	Comparison of duty cycles
Learning daily patterns	Comparison of reward functions

Table VI.1: Evaluative and investigative experiments carried out in the present study.

Several experiments were designed to test the ability of the algorithm to learn patterns that repeat with various periods. Both hourly and daily patterns of pedestrian behaviour were simulated. The effectiveness of the high and low duty cycles for making contacts during times of both high and low activity was investigated. The two reward functions defined by Equation V-1 and Equation V-2 were also compared.

B. Simulation Setup

Each of the patterns discussed in Sections VI.C – VII.D were repeated over 10 days of simulated time. Energy consumption by the node was calculated in terms of the percentage of initial battery power. Only energy consumed by the node's radio was modelled. A similar model to the one found in [4] was used.
In the following sections, the exploration policy described by Equation V-3 is referred to as "Exploration Policy 1", whereas the policy described by Equation V-4 is referred to as "Exploration Policy 2".

C. *Simulated Patterns*

Figure VI.1 is an example of a simulated pattern of pedestrian activity. The reference scenario for all simulated patterns is the single-agent environment illustrated in Figure IV.1. The levels of activity indicate the percentage of time in seconds that pedestrians spent within the contact area. Activity is measured over 5 minute intervals because of the bias introduced by the assumption made in Section V.A. It must be proven that the algorithm performs well under simplifying generalisations before more rigorous testing can be performed.

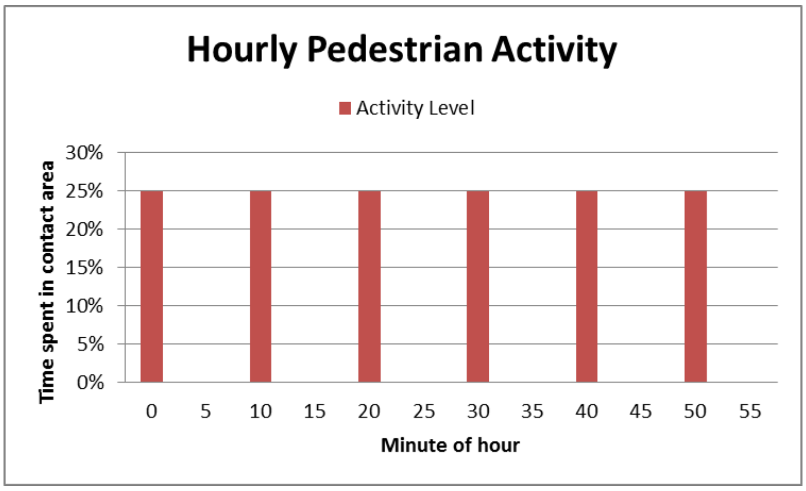


Figure VI.1: A regular pattern of pedestrian activity, repeating with an hourly period. During times of activity, 75 pedestrians are expected to enter the contact area over a 5 minute interval.

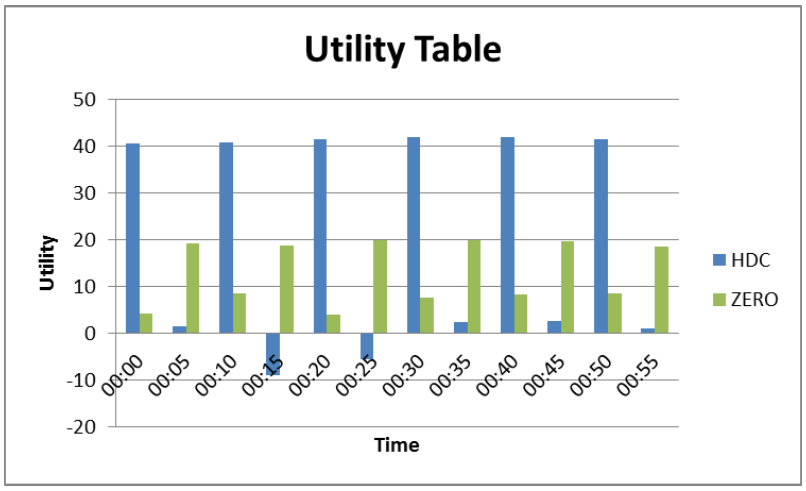
VII. Experimental Results

A. *Learning Hourly Patterns*

The algorithm was tested on the pattern illustrated in Figure VI.1. The purpose of this experiment was to confirm that a periodic hourly pattern, with times of high activity punctuated by times of no activity could be learned. The high and low duty cycles were tested in isolation. Exploration Policy 1 was used, with a constant probability to perform exploration of 0.02, and the zero duty cycle as the highest priority action. For the reward function, nc_{max} was set to 5, β was set to 10, and σ was set to 1000. Originally, a learning rate of 0.5 and a discount value of 0.5 were used in the function Q. Examples of the final state of the utility table after executing the algorithm are shown in Figure VII.1.

It is clear from the figure below that the algorithm was successful in learning the times of activity. It can also be seen that the low duty cycle (LDC) was attributed a higher utility, on average, than the high duty cycle (HDC). With such a high level of pedestrian activity, both duty cycles are very likely to achieve the maximum number of contacts; however, the reduced energy cost of the low duty cycle will receive a higher reward. The uneven distribution of LDC utilities is likely the result of occasionally missing contacts that the HDC could catch.

The graphs also show positive utilities associated with both the high and low duty cycles at times of no activity. This erroneous output is explained by an error in temporal credit assignment [10] caused by the choice of discount value for the function Q.



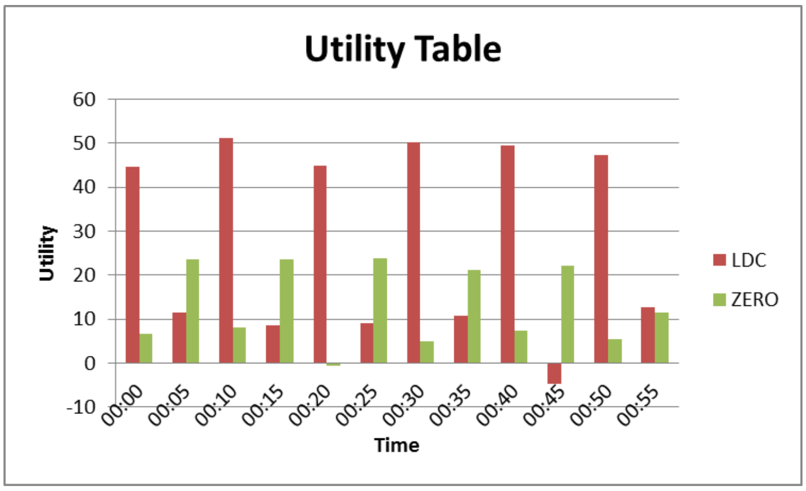
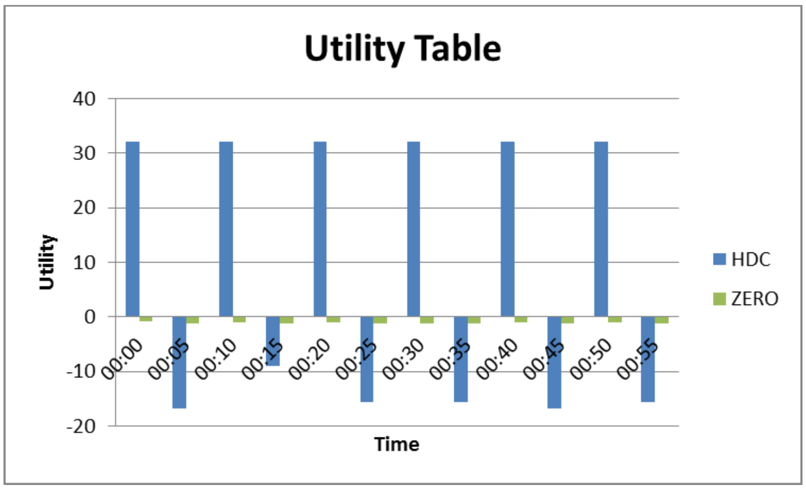


Figure VII.1: Graphs of the utility tables resulting from the pattern illustrated in Figure VI.1, using a discount value of 0.5.

A non-zero discount value in the function Q caused the expected future reward associated with states of high activity to be propagated back in time to preceding states of no activity. Because the state variables used (see Table V.1) are temporal in nature, the new state observed after performing an action is determined by moving forward in time by the duration of action, and is independent of the actual action performed. Therefore, the expected future reward for arriving in the new state should be disregarded, since one would arrive in that state regardless of the previous action. Similar temporal variables are used in the experiments described in [4]; however, the authors make no mention of this temporal credit assignment problem. Further investigation is recommended for future work. Figure VII.2 shows the same experiments with a discount value of zero.



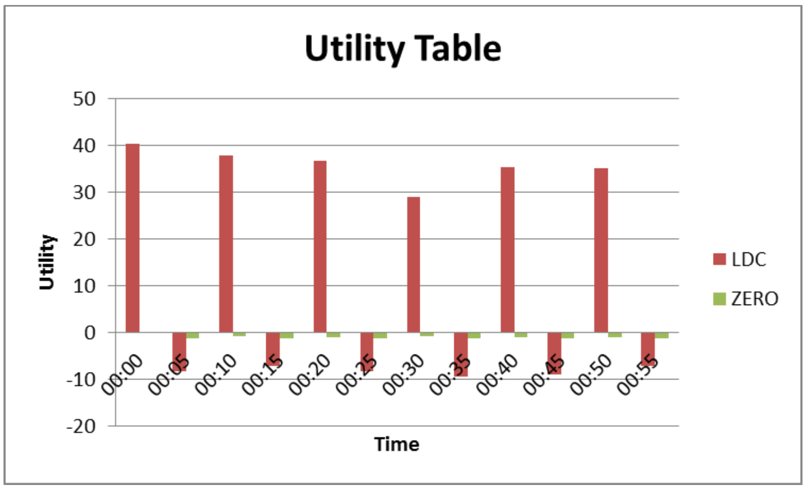


Figure VII.2: Erroneous positive utilities are eliminated by setting the discount value to zero when updating utilities. The rewards associated with times of activity are not distributed to the surrounding times of no activity.

B. *Comparing High & Low Duty Cycles*

As described in Section V.B, the high duty cycle and low duty cycle serve the purpose of making contacts during times of low activity, and high activity respectively. The goal of testing the algorithm on the pattern in Figure VII.3 was to compare the utility of the two duty cycles as the level of pedestrian activity changes from high to low. Exploration Policy 2 was used in this experiment.

The chart in Figure VII.4 shows an example of a single execution of the algorithm. At times of high activity, the utility of performing the LDC is higher than that of performing the HDC. There is a clear turning point when pedestrian activity drops to 10% or below, and the utility of performing the HDC becomes greater. As the activity level drops even lower, there is also a decrease in the magnitude of the utility associated with the HDC, as it becomes increasingly difficult to make contacts.

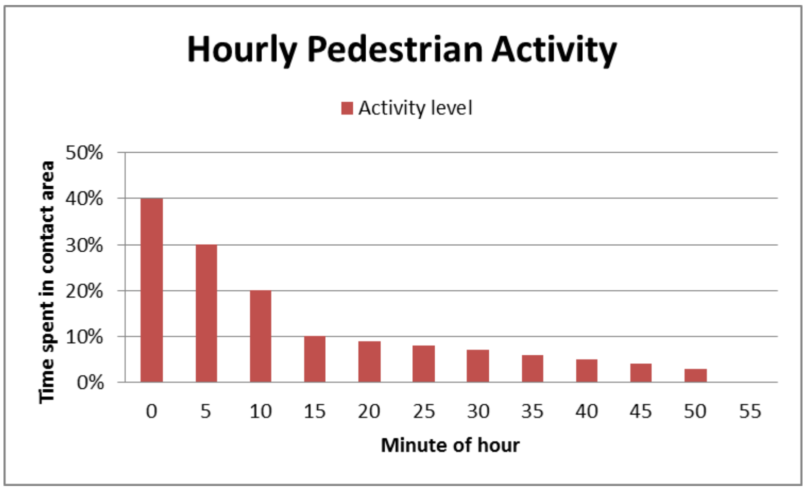


Figure VII.3: An hourly pattern of gradually decreasing pedestrian activity. The maximum and minimum numbers of pedestrians expected within the contact area are 120 and 9 respectively.

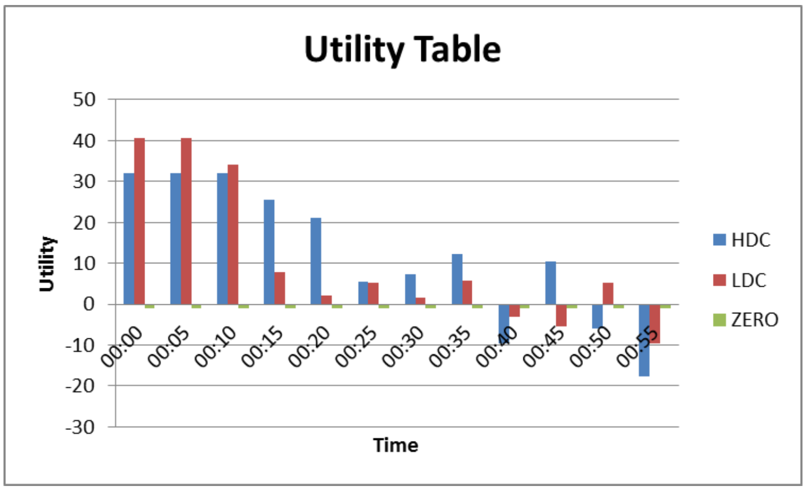


Figure VII.4: Example of a utility table resulting from the pattern in Figure VII.3.

The results were similar across multiple executions, though the average utility associated with the duty cycles at each time was not calculated. Such an aggregation will be performed when this experiment is repeated in the future.

C. Comparing Reward Functions

The algorithm was executed again on the pattern shown in Figure VII.3, this time using the reward function defined in RADA (Equation V-1.) This reward function is biased towards the HDC (see Figure VII.5,) which is able to make more contacts than the LDC and has a higher energy cost. As observed in previous experiments, the utility of a zero duty cycle assumes a final value close to 0 at all times. Even though no contacts can be made using a zero duty cycle, the energy spent is minimal, and this causes both reward functions to generate a negligible negative reward.

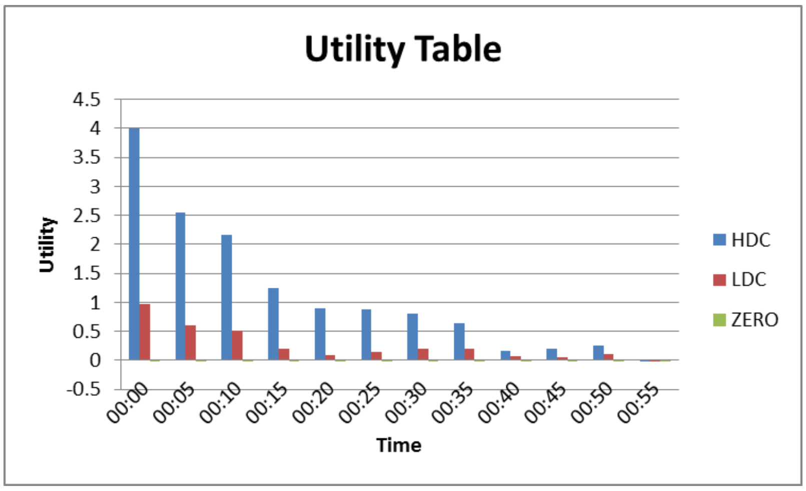


Figure VII.5: Example of a utility table resulting from the use of the RADA reward function on the pattern in Figure VII.3.

## *D.	Learning Daily Patterns*

Table VII.1 shows times of the day used to create a repeating pattern with a daily, rather than hourly period. This pattern represents the rush hours of activity during a working day, firstly in the morning, then around the beginning and end of lunch-time, and finally in the evening. Figure VII.6 shows the trend of utility values over time for each duty cycle that was learned by the algorithm. This pattern was chosen as a more realistic test of the performance of the algorithm. An extension of this experiment might involve the inclusion of noise around the peak times listed in the table below. If this pattern is supposed to represent the flow of people in and out of a certain area within an office building, then it is unlikely that there would be extended periods of complete inactivity.

Time:	08:55	09:00	09:05	12:55	13:00	13:55	14:00	16:55	17:00	17:05
Activity Level:	7%	15%	4%	15%	7%	4%	15%	15%	4%	7%

Table VII.1: Daily times of pedestrian activity.

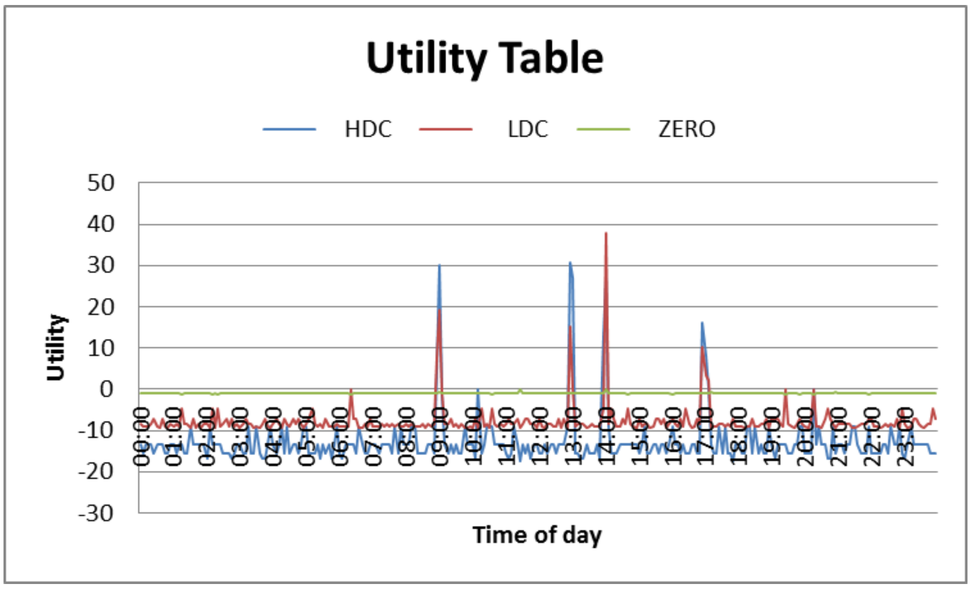


Figure VII.6: Example of a utility table resulting from learning a repeating pattern with a daily period.

VIII. Software Design

A. Language & Tools

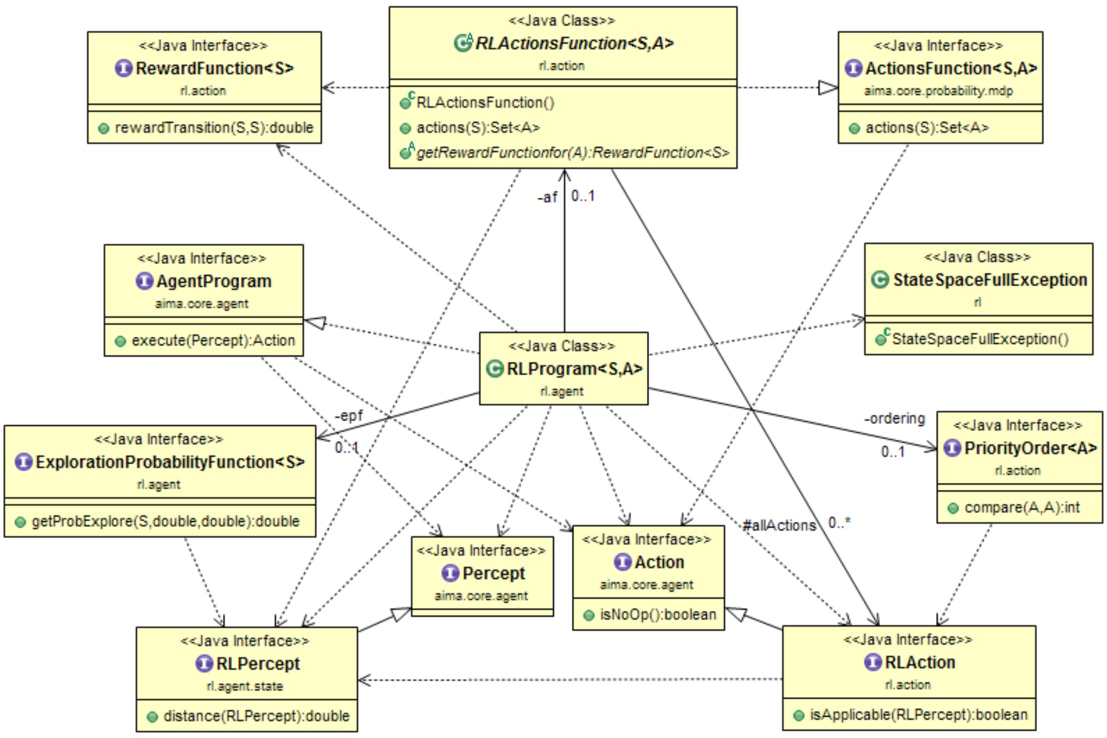
The algorithm presented in section V has been implemented in Java 7 using the Eclipse IDE. Programming in Java promotes object oriented design, which is preferred when implementing systems of many interacting components. Java's Collections framework offers a rich set of predefined data structure classes essential for implementing complex algorithms.

A Java implementation of the algorithms from Artificial Intelligence: A Modern Approach is available from the aima-java project [11]. The classes provided therein can be used much like a framework for implementing one's own programs. Several classes from the aima-java project that provide general definitions of agents, actions, environments etc. were extended to specify components of the system developed in the current study (see System Diagram VIII-1.) The Eclipse IDE automates most of the configuration of Java projects and is useful for importing external libraries. The IDE also has a built-in javadoc compiler and advanced features for quickly viewing documentation and source files by highlighting objects within the text editor. The ObjectAid UML Explorer plugin for Eclipse was used to generate UML diagrams (as seen in System Diagram VIII-1.)

The implementation was tested in a simulation also created in Java. For future testing on genuine wireless sensor platforms, one could program ContikiOS nodes using the Cooja simulator [12], or purchase one's own hardware.

B. Algorithm Class Structure

The algorithm presented in Section has been given a flexible implementation using object oriented design techniques such as generic types, interface types, and delegation. System Diagram VIII-1 is a UML class diagram showing the class 'RLProgram', in which the algorithm is implemented, and its dependencies. This design follows the Template Method Pattern [13]. Specific behaviour can be achieved by implementing the interfaces defined for the components of the algorithm. The methods exposed by the various interfaces are flexible in terms of the contract defined by their signatures. Most of these methods are also designed to delegate work to their inputs.



System Diagram VIII-1: A UML class diagram showing dependencies of the 'RLProgram' class.

IX. Software Implementation

A. Generic Algorithm Implementation

An instance of the RLProgram class can be stored as a program in another object that simulates a wireless sensor node agent. The program takes 7 inputs when it is created (more inputs are required for different policies of exploitation and exploration.) Table IX.1 shows inputs to the program.

Input	Description
S:	The type of states stored in the utility table.
A:	The type of actions stored in the utility table.
Max # States:	The number of states allowed to store in the table.
Actions Function:	Returns the actions applicable in any state and their reward functions.
Distance Threshold:	A strict upper bound on the hamming distance between two similar states.
Learning Rate:	A parameter to the function Q
Discount:	A parameter to the function Q

Table IX.1: Inputs provided to the algorithm.

The RLProgram class has been designed to accept different kinds of state and action definitions. Thus, it is a generalised implementation of the algorithm as illustrated in Figure V.1. Listing IX-1 shows an excerpt from RLProgram.java.

A built-in implementation of a map data structure using a hash map and doubly linked list was chosen to represent the utility table. This implementation has the benefit of preserving an order of iteration over items in the table [14], when searching for similar states or choosing from a set of actions. The order of iteration can be based on the order of insertion into the table or the number of times an item has been accessed. The latter configuration may be useful in future optimisation efforts, as the actions selected most often are generally those with the highest utility.

```java
/* Use the Q-Learning algorithm to select an action to execute.
 *
 * @param sPrime The current state of an agent.
 * @return An action to execute in the current state.
 */
protected A learning( S sPrime ) {

        if ( isTerminal( sPrime ) ) {
                s = null;
                a = null;
                return null;
        }

        // can sPrime be aggregated into the table?
        S aggregateSPrime = aggregateState( sPrime );
        if ( aggregateSPrime == null ) {
                // cannot be aggregated.
                if (! reachedMaxNumStates() ) {
                        // cannot be aggregated, can add new state.
                        addToUtilityTable( sPrime );
                        aggregateSPrime = sPrime;
                } else {
                        // cannot be aggregated, cannot add new state.
                       // Log the final state of the utility table.
                        System.out.println( utilityTable() );
                        throw new StateSpaceFullException();
                }
        }

        if ( s != null ) {
                // Compute the reward.
                RewardFunction<S> rewardFunction = af.getRewardFunctionfor( a );
                double r = rewardFunction.rewardTransition( s, sPrime );
                // update table(as, a)
                updateUtility( aggregateS, a, aggregateSPrime, r );
        }

        // choose an action to execute.
        s = sPrime;
        aggregateS = aggregateSPrime;
        a = policy( aggregateSPrime );

        return a;
}
```

Listing IX-1: A java method in which an action is chosen to perform as part of the algorithm illustrated in Figure V.1

B. Simulation Implementation

Repeating patterns of pedestrian activity within the single-agent environment described in Section IV.B was simulated in Java for the purpose of experimentation. In the simulation, time is divided into units of 1 second duration. Five minutes of simulated time (the duration of an action performed under the algorithm) consists of 300 consecutive time units (e.g. Figure IX.1.)

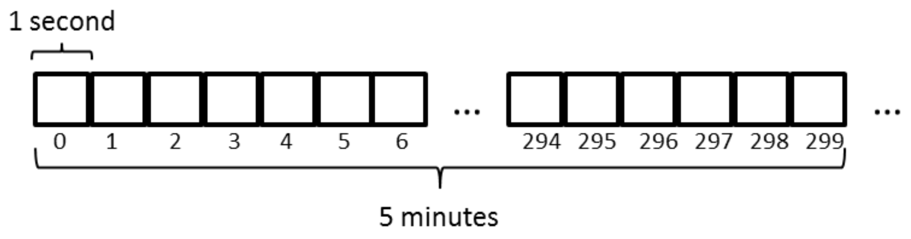


Figure IX.1: The first 5 minutes of simulated time, divided into units of 1 second duration.

A percentage of time units over a time interval are designated as times of activity within the contact area. The percentage of time units determines the level of activity over that interval. A bitmap data structure, illustrated in Figure IX.2, was used to represent this timeline. Each bit in the map represents the ability of the node to make contact with a pedestrian within its contact area at that point in time: a 1-bit signifies that a contact can be made, whereas a 0-bit signifies that a contact cannot be made.

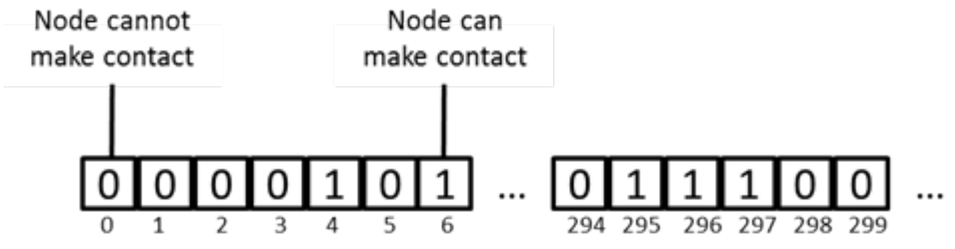


Figure IX.2: Part of the bitmap representing the first 5 minutes of simulated time.

The simulation starts at bit 0, which corresponds to a certain time (e.g. 09:00 hours.) The simulation progresses based on two operations that can be called on the bitmap; one called 'probe', and the other called 'skip'. The probe operation returns the value of the current bit and moves time forward by a single unit (bit.) The skip operation moves time forward by a specified number of units without returning the value of any bits that are skipped. As stated previously, this design is quite limited. The size of the bitmap required to simulate a weekly pattern is too large in practice.

A pattern of bits, representing the pattern of pedestrian movement through the contact area of the node, is entered into the bitmap before running a simulation. A random number generator is used in Listing IX-2 to set a certain percentage of bits over an interval to 1 in order to simulate different levels of pedestrian activity.

```
    /**
      * Simulate human activity in the contact area of a node over a certain time interval.
      * @param day The day of activity.
      * @param hour The hour of activity.
      * @param minute The minute of activity.
      * @param numMinutes The number of minutes of activity to simulate.
      * @param p The percentage of bits that will be set to 1. (The level of activity)
      */
     public void simulateActivity( int day, int hour, int minute, int numMinutes, double p ) {
                // Select the correct bits in timeline.
                int start = ( day * A_DAY ) + ( hour * AN_HOUR ) + ( minute * A_MINUTE );
                int finish = start + ( numMinutes * A_MINUTE );
                for ( int i = start; i < finish; i++ ) {
                       // Set p% of the time units in the interval to 1.
                       if ( Math.random() < p ) {
                              timeUnits.set( i );
                       }
                }
     }
```

Listing IX-2: Java method used to automate the setting of a pattern in the bitmap.

C. *Implementing Duty Cycles*

As stated in the previous subsection, the duty cycle actions can be translated into a sequence of calls to probe and skip operations on a bitmap. Each duty cycle specifies the length of time between consecutive contact probing attempts. This information has been encapsulated in instances of an enumerated type that represent the duty cycles. The value that is stored in these duty cycle objects is used as an argument to skip operations, specifying the number of time units to skip before the next probe operation. The use of duty cycle objects is exemplified in Listing IX-3.

```
int i = ACTION_DURATION;
while ( i > 0 ) {
    // Cut the action short if it will exceed ACTION_DURATION.
   int probesToSkip = ( i < dutyCycle.getToff() ) ? i : dutyCycle.getToff();
   contactArea.skip( probesToSkip );
   recordHistory();
   i -= probesToSkip;
   if ( i > 0 ) {
       isInContact = contactArea.probe();
       // log the contact.
       isInContact = false;
       i--;
   }
}
```

Listing IX-3: The skeleton of a Java method that uses data stored in duty cycle objects to perform contact probing.

X. Conclusion & Future Work

In this project, a reinforcement learning algorithm for implementation on constrained wireless sensor nodes attempting opportunistic data transfer to mobile data collectors was designed and evaluated. A description of the learning task facing such sensor nodes was also formulated.

The algorithm, when tested in simulation on simple patterns of pedestrian behaviour, was shown to successfully learn both hourly and daily patterns. The reward function specified for the algorithm was compared with a function from the literature. In the reference scenario of the present study, the custom reward function was observed to be more cost effective in terms of energy expenditure at the node, as it granted a greater reward to a low duty cycle during times of high activity. The reward function defined in the literature was also found to be biased towards a high duty cycle. An instance of the temporal credit assignment problem was discovered that, through further investigation might also be discovered in works from the literature.

The algorithm designed in this project can be extended in order to facilitate the implementation of Data Pre-Forwarding in dense wireless sensor networks. A possible extension can involve increasing the dimensions of the utility table to include models of pedestrian activity learned, and shared by neighbouring nodes. In this way, at each point in time a decision could be made based on the best route to a data collector, rather than limited local information.

More rigorous experiments can be carried out to further evaluate the algorithm in realistic simulations of WSNs and their environments created in Cooja. The ability of the algorithm to learn effectively in the presence of noisy information is recommended for investigation. This noise could be the result of repeating patterns with variance or a low level of background activity distinguished from rush hours. Examples of patterns that the algorithm is unable to learn due to the bias introduced in the design process should be found and analysed. Finally, the two exploration policies discussed in this report should be compared with each other.

XI. Acknowledgements

I would like to thank Dr Ken Brown for his supervision during this project. I would also like to thank Dr Xiuchao Wu for his assistance.

XII. Bibliography

[1] I. F. Akyildiz, Y. Sankarasubramaniam, E. Cayirci and W. Su, “Wireless Sensor Networks: A Survey,” *Computer Networks,* vol. 38, pp. 393-422, 2002.

[2] X. Wu, K. Brown and C. Sreenan, “Data Pre-Forwarding for Opportunistic Data Collection in Wireless Sensor Networks,” Cork, 2012.

[3] K. Shah and M. Kumar, “Distributed Reinforcement Learning Approach to Resource Management in Wireless Sensor Networks,” Arlington, 2007.

[4] K. Shah, M. Di Francesco, G. Anastasi and M. Kumar, "A framework for Resource-Aware Data Accumulation in sparse wireless sensor networks," *Computer Communications,* vol. 34, 2011.

[5] C. J. Watkins and P. Dayan, “Q-Learning,” *Machine Learning,* vol. 8, pp. 279-292, 1992.

[6] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 2 ed., Prentice Hall, 2002.

[7] X. Wu, K. N. Brown and C. J. Sreenen, “SNIP: A sensor node-initiated probing mechanism for opportunistic data collection in sparse wireless sensor networks.,” in *IEEE INFOCOM Workshops*, 2011.

[8] M. C. Gonzáles, C. A. Hidalgo and A.-L. Barabási, “Understanding individual human mobility patterns,” *nature,* vol. 453, pp. 779-782, 5 June 2008.

[9] T. M. Mitchell, “The need for biases in learning generalizations,” 1980.

[10] R. S. Sutton, “Temporal Credit Assignment in Reinforcement Learning,” 1984.

[11] C. O Reilly, R. Mohan and R. Lunde, “aima-java source code repository,” [Online]. Available: https://code.google.com/p/aima-java/.

[12] “Get Started With Contiki,” [Online]. Available: http://www.contiki-os.org/start.html.

[13] G. e. a. Erich, Design Patterns: Elements of Reusable Object-oriented Software, Addison Wesley, 1995.

[14] Oracle Corporation, “LinkedHashMap (Java Platform SE 7),” [Online]. Available: http://docs.oracle.com/javase/7/docs/api/java/util/LinkedHashMap.html.