FAKULTÄT FÜR
INFORMATIK

Otto-von-Guericke-University Magdeburg

Faculty of Computer Science

Chair of Computational Intelligence

Predicting Future Network Topology in Wireless Sensor Networks with Mobile Nodes

Master Thesis

Author:

David Atienza Fernandez

Supervisor:

Sanaz Mostaghim

Advisors:

Christoph Steup and Dominik Weikert

Magdeburg, 24.03.2022

Contents

Acronyms

MOTA Multi-Objective Task Allocation

A-MOTA Availability-Aware Multi-Objective Task Allocation

M-MOTA Mobility-Aware Multi-Objective Task Allocation

CTMC Continuous-Time Markov Chain

DAG Directed Acyclic Graph

DTMC Discrete-Time Markov Chain

IoT Internet of Things

LSTM Long Short-Term Memory

M2M Machine to Machine

NSGA-II Non-Dominated Sorting Genetic Algorithm II

PSO Particle Swarm Optimization

RFID Radio Frequency Identification

RNN Recurrent Neural Network

WSN Wireless Sensor Network

A Availability

L Latency

NL Network Lifetime

PC Power Consumption

R Reliability

Abstract

This paper proposes multiple network topology prediction algorithms which can be used for Mobility-Aware Multi-Objective Task Allocation (M-MOTA) within wireless sensor networks with mobile nodes. The algorithms all predict future network topology so that M-MOTA can optimize task deployment for possible network states. To this end, a scenario where cars drive around a smart city, acting as mobile nodes in a network, was thought of to create a wireless sensor network with high mobility. Multiple task allocations which were optimized using the prediction algorithms are evaluated in that scenario. The mobility data for this scenario is provided by newly created mobility models, which are also introduced in this paper. Using a simulation environment with the introduced mobility models, the prediction algorithms are evaluated measuring Power Consumption, Latency and successful task execution. Multiple approaches show promising performance in all three categories when compared to the theoretical best.

1 Introduction

The Internet of Things (IoT) is a rapidly spreading technological advancement within the second and third decade of the 21st century. It describes group of physical objects equipped with hardware to communicate with each other via the Internet [1], enabling information sharing between devices on larger scale than before.
The range of applications the Internet of Things can help enhance is broad. Consumer applications like smart home devices, smart thermostats or lights [2] can increase the quality of life for their consumers.
In industrial applications the Internet of Things can be used to monitor soil properties for more effective farming [3] or to monitor the wear of heavy machinery [4] in order to increase safety and cost effectiveness when replacing parts.
The public sector can use IoT systems for infrastructural monitoring in order to optimize the maintenance and expansion of roads, monitoring the environment in hospitals [5] and enhancing other critical infrastructure.
In smart cities [6], the Internet of Things can help reduce traffic congestion [7], optimize parking situations for cars, make roads safer with intelligent traffic regulations and help lessen the burden on the climate by intelligently switching traffic lights [8] and improving public transportation.
Wireless Sensor Networks (WSNs) are closely connected to the IoT. The term describes a wireless network of nodes, each one monitoring the environment around it via a sensor and sharing it with other nodes in a decentralized manner [9] . Usually, these nodes are cheap, small and run via a low power battery, making them easy to replace.
WSNs can not only be achieved by scattering nodes throughout the environment, but also by integrating them into things which are distributed over an area, for example in pipes, machinery, cars and other things. It is assumed by [10] that the IoT will be so widespread in a few years that entire cities will transform by connecting nearly every part of them to a big network, transforming them into what we call **smart cities**.

1.1 Motivation

This work considers the scenario of a smart city. Its focus will be on the cars driving around in the city, which are assumed to be part of the Internet of Things and act as mobile nodes in a Wireless Sensor Network.
Given this scenario, trying to maximize the quality of the network is vital. A poor quality network affects the communication among its nodes in a negative way, which gives rise to problems. In our case a negative influence on node communication could interrupt traffic. The cars in our city are assumed to do IoT assisted route planning [7], whose interruption can result in traffic congestion in the streets. This in turn causes loss of time and money for the city inhabitants and can even cause potentially dangerous traffic situations because a higher car density makes it more likely that accidents occur [11].

In order to evaluate the quality of a Wireless Sensor Network, quality metrics which consider the unique properties and problems a smart city WSN has need to be defined.
These quality metrics for Wireless Sensor Networks we define are Reliability (R), Availability (A), Latency (L) and Power Consumption (PC). The three metrics R, A and L were all used in [12] and [13] and are generally applicable to measure the quality of a WSN. Additionally, a fourth metric called Network Lifetime (NL) was introduced in [13], which we will be replacing with the PC metric.
Since the cars considered in this scenario have high capacity batteries, the network lifetime should not be affected by the Power Consumption of the WSN. Therefore it makes more sense to consider Power Consumption directly instead.

The movement of the cars in a smart city introduces dynamic and frequent change of network topology by frequently breaking existing connections between the cars, likely compromising the quality of the network as defined by our metrics.
Considering that, a method to improve the network quality is desirable. One way to do that is predicting future topology changes within the network and adapting to them before they occur. One could for example reallocate tasks within the network using the predicted topology changes, thereby lessening the negative impact introduced by the high mobility of the nodes.

1.2 Goals

1.2.1 Mobility Models

In order to create a prediction method for topology changes within a Wireless Sensor Network spread throughout a smart city, data with which to evaluate the quality of the prediction and the resulting network quality is needed. Since real data traces for many connected cars in a smart city are very time-intensive to collect, we aim to introduce a mobility model within a simulation of a Wireless Sensor Network which produces similar data instead. This synthetic data can then be used instead of real traces to evaluate the predictions.
Approximating a smart city, the mobility model will have characteristics of a Manhattan model, whose topography approximates many cities in the real world. The classical Manhattan mobility model however has nodes moving around randomly within its topographic constraints, which makes the data generated suboptimal when trying to predict future actions.
The cars in our scenario do not drive around randomly, deciding where to turn anew at every intersection but instead follow predictable patterns. Said patterns are created by the dynamic routing [7] introduced by the cars being connected to the Internet of Things. Therefore our first goal is to introduce a new mobility model whose aim is to produce data which follows a predictable pattern approximating the real life behaviour of cars. This will then make the prediction of future node positions and network topology within a simulation using this mobility model meaningful.

1.2.2 Prediction Methods

The prediction methods introduced in this paper have the goal of providing prediction data which can be used by the Mobility-Aware Multi-Objective Task Allocation (M-MOTA) algorithm. Said algorithm distributes tasks to nodes, optimizing for the quality metrics of Wireless Sensor Networks mentioned before and will be slightly modified in order to optimize for Power Consumption instead of Network Lifetime.

The Mobility-Aware Multi-Objective Task Allocation algorithm, which considers node mobility when allocating tasks, is an extension of Multi-Objective Task Allocation (MOTA). By considering future node positions, it can better allocate tasks throughout a WSN than the original MOTA algorithm, but in turn it needs a predictor for the future network topology. The task allocation in a WSN featuring high mobility, like the smart city scenario we use here, should be improved when using the prediction methods introduced in this paper.
The goal is to come up with multiple prediction algorithms using different established techniques and comparing them with each other in order to determine which one is best suited for use with M-MOTA.
Additionally, other criteria will be introduced in order to evaluate and compare the prediction techniques other than the performance of each predictor in combination with M-MOTA. These criteria include:

- The accuracy of the prediction method for the position of each node.
- The accuracy of the prediction method for the topology.
- The accuracy decrease of the prediction method depending on the length of the look-ahead into the future.

The accuracy of the prediction for each node and for the whole topology will make it possible to evaluate how the absolute accuracy of a prediction affects the performance of the M-MOTA algorithm and will also provide an overview of how good a prediction method is without looking at the quality of the task allocation.
Additionally, the accuracy decrease when increasing the look-ahead time will be a useful evaluation tool in order to see if different algorithms might be useful for different types of Wireless Sensor Networks and different kinds of use cases, depending on how far the look-ahead time needed is.

1.3 Thesis Outline

In Chapter 2 all fundamentals required to understand the algorithms and concepts introduced in later chapters can be found.
Chapter 3 discusses related works such as the Availability-Aware Multi-Objective Task Allocation (A-MOTA) algorithm which is another variation of the original MOTA, as well as work predicting car positions based on their previous GPS data.
The new mobility model and prediction methods mentioned above will be introduced in Chapter 4.
The newly created mobility models and prediction methods will be evaluated in Chapter 5, after which the conclusion and possible future work will be discussed in Chapter 6.

2 Prerequisites

This chapter describes concepts that are necessary to understand the remaining chapters of this thesis. Readers that are already familiar with the topics may skip ahead at their own discretion.

2.1 The Internet of Things and Smart Cities

The Internet of Things is a term for a set of technologies, systems and a vision to connect things in our physical environment via the internet [14]. It is about remotely monitoring, controlling and automating things by equipping them with sensors and means of communication [1], enabling them to interface and interact with one another via the internet.
This means that the internet will change from its form today primarily consisting of human to human communication. In an IoT future, the primary communication in the internet will be Machine to Machine (M2M) [15] with human to human communication only making up a small part of the whole.
The primary enablers of the rise of IoT systems have been the miniaturization of electronic devices, as well as Radio Frequency Identification (RFID) and other wireless technology becoming cheap enough to be included in many things [16].
In the future, entire cities will potentially turn into a part of the Internet of Things known as **smart cities**. These smart cities make use of the Internet of Things by collecting and analyzing data in order to improve their infrastructure, public services and general quality of life for their inhabitants [17].

Examples for the applications of IoT solutions in a smart city include:

Smart utility meters and grids [17] These tools allow citizens to manage points of high energy consumption in their households and give energy providers the possibility to manage energy flow throughout a city, which is especially important with the rise of green energy solutions and their decentralized and fluctuating power generation.

Smart transportation methods [17] These methods include public transport innovations which let citizens see the location and arrival time of public transport ahead of time, as well as connected cars and traffic lights making traffic flow more smoothly and helping to prevent accidents.

Smart waste management solutions [10] Smart waste management solutions help plan the routes of waste managements services, making them more efficient.

Smart air quality monitors [10] Smart air quality monitors can alert people of unsafe pollutant levels in parts of a city and help identify and eliminate choke points with high pollution.

Smart emergency response systems [9] Smart emergency response systems can help identify the areas most affected by emergencies like natural disasters or fires and optimize response time and effectiveness of emergency services.

Although these examples already cover a wide variety of use-cases, there are many more being explored [18]. Smart car sharing services [19] for example can help lessen the huge traffic problems many modern cities face by providing an easy way to access a car without owning one, thereby decreasing the space required in a city for parking. This, combined with self driving cars, one of the most well known examples of IoT technology, can change the way transportation works fundamentally.
Widely spread, IoT assisted driving or fully self driving cars can exchange their positions with each other, making traffic in a city fully transparent for every participating machine. This in turn can be used to predict where heavy traffic will happen and makes adaptive rerouting [7] in order to avoid traffic jams possible. These connected cars combined with smart traffic lights [8], which would also know the positions of cars in advance, could also be used to adapt red-green cycles at intersections, significantly improving upon one of the most common choke points in traffic, thereby increasing the traffic flow overall and improving traffic safety [20].
The Internet of Things however does not only present new opportunities. In [21] many challenges that have to be addressed in order for the Internet of Things to fulfill its potential are pointed out. For the Internet of Things to work, internet has to be made available everywhere and at no cost. Additionally, power generation and storage have to be available for all devices connected. Security issue concerns also arise, as critical IoT will increasingly come into the focus of malicious hacking groups. With the rise of M2M communication, magnitudes of data never seen before will have to be exchanged and processed as well. One big new source of data will be wireless sensor networks.

2.2 Wireless Sensor Networks

Wireless Sensor Networks are an emerging type of wireless network that contain distributed and independent sensor devices which monitor physical or environmental conditions around them [9]. They consist of small-scale, cheap sensor nodes that communicate with each other, exchanging the information each of them obtains. The strength of WSNs arises from the huge number of interconnected nodes that can be employed simultaneously [9], making the failure of an individual one noncritical and replacement cheap.
The main difference between the sensor nodes used in WSNs and devices in classical wireless networks is that the sensor nodes are small, cheap devices, usually only powered by a small battery. This makes energy usage a new problem that is not present in other wireless networks [22]. Arising from this problem are different mitigations and solutions to make these networks last longer. One important thing is the invention of many **low-power network protocols**. These protocols include *ZigBee*, *Bluetooth LE*, *6LoWPAN*, *RFID*, *NFC* [23] and many more examples widely used in current IoT applications, which all include specific design decisions that facilitate low power consumption.

To not only decrease the consumption of energy but actually gain back depleted energy over time, energy harvesting techniques are also being researched and deployed [24]. These techniques include the conversion of solar, wind, tidal and geothermal energy to electricity in order to keep a node from running out of power.
Wireless Sensor Networks are integral to the Internet of Things in many cases, for example in weather monitoring [25], soil quality monitoring [26], battlefield surveillance [27], tracking of human and animal movement [9], and monitoring of equipment condition [28].

The main applications of WSNs can be split into three categories: **area monitoring**, **entity monitoring** and the combination of both, **area-entity monitoring** [22]. Different circumstances make WSNs important for all these use cases.
For area specific applications like weather or soil quality monitoring [25][26], a large area has to be monitored. A WSN created by scattering sensors over a wide range makes it possible to gather many data points for the whole area and send them all back to a base station to be processed and create a big picture view. This can also facilitate the early detection of natural disasters like hurricanes or wildfires by combining many data points over a wide-spread area in order to detect anomalies [9].
For entity specific applications like animal monitoring [29], a WSN can be employed in order to observe large scale migrations and other behaviour. By using a large amount of sensors, more animal behaviour and movement data can be gathered than any group of humans could ever reasonably collect manually and without the need for a group of humans to interact with the ecosystem.
Additionally, WSNs can be employed to spread out sensors in areas otherwise not easily accessible to humans, for example to observe ocean currents or animal behaviour, or to collect data from a battlefield [27] or area devastated by a natural disaster [9] in order to deploy humanitarian aid more quickly and efficiently.

2.2.1 Task Allocation

Because Wireless Sensor Networks contain a large number of decentralized nodes, sending raw data out of the network is undesirable due to the fact that wireless data transmission often is more resource intensive than local computation [30]. One can reduce the power consumption by aggregating and processing the information gained from the nodes inside the network, making it the better alternative [30].
If one considers a typical WSN scenario as laid out in the examples above, data is collected by sensor nodes throughout some area and needs to be made available at central sink nodes where it is further processed. In many such cases, the data collected by different sensors can be jointly processed while being forwarded towards the sink [31], which is where in-network aggregation and processing comes into play.
How this in-network data processing is executed (i.e. which tasks are allocated to which node) has significant impact on the energy consumption and overall network efficiency due to the increase or reduction of the number of data transmissions and length of individual packets depending on the allocation [31].

A complication with optimal task allocation for WSNs is that it is an instance of the generalized assignment problem. This kind of problem has been proven to be NP-hard[32], making finding the optimal solution by classical means unviable given a non-trivial amount of nodes and tasks [33].

To see why that is the case, one can consider an example network [13] of homogeneous nodes $\mathcal{N} = \left\{N_0, \ldots, N_i, \ldots, N_{|\mathcal{N}|-1}\right\}$ of size $|\mathcal{N}|$ to which a set of tasks $\mathcal{T} = \left\{T_0, \ldots, T_i, \ldots, T_{|\mathcal{T}|-1}\right\}$ of size $|\mathcal{T}|$ are assigned. We can calculate the total number of possible assignments to be $|\mathcal{N}|^{|\mathcal{T}|}$. If all tasks are independent of one another, this gives us a time complexity of $\mathcal{O}(|\mathcal{N}| \cdot |\mathcal{T}|)$ to distribute all tasks optimally, because the best node for each task can be selected individually. However, as soon as soon as there exist some sub-tasks of a single global task, the search space increases exponentially, making finding the optimal solution impossible in a reasonable amount of time if $|\mathcal{T}|$ is big enough.
These circumstances make heuristic algorithms necessary to approximate good solutions for task allocation instead [33]. Examples of meta-heuristic algorithms which can and have been employed to find solutions for task allocation problems include genetic algorithms like Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [12] and Particle Swarm Optimization (PSO) [33].
The following, more formal problem formulation was introduced in [12] and [13].

A WSN and its nodes can be modelled as an undirected graph $G_{Net} = (\mathcal{N}(t), E_{Com}(t))$. Each node $N_i \in \mathcal{N}$ has a position $\vec{x}_i(t)$ and energy $E_i(t)$ at any given time t.
Every node also has an inherent node latency l_i given by the function $L_i(p_j)$, which depends on the processing cost for a given task, as well as an inherent power consumption c_i given by the function $C_i(p_j)$ which also depends on the processing cost for the task. Two nodes that share an edge $e_{ij} \in E_{Com}(t)$ are able to communicate with each other with a certain amount of latency $l_{ij}(t)$ between them using a specific amount of energy $E_{ij}(t)$.

The global task and its dependent sub-tasks can be represented as a Directed Acyclic Graph (DAG) $G_{Task} = (\mathcal{T}, E_{Task})$ with each vertex in the graph being a task $T_i \in \mathcal{T}$ and each edge being $e_{ij} \in E_{Task}$ being a dependency between the two tasks T_i and T_j.
Additionally, each vertex has a weight p_i which represents its associated processing cost and each edge e_{ij} has a weight w_{ij} which represents the cost to exchange information between the two tasks T_i and T_j if they are being executed on different nodes in the network. Finally, each task may also have a constraint S_i which restricts the space it has to be executed in.

The task allocation itself can be represented by a function $a : \mathcal{T} \to \mathcal{N}(t)$ which maps each vertex in task graph to a vertex in the network graph at a specific time t while fulfilling the constraint that any node executing a task needs to be connected to all nodes executing directly connected tasks.
The goal when allocating tasks is to find a series of valid allocations $\mathcal{A} = (a_0, a_1, \ldots, a_n)$ with associated start times t_i^{start} and end times t_i^{end}, that maximizes a certain set of quality parameters for the network.

2.2.2 Quality Metrics

The performance of a WSN can be impacted by bad task allocation, degradation of its physical components over time and other internal and external influences temporarily or permanently disabling nodes in the network or disrupting communication between nodes. To measure the quality of a network in these circumstances, multiple metrics can be used. Metrics which are needed for this work will be defined in this chapter.

Network Lifetime (NL) is defined by Equation 2.1. It is the time between the first and last valid allocations of tasks within a WSN.

$$NL(\mathcal{A}) = t_n^{end} - t_0^{start} \tag{2.1}$$

Power Consumption (PC) of an allocation series $\mathcal{A}$, as seen in Equation 2.2 is equal to the sum of the Power Consumption of all allocations within the series. The Power Consumption $PC(a_t)$ of one allocation a_t is defined as the sum of the Power Consumption of all connected tasks.
The Power Consumption E_{ij} needed between two tasks T_k, T_l is the sum of all the power consumed along the path $P_{kl}(a_t)$ plus the inherent power consumption c_k.

$$PC(\mathcal{A}) = \sum_{a_t \in \mathcal{A}} \left[\max_{T_k, T_l \in V_{Tasks}} \sum_{e_{ij} \in P_{kl}(a_t)} E_{ij} + c_i \right] \tag{2.2}$$

The **Availability (A)** is the fraction of time a network is working on a global Task T. It is defined by Equation 2.3.

$$A(\mathcal{A}) = 1 - \frac{\int_{t_0^{start}}^{t_n^{end}} e(\mathcal{A}, t) dt}{NL(\mathcal{A})} \tag{2.3}$$

with $e(\mathcal{A}, t)$ being a boolean function on whether the network has encountered an error, as shown in Equation 2.4.

$$e(\mathcal{A}, t) = \begin{cases} 0, & a_i \text{ is valid}, t_i^{start} \leq t < t_i^{end} \\ 1, & \text{otherwise} \end{cases} \tag{2.4}$$

The **Reliability (R)**, as shown in Equation 2.5 is the time a network needs until encountering its first fault $t_{\mathcal{A}}^{error} = \underset{t}{\operatorname{argmin}} \left(e(\mathcal{A}, t) = 1 \right)$, at which point it can no longer work on the global task T.

$$R(\mathcal{A}) = \frac{t_{\mathcal{A}}^{error}}{NL(\mathcal{A})} \tag{2.5}$$

Finally, the **Latency (L)** of an allocation series $\mathcal{A}$ is the maximum latency of all allocations within the series. The latency $L(a_t)$ of one allocation a_t is defined as the maximum latency of all connected tasks. The latency l_{ij} between two tasks T_k, T_l is the sum of all latencies along the path $P_{kl}(a_t)$ plus the inherent node latencies l_k. All of this combined is represented by Equation 2.6.

$$L(\mathcal{A}) = \max_{a_t \in \mathcal{A}} \left[\max_{T_k, T_l \in V_{Tasks}} \sum_{e_{ij} \in P_{kl}(a_t)} l_{ij} + l_i \right] \tag{2.6}$$

2.3 Mobility Models

Mobility models describe the movement behaviour of entities over time and attempt to mimic the real life behaviour of those entities [34]. This can either be done by recording and playing back movement of entities that were actually observed, or by building a synthetic model which attempts to recreate real movement patterns without the use of traces [34].
The former approach has the advantage of perfectly representing real life movement behaviour, but can only ever cover the data points that were actually traced. For use-cases which require more data than can be acquired by traces, or if acquiring traces is not desirable for other reasons, synthetic models are needed.
Recreating movement with synthetic models has the advantage of being able to generate any amount of data, but comes with the the downside of synthetic models usually only being implemented as simple random traffic generators [35]. More advanced models require more time to develop, as well as more resources to execute.
Synthetic mobility models can broadly be split into two distinct categories [36]: Homogeneous and Heterogeneous models. Additionally, they can be further divided by their properties in order to more easily find a suitable model for a desired use-case.

Bai et al. [37] classify models by their basic mobility characteristics:

Random models — Entities in these models move randomly. The models can be subdivided further by the degree of randomness.

Models with temporal dependency — In these models, future movement is influenced by past movement.

Models with spatial dependency — Here, entities are influenced by each other and move in a correlated manner.

Models with geographical restrictions — These models present constraints like streets, roads, obstacles, etc., that hinder the movement of nodes in some way.

Roy [38] provides an alternative classification, dividing models into seven groups:

Individual mobility models — The movement for each entity is calculated individually, making entities act independent from one another.

Group mobility models — The movement for a group of cooperative entities is calculated, making them move in a correlated manner.

Autoregressive mobility models — The mobility patterns are correlated with the mobility states.

Flocking and swarm mobility models — These models imitate self-organizing swarms (like birds, ants, etc.) in nature.

Virtual game-driven mobility models — Interaction with all other entities within a network is taken into consideration when calculating the movement behaviour of a node.

Non-recurrent mobility models Here, it is assumed that the network permanently changes its topology over time, making entities move in ways that are not correlated to previous movement patterns at all.

Social-based mobility models For these models, the entities within it are considered as a community of groups within a society.

Furthermore, specific metrics which can be used to measure the properties of any given mobility model exist more exactly as well [39]:

Network diameter By modeling a network as a graph, the graph diameter, i.e. the maximum distance in hops between two nodes, can be measured.

Neighborhood instability The neighborhood instability can be estimated as the number of radio link appearances and disappearances.

Node distribution When representing the node density in some small surface areas, the geographical nodes distribution can be measured.

Repetitive behavior A quantitative metric related to the property of a node to exhibit the same movement after a given time.

Clustering coefficient The clustering coefficient is the ratio of the radio links among neighbors and the number of neighbors.

Using all of these classifications and metrics, a suitable mobility model for any given problem can be chosen based on its characteristics.
An important thing to note, however, is that a trade-off must be made every time a synthetic model is chosen. Less complicated models tend to generate synthetic data approximate real data less accurately, but more complex models have other problems that might not make it desirable to use them.
For one, the more complex a model is, the more computational time is needed in order to simulate the movement of the nodes within the model. Secondly, developing a more complex model explicitly designed for a specific use-case needs time which could be used to trace real data instead, which might make a complex synthetic model the worse option. Thirdly, even using an already existing more complex model requires tuning many parameters to generate data suitable for the desired use-case, again making it a time and effort trade-off versus using a simpler model or collecting real data traces.
This means that a more accurate model might not always be the correct choice even though it generates data that more closely approximates real life movement. Each use-case therefore must always be evaluated beforehand in order to find what quality of data is needed before choosing a specific model.

2.3.1 Homogeneous Models

Homogeneous models are based on having a group of cooperating entities all generally moving as a group according to the specific model [36]. Within the general group, each node then has smaller individual fluctuations for its movement. An example for an application making use of a homogeneous models would be a flock of birds migrating.

This category primarily includes Bai et al.'s [37] *Models with spatial dependency* and Roy's [38] *Group mobility models*, *Flocking and swarm mobility models*, *Virtual game-driven mobility models* and *Social-based mobility models*. In addition some models might fall into other categories of Bai et al. [37], as they are not mutually exclusive.

Important models in this category are:

Reference point mobility [40] In this model each node belongs to a group in which every node follows a group leader who determines the group's movement behavior. The nodes in a group are usually randomly distributed around the group leader and then use their own mobility model to calculate smaller individual fluctuations.

Group force mobility [41] Each node in this model evaluates what action it should take based on the properties of its neighbors. For all neighboring nodes which are located within a certain range, the node determines if it belongs to the same group or different group and then calculates a repulsion or attraction force.

Autoregressive group mobility Here, the movement of nodes is calculated using an autoregressive model [42]. This means real-life training data is used, from which a model is generated by using autoregression.

Swarm group mobility [43] This is not a single model, but a group of models which all aim to emulate different kinds of swarms. Each model simulates one type of swarm, which consist of animals, soldiers, cars on a highway, or other things behaving in a swarm-like way.

Community based mobility [44] This model groups nodes together in a way that is based on social relationships among them. This grouping is then mapped to a topographical space, with movements influenced by the strength of social ties.

Orbital based mobility [45] Nodes in this model are moving in an orbit, mostly representing satellites.

Virtual game driven mobility [38] In virtual game-driven mobility models, a group of mobile nodes based on player strategies are mapped from the real world to virtual agents.

2.3.2 Heterogeneous Models

In heterogeneous models, every entity moves independently from any other entity that exists. Thus, as opposed to homogeneous models, entities are not cooperating as a group and can generally be seen as fully autonomous individuals. This category includes every category of Bai et al. [37] that is not spatially dependent and primarily includes Roy's [38] *Individual mobility models*.

Notable models in this category are:

Random walk mobility [38] Each movement in the Random Walk Mobility Model occurs in either a constant time interval or a constant distance traveled, after which a new direction and speed are chosen from a predefined lower and upper bound for the speed and a randomly chosen angle.

Random waypoint mobility [34] In the Random Waypoint Mobility Model, a node chooses a random destination in the simulation area and a speed that is uniformly distributed between a predefined lower and upper bound. The node then travels toward the chosen destination at the selected speed. Upon arrival, the node pauses for a specified time period before starting the process again.

Gauss-Markov mobility [46] The Gauss–Markov Mobility Model was designed to adapt to different levels of randomness via one tuning parameter. Each node is initially assigned a speed and direction. At fixed intervals of time, the speed and direction of each node is updated in accordance to a Gauss-Markov model.

Geographic constraint mobility [47] This model introduces obstacles and other geographic constraints that limit how nodes can move.

Graph-based mobility [48] In this model, nodes move in accordance to a graph. A node may only move from one vertex on the graph to another vertex connected to it via an edge. The vertices therefore represent points of interest nodes can move to and the edges represent the paths they can take.

Fluid flow mobility [38] The movement of nodes is modeled like a fluid, creating smooth non random trajectories and smooth continuous velocity for all nodes.

Manhattan mobility Explained in further detail below.

2.3.3 Manhattan Mobility Model

The name of the Manhattan Mobility Model is, just like the name of the Manhattan norm (otherwise known as l_1 norm), inspired by the grid-like street layout of the island of Manhattan. The model assumes a topology of a grid of equidistant streets, giving every intersection four equally distant neighbouring intersections.
The model can take on two different forms, either repeating itself infinitely by making nodes reappear at the opposite end of the model when leaving the bounds on one side, or by having hard model boundaries where intersections have three neighbours at the edges and two neighbours at the corners.
Every node within the model initially gets assigned a random starting point at an intersection within the model boundaries, as well as a randomly chosen starting direction within the topographic constraints. The starting speed for each node is also randomly chosen within predefined upper and lower bounds [49]. Each of those random starting parameters is generally generated using a uniform distribution.

Given its start parameters, each node then starts moving in a straight line until it enters an intersection. After entering an intersection, a random choice is made as to where the node will turn. The chance to continue straight at any given intersection is 50%, while turning either left or right have a 25% chance each [49].
When using hard boundaries, a node has an equal probability to turn left or right when encountering an intersection at an edge and no choice where to turn at a corner, since nodes can not turn around at an intersection in this model.
The model can be tuned to have nodes that ignore each other completely, or to simulate nodes that can not overtake each other and instead decrease their speed as long as they move behind another node.
A visualization of how nodes in a Manhattan mobility model move can be seen in Figure 2.1.

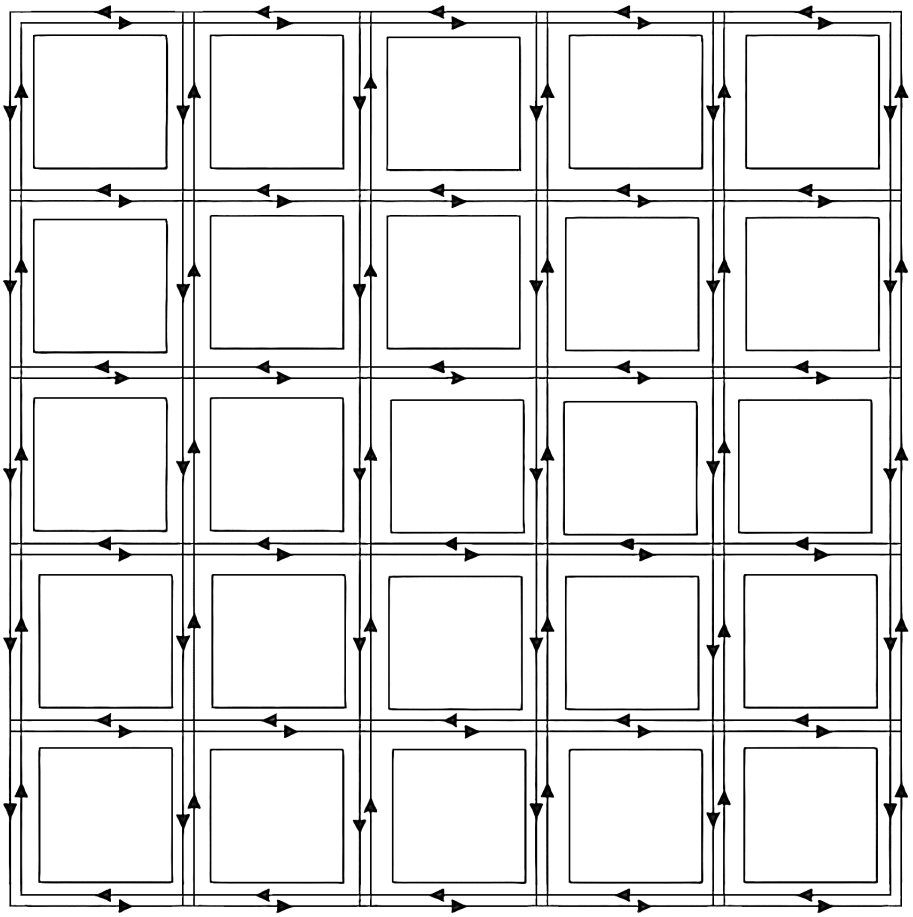

Figure 2.1: Example of a Manhattan Model

Using the classification [36] above, this model is heterogeneous, since its nodes do not move in a cluster or are otherwise cooperating in a group.
Using Bai et al.'s [37] classification, this is a random model with geographical restrictions where movement has temporal dependence. Depending on whether nodes can slow down for each other or ignore one another when overtaking, there also is a degree of spatial dependence in the model.
Using Roy's [38] classification, this would be an individual mobility model because the movement for each node is calculated individually.
The Manhattan mobility model can be used very effectively to model nodes moving around in urban environments, because it approximates the geographical restrictions urban environments impose on its mobile actors [36].

2.4 Data Forecasting

Data forecasting describes the act of predicting future data a system will produce as accurately as possible using all information that is known about it [50]. In order to do that, all information available can be used, including known information about the inner workings of the system and historical data produced by it.
There exist two main data forecasting methods, one being qualitative forecasting and one being quantitative forecasting. Qualitative forecasting describes the act of trying to forecast data without having information about the past available, while quantitative forecasting is employed to derive predictions about future data from information about the past and the system producing the data.
Following from here, only quantitative data forecasting will be further explained and will simply be referred to as data forecasting [50].
Examples of data forecasting include predicting future weather data, economic data, positional data, and many others. Resulting from that are many kinds of methods used for data forecasting, which can be used depending on how much historical data exists and what is known or assumed about the system generating the data. In the following, two methods relevant for this work will be explained.

2.4.1 Markov Chains

A Markov model is a stochastic model describing a process that experiences transitions from one state to another by following a set of probabilistic rules [51]. The defining characteristic of any Markov model is that the possible future states are only dependent on the current state of the process, no matter how it arrived at its present state, which is called the Markov property.
Markov chains are the simplest type of Markov model. They model the state of a system with a random variable that changes over time. Since the Markov property is assumed to hold true, this implies that the distribution for this variable depends only on the distribution of a previous state.
If the process moves from one state to another in discrete timesteps, it results in a Discrete-Time Markov Chain (DTMC), whereas a continuous time process is called a Continuous-Time Markov Chain (CTMC) [51]. A visual example of a simple, three state Markov chain can be seen in Figure 2.2. In the example, every state has a transition probability of 0.5 to transition to each of the other two states and no state can transition to itself.

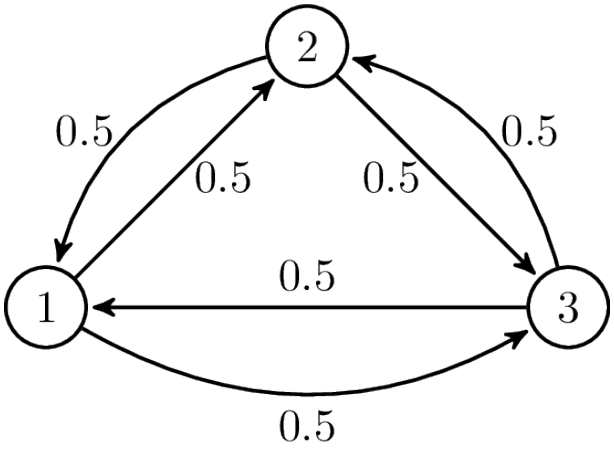

Figure 2.2: Example of a simple Markov chain [52]

The transition between states for a markov chain X at a given time t can be represented by a transition matrix P_t. Given an ordering of a matrix's rows and columns by the state space S, the $(i, j)^{th}$ element of the matrix P_t is given by the Equation 2.7 [51]:

$$(P_t)_{i,j} = P(X_{t+1} = j | X_t = i) \tag{2.7}$$

Each row of this matrix is a probability vector, and the sum of its entries is one.
If data is produced by a system which can be described by a Markov chain, one can predict future data by considering the most probable future state of the system, which is easily achieved by using the transition matrix, to predict the next data points.

Other types of Markov models with different use-cases exist as well, for example:

Hidden Markov Models [54] Hidden Markov Models are Markov chains in which the system of the model has unobservable states.

Markov Decision Processes [53] Markov decision processes model decision making in discrete, stochastic, sequential environments which an agent inhabits. The environment changes state randomly in response to action choices made by the agent. The state of the environment affects a reward obtained by the agent and the probabilities of future state transitions.

Markov Random Fields Markov Random Fields are random fields that satisfy the Markov property. A random field is the representation of the joint probability distribution for a set of random variables [55].

The main weakness of Markov models is that the states of a system have to be identified beforehand in order to model it, which gives rise to the desire for a method which requires less human effort.

2.4.2 Neural Networks

Artificial neural networks have been used for a wide variety of things, one of them being data forecasting. An artificial neural network in its essence is an imitation of the way the human brain processes data [56]. It consists of artificial neurons which are inspired by real biological neurons, taking in data and processing it. These neurons are also linked to each other, exchanging information, again inspired by the way a real human brain has its neurons connected.

Perceptrons

One of the most basic artificial neurons is called a perceptron. Perceptrons are the building block of nearly all neural networks employed today. A perceptron unit works in the following way [57], as can also be seen in Figure 2.3:

- A perceptron gets real inputs $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ via n edges.
- Each edge has an associated real weight, resulting in a weight vector $\vec{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$
- $\vec{x} \cdot \vec{w}$ is computed and compared to the threshold θ of the unit. If the result is equal to or greater than θ, the output of the cell is 1, otherwise it is 0.

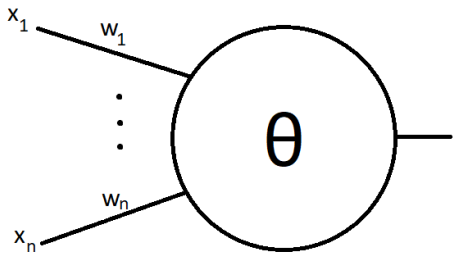

Figure 2.3: Perceptron

In addition, in many cases it is more convenient to always use 0 as a threshold and add an additional edge with the weight $-\theta$, whose input is always one. This does not affect the functionality of the perceptron and is only done for convenience [57].

In essence, a perceptron separates input data into 2 categories, which can also be represented in a geometric sense.

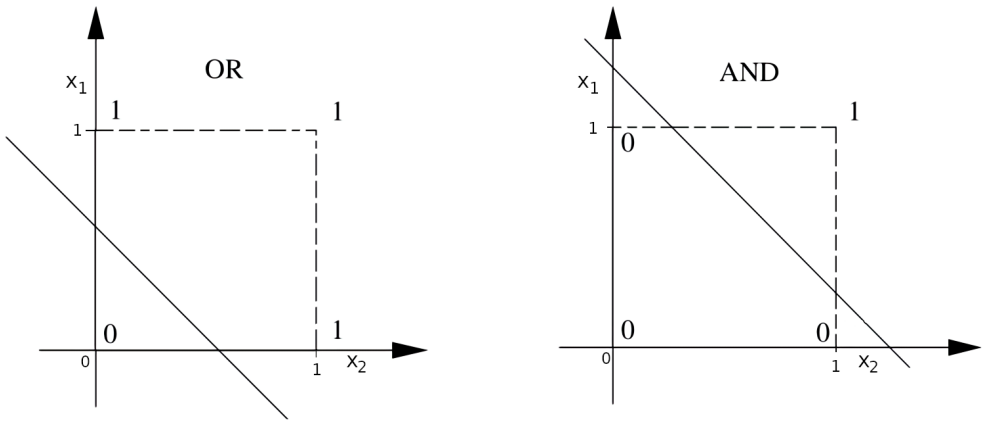

Figure 2.4: AND and OR functions, each separated by a Perceptron

As can be seen in figure Figure 2.4, the input data for the OR-function and AND-function gets separated by a boundary into two categories, making it possible to map each input to the correct output. The boundary is given by the weights $w_1, w_2, ..., w_n$ and can vary, since x_1 and x_2 are binary inputs which makes multiple boundaries valid for separation. Additionally, an important thing becomes clear when looking at Figure 2.4: The input gets separated *linearly*.
Any linearly separable task can be modeled by a perceptron given the right weights [57]. This is what allows for logical functions like the OR-function and AND-function to be modelled.
The algorithm to find said weights is called the perceptron learning algorithm, which will not be discussed here further.

Multilayer Perceptrons

A single perceptron can separate input data linearly, as was elaborated above. Unfortunately, there are functions whose input space can not be separated in such a way. One of those functions is the basic XOR-function. The reason as to why it can not be separated in such a way becomes apparent when represented visually, like in Figure 2.5.

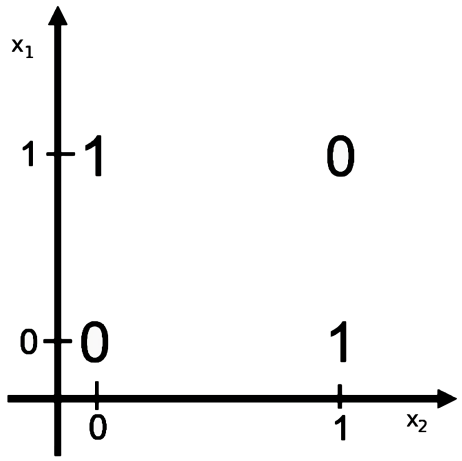

Figure 2.5: Geometrical Representation of the XOR-problem

Functions like these are known as non linearly separable functions and can not be separated by a single perceptron. One way to deal with these functions is by combining multiple perceptron units in multiple *layers* [57] .

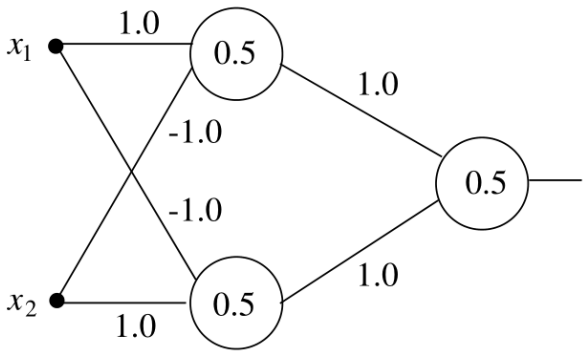

Figure 2.6: Example of a Multi-Layer Perceptron for the XOR-Function [57]

When combining multiple units like in Figure 2.6 , a region of space can be isolated, making problems like classifying the XOR-function solvable, which can be seen in Figure 2.7.

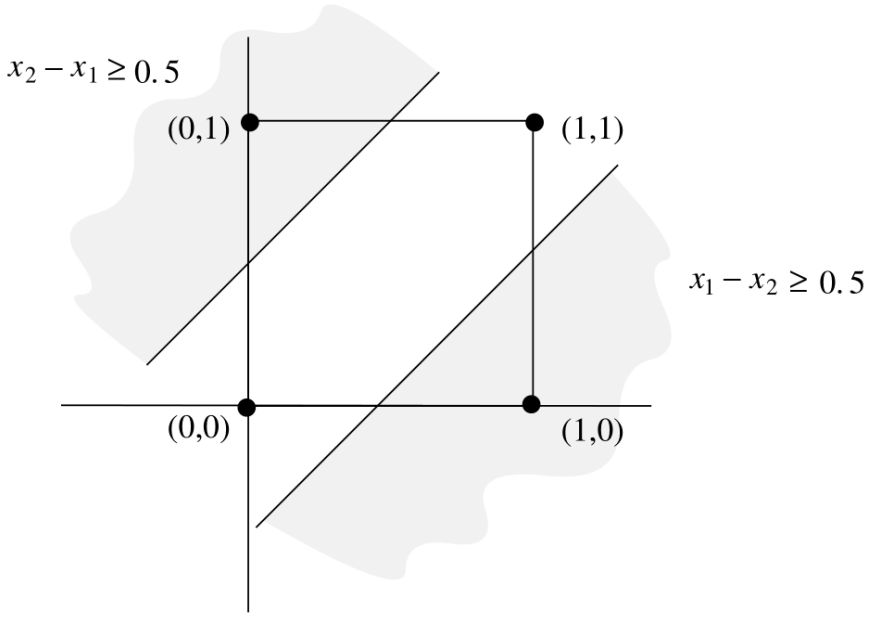

Figure 2.7: Solution to the XOR-Problem [57]

These networks of perceptrons are called multi-layer perceptrons and are one of the simplest types of artificial neural networks [57]. They consist of one input layer, one output layer and zero or more hidden layers (an example can be seen in Figure 2.8), each consisting of at least one perceptron unit. One important thing to note is that the data flows through the layers only in one direction, giving the network a clear hierarchy of layers. These multi-layer perceptrons are able to approximate any continuous function [57] and can therefore solve problems which are not linearly separable.

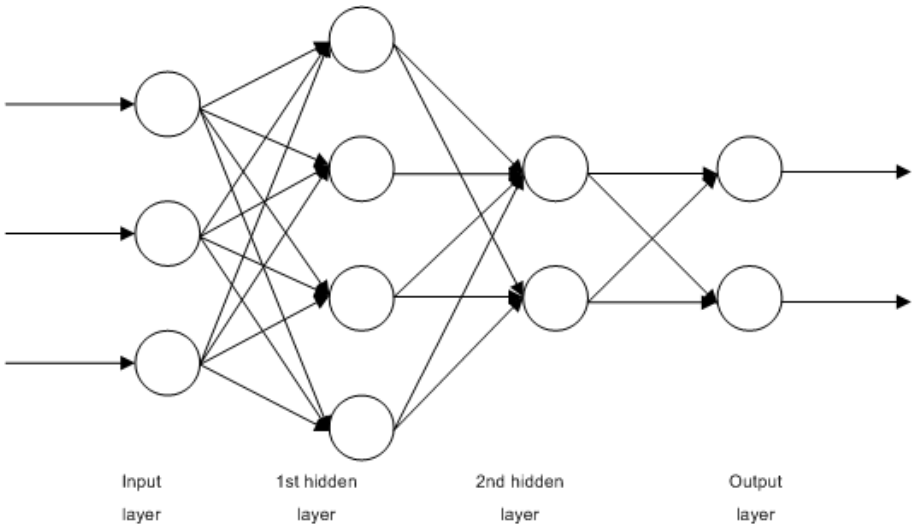

Figure 2.8: Example Layout of a Multi-Layer Perceptron [58]

Finding the correct weights for a multi-layer perceptrons to solve a given problem is harder than finding the weights for a single perceptron unit. The algorithm to find the weights for a whole perceptron network is called backpropagation. Backpropagation works by introducing a loss function with respect to the networks weights, which represents the error between the classification the network does for a given input and an expected output.
By computing the loss function for an input-output pair, the weight finding becomes an optimization problem where minimizing the cost function optimizes the weights. Minimizing the loss function can be performed by calculating the gradient and then applying a gradient descent method.
The specifics for all of these features will not be discussed here, as there are various implementations and modifications that can and have been employed to optimize the backpropagation algorithm.

Activation Functions

The classical method of calculating $\vec{x} \cdot \vec{w}$ and comparing the result to the threshold of a perceptron is outdated and rarely used anymore. The reason is that comparing the result against a threshold and using only zero or one as the output of the perceptron essentially corresponds to a Heaviside step function, which provides no useful gradient to use for backpropagation.
Instead, after calculating $\vec{x} \cdot \vec{w}$, a so called **activation function** is applied to the result. The output of the activation function is then used to determine how active a unit is. This is done because it allows for better training of the network and because the activation functions commonly used have usable gradients, unlike the classical method [59].
Commonly used are the sigmoid function (shown in Figure 2.9), the Rectified Linear Unit (ReLU) activation function (shown in Figure 2.10) and the tanh function.

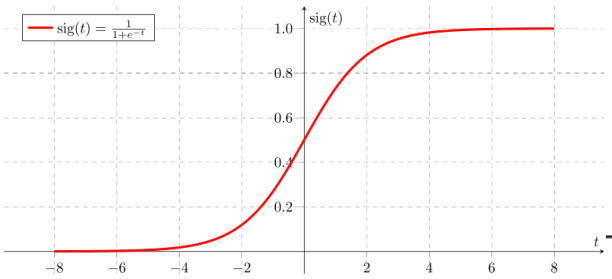

Figure 2.9: Sigmoid function [60]

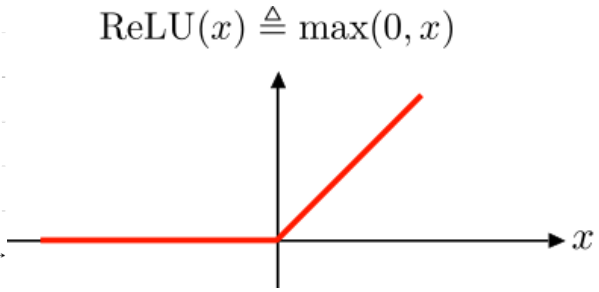

Figure 2.10: ReLU function [61]

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural network that are commonly used to work with and predict sequential data or time series data [62]. They are employed because they have a sort of "memory", taking information from prior inputs to influence the current input and output. This means that unlike in other neural networks, data does not flow linearly through them.
To train Recurrent Neural Networks (RNNs) a variant of the normal backpropagation algorithm called backpropagation through time [62] is used in order to determine the gradients. Backpropagation through time differs from the original algorithm in that it sums errors at each timestep, which is not needed in non-recurrent neural networks.

Because of this RNNs run into two problems [62]: Exploding and vanishing gradients. Both issues arise due to the size of the gradient. When the gradient is too small, it continues to become smaller, updating the weight parameters until they become so close to zero that they can not be worked with.
Exploding gradients on the other hand are the exact opposite of this. For this problem the gradient becomes too large, creating an unstable model. The weight parameters grow so large that they eventually "explode" and become virtually infinite, again breaking the model.

Long Short-Term Memory

One solution for the vanishing gradient problem are Long Short-Term Memory (LSTM) networks. LSTM networks are a special kind of RNN capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber in [63].
A LSTM contains cells which can store data outside the normal flow of an RNN. This happens via three gates: An input gate, an output gate and a forget gate. The inner workings of a LSTM cell can bee seen in Figure 2.11.

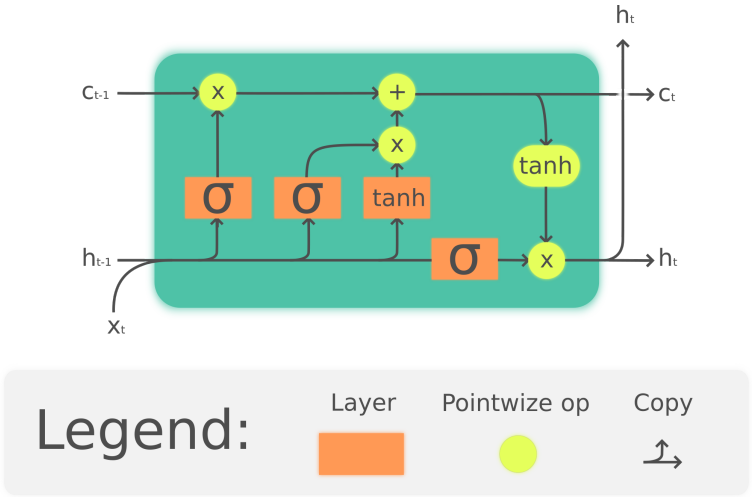

Figure 2.11: The LSTM cell [64]

The following is a short explanation on how LSTM cells function [65]. The main addition here is the cell state, represented by the the horizontal line running through the top of the figure.
The first step in the cell is to decide what information is being left out from the cell state. This decision is made by the leftmost layer and is called the forget gate. It looks at the output of the network connections to the cell h_{t-1} and at the input x_t and decides on how much to keep in the cell state via a sigmoid function.
In the next step it is decided upon how much new information is going to be stored in the cell state. This consists of two parts. First, a layer called the input gate decides which values are going to be updated using a sigmoid function. Next, a layer using a tanh function creates new candidate values that could be added to the state. Then the output of the two functions get multiplied to update the cell state.
In the final step the output of the cell is calculated. This output is based on the cell state, but will be filtered. First, an output gate using a sigmoid function decides which part of the cell are going to be the output. Then, using a tanh activation function, the cell state decides how active each output is. This is then multiplied with the output of the output gate.
The addition of the cell state enables LSTM networks to easily classify, process and make predictions based on time series data.

3 Related Work

This chapter provides an overview of the original MOTA algorithm and its variations, A-MOTA and M-MOTA.
It also provides an overview of a paper using Markov Chains to successfully predict where cars will turn at intersections.

3.1 Multi-Objective Task Allocation (MOTA)

The Multi-Objective Task Allocation algorithm proposed in [12] is a task allocation algorithm based on the genetic algorithm NSGA-II [66].

Genetic algorithms are metaheuristics inspired by the process of natural selection. They encode different solutions to an optimization problem into strings of binary numbers, real numbers or similar representations and treat them as individuals in a population. Each individual is assigned a **fitness**, which is a function approximating how good the solution solves the given problem. algorithm 1 is an example of how genetic algorithms work in general, based on [67].

Algorithm 1: `Genetic Algorithm Example`

Input: $maxGen, popSize, ne, nc, mc$;
$currentGen = 0$;
$pop =$ `initializePop(`$popSize$`)`;
while $currentGen < maxGen$ **do**
    $newPop =$ `selectBestSolutions(`ne`)`;
    for $i = 0$ ***to*** nc **do**
        $X_a, X_b =$ `randomlySelectTwoPops()` $X_{new} =$ `crossOver(`X_a, X_b`)`;
        $X_{new} =$ `mutate(`X_{new}, mc`)`;
        $newPop \mathrel{+}= X_{new}$;
    $pop = newPop$;
    $currentGen \mathrel{+}= 1$;
return `bestIndividual(`pop`)`;

`initializePop` randomly initializes a number of individuals given by the $popSize$, which then form the initial population pop.
After that, the steps inside the loop are repeated until the termination condition is fulfilled. In this case, the algorithm runs for a number of **generations** $maxGen$. Alternatively the algorithm could run for a fixed time or until another condition is fulfilled.
First, the ne best solutions of the population are preserved for the next generation in $newPop$ using the $selectBestSolutions$. The best solutions are always those with the highest fitness.
Then, nc new individuals are created. Each new individual has two "parents" randomly chosen by `randomlySelectTwoPops`. Those two individuals are then merged into the new individual

using a `crossOver` operation, of which there exist many widely used ones [68].
After that, the newly created individual gets mutated with a certain probability mc in the `mutate` function. Mutation operations transform a specific individual randomly, without using a second individual [69]. Finally, the new individual is added to the generation.
After all new individuals have been added to the new generation, the population pop gets updated to be the new generation $newPop$ and the process begins anew.
After the termination condition is fulfilled, the individual with the highest fitness gets selected as the winner in `bestIndividual` and gets returned as the output of the algorithm.

In the MOTA algorithm, each individual in the population represents an allocation $\mathcal{A}$. Within each allocation, every task gets assigned a node of the WSN.
An encoded solution is a list of Integers representing which entry in it is assigned to which task, as can bee seen in Equation 3.1.

$$I = (x_0, x_1, \ldots, x_n | x_i \in [0, |V_{nodes}| - 1]) \tag{3.1}$$

Each entry $I_i = N_j$ indicates the node N_j to which the task T_i will be assigned. The crossover operator is a variation of a simulated binary crossover, to which a final calculation step was added in order to adapt it to an integer-based solution encoding. The final calculation step rounds to the nearest integer.
Mutation is performed by randomly reassigning genes with a certain probability on each gene of the individual. This means that with a certain probability an existing assignment of a task is replaced with a random new assignment. In the original work, a crossover rate of $p_c - 0.9$, a mutation rate of $p_m = 0.5$ along with crossover and mutation distribution indices of $\eta_c = \eta_m = 20$ were chosen.
The fitness of each individual is evaluated by performing a simulation with the corresponding task allocation in a modified variant of the NS-3 network simulator, which provides the metrics required for calculation.
The goal of the original MOTA algorithm is to optimize the metrics Latency (L) and Network Lifetime (NL), as can be seen in more detail in Equation 3.2.

$$\underset{\mathcal{A}}{minimize} \quad (-NL(\mathcal{A}), L(\mathcal{A})) \tag{3.2}$$

Each allocation experiences packet loss due to packet collision and interference. This might affect the results, as fewer received packages result in less energy costs due to the removed retransmission and actuation costs. In order to counteract that, a penalty term proportional to the percentage of missed packages was introduced.
The basic concept of the algorithm is to start the genetic algorithm, initialize the population, let each individual be evaluated by letting the NS-3 simulation run until a node in the network is depleted, then let the genetic algorithm run its course.
The winner is chosen after the genetic algorithm has reached the desired amount of generations. It is selected by applying cone-domination to all non-dominated solutions. In the original work, a population of size 100 was chosen and the genetic algorithm was run for 200 generations.
Afterwards, a stopping condition checked: All nodes whose energy is depleted are removed from the network and the resulting network is checked for connectivity.
If all nodes in the network can still be reached from all other nodes, the algorithm is restarted

on this new network. If the network is no longer fully connected, it is considered dead and the algorithm stops and outputs the quality metrics obtained.

The original work compares its results against the DTAS algorithm [70], which is a single-objective approach and was chosen due to the lack of multi-objective state-of-the-art approaches. The results for a grid setup with distances assigned to generate 4-neighbourhood communication links can be seen in Figure 3.1 and Figure 3.2. Evaluated was a grid size of 9x9 and two kinds of task setups: A Single-Sink setup consisting of 10 non-constrained sensing tasks connected to a single actuating (sink) task, which is constrained to a 3x3 grid located in the centre of the Grid network and a Sink-Source setup for which the sensing tasks are constrained to the left half of the grid while the actuating tasks are constrained to the right half of the grid - both including the central column.

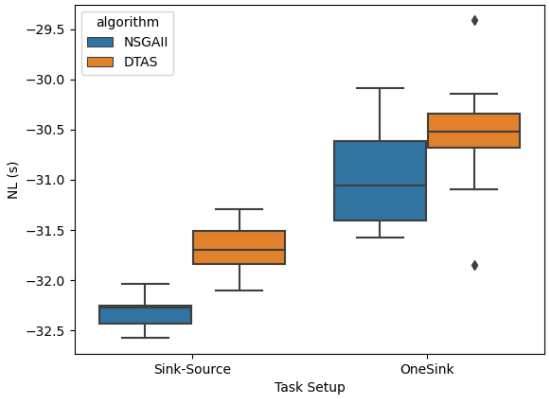

Figure 3.1: Network Lifetime for the Grid Network for the different task setups [12]

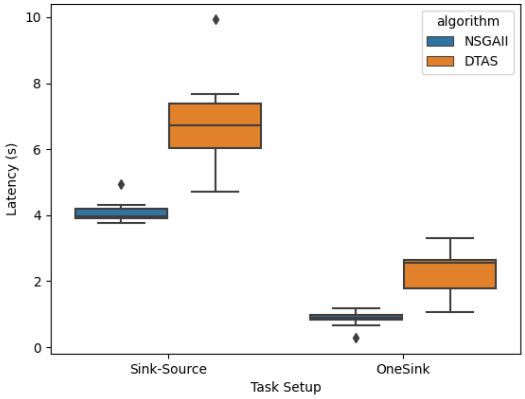

Figure 3.2: Latency for the Grid Network for the different task setups [12]

The MOTA algorithm slightly outperforms the DTAS algorithm for Network Lifetime and noticeably outperforms it for Latency, as can be seen in the figures.

3.1.1 Mobility-Aware Multi-Objective Task Allocation (M-MOTA)

M-MOTA is a variation of the original MOTA algorithm and was introduced in [13]. All things not mentioned here are therefore unchanged from the original algorithm.

The main change of the M-MOTA algorithm is that it incorporates a prediction mechanism to optimize for future node positions, instead of only optimizing for the current network state. This allows the algorithm to optimize ahead of time.

The M-MOTA algorithm is initialized by running the original MOTA algorithm for a number of generations in order to initialize an allocation and population. The algorithm then makes a prediction about the network state a certain amount of time Δt ahead.
For that, algorithm supplies the current network state to a predictor, which returns the predicted node positions. Using those positions, a network graph G_{pred} can be generated.
During the fitness evaluation of the individuals a part is then evaluated on G_{pred} instead of the current network graph G_{net}. The size of this set is proportional to the confidence of the prediction, with a minimum of 20% of the total population.

Another change of M-MOTA are the quality metrics used to evaluate the fitness of an individual. The quality metrics optimized for this algorithm can be seen in Equation 3.3.

$$\underset{\mathcal{A}}{minimize} \quad (-NL(\mathcal{A}), L(\mathcal{A}), -A(\mathcal{A}), -R(\mathcal{A})) \tag{3.3}$$

For the actual fitness evaluation, only NL and L are used directly, while A and R are estimated by the number of packets that did not reach their sink task.

For evaluation purposes in the original work, the network topology chosen was based on a Manhattan grid. A set of static nodes were placed along a grid pattern in the network area, with mobile nodes moving along roads between the static nodes. The static nodes were spaced in a way that results in a 4-neighborhood for direct communication. This setup can be seen in Figure 3.3, with the static nodes shown in orange, their communication links black and the mobile node "roads" in light grey.
The task setup evaluated was a Sink-Source setup again. This time, two different numbers of tasks, namely 19 and 55, were used for evaluation.
The mobility models used for the moving nodes were the ones which will be introduced in this work here. The nodes were able to move at a speed between 2 m/s and 7 m/s, with the roads being 100 m apart from one another.
The network always had 81 fixed nodes and either no mobile nodes, 30 mobile nodes or 50 mobile nodes for the evaluation.
As prediction methods a hypothetical perfect predictor, as well as a predictor specifically made for the mobility model given were used.

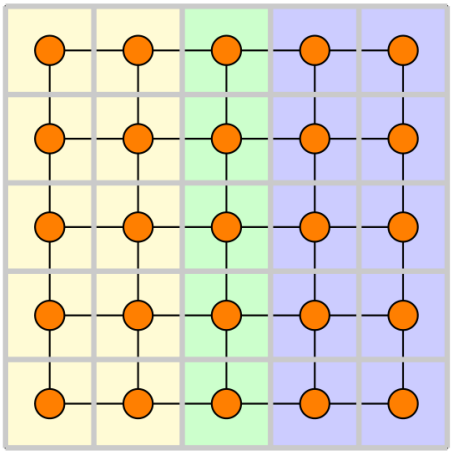

Figure 3.3: The Network Layout [13]

Figure 3.4 show the results obtained for the network lifetime, Figure 3.5 shows the latency and Figure 3.6 shows how many packages were missed.

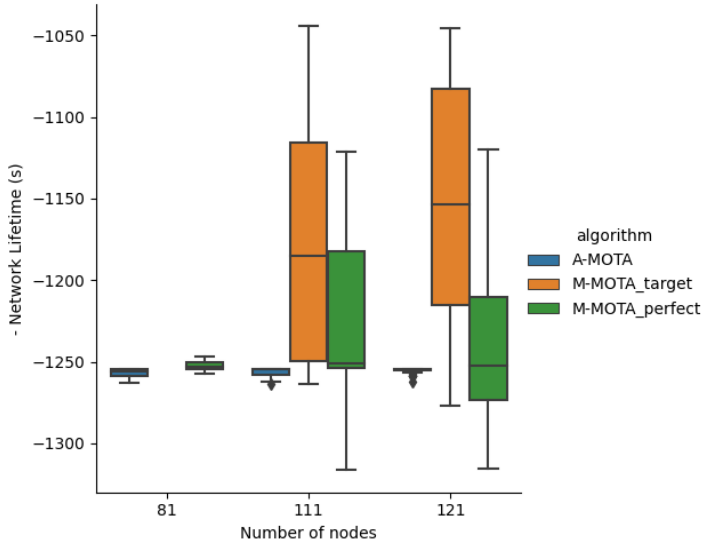

Figure 3.4: The Network Lifetime [13]

The network lifetime is actually inversely correlated with the amount of tasks being able to be executed, which explains why the A-MOTA algorithm has the highest network lifetime. For A-MOTA the lowest amount of tasks get executed, resulting in less power consumption.
This continues for the other two predictors, with the perfect predictor being able to help create an allocation that executes the most amount of tasks and therefore has the highest power consumption and lowest overall network lifetime.

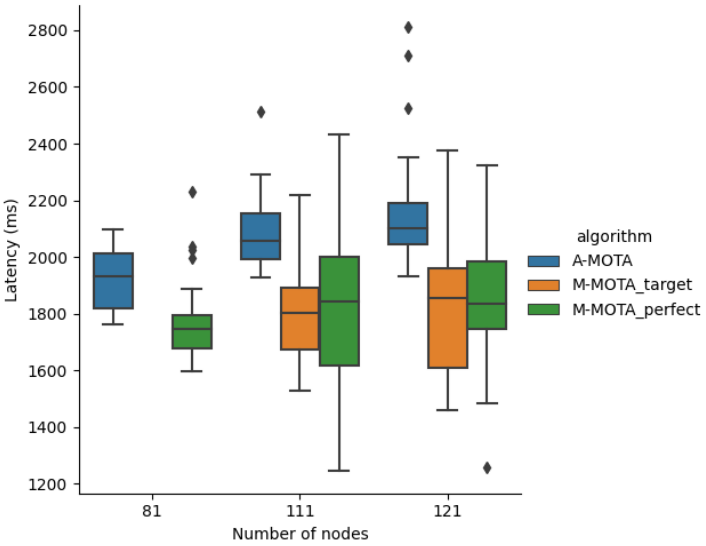

Figure 3.5: The Network Latency [13]

The latency shrinks the better the prediction is, as can be seen in the figure.

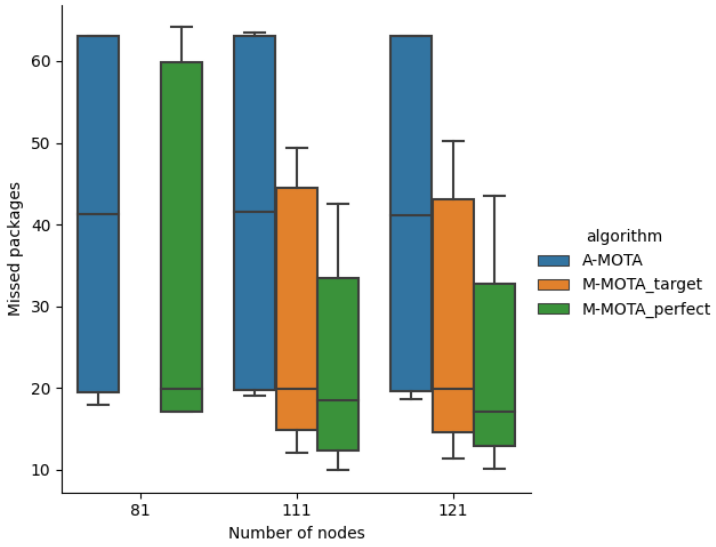

Figure 3.6: The Packets missed [13]

A better prediction also results in a lower amount of packets lost.

Overall, this work clearly shows that taking movement into account and predicting the future state of the network based on the movement results in higher quality metrics obtained for a WSN.

3.1.2 Availability-Aware Multi-Objective Task Allocation (A-MOTA)

Availability-Aware Multi-Objective Task Allocation [71] is another variation of the original MOTA algorithm.
One of the most important features introduced in algorithm has to do with the re-optimization of the task allocation after a node failure. In order to facilitate said task allocation, a diversity-based archive M is introduced here. Said archive retains solutions based on a new similarity metric, f_s, which is defined as follows.
Each solution a is assigned a similarity value determined by the number of unique nodes used in the solution's allocation that are already used by other solutions $I \in M$ in the archive, as shown in 3.4.

$$f_s(a) = \sum_{I \in M} |set(a) \cap set(I)| \tag{3.4}$$

The intended purpose is to only retain solutions with the smallest similarity values. Once the archive reaches a predetermined size, only new solutions with a smaller similarity value than the solution with the maximum similarity value in the archive are added to it, by replacing the least unique solution.
This way, the algorithm is able to keep a set of solutions using a more diverse set of nodes, which should provide additional options whenever a node in the currently optimal allocation fails.
The archive can then always be used to seed the population when re-optimizing. When a node failure is detected, a new solution can immediately be chosen from the archive. If the archive contains no feasible solutions, a valid allocation is constructed from the current allocation by exchanging faulty nodes with other nearby nodes.

An evaluation on how well this method performs in comparison to the M-MOTA algorithm can be seen in Figure 3.4, Figure 3.5 and Figure 3.6 further above.
It is clear that A-MOTA performs worse than M-MOTA. The algorithms are however not mutually exclusive and can be used in combination with each other.

3.2 A Markov Model for Driver Turn Prediction

In the paper A Markov Model for Driver Turn Prediction [72], an algorithm for making short-term route predictions for vehicles using their GPS data is described.
In the work, each trip made by a car is represented as a connected sequence of road segments with no adjacent repeats. Said sequence is used as the basis to create a Markov model.
The prediction of a vehicles near-term future route its based on it near term past route. The sequence of traversed road elements is modeled as $X(i)$ with i being a discrete time variable and $X(i)$ being a road segment represented by an Integer.
The sequence of traversed road segments for a vehicle on a trip is denoted by $\{\dots, X(-2), X(-1), X(0), X(1), X(2), \dots\}$. $X(0)$ represents the road segment at the current time, all Integers in the sequence before that represent past road elements and all elements in the sequence after $X(0)$ represent unknown future road elements that are to be predicted. $P[X(i+1)]$ denotes a discrete probability distribution over all the road segments for which road segment will be encountered after $X(i)$.

The work asserts that for each road segment, a histogram of which road segments were encountered immediately after can be build and normalized to get a discrete probability distribution. Using that, a first order Markov model can be built.
Continuing with the same logic, a second order Markov model can be built by creating histogram over all two-element, ordered sequences.
If one does want to predict further ahead into the future, for example the road element two time steps into the future one can use the distribution over the road segments after the next one, given the current one.
The work then generalized that a n^{th} order Markov model can be built to predict the m^{th} next encountered segment, which can be represented by Equation 3.5.

$$P_n[X(m)] = P[X(m)|X(-n+1), X(-n+2), \dots, X(0)] \tag{3.5}$$

It is noted here that the probabilistic prediction has a measure of its own uncertainty that can be usefully reported to in-vehicle applications.

The results obtained by the work were obtained by observing 100 drivers for a total of 12.21 days and using said data to train the model. This was done via leave-one-out testing, meaning that all but one trip was used to train the model and the remaining trip was used to test how well the model performs. This was then done once for each trip to obtain an average accuracy over all trips.

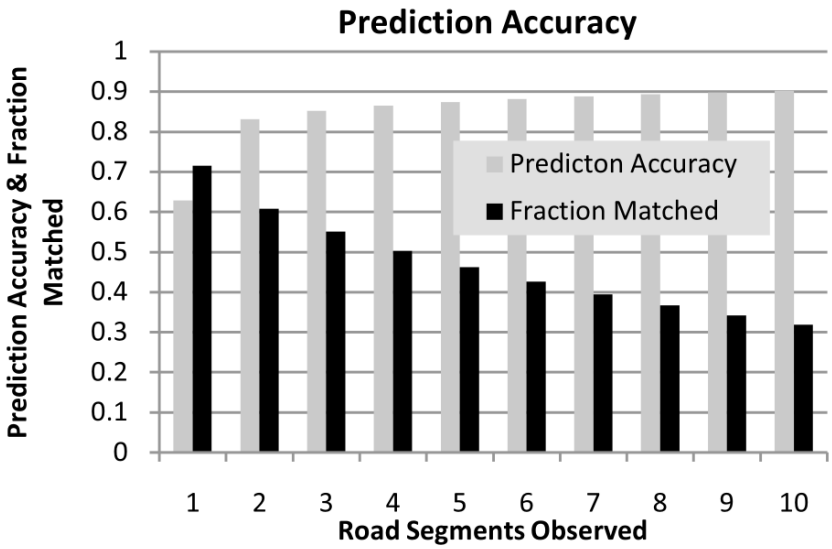

Figure 3.7: Accuracy for predicting the next segment. [72]

The results can be seen in Figure 3.7 and show that the prediction accuracy starts out around 60% when only observing one previous road segment and then rapidly increases when observing more, capping out at an accuracy of around 90%.

4 Mobility Models and Predictors for M-MOTA

4.1 Overview

Inspired by the traditional MOTA algorithm and the improvement of it when incorporating prediction of future topologies as shown in [13], we aim to find general prediction methods for cars in a smart city scenario.
In order to accomplish that goal, data on which the prediction methods are to be evaluated is needed. However, real traces of fleets of cars driving in a city for a significant amount of time in multiple instances are difficult and time consuming to acquire, which leaves synthetic models as the best option available to us.
The first part of this chapter, therefore, will elaborate why we chose the Manhattan mobility model as a baseline for our data generation and why and how we extend it in order to generate data that is suitable for our use-case.
After establishing the synthetic models, we then introduce the actual prediction methods in the second part of this chapter and elaborate why each of those methods was chosen as a potential candidate.

4.2 The Mobility Models

As laid out above, a mobility model that produces data on which the different prediction methods can be evaluated is needed. The classical Manhattan mobility model, as already described in further detail in Chapter 2 was chosen as a baseline for these predictors. It already provides topographic constraints which are approximating the topography of real cities, which immediately makes it a potential candidate as a synthetic model.
Cars in an actual smart city can be assumed to act independently of each other, each one with its own target in the city which is chosen by each individual. The route to a chosen target can be assumed to be calculated by an IoT-connected GPS system similar to [7]. This immediately rules out any homogeneous mobility model as a potential candidate, as cars clearly act as individuals and not in any kind of cooperating manner. Taking a closer look at the already established heterogeneous models, either a geographic constraint mobility model, a graph-based model or a Manhattan mobility model are the closest to a potential real world smart city from a topographic perspective.
Random walk, random waypoint and Gauss-Markov models can immediately be ruled out, as the high amount of randomness for node movement and the lack of topographical restrictions do not generate data that approximates our scenario in any meaningful way. The lack of topographical restrictions and the general fluidlike movement in fluid flow mobility models rule those out as well.
A geographic constraint mobility model incorporates obstacles of arbitrary shapes and pathways between said obstacles [47], making the Manhattan mobility model essentially a sub-type of geographic constraint model, since a geographic constraint model could be set up in a way that imitates a Manhattan mobility model. This poses the question which of the two is better suited for the following task.

While a general geographic constraint model can be set up to imitate any one specific city, what we are looking for is an approximation of a generic smart city. The Manhattan model has been deemed a good approximation for general high density urban areas before [73][74][75], which is why it was deemed to be a good enough choice as opposed to building a custom geographic constraint model.
Graph-based models were ruled out on the same basis, as they are also generalizations of Manhattan models, if one models the intersections as vertices and the streets as edges.
The Manhattan mobility model however produces insufficient data when it comes to evaluating different methods of predicting the future topology of a network with nodes in our scenario. In the classical model, nodes choose the direction they move in randomly at each intersection. While the 50% forward, 25% left and 25% right ratios when it comes to turning at an intersection give nodes a slight preference in the way they move, it is still not at all representative of the clear driving patterns our scenario would have. Driving patterns in smart cities should have a clearly defined target, as well as a predetermined path to said target calculated by a GPS connected to the Internet of Things [7]. This should lead to a very different movement behaviour than the one in a Manhattan mobility model.
Since any time series prediction relies on patterns in the data that makes it possible to predict future states in the first place, a variation of the classical Manhattan mobility model which incorporates patterns that approximate the real world is needed.
In order to achieve that, assumptions about the movement patterns in our scenario are laid out below.

- Cars moving in our scenario are driven from a starting point A to an endpoint B, both of which are known at the time the cars begins to move.
- In our scenario, pathfinding will always be done by an IoT powered GPS system [7], which will compute the route with the fastest time to a target location.
- The travel time to a target location is only influenced by the path, the traffic and the speed at which the driver chooses to travel.

Using these assumptions, there are multiple variations of the Manhattan model that can be created in order to increase the suitability of the data for our task.

4.2.1 Manhattan Target Model

For the first extension, we primarily consider the way a car would always move to a specific end point guided by a GPS system and combine that concept with the already existing Manhattan model. The speed with which nodes in a Manhattan model move is intrinsic, meaning different streets segments have no influence on it. Not considering any traffic, the fastest travel time therefore always is achieved by calculating the shortest path to the target location.
Using that knowledge, a model can be created which modifies node movement to include a target location.

Every node in this model moves according to the algorithm shown in algorithm 2.

Algorithm 2: `Manhattan Mobility Target Model`

```
Input: pos, spd, Δt;
while simulationRunning() do
    target = generateNewTargetLocation();
    dir = directionLeadingToShortestPath(pos, target);
    while targetNotReached(pos, target) do
        if atIntersection(pos) then
            dir = directionLeadingToShortestPath(pos, target);
            pos = crossIntersection(pos, spd, dir, Δt)
        else
            pos = pos + dir · spd · Δt;
```

The three input variables to the algorithm are the initial position *pos* at an intersection of a node, the initial speed *spd* of the node and the time Δt that elapses with each step of the simulation. Δt is always assumed to be a small enough simulation step so that precision errors errors do not affect the algorithm.
As long as the simulation is running, the `simulationRunning` function will return true and the nodes move by the instructions within the loop.
First, a new target gets created for each node by the function `generateNewTargetLocation`. This function chooses an intersection within the bounds of the simulation and returns an Integer representing the chosen intersection.
Using the generated intersection, the `directionLeadingToShortestPath` function gets invoked and searches for the shortest path from the current position of the node to the intersection. Then the direction in which the node has to move can easily be calculated and gets returned as the output of the function.
The algorithm then goes into a loop with the `targetNotReached` function as its exit condition, which checks if the current position of the node is within the bounds of the target intersection. If that is the case, the loop ends and the algorithm continues from there.
If the node is not yet at its target intersection, it first gets checked if the current position of the node is within the bounds of *any* intersection.
If that is not the case, the `atIntersection` function returns false and the position gets updated using the equation seen in algorithm 2.
Alternatively, if the node is within the bounds of an intersection, its direction gets updated using `directionLeadingToShortestPath` again. After that the node chooses the exit at which to leave the intersection using the new direction calculated. This is done in `crossIntersection`, which then also advances the position of the node towards the chosen exit of the intersection and returns its new position.

One notable thing about the calculation of the shortest path is that if the two points between which the shortest path calculation on the Manhattan grid is done are not connected by a straight line multiple shortest paths exist, making multiple directions valid.
It can be chosen if this should always result in a specific path being taken, or if the path taken should be randomized.
Another thing that can be varied is by what distribution the targets for each node should be generated, or if they should not be random at all. By default the targets are generated with a uniform distribution, but many more generation methods are possible here.

Overall, this extension of the Manhattan mobility model should improve the similarity between synthetic and real data. It also gives the generated data a clear pattern, making prediction of future data more meaningful than in the original model.

4.2.2 Manhattan Target Highway Model

Based on the newly created model above, further modifications can be made. The general assumption of cars moving to a target location will not change in this variation, but another part of the assumptions made above will be incorporated.
In the Manhattan mobility model, the speed of a car is completely detached from the road segment it is on. This is a simplification of reality, where each section of a road has speed limits and physical circumstances which influence the speed at which a car is driving.
Using that knowledge, a model can be created which uses the Manhattan Mobility Target Model as a baseline and incorporates some kind of influence of road segments on the speed with which a node moves.
In our case the outer ring of the Manhattan grid was chosen to be a highway. This means that the outermost road segments of the grid allow for an increased movement speed of the nodes. This an abstraction of the way certain highways can connect to cities in the real world, where they are sometimes built at the edges of a city and connect to the lower speed inner city roads via ramps.
The fundamental algorithm used to calculate the movement of each node does not change significantly from the Manhattan Mobility Target Model, as can be seen in algorithm 3.

Algorithm 3: `Manhattan Mobility Target Model with Highways`

Input: $pos, spd, \Delta t$;
while *simulationRunning()* **do**
    $target =$ `generateNewTargetLocation()`;
    $dir =$ `directionLeadingToPathWithShortestTravelTime`$(pos, target)$;
    while *targetNotReached(pos, target)* **do**
        if *atIntersection(pos)* **then**
            $dir =$ `directionLeadingToPathWithShortestTravelTime`$(pos, target)$;
            $pos =$ `crossIntersection`$(pos, spd, dir, \Delta t)$
        else
            $pos = pos + dir \cdot spd \cdot \Delta t$;

Instead of simply calculating the shortest path towards the target location of each node, we now calculate which path would have the shortest travel time in the `directionLeadingToPathWithShortestTravelTime` function. To represent which roads have a higher or lower travel speed each road segment can simply be given a weight.
The increased highway travel speed in our case is implemented as a multiplier of the base travel speed which is generated for each node. This means that nodes use a multiple of their originally generated travel speed when travelling on a highway, giving every node a distinct travel speed even when using the highways.
A customization of this model might be the way in which the speed of each node gets affected. Possibilities could for example include a set travel speed for all nodes while on highways or differing multipliers for individual nodes.
Other variations may include more or even every road segment in the grid having different movement speed influences on the nodes, instead of only the outermost layer.

This extension of the Manhattan Mobility Target Model should increase the similarity between real world and synthetic data again by introducing roads which affect the speed of cars.
Additionally, it can be used to compare how different generation methods influence the accuracy of the prediction methods used in the next step.

4.3 Future Topology Predictors for Wireless Sensor Networks

As explained in Chapter 1, the ultimate goal of the prediction methods below is to be used with the M-MOTA algorithm by creating algorithms which can predict the future topology of a WSN in our smart city scenario. This means that in theory only the future topology and not the specific positions of the nodes need to be predicted.
In reality, predicting topology without predicting the specific position of each node in the network is quite difficult because the topology emerges from the positions of all nodes in relation to each other. Therefore the goal of the prediction methods will be to predict the position of all nodes in the network, which then allows for us to make a prediction about the topology.
Predicting future positions of moving nodes can and has been done in different ways using the predictions methods in Chapter 2, one such example being the work shown in Chapter 3, where the movement of cars was predicted using Markov models.
Specifically for our scenario, predicting the future positions of nodes becomes more doable than for other use cases of Wireless Sensor Networks. Firstly, cars, unlike drones or other types of nodes which might be part of a WSN, are bound to the ground. Therefore, with the notable exception of tunnels or bridges, they can not move over or underneath each other.
This means that the vertical positions of the nodes can mostly be disregarded for our scenario, as it can not make a significant difference in the topology of the network.
Furthermore, the nodes in our network are bound by topographic constraints our scenario gives us in the form of streets, making them move in a straight line whenever they are not at an intersection. This means that whenever a node does not come across an intersection its future position can be predicted easily as long as its current velocity is known and no acceleration or deceleration occurs.

4.3.1 Node Predictor

Using the constraints mentioned and also the assumptions made about the general movement of cars in a smart city, an initial idea for a prediction method can be formed.
When not at an intersection, a node should move at a constant speed in one direction. When at an intersection, a node should change to or stay on a path with the shortest travel time towards its target location.
Considering this, the assumption can be made that the future decision of a node at an intersections is most likely similar to its past decisions. Given a node N which has taken the decisions to cross an intersection straight F_N number of times, has turned left L_N number of times and has turned right R_N number of times recently, a naive way to predict its future decision would be to assume that it will do what it has done in the past the most amount of times.

Given the current position p_N, current direction d_N and current velocity v_N and us wanting to predict the future position of the node a certain time span t ahead, this naive way of prediction can be modeled by the following algorithm shown in algorithm 4.

Algorithm 4: `Naive prediction method`

```
Input: p_N, d_N, v_N, L_N, R_N, F_N, t;
while t ≫ 0 do
    if notAtIntersection(p_N) then
        d = distanceToIntersection(p_N, d_N);
        Δt = d/v_N;
        t = t − Δt;
        /* Early exit if t gets negative                    */
        p_N = p_N + d_N · v_N;
    else
        d_N = mostLikelyDirection(L_N, R_N, F_N);
        t = t − intersectionCrossingTime(v_N);
        /* Early exit if t gets negative                    */
        p_N = positionAfterCrossingIntersection(p_N, d_N);
return p_N;
```

In the algorithm, we enter a loop while our look-ahead time t has not reached zero.
First, we check if the node is within the bounds of an intersection or not.
If it is not, we calculate the distance to the next intersection within the direct path of the node using `distanceToIntersection`. Using that distance and the velocity at which the node is moving we can then determine the time it needs to reach said intersection. We then subtract that time from the original look-ahead time and advance the node position.
If the node is at an intersection when performing the check, we invoke the `mostLikelyDirection` function. For this algorithm, the function only compares the number of times our node crossed an intersection straight, has turned left and has turned right. It looks at which of the three things the node has done most often and picks that to be the choice of exit the node will make at the current intersection. This choice is then used to calculate the direction in which the node will move in and the node position is advanced towards the chosen exit of the intersection using the `positionAfterCrossingIntersection` function. We then calculate the time the node has needed to cross the intersection based on its velocity in `intersectionCrossingTime` and subtract that from the original look ahead time.
This is done until t reaches zero, at which point the prediction has reach the desired look-ahead point in time and returns the predicted position.
Both cases within the loop have an early exit condition if t is calculated to be negative after moving, as indicated by the two comments. Should that happen the fraction of the distance which is needed to get t to exactly zero is calculated and the position is then updated accordingly, after which the loop ends.

This algorithm gives an innate confidence for its predictions. At each intersection, the confidence that the chosen exit is correct is equal to the number of times the chosen exit was taken divided by the total number of times the node has taken any exit.
Therefore, the total confidence for a prediction made by this algorithm is the product of confidences for predictions made at all intersections by the current node.

The most important thing to choose in this algorithm is how many steps into the past we look. Only the last decision of the node could be considered, all decisions a node has made in the past could be considered, or any amount of decisions in between could be considered.
Regardless of how many steps are considered, it becomes obvious very quickly that this prediction method will get the direction a node chooses at an intersection wrong in many cases. It will always predict the exit a node will choose wrongly when its current decision differs from the average of the decisions made in the past.
This prediction method, while not very sophisticated, can serve as a baseline and comparison for more advanced ones.

4.3.2 Intersection Predictor

Trying to find a less naive approach than the one considered above, one could think of not simply using the memory of a node to predict where it will turn at a future intersection. Instead each intersection could be given a memory of how many nodes have chosen which exit when crossing it.
This can be justified by going back to our scenario. One can assume that in the given scenario, the targets where cars drive to are not distributed equally, but that certain hot spots will be present in any given city [75]. Also, certain path through a city are faster than others and will be recommended more by GPS routing systems [7].
This could lead to intersections having preferred exits, which in turn can be used to predict where future cars are more likely to exit.
Given an intersection I with a memory of how many nodes have crossed straight F_I number of times, have turned left L_I number of times and have turned right R_I number of times, and the same node N and all its properties as defined in the last prediction method, the algorithm algorithm 5 can be employed.

Algorithm 5: `Prediction method using intersection memory`

Input: $p_N, d_N, v_N, L_I, R_I, F_I, t$;
while $t \gg 0$ **do**
  if *notAtIntersection*(p_N) **then**
    $d =$ `distanceToIntersection`(p_N, d_N);
    $\Delta t = d / v_N$;
    $t = t - \Delta t$;
    `/* Early exit if t gets negative */`
    $p_N = p_N + d_N \cdot v_N$;
  else
    $d_N =$ `mostLikelyDirection`(L_I, R_I, F_I);
    $t = t -$ `intersectionCrossingTime`(v_N);
    `/* Early exit if t gets negative */`
    $p_N =$ `positionAfterIntersection`(p_N, d_N);
return p_N;

The only change in this algorithm is the `mostLikelyDirection` function with which the exit and therefore direction chosen at an intersection will be predicted, as explained above. While a node is not at an intersection, the prediction of this method will therefore be exactly the same as the naive prediction method and will only differ after at least one intersection has been crossed by a node.

In this algorithm, the `mostLikelyDirection` function compares the number of times all nodes at the specific intersection the current node is at have crossed straight, have turned left and have turned right. It then picks the choice which was made most often in the past of the intersection and picks that to be the choice of exit the current node will take as well. Based on that, the new direction for the node can then be calculated.
This algorithm, just like the last, gives a confidence for its predictions. At each intersection, the confidence that the chosen exit is correct is equal to to the number of times the chosen exit was taken divided by the total number of times the intersection was visited.
Therefore, the total confidence for a prediction made by this algorithm again is the product of confidences for predictions made at all intersections by the current node.
While this prediction method may seem more advanced in comparison to the naive prediction method explained before, it still has flaws introduced by the assumptions made.
First, the assumption made about targets in a city being in certain hot spots and not being distributed equally is not verified to hold true in general for our scenario.
Second, intersections having preferred exits is reasonable when using static GPS routing, but for our smart city scenario this might not hold true. With IoT assisted GPS systems like in [7], traffic will most likely be spread out more evenly in order to reduce travel time for all cars and to reduce the burden on individual street segments. This in turn means that intersection exits will be more evenly spread as well, reducing the effectiveness of this approach.
While this method will probably be more effective than the naive method in general, it still uses too many assumptions made naively, which is why in the next steps models with different approaches will be used.

4.3.3 Markov Predictor

Inspired by the the general concept of Markov models and their successful usage in "A Markov Model for Driver Turn Prediction" [72] as described in chapter Chapter 3, a prediction method employing Markov chains can be thought of.
Modeling the sequence of traversed road segments of a node as $X(t)$, with t being a discrete time variable and $X(t)$ being a road segment represented by an Integer, we can build an n^{th} order Markov model to predict the next road segment a node will chose to turn to, as was done in the work that inspired this algorithm.

$$P_n(X(1)) = P(X(1)|X(-n+1), X(-n+2), ..., X(0)) \tag{4.1}$$

Using the probability for the next road segment as shown in Equation 4.1, we can create the Markov-based predictor by saving as many previously encountered road segments $X(-n+1), X(-n+2), ..., X(0)$ as needed for the n^{th} order Markov model.

Algorithm 6: `Markov-based predictor`

Input: $p_N, d_N, v_N, t, X(-n+1), X(-n+2), ..., X(0)$;
while $t \gg 0$ **do**
  if *notAtIntersection*(p_N) **then**
    $d =$ `distanceToIntersection`(p_N, d_N);
    $\Delta t = d / v_N$;
    $t = t - \Delta t$;
    `/* Early exit if t gets negative */`
    $p_N = p_N + d_N \cdot v_N$;
  else
    $segment =$ `getRoadSegmentWithMaxProbability`$(X(-n+1), X(-n+2), ..., X(0))$;
    $d_N =$ `directionTowardsSegment`$(segment, p_N)$;
    $t = t -$ `intersectionCrossingTime`(v_N);
    `/* Early exit if t gets negative */`
    $p_N =$ `positionAfterIntersection`(p_N, d_N);
    $X(-n+1), X(-n+2), ..., X(0) =$ `shiftSegmentsAndAddNewOne`$(segment)$;
return p_N;

The algorithm shown in algorithm 6 shows that again, there are only slight differences to the other two predictors. In this predictor, when a node is at an intersection, the n previously encountered road segments get fed into the n^{th}-order Markov model in the `getRoadSegmentWithMaxProbability` function. Said function then gets the transition matrix which gives us the probabilities for the node to transition to each state (i.e. to end up on on each road segment).
We can then pick the state/road segment with the highest probability and look up which exit and by extension direction lead to said road segment in `directionTowardsSegment`. As in [72], we also know that all road segments that have a probability to be reached in the transition matrix can physically be reached by our node.
After that, we add the segment of the street that the predictions leads us to as the latest element in the sequence of segments and remove the oldest segment in order to have the sequence ready for the next prediction that has to be made in the `shiftSegmentsAndAddNewOne` function.
The Markov model itself is trained in nearly the same way as it is in [72]. A notable difference is that our scenario does not provide us with a finite amount of data collected beforehand, instead we can use data collected by running our simulation using the created mobility models.

For this algorithm, the transition matrix directly gives us the confidence for a prediction at each intersection. The total confidence is the product of confidences for predictions made at all intersections for a node again.

A notable choice to be taken for this prediction method is what order of Markov model should be used with a higher order model being expected to help find more accurate predictions [72]. Based on the accurate predictions achieved in [72], it is expected that this method to predict future positions and by extension the future topology of the network performs well.

4.3.4 Neural Network Predictor

A different method for time series prediction is neural networks, which have been used to predict different kinds of driver actions before [76][77]. In the following, we will employ a Long Short-Term Memory network, which has been successfully used for time series prediction in particular, as described previously in Chapter 2.
One method to model our prediction problem using neural networks, would be to again use a time series of road segments which can then be fed into a neural network instead of a Markov model. We, however, decided to utilize the full extent of the capabilities of neural networks, which is why we instead use a different approach. Instead of using road segments as our sequence, for this approach we use the position $\vec{x}(t)$ of each node, with $\vec{x}(t)$ being a positional vector of our node at time t.
While this will make the model more difficult to train, it should simultaneously allow it to generalize more. Modelling the problem with this approach, the algorithm looks like algorithm 7.

Algorithm 7: `Prediction method an` LSTM

Input: $t, \vec{x}(-n+1), \vec{x}(-n+2), ..., \vec{x}(0)$;
$p =$ `lstmPrediction`$(t, \vec{x}(-n+1), \vec{x}(-n+2), ..., \vec{x}(0))$;
return p;

Instead of making any assumptions about how the nodes move in general, we feed the network a sequence of positions of a node and let it figure out where the node should be next, making the `lstmPrediction` give us a position directly.
Thus the algorithm also differs significantly from the other methods in the sense that we do not actually know what the algorithm is doing, we only know the sequence we give it and the position it gives us. This makes the way the network was modeled and trained the most relevant thing to know in order to understand the output of the network.
The model chosen was a simple LSTM model. It contains one input layer taking in the data sequence, one LSTM-layer containing 128 units and one output layers returning two real values representing the positional vector of a node.

The disadvantage of this approach is that any given prediction does not return a confidence score, which is why the accuracy for any prediction made by the neural network was chosen to be the accuracy of the network when testing.

Overall, this approach is expected to provide good prediction results, as LSTM networks have been found to produce meaningful results when predicting time series data.

5 Evaluation

In this chapter we will evaluate the implemented algorithms using different methods. First, every predictor will be compared in terms of its prediction accuracy for node positions. After that, the accuracy of prediction for the future topology will be compared, followed by the final comparison which is done by using the M-MOTA algorithm with every predictor.

5.1 Test Setup and Parameter Overview

To generate test data for the predictors to use, the original Manhattan mobility model as well as the two new ones are used to acquire an overview of the capacity of each prediction method. All following simulations for the Wireless Sensor Network were done using a modified version of NS3 network simulator v.3.34 [78].

5.1.1 Setup for the positional and topological Comparison

Table 5.1: Parameters for the models for the position and topology evaluation

Category	Parameter	Value
Mobility	Minimum Speed (km/h)	30.0
	Maximum Speed (km/h)	50.0
	Road Distance (m)	100
	Grid Size	10x10
	Number of Mobile Nodes	100
	Number of Static Nodes	100
Predictors	Look-Ahead Time	{10,20,30,40,50,60}
	Discrete Time Points predicted (per Simulation)	50
	Simulation Runtime (s)	3000
	# Simulations run (per Combination)	31

The parameters used for the mobility models and predictors to compare the general positional and topological prediction accuracy can be seen in Table 5.1. The speed of the nodes was generated using a uniform distribution.
The simulation area for all experiments was chosen to have a hard limit, so the Manhattan models all have an outermost ring of streets at the models bounds.
The starting and target locations were generated using a uniform distribution with every intersection in the Manhattan Grid being considered.
The speed of a node when driving on a highway was chosen to be three times its base speed, giving nodes on a highway velocities from 90 km/h to 150 km/h.

In addition to mobile nodes, a set of static nodes are placed along a grid pattern in the network area, as was done in the original evaluation of M-MOTA. For a visualization of the placement of the static nodes see Figure 3.3. Data for every combination of previously introduced predictors and mobility models, as well as the original Manhattan model was acquired for comparison purposes.

5.1.2 Neural Network Training and Parameters

Table 5.2: Parameters for the Neural Network Training

Category	**Parameter**	**Value**
Neural Network	Optimizer	Adam
	Learning Rate	0.001
	Loss function	MSE
	Batch Size	32
	Epochs	5000
	Sequence Length	20
Simulation	Runtime (s)	10000
	# Simulations run (per Model)	10

The data required to train the model was acquired by simulating a WSN using all three variants of the Manhattan mobility model. All of them were configured as described in Table 5.1.
The parameters used for training the neural network and the number of data points acquired can be seen in Table 5.2.
In order to create usable data for the neural network, the position of every node was saved after each second of the simulation. The data was then processed to create sequences of positions for all nodes. The timesteps within the sequences correspond to the possible look-ahead times in Table 5.1.
The network was trained to minimize the difference between the output of the network (i.e. the predicted position of a node) and the actual position of the node.

5.1.3 Setup for the Comparison using M-MOTA

Table 5.3: Parameter values used for the M-MOTA experiments

Category	Parameter	Value
Task Parameters	Processing Time (s)	0.5
	Send Task Interval (s)	10
	# of Tasks	{19,55}
MOTA	Crossover rate p_c	0.9
	Mutation rate p_m	0.5
	Population size	100
Mobility	Minimum Speed (km/h)	30.0
	Maximum Speed (km/h)	50.0
	Road Distance (m)	100
	Grid Size	10x10
	Number of Mobile Nodes	100
	Number of Static Nodes	100

To evaluate the performance on the M-MOTA algorithm, each test case was run eleven times. The general setup for the mobility models is the same as for the other experiments above.
A task setup with two different numbers of tasks and the goal of minimizing Power Consumption, Latency and the amount of missed packets was evaluated. The amount of missed packets are the number of packets sent by any source task but not received by the respective sink tasks. They are used as an indicator for Availability and Reliability, as both of these metrics are heavily influenced by packet loss.
We compare the results of all prediction methods with each other, as well as to a hypothetical perfect predictor which can predict the future network topology perfectly. This is achieved by letting the simulation continue for the length of the look-ahead time and then reading all node positions.
The task setup chosen is similar to one of the setups used to evaluate the original MOTA algorithm (see Chapter 3). It is a Single-Sink setup consisting of 10 non-constrained sensing tasks connected to a single actuating (sink) task, which is constrained to a 4x4 grid located in the centre of the Grid network. The simulation time when evaluating individual solutions is restricted to 100 seconds in order to decrease the computing time for fitness evaluations and the experiments as a whole.

5.2 Performance of the Predictors

5.2.1 Positional Accuracy

Predictability Differences of the three Mobility Models

First we look at the impact the different mobility model variants have on the ability to predict where a node will be. For that we will look at the prediction results 20 seconds, 40 seconds and a full minute ahead into the future.

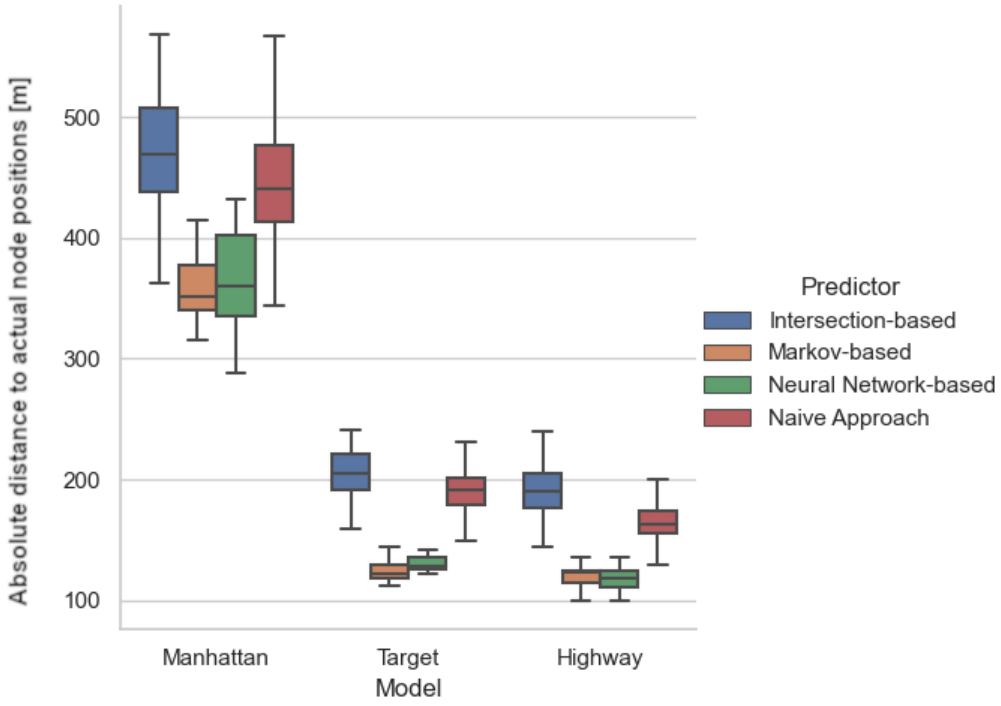

Figure 5.1: Predictability of the three mobility models 20 seconds ahead

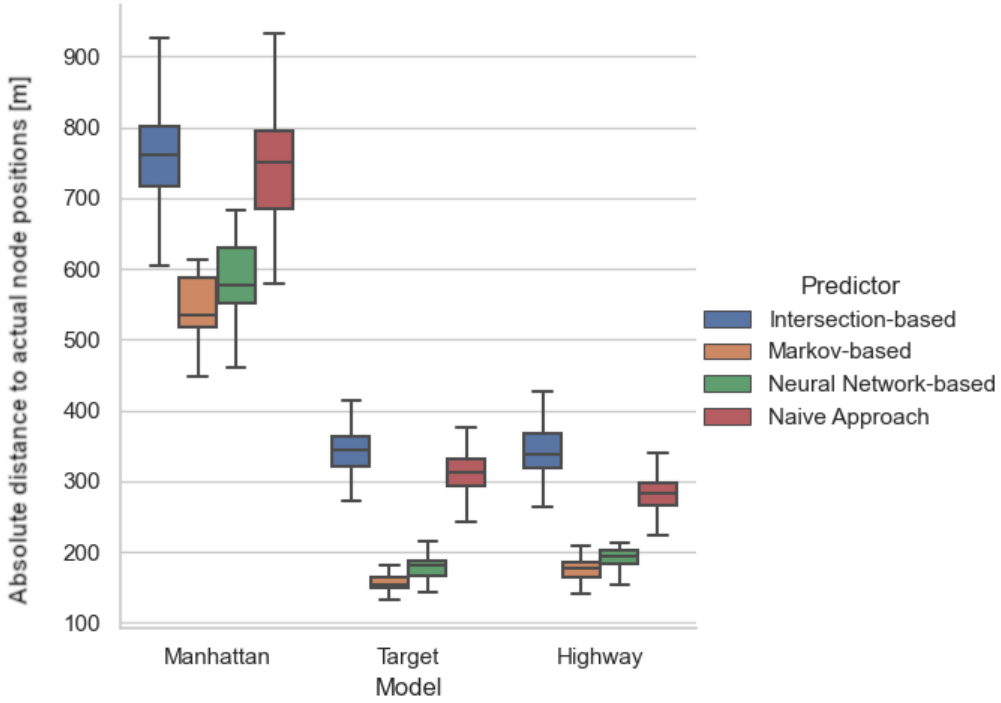

Figure 5.2: Predictability of the three mobility models 40 seconds ahead

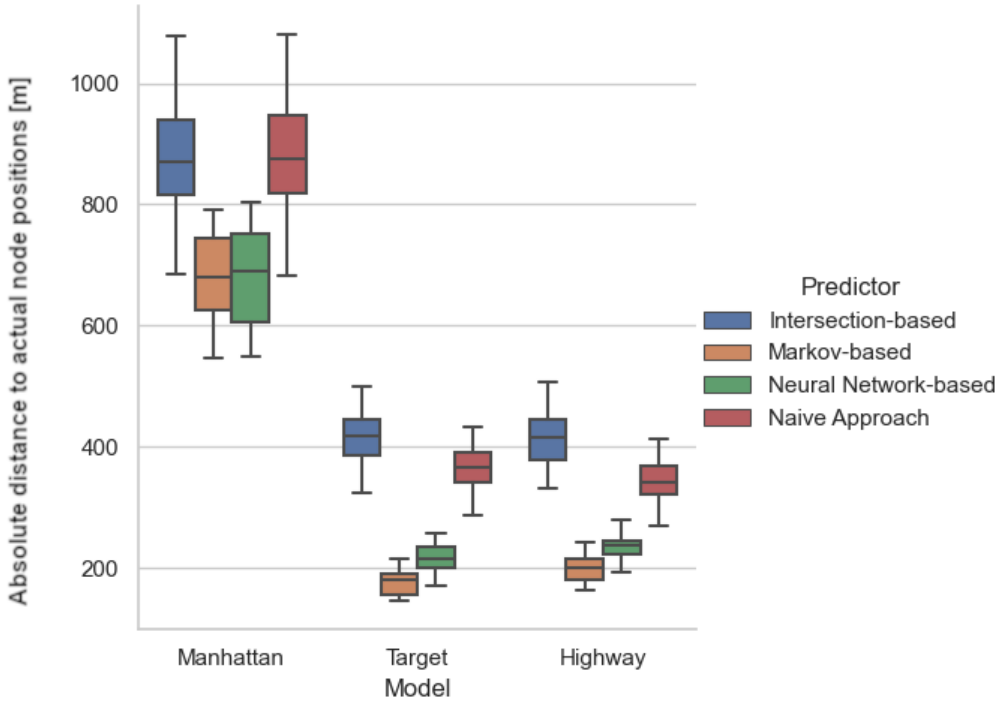

Figure 5.3: Predictability of the three mobility models 60 seconds ahead

Figure 5.1, Figure 5.2 and Figure 5.3 show the experimental results for the predictability of all models used. It becomes immediately apparent that the classical Manhattan model with its lack of patterns in the node movement can not really be predicted by any method, with every predictor being nearly equally unable to give any meaningful results. This clearly shows why other mobility models were needed in order to evaluate the prediction methods.
The two newly created models on the other hand both show clear signs of being predictable even 60 seconds ahead into the future, with the two simpler methods of prediction being around 400m off on average and the more advanced predictors being around 200m off on average. This indicates usefulness for all four of these models, with the two models using advanced methods expectedly performing better than the other two.

Performance of the Predictors for different Look-Ahead Times

Next we look at the decrease in performance for each predictor when increasing the look-ahead time. Since we judged the original Manhattan model to be too unpredictable based on the data we showed above, going forward we will only use the two models we created.

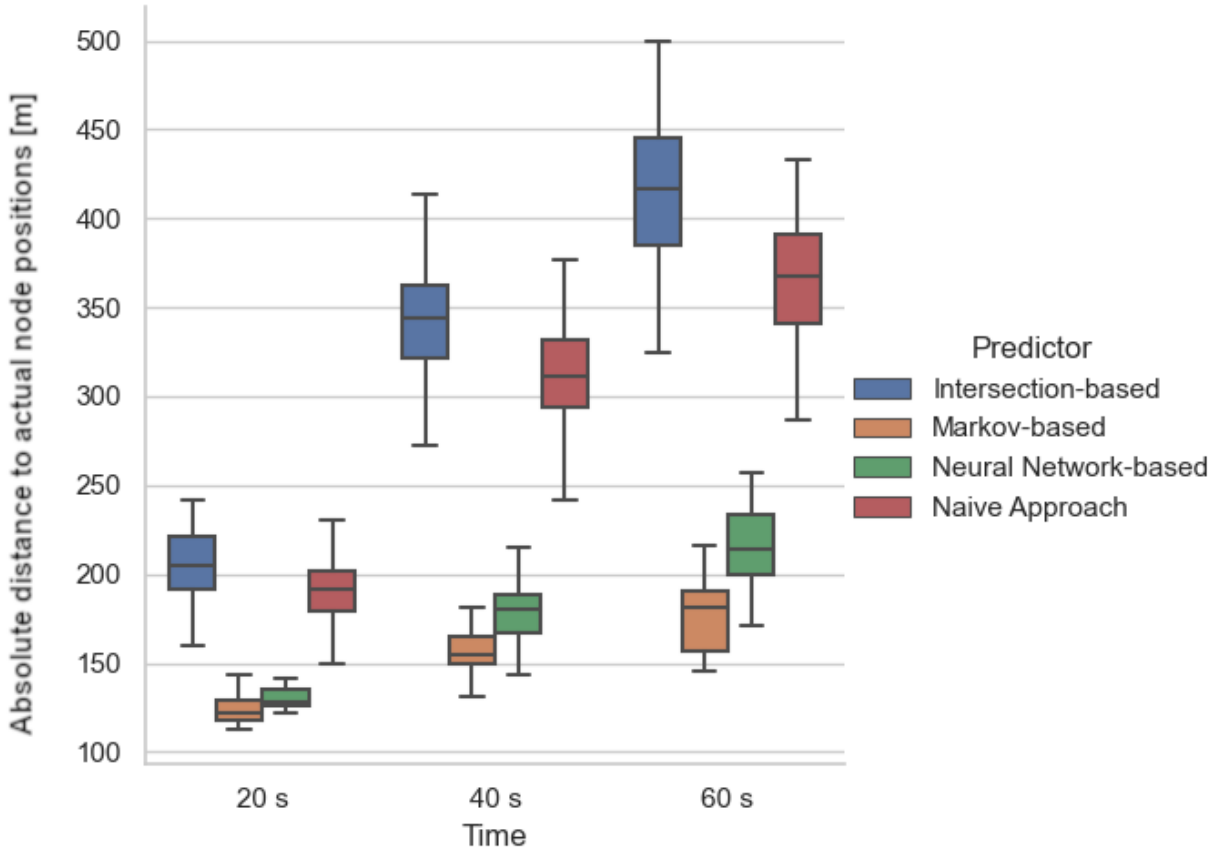

Figure 5.4: Decrease of the prediction accuracy over time for the Target Model

Figure Figure 5.4 shows the experimental results for the accuracy decrease when predicting the target model. The average accuracy for all prediction methods decreases, the larger the look-ahead time gets, as is to be expected. In addition the variance of the predictions also increases, which is to be expected as well, as luck plays a bigger role in each prediction over a longer period of time. One such example of luck influencing the outcome of a prediction massively would be a node reaching its destination within the look-ahead time and generating a new target location. In that case, there is no way to predict which way it will go, as the newly generated target has no correlation to the old target, therefore the path to the newly generated target does not correlate to the previous path the node has taken.

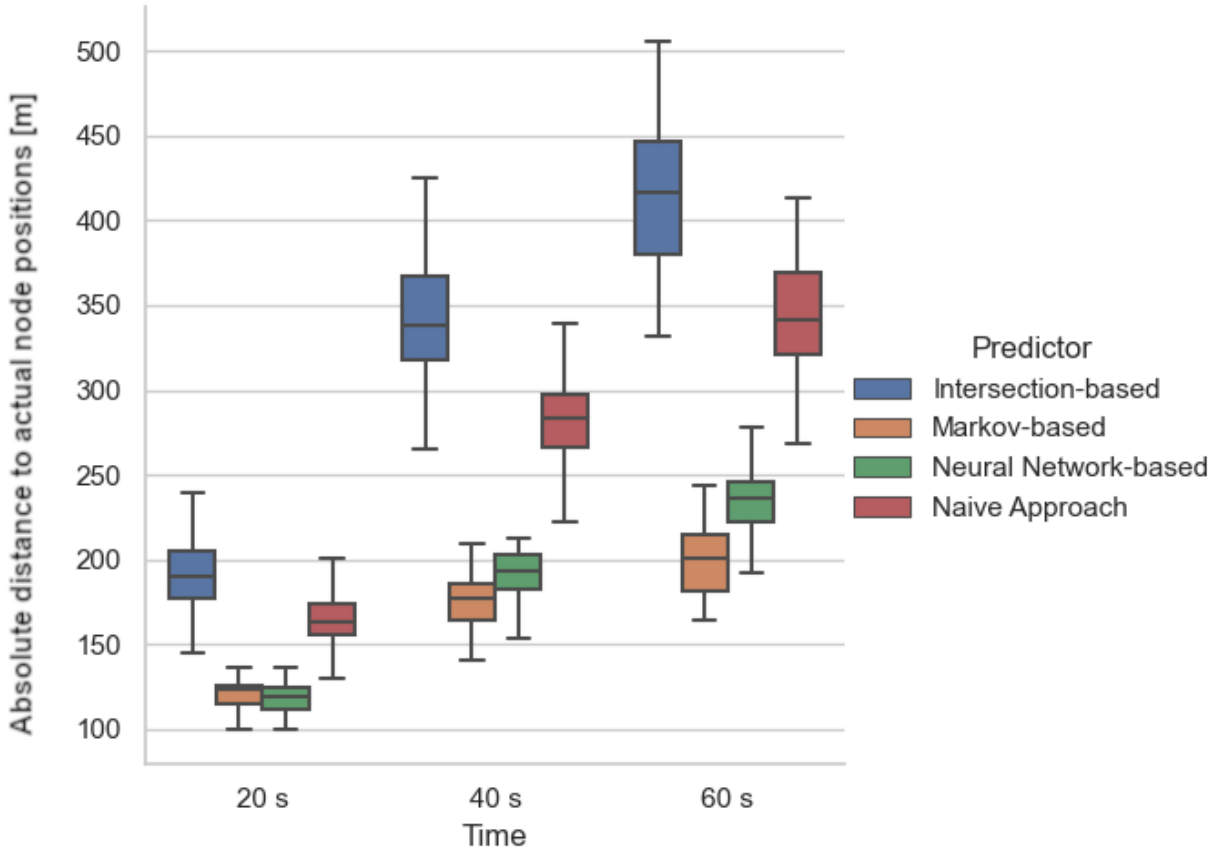

Figure 5.5: Decrease of the prediction accuracy over time for the Highway Model

Figure Figure 5.5 shows the experimental results for the accuracy decrease when predicting the highway model. The predictions for the Highway model behave very similarly to the predictions of the target model, as is to be expected because the models only differ mildly. In general, the prediction results for longer look-ahead times are slightly worse when comparing the highway to the target model, which can be explained by the fact that nodes on the highway move faster, making them cross intersections and generate new target locations more often.

Comparison of the different Prediction Methods

Finally, we compare the accuracy of every prediction method. As can be seen in all the experimental results above, is the intersection-based predictor the worst performing one when it comes to predicting the absolute positions.
This method of prediction relies on the nodes having clear exit preferences at intersections which might or might not hold true, as laid out in the last chapter.
For our test-case, we generated the target locations in the city uniformly, making this prediction method perform worse than it might have, had we chosen a method to generate clusters of target locations. Nevertheless, it is clearly the worst predictor in our scenario, with its average prediction accuracy being worse than the third quartile of the naive prediction method.
The naive approach for prediction performs noticeably better than the intersection-based method, but is clearly still outdone by the more advanced approaches. This was to be expected, as we already elaborated on this method's problems in the last chapter.

Its average prediction is nearly two times as far away from the ground truth than the Markov and neural network based predictors, indicating that those two are clearly better.
The two winners of this comparison are the Markov and neural network based models. Both perform only mildly differently when predicting the positions of nodes. The Markov model seems to have a general edge over the neural network, which can be explained by the fact that its algorithms had more constraints, which make it less generally applicable but more accurate in our specific case. Both approaches seem to be promising in general.

5.2.2 Topological Accuracy

In order to evaluate if the topological prediction quality matches the position prediction quality, adjacency matrices were created for the Wireless Sensor Networks.
The adjacency matrices of the predictions can be compared to the ground truth of the network topology at the predicted point in time in order to gain an overview of how well the prediction methods perform in matching the actual topology.
To give an overview of the general topological accuracy, we will show the prediction results after looking ahead 20, 40 and 60 seconds into the future and comparing them with the ground truth.

Comparison of the Topology for the Target Model

Table 5.4: Prediction Statistics for the Target Model Topology 20 seconds ahead

Model	Accuracy	Precision	Recall
Naive	90.97%	90.51%	70.65%
Intersection-based	93.81%	89.76%	84.52%
Markov-based	98.96%	93.80%	89.77%
Neural Network-based	94.85%	92.28%	89.63%

Table 5.5: Prediction Statistics for the Target Model Topology 40 seconds ahead

Model	Accuracy	Precision	Recall
Naive	88.06%	84.07%	63.99%
Intersection-based	90.69%	81.92%	79.48%
Markov-based	95.95%	88.38%	86.35%
Neural Network-based	94.52%	86.56%	86.08%

Table 5.6: Prediction Statistics for the Target Model Topology 60 seconds ahead

Model	Accuracy	Precision	Recall
Naive	86.77%	82.90%	61.45%
Intersection-based	89.83%	80.99%	74.29%
Markov-based	93.99%	87.11%	84.55%
Neural Network-based	92.46%	86.94%	83.83%

Tables Table 5.4, Table 5.5, Table 5.6 confirm that the prediction methods that predict node positions better, are generally also better at predicting the topology of the network.

The Markov based predictor predicts the topology best at all timesteps, followed by the neural network based approach in close proximity, with both of them clearly outperforming the other two predictors especially when it comes to predictions with a longer look-ahead time.
The first interesting thing to note is that after 20 second all four prediction methods are still relatively close in terms of precision and accuracy, but after 40 second the naive and intersection based predictors both rapidly decline in the quality of predictions they make. The second interesting thing to note is that while the naive predictor has a precision that is on par and even slightly better than the intersection based predictor, its recall is very low in comparison to every other predictor. This means that the naive predictor misses many connections between nodes that exist in the ground truth.
The intersection based predictor on the other hand has a higher Recall, even though none of its metrics are on par with the two better predictors.

Comparison of the Topology for the Highway Model

Table 5.7: Prediction Statistics for the Highway Model Topology 20 seconds ahead

Model	Accuracy	Precision	Recall
Naive	91.10%	89.45%	77.83%
Intersection-based	94.60%	84.25%	87.79%
Markov-based	98.10%	97.59%	93.76%
Neural Network-based	94.71%	92.43%	89.03%

Table 5.8: Prediction Statistics for the Highway Model Topology 40 seconds ahead

Model	Accuracy	Precision	Recall
Naive	88.99%	83.52%	70.08%
Intersection-based	90.90%	73.62%	84.20%
Markov-based	96.95%	88.97%	91.25%
Neural Network-based	92.52%	83.65%	84.02%

Table 5.9: Prediction Statistics for the Highway Model Topology 60 seconds ahead

Model	Accuracy	Precision	Recall
Naive	87.25%	80.14%	68.39%
Intersection-based	89.38%	69.57%	83.80%
Markov-based	95.38%	87.79%	88.53%
Neural Network-based	91.67%	81.24%	83.50%

Tables Table 5.7, Table 5.8, Table 5.9 show the experimental results for the topological prediction quality using the Highway model.
The general trend of the prediction results for the target model continues for this model as well. One notable thing is that the Markov based model actually seems to perform better here, while the neural network based model seems to struggle a bit more with this mobility model.
Another thing to note is the low precision for the intersection based method for this model specifically. This might be due to the fact that the intersection based predictor is bad at predicting where nodes will exit a highway, which might be the reason the prediction suffers here.

In general these metrics indicate that the Markov based approach as well as the neural network based approach are both suitable to predict future network topology, with the Markov based predictor showing especially promising results with the highway model with which the neural network based predictor seems to struggle a bit. The target model on the other hand shows the Markov predictor and the neural network predictor performing nearly evenly.

5.3 Performance of M-MOTA using the Predictors

Finally, in the following we directly compare all prediction methods when using them in the M-MOTA algorithm. Additionally we compare the prediction methods against the hypothetical perfect predictor explained in the test setup. This will give us the most indicative results on how well each predictor actually performs in the scenario introduced at the very beginning.

Results regarding Latency

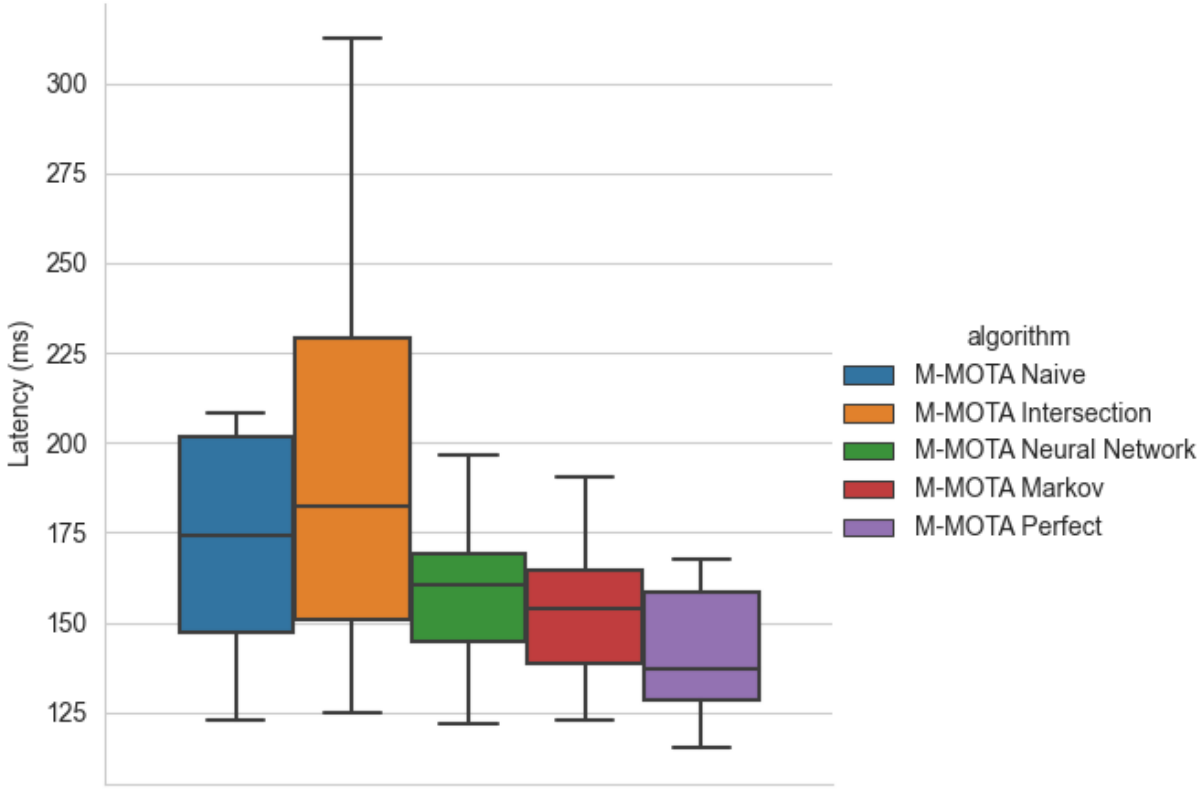

Figure 5.6: Experimental results for the Latency metric

Figure 5.6 shows the experimental results concerning the Latency of the evaluated network. The results using all methods mentioned above are displayed side by side to make the comparison easier.

The worst average latency and the highest variance in latency are both achieved by the intersection based predictor, which, as indicated in the previous sections already, has problems in our scenario.

The naive prediction method performs a bit better, achieving a slightly lower average latency and a noticeably lower variance.
The two winners, as already theorized based on the positional and topological predictions, are the Markov and neural network based prediction methods. The Markov predictor performs a bit better on average than the neural network based predictor, but both have a significantly smaller interquartile range than the other two prediction methods in addition to a better average latency.
Additionally, both predictors are not far off of the theoretical best, with the average of the Markov based approach just barely being lower than the third quartile of the theoretical best and the neural network based average being slightly higher than the third quartile of the perfect predictor.
This shows that both the Markov predictor as well as the neural network predictor are well suited to decrease the latency within a WSN by enhancing the task allocation.

Results regarding Packet Loss

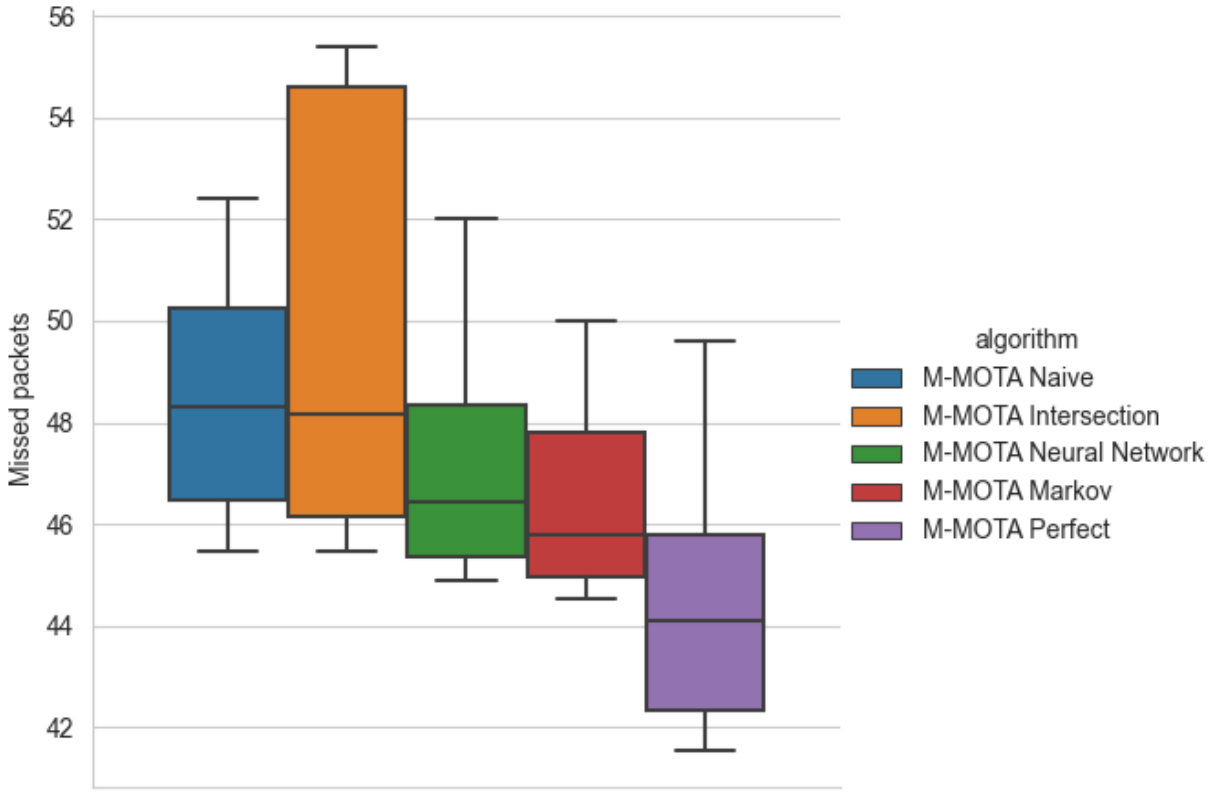

Figure 5.7: Experimental results for the Packet Loss

The amount of missed packets continues the trend for predictor performance, as seen in Figure 5.7. As expected, the perfect predictor delivers packets more reliably than all realistically achievable predictors, since the node position used for the optimization is always accurate and the algorithm can always allocate tasks to the correct nodes.
For the other predictors, the ones with a better prediction help the algorithm allocate tasks to make less packets miss.
The intersection based predictor notably is very unstable here and results in some good and some very bad allocations.

The naive predictor performs nearly equally on average as the intersection based predictor, but its interquartile range is nearly halved in comparison to it.
The two clearly better predictors are the neural network predictor and the Markov predictor again, with the neural network predictor being nearly equal to the Markov predictor with only very slightly increased packet loss. Both are clearly worse than the hypothetical best, but their averages are still only barely above the third quartile of it.
Overall, both of these methods seem suitable again.

Results regarding Power Consumption

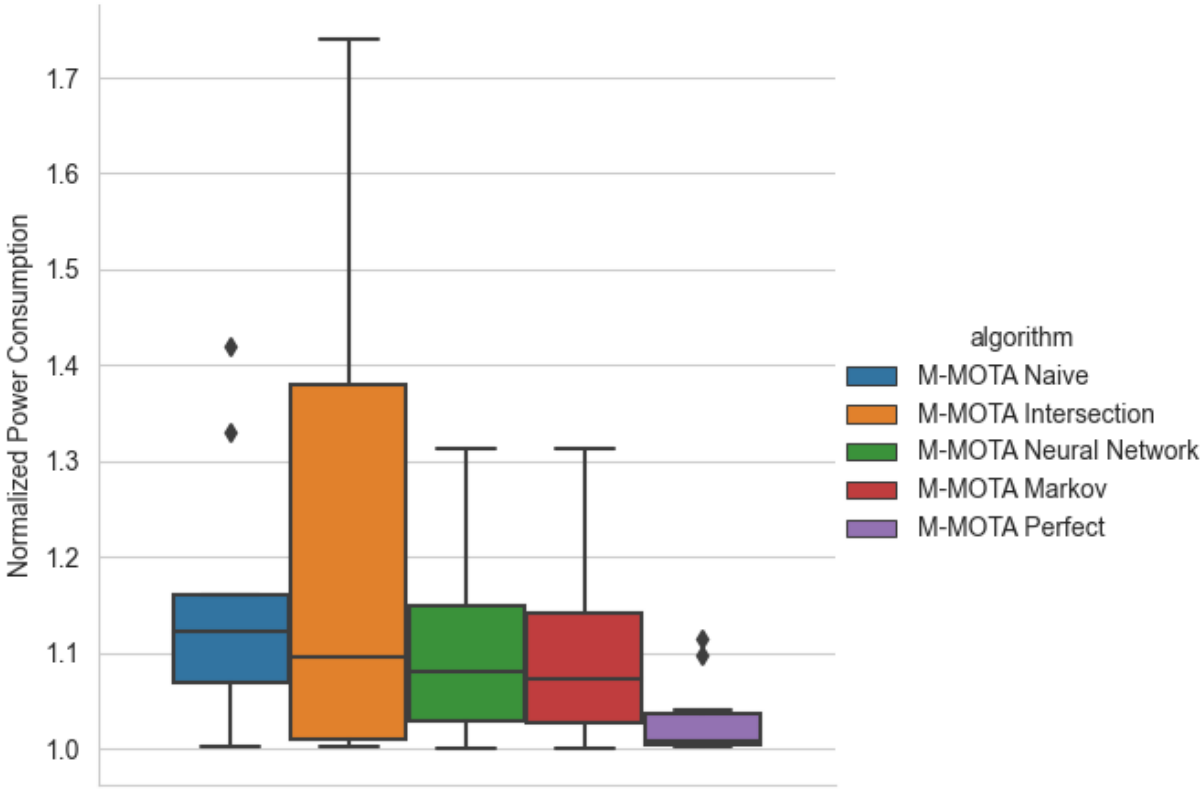

Figure 5.8: Experimental results for the Power Consumption metric

The power consumption of the allocations using each predictor is shown in Figure 5.8. It has been normalized with the packet loss as to prevent the problem that showed up in the original M-MOTA experiments [13], when worse allocations had lower power consumption due to high packet loss and the resulting non-execution of tasks.
The intersection predictor shows very unstable power consumption, just like for the package loss.
The naive predictor, neural network predictor and Markov predictor all perform well for power consumption, with all of them being clearly behind the perfect predictor.
Overall it seems that for power consumption any vaguely competent prediction method is fine, but even very good prediction methods struggle to reach the low power consumption a perfect prediction would enable.

6 Conclusion and Future Work

In this work four prediction methods based on node memory, intersection memory, a Markov model, and an artificial neural network were introduced, that can be used in combination with the M-MOTA algorithm to allocate tasks in a WSN. Two new mobility models based on the Manhattan mobility model which can be used to evaluate the prediction methods and the M-MOTA algorithm in a smart city scenario were introduced as well.
All of the predictors and mobility models introduced were used to evaluate which prediction methods most enhance the task allocation for the described scenario. Results show that the Markov predictor and the neural network predictor show good results for all quality metrics introduced when compared to what is theoretically possible.
The introduced Markov predictor was shown to be the best in terms of all quality metrics and clearly performs well in our scenario.
The neural network predictor was demonstrated to not fall far behind the Markov based approach for all quality metrics, while having the potential to be more generalizable for other scenarios and for real world data by relying less on human modelling of a scenario than the Markov predictor.

In the future, more complex mobility models or real traced data could be used to re-evaluate all prediction methods and see if the results differ from what we have found using our introduced mobility models.
In addition, the neural network used as a predictor in this work was based on a relatively simple model when compared to state of the art models used in other applications. There were no cutting edge methods used to fully optimize its predictions, as that was outside the scope of this work. Therefore, room for improvement most likely still exists for the neural network predictor and further investigation is needed to observe how accurate this prediction method can become when modelled and trained optimally. Given the general success of neural networks for time series forecasting, it might be reasonable to assume that a more optimal model and training method could outperform the Markov predictor.

Optimally allocating tasks in mobile Wireless Sensor Networks remains a challenging task due to the complex nature of the overall task allocation problem, as well as the many internal and external factors that influence all nodes in the network and by extension the way tasks need to be allocated.
Ultimately, this work provides an overview over which prediction methods are useful to help M-MOTA better allocate tasks in Wireless Sensor Networks and can be used as a baseline when using the algorithm in other scenarios and future work.

Bibliography

[1] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, "From machine-to-machine to the internet of things," ch. 2, p. 9 ff., Oxford: Academic Press, 2014.

[2] M. Maier, "The internet of things (iot): what is the potential of internet of things applications for consumer marketing?," June 2016.

[3] A. Ouadjaout, N. Lasla, M. Bagaa, M. Doudou, C. Zizoua, M. Kafi, A. Derhab, D. Djenouri, and N. Badache, "Dz50: Energy-efficient wireless sensor mote platform for low data rate applications," *Procedia Computer Science*, vol. 37, 2014.

[4] M. Sommarberg, R. Gustafsson, Z. Cheung, and E. Aalto, *Value Creation from the Internet of Things in Heavy Machinery: A Middle Manager Perspective.* Singapore: Springer Singapore, 2018.

[5] U. Iqbal, M. A. Dar, and S. Nisar Bukhari, "Intelligent hospitals based on iot," in *Fourth International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics*, 2018.

[6] A. Gaur, B. Scotney, G. Parr, and S. McClean, "Smart city architecture and its applications based on iot," *Procedia Computer Science*, vol. 52, 2015.

[7] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery, 2011.

[8] K.-H. N. Bui, J. E. Jung, and D. Camacho, "Game theoretic approach on real-time decision making for iot-based traffic light control," *Concurrency and Computation: Practice and Experience*, vol. 29, 2017.

[9] C. S. Raghavendra, K. M. Sivalingam, and T. Znati, *Wireless sensor networks*, p. 7 ff. Springer, 2004.

[10] Insider Intelligence, "How iot and smart city technology works: Devices, applications and examples." https://www.businessinsider.com/iot-smart-city-technology, 02 2022. Accessed: 2022-03-17.

[11] R. Marzoug, H. Ez-Zahraouy, and A. Benyoussef, "Simulation study of car accidents at the intersection of two roads in the mixed traffic flow," *International Journal of Modern Physics C*, vol. 26, 2015.

[12] D. Weikert, C. Steup, and S. Mostaghim, "Multi-Objective Task Allocation for Wireless Sensor Networks," in *Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2020.

[13] D. Weikert, C. Steup, D. Atienza, and S. Mostaghim, "Mobility-aware multi-objective task allocation for wireless sensor networks," in *Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2021.

[14] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, ch. 13, p. 307 ff. Springer US, 2011.

[15] L. Tan and N. Wang, "Future internet: The internet of things," in *3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, 2010.

[16] V. Mulloni and M. Donelli, "Chipless rfid sensors for the internet of things: Challenges and opportunities," *Sensors*, vol. 20, 2020.

[17] J. A. BK Tripathy, *Internet of Things (IoT) Technologies, Applications, Challenges and Solutions*, pp. 42–45. CRC Press, 2020.

[18] H. U. Rehman, M. Asif, and M. Ahmad, "Future applications and research challenges of iot," in *International conference on information and communication technologies (ICICT)*, IEEE, 2017.

[19] S. Auer, S. Nagler, S. Mazumdar, and R. R. Mukkamala, "Towards blockchain-iot based shared mobility: Car-sharing and leasing as a case study," *Journal of Network and Computer Applications*, vol. 200, 2022.

[20] V. Miz and V. Hahanov, "Smart traffic light in terms of the cognitive road traffic management system (ctms) based on the internet of things," in *Proceedings of IEEE East-West Design Test Symposium (EWDTS)*, 2014.

[21] S. C. Mukhopadhyay and N. K. Suryadevara, *Internet of Things: Challenges and Opportunities*, ch. 1. Springer International Publishing, 2014.

[22] M. S. Obaidat and S. Misra, *Principles of Wireless Sensor Networks*, ch. 1. Cambridge University Press, 2014.

[23] G. Strait, "The complete list of wireless iot network protocols." https://www.link-labs.com/blog/complete-list-iot-network-protocols, 02 2016. Accessed: 2022-03-17.

[24] Nayak, Pani, Choudhury, Satpathy, and Mohanty, *Wireless Sensor Networks and the Internet of Things: Future Directions and Applications*, p. 21 ff. Apple Academic Press, 2021.

[25] C. Yawut and S. Kilaso, "A wireless sensor network for weather and disaster alarm systems," in *International conference on information and electronics engineering, IPCSIT*, 2011.

[26] Q. Wang, A. Terzis, and A. Szalay, "A novel soil measuring wireless sensor network," in *IEEE Instrumentation Measurement Technology Conference Proceedings*, 2010.

[27] T. Bokareva, W. Hu, S. Kanhere, B. Ristic, N. Gordon, T. Bessell, M. Rutten, and S. Jha, "Wireless sensor networks for battlefield surveillance," in *Proceedings of the land warfare conference*, 2006.

[28] A. Tiwari, F. L. Lewis, and S. S. Ge, "Wireless sensor network for machine condition based maintenance," in *8th Control, Automation, Robotics and Vision Conference*, IEEE, 2004.

[29] J. P. Dominguez-Morales, A. Rios-Navarro, M. Dominguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, D. Cascado-Caballero, A. Jimenez-Fernandez, and A. Linares-Barranco, "Wireless sensor network for wildlife tracking and behavior classification of animals in doñana," *IEEE Communications Letters*, vol. 20, 2016.

[30] A. Niccolai, F. Grimaccia, M. Mussetta, and R. Zich, "Optimal task allocation in wireless sensor networks by means of social network optimization," *Mathematics*, vol. 7, 2019.

[31] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 14, 2007.

[32] M. Yuan, C. Jiang, S. Li, W. Shen, Y. Pavlidis, and J. Li, "Message passing algorithm for the generalized assignment problem," in *Int. Conf. on Network and Parallel Computing*, Springer, 2014.

[33] J. Yang, H. Zhang, Y. Ling, C. Pan, and W. Sun, "Task allocation for wireless sensor network using modified binary particle swarm optimization," *IEEE Sensors Journal*, vol. 14, 2014.

[34] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless Communications and Mobile Computing*, vol. 2, 08 2002.

[35] M. López and P. Manzoni, "Anejos: a java based simulator for ad hoc networks," *Future Generation Computer Systems*, vol. 17, 2001.

[36] A. abu taleb, T. Alhmiedat, O. Al-Haj Hassan, and N. Turab, "A survey of sink mobility models for wireless sensor networks," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 4, 01 2013.

[37] F. Bai, N. Sadagopan, and A. Helmy, "Important: A framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks," *IEEE INFOCOM*, 2003.

[38] R. R. Roy, *Handbook of Mobile Ad Hoc Networks for Mobility Models*. Springer-Verlag, 2010.

[39] F. Theoleyre, R. Tout, and F. Valois, "New metrics to evaluate mobility models properties," in *2nd International Symposium on Wireless Pervasive Computing*, 2007.

[40] G. Jayakumar and G. Ganapathi, "Reference point group mobility and random waypoint models in performance evaluation of manet routing protocols," *Journal of Computer Networks and Communications*, 2008.

[41] S. A. Williams and D. Huang, "Group force mobility model and its obstacle avoidance capability," *Acta Astronautica*, vol. 65, 2009.

[42] I. Pardoe, L. Simon, and D. Young, "Stat 501: Regression methods - lesson 14." https://online.stat.psu.edu/stat501/lesson/14/14.1. Accessed: 2022-03-17.

[43] D.-S. S. Kim and S.-K. Hwang, "Swarm group mobility model for ad hoc wireless networks," *Journal of Ubiquitous Convergence Technology*, vol. 1, 2007.

[44] M. Musolesi and C. Mascolo, "A community based mobility model for ad hoc network research," in *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, 2006.

[45] H. Tsunoda, K. Ohta, N. Kato, and Y. Nemoto, "Geographical and orbital information based mobility management to overcome last-hop ambiguity over ip/leo satellite networks," *IEEE International Conference on Communications*, 07 2006.

[46] J.-D. M. M. Biomo, T. Kunz, and M. St-Hilaire, "An enhanced gauss-markov mobility model for simulations of unmanned aerial ad hoc networks," in *7th IFIP Wireless and Mobile Networking Conference (WMNC)*, 2014.

[47] S. Ahmed, G. C. Karmakar, and J. Kamruzzaman, "Geographic constraint mobility model for ad hoc network," in *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, 2008.

[48] J. Tian, J. Hahner, C. Becker, I. Stepanov, and K. Rothermel, "Graph-based mobility model for mobile ad hoc network simulation," in *Proceedings of the 35th Annual Simulation Symposium*, April 2002.

[49] S. Ranka, S. Aluru, R. Buyya, Y.-C. Chung, S. Gupta, A. Grama, R. Kumar, V. V. Phoha, and S. Dua, *Contemporary Computing*, ch. 4, p. 170 ff. Springer Science & Business Media, 2009.

[50] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*, ch. 1. OTexts, 2018.

[51] P. A. Gagniuc, *Markov chains: from theory to implementation and experimentation*. John Wiley & Sons, 2017.

[52] Elenktik, "Corrected markov chain." https://commons.wikimedia.org/wiki/File:Mchain_simple_corrected_C1.png, 2015. Accessed: 2022-03-19.

[53] M. L. Littman, "A tutorial on partially observable markov decision processes," *Journal of Mathematical Psychology*, vol. 53, 2009.

[54] S. R. Eddy, "What is a hidden markov model?," *Nature biotechnology*, vol. 22, 2004.

[55] E. Vanmarcke, *Random fields: analysis and synthesis*. World scientific, 2010.

[56] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, 1982.

[57] R. Rojas, *Neural Networks - A Systematic Introduction*. Springer, 1996.

[58] J. Salatas, "Multilayer neural network." https://commons.wikimedia.org/wiki/File:Multilayer_Neural_Network.png, 2011. Accessed: 2022-03-19.

[59] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," vol. 15, 2010.

[60] MartinThoma, "Sigmoid function." https://de.wikipedia.org/wiki/Datei:Sigmoid-function-2.svg, 2014. Accessed: 2022-03-19.

[61] Renanar2, "Contrast between the rectified linear unit function and the nonnegative soft thresholding pointwise nonlinearities." https://commons.wikimedia.org/wiki/File:ReLU_and_Nonnegative_Soft_Thresholding_Functions.svg, 2018. Accessed: 2022-03-19.

[62] IBM Cloud Education, "Recurrent neural networks." https://www.ibm.com/cloud/learn/recurrent-neural-networks. Accessed: 2022-03-17.

[63] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, 1998.

[64] G. Chevalier, "The long short-term memory (lstm) cell can process data sequentially and keep its hidden state through time." https://commons.wikimedia.org/wiki/File:The_LSTM_cell.png, 2018. Accessed: 2022-03-19.

[65] C. Olah, "Understanding lstm networks." https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. Accessed: 2022-03-19.

[66] K. Deb, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-2," *IEEE Trans. Evol. Comput.*, vol. 6, 2002.

[67] J. Xuan, H. Jiang, and Z. Ren, "Pseudo code of genetic algorithm and multi-start strategy based simulated annealing algorithm for large scale next release problem," *Dalian University of Technology*, 2011.

[68] T. D. Gwiazda, *Crossover for single-objective numerical optimization problems*, vol. 1. Tomasz Gwiazda, 2006.

[69] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *International journal of machine learning and computing*, vol. 7, no. 1, 2017.

[70] Y. Jin, S. Vural, A. Gluhak, and K. Moessner, "Dynamic task allocation in multi-hop multimedia wireless sensor networks with low mobility," *Sensors*, vol. 13, 2013.

[71] D. Weikert, C. Steup, and S. Mostaghim, "Availability-aware Multi-Objective Task Allocation Algorithm for Internet-of-Things Networks," *IEEE Journal of Internet of Things*, vol. in revision, 2021.

[72] J. Krumm, "A markov model for driver turn prediction," *Society of Automotive Engineers (SAE) World Congress*, 2008.

[73] P. Crescenzi, M. D. Ianni, A. Marino, G. Rossi, and P. Vocca, "Spatial node distribution of manhattan path based random waypoint mobility models with applications," in *Structural Information and Communication Complexity, 16th International Colloquium*, Lecture Notes in Computer Science, Springer, 2009.

[74] F. J. Martinez, J.-C. Cano, C. T. Calafate, and P. Manzoni, "Citymob: A mobility model pattern generator for vanets," in *ICC Workshops - IEEE International Conference on Communications Workshops*, 2008.

[75] B. Ramakrishan, M. M. Joe, and R. B. Nishanth, "Modeling and simulation of efficient cluster based manhattan mobility model for vehicular communication," *Journal of emerging technologies in web intelligence*, vol. 6, 2014.

[76] J. Peng, Y. Guo, R. Fu, W. Yuan, and C. Wang, "Multi-parameter prediction of drivers' lane-changing behaviour with neural network model," *Applied ergonomics*, vol. 50, 2015.

[77] O. Olabiyi, E. Martinson, V. Chintalapudi, and R. Guo, "Driver action prediction using deep (bidirectional) recurrent neural network," *ArXiv*, vol. abs/1706.02257, 2017.

[78] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*, pp. 15–34, Springer, 2010.

Statement of Authorship

Thesis: Predicting Future Network Topology in Wireless Sensor Networks with Mobile Nodes

Name: David Surname: Atienza Fernandez

Date of birth: 27.06.1997 Matriculation no.: 222275

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency.

Signature Date