SPATIO-TEMPORAL GRAPH-BASED INTERPOLATION FOR MOBILE SENSING NETWORKS

Bachelor's Degree in Computer Science

Bachelor's Degree in Telecommunications Engineering

May, 2022

By

Adrià Guinovart Plana

Director: Dr. Antonio Ortega

Co-director: Dr. Montse Pardàs Feliu

ACKNOWLEDGMENTS

This thesis is the result of eight months of research at the Ming Hsieh Department of Electrical and Computer Engineering at USC Viterbi, developed as the final bachelor thesis of my Computer Science and Telecommunications Engineering Degrees at CFIS-UPC.

I wish to extend my special thanks to my tutor Dr. Antonio Ortega, who has supervised this research project through all of its stages, providing me with essential guidelines and enriching this project. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research and daily life. I would also like to thank Dr. Montse Pardàs Feliu for her valuable insights, support and feedback during the development of this thesis. To my school at CFIS-UPC, for giving me the opportunity and support to dedicate a semester of my bachelor's program at USC.

Finally, I would like to express my gratitude to my parents and friends for their encouragement and support all through my studies.

ABSTRACT

Mobile sensor networks are attracting a lot of interest in the research community, since they allow to exploit mobility and cover larger areas than conventional fixed sensor networks. This thesis approaches the main issues that arise in these mobile sensor frameworks, such as irregular sampling in both space and time or low spatio-temporal resolution, from a Graph Signal Processing (GSP) perspective. We present a tree graph-based interpolation method for mobile sensing networks as a solution to increase the spatio-temporal resolution that these sensor networks provide. The proposed tree graph method parameters are optimized for an air quality dataset gathering ozone levels in an urban area in Zurich, Switzerland. We found that our solution is able to adapt to local changes and outperform classical spatial interpolation techniques such as Kriging and Inverse Distance Weighting (IDW).

Keywords

Graph signal processing, interpolation, mobile sensor networks, spatio-temporal resolution.

RESUMEN

Las redes de sensores móviles están atrayendo mucho interés en la comunidad investigadora, ya que permiten explotar la movilidad y cubrir áreas más grandes que las redes de sensores fijas convencionales. Esta tesis aborda los principales problemas que surgen en los escenarios con sensores móviles, como el muestreo irregular tanto en espacio como tiempo o la baja resolución espacio-temporal, desde una perspectiva de procesado de señal mediante grafos (GSP). Presentamos un método de interpolación basado en grafos con topología de árbol para redes de sensores móviles como solución para aumentar la resolución espacio-temporal que proporcionan estas redes de sensores. Los parámetros del método basado en grafos con la topología de árbol propuesto están optimizados para una base de datos de calidad del aire que recopila los niveles de ozono en un área urbana de Zúrich, Suiza. Descubrimos que nuestra solución puede adaptarse a los cambios locales y superar las técnicas clásicas de interpolación espacial como Kriging e Inverse Distance Weighting (IDW).

Palabras clave

Procesado de señal mediante grafos, interpolación, redes de sensores móviles, resolución espacio-temporal.

RESUM

Les xarxes de sensors mòbils estan atraient molt interès a la comunitat investigadora, ja que permeten explotar la mobilitat i cobreixen àrees més grans que les xarxes de sensors fixes. Aquesta tesi aborda les principals qüestions que sorgeixen en aquests marcs de sensors mòbils, com ara el mostreig irregular tant en l'espai com en el temps o la baixa resolució espai-temporal, des d'una perspectiva de processament de senyal mitjançant grafs (GSP). Presentem un mètode d'interpolació basat en grafs amb topologia d'arbre per a xarxes de detecció mòbil com a solució per augmentar la resolució espai-temporal que proporcionen aquestes xarxes de sensors. Els paràmetres del mètode de basat en grafs amb topologia d'arbre proposat estan optimitzats per a una base de dades de qualitat de l'aire que recull els nivells d'ozó en una àrea urbana de Zuric, Suïssa. Hem trobat que la nostra solució és capaç d'adaptar-se als canvis locals i superar les tècniques clàssiques d'interpolació espacial com ara Kriging i Inverse Distance Weighting (IDW).

Paraules clau

Processat de senyal mitjançant grafs, interpolació, xarxes de sensors mòbils, resolució espai-temporal.

Table of Contents

Acknowledgments **ii**

Abstract **iii**

Resumen **iv**

Resum **v**

Table of Contents **vi**

List of Figures **x**

Chapter 1:

Introduction **1**

1.1 State of the art and related work 2

1.2 Outline and main contributions 3

Chapter 2:

Graph Signal Processing Fundamentals **5**

2.1 Graphs 5

2.1.1 Basic definitions 5

2.1.2 Algebraic representation of graphs 7

2.2 Graph signals 7

2.3 Tree graphs 8

2.3.1 Maximum spanning tree 8

Chapter 3:

Problem definition **10**

3.1 Air quality dataset 12

Chapter 4:

Proposed solution **14**

4.1 Data pre-processing 14

4.1.1 Timestamp reference 15

4.1.2 Coordinates to distance conversion 15

4.2 Graph learning 16

4.2.1 Topology inference 17

4.2.2 Weight estimation 18

4.2.3 Graph learning algorithm 24

4.3 Interpolation 26

4.4 Solution optimization 28

4.4.1 Temporal complexity analysis 28

4.4.2 Removal of candidate nodes without samples 29

4.4.3 Maximum edge distance 30

Chapter 5:

Experiments and results **33**

5.1 Tree model validation 33

5.1.1 Performance metrics 33

5.1.2 Cross-validation 35

5.2 Hyper-parameter tuning 37

5.2.1 Hyper-parameter tuning: Δt 37

5.2.2 Hyper-parameter tuning: Δs 42

5.2.3 Hyper-parameter tuning: α 45
5.2.4 Hyper-parameter tuning: maximum edge distance 47
5.3 Comparison with state of the art interpolation methods 49
5.3.1 Inverse Distance Weighting 50
5.3.2 Kriging 51
5.3.3 Results 55

Chapter 6:

Conclusions **60**

6.1 Conclusions 60
6.2 Future Work 61

References **62**

Appendix A: Implementation **67**

List of Figures

4.1 Spatial distribution of the air quality data. 17

4.2 Illustrative example of every possible combination of squared differences. . 22

4.3 Illustrative example of missing observations approach. 24

4.4 Simple graph used to exemplify the tree interpolation scheme. 27

4.5 Node distribution and frequency of observations with $\Delta s = 100m$. 31

4.6 Example of tree graph obtained with $\Delta t = 30min$, $\Delta s = 150m$, $\alpha = 0.1$. . 32

5.1 Illustrative example of k-fold cross-validation with $k = 5$. 36

5.2 Nested cross-validation scheme. 36

5.3 Temporal autocorrelation plots for different time periods. 39

5.4 RMSE as a function of Δt. 41

5.5 Spatial autocorrelation. 43

5.6 RMSE as a function of Δs. 45

5.7 RMSE as a function of α. 47

5.8 RMSE as a function of the maximum edge distance. 48

5.9 Weights computation time as a function of the maximum edge distance. . 49

5.10 Example of different Kriging variogram models. 53

5.11 RMSE for different interpolation methods as a function of Δt. 56

5.12 RMSE for different interpolation methods as a function of Δt. 56

5.13 MSE with error bars for different interpolation methods. 57

5.14 Edge weights in the tree graph as a function of edge distance. 58

5.15 Execution time for different interpolation methods as a function of Δt. . . 59

Chapter 1

Introduction

Large amounts of data and information are surrounding us. Almost every part of human existence is being recorded and gathered as information by analysts, governments and companies: our own information through applications, monetary and banking information, our social networks, mobility patterns, advertising inclinations and more. All this information that is gathered needs to be processed to be valuable for its purpose. The complexity of these networks and interactions implies that the information dwells on irregular and complex structures that do not lend themselves to standard tools [1].

To confront this issue, graphs are being utilized to address the construction of information domains in different applications. For example, on account of social networks, individual users lay out associations with one another and interact producing a lot of information. In a graph representation, users can be represented by nodes and social interactions or connections between users can be represented by graph edges. The information on these graphs can be envisioned as a finite set of samples with one sample at each graph node. This arrangement could be utilized, for instance, to deduce the structure of a community by its relations through game theoretic models [2], evaluate the connectedness of the world or study the significance or impact of specific users [3].

The emerging field of Graph Signal Processing (GSP) merges graph theory concepts with signal processing in order to process all the data generated and structured as a complex network, which will be represented by a graph signal. An important area of research in GSP is the interpolation problem in graph structured data. This arises in different forms, such as in semi-supervised learning of categorical data, ranking problems,

and missing value prediction such as matrix completion problems [4]. One of the main sources of data and information are sensor networks. The conventional approach consists on deploying *fixed* sensor networks, where sensors are installed in certain locations or monitoring stations [5]. These sensor networks usually require a significant amount of nodes to cover the area of interest with a high spatial resolution, which is desired in many applications such as the development of hydraulic models [6]. Nevertheless, *mobile* sensor networks have gained importance among the research community over the last few years. These sensor deployments are very appealing because the sensor nodes capturing the data are mobile so that they can cover wider areas. However, modeling the data captured from mobile sensing networks can be complex for multiple reasons, including for example the inherent irregularity of their spatio-temporal sampling [7].

The main purpose of this research is to develop a graph-based interpolation method that is capable of modeling the data captured by mobile sensor networks and making predictions. For example, this method enables providing estimated values at sensed locations at times when sensors in those locations are not active, so that the spatio-temporal resolution of the data can be increased.

1.1 State of the art and related work

In recent years, a lot of work has been done in Graph Signal Processing (GSP). Although GSP is a very broad area of study, we will mainly focus on the part that is most related to the use of graphs for signal interpolation. We will also present some relevant work on mobile sensor networks and interpolation techniques not related to graphs.

In GSP, graph learning from data is a crucial problem. There exist many ways in which we can obtain a graph from data samples, but what we really want to accomplish is to learn a graph that represents the information in the data through meaningful connections. As an example, a signal smoothness-based technique together with Laplacian interpolation

has been proposed in air pollution monitoring [8].

In some cases, when working within a GSP framework, research has shown that fixing the graph topology before learning the graph can yield an optimal graph construction for that particular topology. In other words, the best graph of a given topology can be found optimally. Research presented in [9] introduces an optimal graph learning process when the graph topology is fixed beforehand to a tree topology, i.e., a graph without any cycles.

In the spatial interpolation field, there exist a lot of interpolation techniques. The most commonly used interpolation methods are distance-weighting methods, interpolating polynomials and kriging, which is a well-known geostatistical estimator [10]. Some research combines these classic data-driven models adopting a random forest algorithm to adaptively choose the more accurate models in order to achieve an error reduction [11].

1.2 Outline and main contributions

This thesis has been structured in six parts. This Introduction to the topic of Graph Signal Processing together with the state of the art and the outline of this thesis is the content presented in Chapter 1.

In Chapter 2 we present the framework in which GSP is developed, stating some basic definitions in the field of graph theory. Then, we introduce how signals are represented in a graph environment as well as the representation of time-varying graph signals. We also dive into tree graphs, which will be of extreme importance in the proposed solution.

Chapter 3 describes the problem to be solved and the actual motivation of the research: how to increase the spatio-temporal resolution of a signal from a GSP perspective. Also, the data with which we worked during this research is presented and the main aspects of this data captured in a mobile sensing network are highlighted.

In Chapter 4 we present our proposed solution to perform interpolation in mobile sensing networks. Some existing theory in the tree graph learning from [9] is presented in

more detail. Then, we discuss the issues that arise when using this approach in a mobile sensing environment and propose a solution for each of them so we can adapt the method to the kind of data that is captured by mobile sensor networks, which is very irregular.

In Chapter 5 we describe the performance metrics and the cross-validation techniques used to tune the model hyper-parameters. Moreover, we compare the proposed tree graph-based interpolation method with state of the art interpolation methods such as Inverse Distance Weighting (IDW) and Kriging.

Finally, Chapter 6 presents the conclusions and describes future work to be done related to the research presented in this thesis. Appendix A describes the GitHub repository where the code used to implement the presented techniques can be found.

Chapter 2

Graph Signal Processing

This chapter covers the fundamentals of Graph Signal Processing and presents the necessary graph theory concepts and definitions that are essential to understand the development of this research.

2.1 Graphs

2.1.1 Basic definitions

In this section, some basic definitions related to basic graph theory are provided.

Definition 2.1. *A **graph** $\mathcal{G}(V, E)$ is a set of nodes V (also called vertices) and edges $E \subseteq \{(i, j) \mid (i, j) \in V^2\}$, where an edge e_{ij} represents a connection between the node i and the node j.*

Definition 2.2. *A graph $\mathcal{G}(V, E)$ is **weighted** if any edge $e_{ij} \in E$, has a defined real positive weight w_{ij}, or **unweighted**, if the weight for all its edges is equal to 1.*

If an edge between two nodes does not exist it can be represented by a weight equal to zero, both in weighted and unweighted graphs.

Definition 2.3. *A graph $\mathcal{G}(V, E)$ is **undirected** if for every edge $e_{ij} \in E$, it holds that $e_{ji} \in E$ and $w_{ij} = w_{ji}$. Conversely, a graph is **directed** if there exists any edge $e_{ij} \in E$ such that $e_{ji} \notin E$ or $e_{ji} \in E$ and $w_{ij} \neq w_{ji}$.*

In other words, edges in an undirected graph do not have a direction or can be traversed in both directions. On the other hand, a directed graph has edges with direction that indicate a one-way connection.

Definition 2.4. *A graph $\mathcal{G}'(V', E')$ is a **subgraph** of $\mathcal{G}(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.*

Definition 2.5. *Given an undirected graph $\mathcal{G}(V, E)$, the **node degree** is defined as the sum of the weights of its edges:*

$$d_i = \sum_{\forall j \in V} w_{ij}$$

In the case of directed graphs, the in-degree and out-degree are defined as follows:

$$d_i^{in} = \sum_{\forall j \in V} w_{ij},\ d_i^{out} = \sum_{\forall \mathrm{j} \in V} w_{ji}$$

Definition 2.6. *A **regular** graph is a graph where each node has the same number of neighbors.*

Definition 2.7. *A **complete** graph is a particular case of regular graph in which every pair of distinct nodes is connected by an edge.*

The number of edges of a complete graph with N nodes is $\frac{N(N-1)}{2}$.

Definition 2.8. *A graph is said to be **strongly connected** if every node is reachable from every other node.*

The strongly connected components of a graph form a partition into subgraphs that are themselves strongly connected.

Definition 2.9. *A graph is said to be **dense** if the number of edges is close to the maximal number of edges. Conversely, a graph is **sparse** if the number of edges is close to the minimal number of edges.*

2.1.2 Algebraic representation of graphs

Definition 2.10. *Given a graph $\mathcal{G}(V, E)$ with $|V| = N$ nodes, the* ***adjacency matrix*** $\mathbf{A}$ *is an $N \times N$ square matrix where the entry corresponding to the i-th row and j-th column is $a_{ij} = w_{ij}$.*

Note that for undirected graphs, the adjacency matrix will be symmetric, given that $a_{ij} = a_{ji},\ \forall\ i, j \in V$. Moreover, if there are no self-loops, $a_{ii} = 0,\ \forall\ i \in V$, i.e. all the elements in the diagonal will be zero.

Definition 2.11. *For any undirected graph with adjacency matrix* $\mathbf{A}$ *we can define an* ***upper triangular*** *matrix* $\mathbf{U}$ *containing all the weight values as:*

$$u_{ij} = \begin{cases} a_{ij} & \text{if } i \leq j \\ 0 & \text{otherwise} \end{cases}$$

Note that for an undirected graph, the upper triangular part $\mathbf{U}$ of the adjacency matrix A contains all the graph information, since $\mathbf{A}$ is symmetric. Thus, it is only required to visit $\mathbf{U}$ to traverse the adjacency matrix of an undirected graph (which is twice as fast computationally than traversing the entire matrix).

2.2 Graph signals

Now that the basic graph theory concepts have been defined, let us present signals on graphs.

Definition 2.12. *A* ***graph signal*** *is a real vector $\mathbf{x} \in \mathbb{R}^N$, where the entry x_i is the real scalar corresponding to the node i.*

Definition 2.13. *A* ***time-varying graph signal*** *can be represented as a real matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$, where N is the number of nodes and M the number of observations in time.*

2.3 Tree graphs

Definition 2.14. *A graph $\mathcal{G}(V, E)$ is a **tree** if it is an undirected graph in which any two vertices are connected by exactly one path. Equivalently, a tree is a connected acyclic undirected graph [12].*

The number of edges in a tree graph is $|E| = |V| - 1$.

2.3.1 Maximum spanning tree

Definition 2.15. *Given a weighted undirected graph $\mathcal{G}(V, E)$, a **spanning tree** $\mathcal{T}$ is a subgraph of $\mathcal{G}$ that forms a tree and includes all of the vertices of $\mathcal{G}$.*

A maximum spanning tree is a spanning tree with the maximum total edge weight. Note that for a given graph there may be several spanning trees but, in general, there is a unique maximum spanning tree. There exist different algorithms to find the maximum spanning tree. Among them, the most popular ones are Prim's algorithm [13] and Kruskal's algorithm [14], presented next.

2.3.1.1 Prim's algorithm

Prim's algorithm can be used to find the maximum spanning tree of a graph. It can be described as performing the following steps:

1. Initialize a tree with any single node from the graph.
2. Grow the tree by one edge by finding, among the edges that connect the tree to vertices not yet in the tree, the maximum-weight edge and adding it to the tree.
3. Repeat step 2 until all vertices are in the tree.

Prim's algorithm has $\mathcal{O}(|V|^2)$ time complexity, with $|V|$ being the number of vertices or nodes in the graph.

2.3.1.2 Kruskal's algorithm

Kruskal's algorithm is an alternative method that can be used to find the maximum spanning tree of a graph. The steps are:

1. Sort all edges in decreasing order of their edge weights.
2. Pick the maximum-weight edge and check if it creates a cycle in the maximum spanning tree. If it does not form a cycle, then include that edge in maximum spanning tree. Otherwise, discard it.
3. Repeat step 2 until the spanning tree includes $|V| - 1$ edges.

Kruskal's algorithm time complexity is $\mathcal{O}(|E| \log |V|)$, with $|E|$ the number of edges and $|V|$ the number of vertices or nodes in the graph. Note that, for dense graphs, Prim's algorithm is faster than Kruskal's. Conversely, Kruskal's algorithm is faster for sparse graphs, where the number of edges $|E|$ is smaller.

Chapter 3

Problem definition

A *mobile* sensor network is a sensor network in which the sensor nodes capturing the data are mobile, that is, they do not remain in a fixed location. For instance, sensor nodes installed on vehicles such as buses, trains or taxis can be part of a mobile sensor network. Over the last few years, mobile sensor networks are attracting a lot of interest in the research community due to advances in hardware design, communication protocols and resource efficiency [15]. Economics, environmental or seismic monitoring, healthcare applications and inventory tracking are some of the main applications of mobile sensor networks [16].

The research community is becoming interested in exploiting mobility in a wireless environment, rather than considering it as a sort of disturbance. In order to take advantage of mobility, three main types of mobility have been defined: random, controlled and predictable. In the random mobility category, mobile sensors are supposed to move according to a random mobility pattern. A drone that moves over space following a random pattern is an example of this mobility type. Random mobility can be challenging to work with and often can be seen as a source of problems rather than providing advantages over forms of mobility. Controlled mobility consists of mobile sensors introduced in the sensing network and moving to specific destinations with defined mobility patterns for specific objectives. For instance, taxis can be an example of controlled mobility, since they move to specific destinations but the paths they follow are not necessarily always the same. Lastly, a sensor node mounted on a means of public transportation that moves with a periodic schedule represents a case of predictable mobility. Trains or buses are

examples of predictable mobility, specially if they are on time, because then it is much easier to predict their position at any given time. Predictable mobility makes it easier to program and accomplish specific desired target tasks [17].

Mobile sensor networks are much more versatile than static sensor networks as the sensor nodes can be deployed in any scenario and quickly provide observations in different locations [18]. Moreover, the coverage area of the sensors can be significantly larger in a mobile sensor network compared to a fixed one. Consequently, if the goal is to cover a certain area, a mobile sensor network will require fewer sensor nodes to be deployed than a static one. Given two systems with the same number of sensors, a mobile sensor network will generally provide a much better coverage in space, i.e., a better spatial resolution, than a conventional fixed sensor network.

The main disadvantage of mobile sensor networks is that simultaneous measurements at different locations are unlikely. Instead, the data gathered by a mobile sensor network usually contains observations that are not simultaneous and are not evenly spaced over time at different spatial locations. As an example, let us consider a mobile sensor network deployed in an urban environment whose purpose is air quality monitoring. As stated above, it is expected to cover all the urban area with a few sensor nodes, but one of the issues that may come up is that at any given time, each sensor will be located in a certain position, so that the sensors will not cover all the target area at that time. Thus, it will be necessary to wait until additional measurements are made in order to achieve the desired coverage. This leads to a low spatio-temporal resolution, that is, the sensor network does not provide a high temporal resolution for every location in the urban area. A high spatio-temporal resolution is critical for real-time sensing and monitoring applications. This issue gives rise to the problem we address in this thesis, namely, developing a graph-based model to increase spatio-temporal resolution using sample interpolation.

3.1 Air quality dataset

This section describes the air quality dataset that will be used to train the graph based model and evaluate the performance of the proposed solution. The dataset, produced by the OpenSense project, contains mobile air quality measurements for the city of Zurich [19]. Mobile sensor boxes are installed on top of trams across the city, in order to provide high spatial resolution, and measure ozone (O_3) and carbon monoxide (CO) levels [20]. Even though this is a specific dataset, it is important to note that it is representative of data being captured on rail systems. Thus, the work in this thesis aims to provide a generalized solution for this type of data, rather than limiting the proposed methods to the field of air quality. Table 3.1 includes a few samples of collected data.

Table 3.1: Example of raw data captured by ozone sensors.

timestamp (ms)	ozone (ppb)	temp. (^{o}C)	humidity (%)	latitude	longitude
1318583686494	14.145374	10	64	4724.22833	832.88198
1318583746777	14.523555	10	64	4724.01861	832.61155
1318583766726	14.603173	10	66	4724.01839	832.61003
1318583826330	14.742503	10	66	4723.85901	832.50465
1318583846355	14.603173	10	66	4723.85936	832.50535

Each sample in Table 3.1 includes the following data:

- **Timestamp:** timestamp in milliseconds in the Unix timestamp format. This count starts at the Unix Epoch on January 1st, 1970 at UTC. Note that data observations are ordered by increasing timestamp and that measurements are not evenly spaced over time.
- **Ozone ppb:** level of ozone (O_3) measured in ppb (parts per billion).
- **Temperature:** temperature in degrees Celsius (^{o}C).
- **Humidity:** percentage of humidity.

- **Latitude:** latitude in the WGS84 format *ddmm.sssss*, which would be equivalent to $dd^\circ\ mm'\ (60 \cdot 0.sssss)''$. As an example, the latitude for the first observation in Table 3.1 would be $47^\circ\ 24'\ 13.6998''$ (note that the decimal part of the latitude needs to be multiplied by 60 to get the number of seconds).

- **Longitude:** longitude in the WGS84 format *dmm.sssss*, which would be equivalent to $d^\circ\ mm'\ (60 \cdot 0.sssss)''$. As an example, the longitude for the first observation in Table 3.1 would be $8^\circ\ 32'\ 52.918''$.

A common aspect of most datasets providing obtained from mobile sensor networks is that they measure a certain quantity at a given position in space and time. That is, time and position information are captured for each sensor observation. The particular dataset used in this thesis collects ozone ppb data and contains approximately 200 000 observations taken over the course of three months.

Chapter 4

Proposed solution

This chapter presents and explains in detail the proposed solution. It also explains the data pre-processing required to obtain the desired input for the solution, as well as some possible optimizations to improve its time complexity. The proposed solution can be divided into two parts. The first part is related to the graph construction or graph learning method, while the second one is about the interpolation method used once the graph is constructed.

4.1 Data pre-processing

This section intends to go through the data pre-processing needed in the particular case of the air quality dataset, but it also serves as an example to describe the input required by the solution that we propose. Note that, for the specific case of air quality sensors, it may be possible to improve data interpolation by considering related environmental factors, such as temperature or humidity. However, our goal is to study the general problem of increasing spatio-temporal resolution of sensed data, including cases where other environmental variables may not be relevant. For this reason we only consider interpolation in time and space of the measured variable (ozone concentration in this particular case). Only this information will be used to construct the graph and interpolate the data.

4.1.1 Timestamp reference

As can be seen in Table 3.1, timestamps are such a large number, since they represent the amount of milliseconds from January 1st, 1970 UTC. Let us say that a given timestamp t can be expressed as $t = t_0 + t'$, where t_0 is the smallest timestamp in the dataset, i.e., the reference timestamp. Therefore, if all timestamps are recomputed such that the new timestamp is now t', we get rid of the offset t_0 that is present in every single timestamp and it does not provide any useful information once the reference timestamp is known. Moreover, all timestamps will be converted from milliseconds to seconds for convenience. The high temporal resolution provided by the sensors is not useful given that measurements are not evenly spaced over time and the average time between samples is close to 40 seconds.

4.1.2 Coordinates to distance conversion

The geographic coordinates system is very useful to represent any point on Earth's surface as accurately as the measurement technique allows, but when it comes to computing distances between points, some complex and time-consuming arithmetic calculations are encountered. This is the main reason why it is convenient to transform the positions expressed as latitude and longitude to a projected coordinate system on a Cartesian plane. In order to do so, a rectangular space will be defined such that it is the smallest rectangle containing all the observations available. Then, a reference point $(0, 0)$ is set as the observation with the lowest latitude and longitude in the data. From this, all coordinates in the plane representing an observation can be transformed to Cartesian coordinates (x, y) using the Haversine formula, which determines the great-circle distance between two points on a sphere given their longitudes and latitudes:

$$d = 2R \cdot \sin^{-1}\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)\cos(\phi_1)\cos(\phi_2)}\right), \tag{4.1}$$

where R is the Earth's radius, ϕ_1, ϕ_2 are the latitudes and λ_1, λ_2 the longitudes of points 1 and 2 respectively, and d is the distance between these points.

This translates to an additional two columns in the data, which will be called x and y and will represent the location where the measurement is taken in Cartesian coordinates with respect to the reference point $(\phi_{min}, \lambda_{min})$. Moreover, to make the understanding of latitudes and longitudes easier, they will be converted from the degrees, minutes and seconds format to decimal degrees. The table below shows the transformations applied to the data in table 3.1:.

Table 4.1: Example of pre-processed data transformation.

timestamp (s)	ozone (ppb)	latitude (º)	longitude (º)	x (m)	y (m)
0	14.1454	47.4038	8.54803	4329.23	5949.06
60	14.5236	47.4003	8.54353	3989.22	5560.01
80	14.6032	47.4003	8.54350	3987.33	5559.63
140	14.7425	47.3976	8.54174	3854.84	5263.90
160	14.6032	47.3977	8.54176	3855.85	5264.64

4.2 Graph learning

The way to obtain a graph from data is a fundamental problem in graph signal processing known as graph learning or graph construction [21]. There exist many graph learning techniques, but none of them suits the type of data collected by a mobile sensor network. As a first step, it can be interesting to observe the data distribution across space. The data spatial distribution for the air quality dataset can be observed in Figure 4.1, where each dot represents an observation. Note that a similar spatial distribution of the data can be expected in other mobile sensor networks that are part of a predictable mobility system. In other words, when the sensor goes from point A to point B and the path is mostly determined, measurements along the path will appear forming a line. This is not the case for random mobility, because there is no such thing as origin or destination. Therefore,

in predictable mobility environments (e.g., buses or trains), it might be interesting to construct a tree graph from the data, since it seems an appropriate representation given the spatial distribution observed.

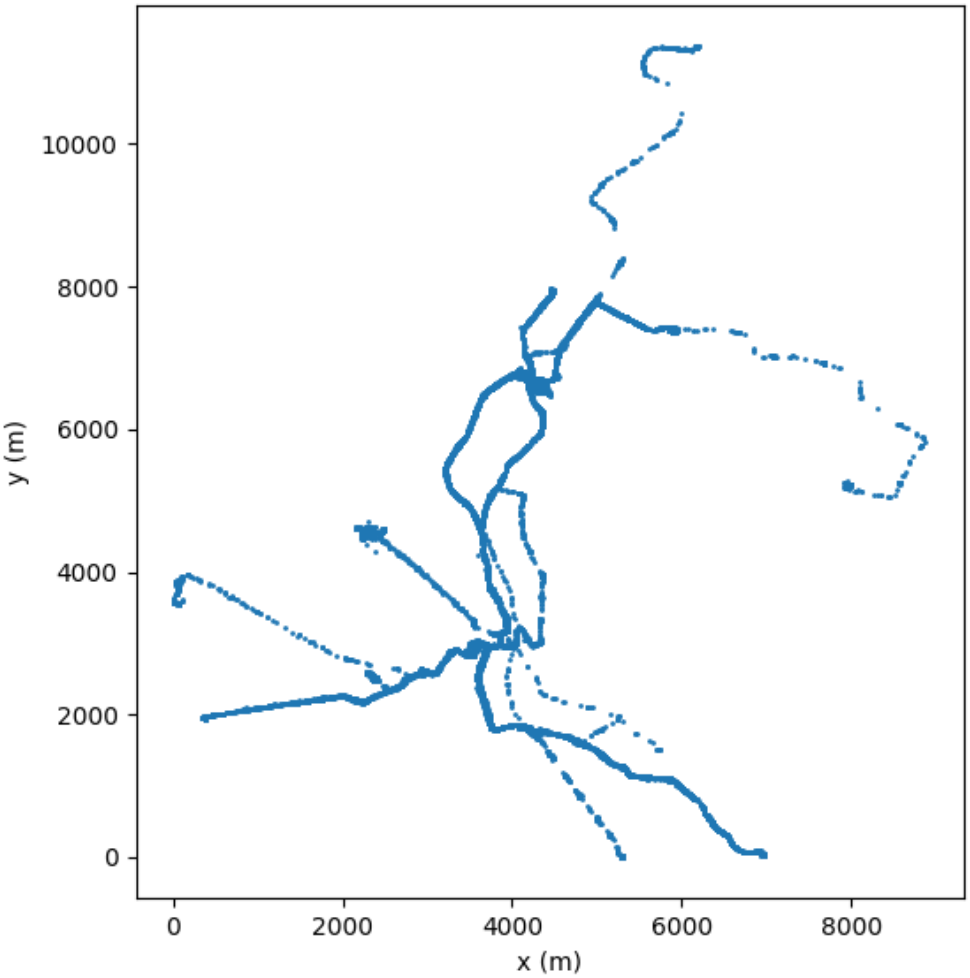


Figure 4.1: Spatial distribution of the air quality data.

Graph learning usually involves two stages: topology inference and weight estimation. Topology inference aims to identify which pairs of nodes should be connected, while weight estimation aims to assign weight values to the edges of a graph given its topology.

4.2.1 Topology inference

In some scenarios, graphs with certain structures are required, such as trees [22] or bipartite graphs [23], and solving the optimal topology can involve solving NP-hard com-

binatorial problems. We will focus on the case where a tree graph is desire, as motivated above, but before diving into weight estimation it is necessary to define more precisely what each graph node represents. Clearly, each node represents a certain location or position in space, but given the sparse and irregular nature of the measurements this has to be defined more carefully.

Some research in graph signal processing has shown that the use of the Voronoi cell area of vertices leads to a more sensible definition of graph signal energy even when sampling is highly irregular [24]. This highlights the fact that we can consider that graph nodes represent an *area*, and not just a specific point. Thus, we propose to divide the rectangular area where all measurements are located into a grid formed by squares, where each square in the rectangular grid will be represented by a node. This way, each observation within the data set can be mapped to a node, namely, the node corresponding to the square region the measurement belongs to. Since there is no ideal way to define the grid partition, a parameter Δs ("delta space") is introduced. From now on, Δs will represent the distance, in meters, of the side of each square in the grid. As a clarifying example: if the lower left corner of the rectangular area containing all observations corresponds to the coordinates $(0, 0)$, and the upper right corner to the coordinates (x_{max}, y_{max}), the grid will contain $\left\lceil \frac{x_{max}}{\Delta s} \right\rceil \times \left\lceil \frac{y_{max}}{\Delta s} \right\rceil$ squares/nodes.

4.2.2 Weight estimation

Research on weight estimation has been focused on, given the empirical covariance matrix

$$\mathbf{S} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^\top$$

from N data samples $\mathbf{x}_i$, estimating the combinatorial graph Laplacian (CGL) as formulated in [25]:

$$\min_{\mathbf{L} \in \mathbb{L}(\mathcal{E})} -\log |\mathbf{L}|_\dagger + \operatorname{trace}(\mathbf{L}\mathbf{S}) + \alpha \|\mathbf{L}\|_{1,\text{off}}, \tag{4.2}$$

where $\mathbb{L}(\mathcal{E})$ is the set of CGLs with edge set $\mathcal{E}$, and $\|\mathbf{L}\|_{1,\text{off}}$ is the absolute sum of all off-diagonal elements of $\mathbf{L}$. Note that the pseudodeterminant (product of nonzero eigenvalues) $|\mathbf{L}|_\dagger$ is required here because CGL matrices are singular. (4.2) describes a MAP (maximum a posteriori) parameter estimation of a Gaussian Markov Random Field (GMRF), $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma} = \mathbf{L}^\dagger)$ [9].

In this thesis, we take as a starting point the work in [9], where a closed form solution in terms of the empirical covariance matrix of the problem formulated in (4.2) is given, for the case where the target graph topology is known and it does not contain any cycles, i.e., it is a tree. Given $\mathbf{S}$ and defining

$$\mathbf{K} = \mathbf{S} + \alpha(\mathbf{I} - \mathbf{1}\mathbf{1}^\top)$$

the resulting weights can be found as follows:

$$w_{r,s} = \left[(\mathbf{e}_r - \mathbf{e}_s)^\top \mathbf{K} (\mathbf{e}_r - \mathbf{e}_s)\right]^{-1} = (k_{r,r} + k_{s,s} - 2k_{r,s})^{-1}, \tag{4.3}$$

or

$$w_{r,s} = \left[\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i(r) - \mathbf{x}_i(s))^2 + 2\alpha\right]^{-1}, \text{ for } (r,s) \in \mathcal{E}, \tag{4.4}$$

where $\mathbf{e}_i$ is the i-th vector of the standard basis, that is, $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0,\ j \neq i$. Note that $\alpha > 0$ is necessary in order to have positive non-infinite weights and, defining

$$\delta_{r,s} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i(r) - \mathbf{x}_i(s))^2,$$

we can see that $\delta_{r,s}$ is the mean-square-difference between the r-th and the s-th elements over all realizations $\mathbf{x}_i$.

To obtain the tree topology we can run a maximum weight spanning tree algorithm on the complete graph with weights $w_{r,s}$ given by (4.3). Thus, the procedure to obtain it consists on the following two steps:

1. Compute the weights using (4.3) or (4.4) for all $(r, s) \in V \times V$, and construct a complete graph with those weights.

2. Apply a maximum weight spanning tree algorithm, such as Kruskal's algorithm [14] or Prim's algorithm [13], to the complete graph.

Note that the closed expressions to compute weights described in (4.3) and (4.4) correspond to an ideal case where each pair of nodes s and t have the same number of observations (N), which will not be the case in most mobile wireless sensor networks. What will actually happen is that some nodes will have more observations than others and these observations will not be simultaneous. In other words, since the weight expression from (4.4) pairs realizations and compares their values, for the estimates to be accurate it will be desirable that those realizations that are being compared be taken either simultaneously or within a small interval of time. Otherwise, the weights may be inaccurate, leading to a graph with no meaningful information, since the differences in (4.4) will correspond to different times and the spatial correlation that leads to estimates in (4.4) will be affected by the lack of temporal consistency (observations at different times). Note that this issue arises because the method in [9] considered simultaneous observations in time.

To take into account the different times at which samples are captured, we need to define time intervals such that all realizations that go into one of the squared difference term in (4.4) belong to the same time interval. Effectively, this means that we need to define a temporal resolution, defined by a time interval Δt ("delta time"). Consequently, when classifying the data to create the graph, each observation will be assigned to the corresponding node according to the position where it is taken, and will also belong to the interval $\left\lfloor \frac{t}{\Delta t} \right\rfloor$, where t is the timestamp at which the observation is taken. The number of temporal intervals considered will be $T = \left\lceil \frac{t_{max}}{\Delta t} \right\rceil$, with t_{max} being the maximum timestamp observed in the data. Then, when all the realizations are classified to their corresponding nodes and temporal intervals, the weights can be estimated using the ap-

proach proposed in [9], but using in (4.4) all those pairs of observations that belong to the same temporal interval. Thus, a modified expression to compute weights will be given as:

$$w_{r,s} = \left[\frac{1}{N} \sum_{t=1}^{T} \sum_{i=1}^{N_t} (\mathbf{x}_{t,i}(r) - \mathbf{x}_{t,i}(s))^2 + 2\alpha \right]^{-1}, \text{ for } (r,s) \in \mathcal{E} \tag{4.5}$$

where T is the number of time intervals, N_t is the number of realizations being compared at time interval t, $\mathbf{x}_{t,i}(r)$ is the i-th realization from node r at time interval t and $N = \sum_{t=1}^{T} N_t$, the total number of pairs of realizations compared along all the time intervals. In other words, (4.5) computes squared differences of realizations within the same time interval and uses all time intervals to estimate the weight between two nodes.

Nevertheless, some nodes will still have more realizations than others at a given time interval, that is, $N_{t,r}$ (number of realizations at time interval t and node r) does not have to be the equal to $N_{t,s}$ (number of realizations at time interval t and node s). This leads to the need of addressing this issue, since every node requires to have the same number of realizations as the node with which is being compared at every time interval [9]. To do so, we consider two approaches: (i) completing the covariance matrix as if there were missing observations ($N_t = \max(N_{t,r}, N_{t,s})$) and (ii) computing all the possible combinations of squared differences ($N_t = N_{t,r} \times N_{t,s}$).

4.2.2.1 Computation of every combination of squared differences

The computation of every combination of squared differences is based on the interpretation of (4.5), which leads to computing squared differences corresponding to all pairs of nodes. For a given choice of temporal resolution, Δt, squared differences are only computed within observations in the same interval. Thus, if during a certain time interval t a pair of nodes r, s have different number of realizations $N_{t,r} \neq N_{t,s}$, the proposed method consists of using all the $(N_{t,r} \times N_{t,s})$ possible squared differences to estimate the weight. Thus, (4.5) can be rewritten as:

$$w_{r,s} = \left[\frac{1}{N} \sum_{t=1}^{T} \sum_{i=1}^{N_{t,r}} \sum_{j=1}^{N_{t,s}} (\mathbf{x}_{t,i}(r) - \mathbf{x}_{t,j}(s))^2 + 2\alpha \right]^{-1}, \quad \text{for } (r,s) \in \mathcal{E} \tag{4.6}$$

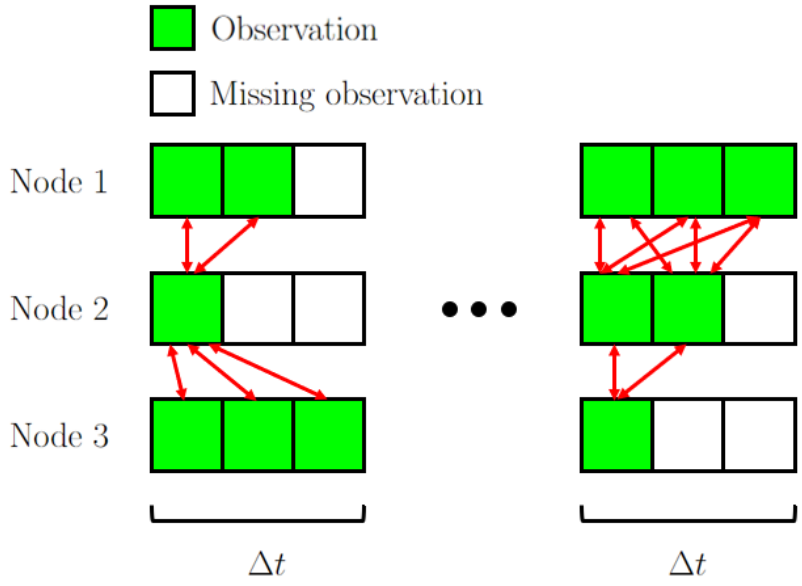


Figure 4.2: Illustrative example of every possible combination of squared differences.

The diagram in Figure 4.2 shows an example of the comparisons that are being made when using this sample removal/ignoring approach according to (4.6). Red arrows represent the realizations that are being compared within the same time interval when graph nodes have different number of observations. Note that, strictly speaking, all weights between each pair of nodes need to be computed, but to keep the figure simple the comparisons needed to compute the weights between nodes 1 and 3 are not shown in Figure 4.2. From this example, it is clear that this approach takes into consideration the maximum possible number of samples at each weight computation between any pair of nodes. This fact makes it flexible and it can be adapted to any data distribution.

4.2.2.2 Covariance matrix estimation with missing observations

A method to perform a covariance matrix estimation with missing observations is proposed in [26]. Assume that the complete data set is represented by the $n \times N$ matrix $\mathbf{X}$, with i-th

row and k-th column given by the vectors $\mathbf{x}_i$ and $\mathbf{x}^{(k)}$ respectively. The $n \times N$ matrices $\mathbf{Y}$ and $\mathbf{\Delta}$ denote the observation and missing data patterns respectively. $\mathbf{\Delta}$ has entries in $\{0, 1\}$. The k-th columns of $\mathbf{Y}$ and $\mathbf{\Delta}$ are denoted by the vectors $\mathbf{y}^{(k)}$ and $\delta^{(k)}$ respectively. These quantities are related through:

$$\mathbf{y}^{(k)} = \delta^{(k)} \odot \mathbf{x}^{(k)}$$

where $\odot$ is the entry-wise product (also known as Hadamard or Schur product) and is defined by $(\mathbf{A} \odot \mathbf{B})_{ij} = a_{ij} b_{ij}$. When the j-th entry of $\delta^{(k)}$ is equal to 0, the corresponding entry of $\mathbf{x}^{(k)}$ is missing. Considering a realistic scenario in which the observation probabilities are unknown and have to be estimated from data, the matrix of empirical observation probability is defined as

$$\widehat{\mathbf{P}} = \frac{1}{N} \sum_{k=1}^{N} \delta^{(k)} \delta^{(k)\top}, \tag{4.7}$$

and its entries are denoted by $\hat{p}_{ij}$. The quantity $\hat{p}_{ij}$ is equal to zero when the i-th and j-th variables are not observed together. Since we can only estimate the entries of the covariance matrix for which there are observations, we define the set $\mathcal{E} = \{(i, j) : \hat{p}_{ij} \neq 0\}$. Let $\Sigma_{\mathcal{E}}$ be a matrix whose entries in $\mathcal{E}$ coincide with the population covariance, and are zero otherwise. Let $\widehat{\mathbf{\Gamma}} = (\hat{\gamma}_{ij})$ be the Hadamard inverse of $\max(\widehat{\mathbf{\Gamma}}, \frac{1}{N}\mathbf{1}\mathbf{1}^\top)$, which satisfies $\hat{\gamma}_{ij}\hat{p}_{ij} = 1$ if $(i, j) \in \mathcal{E}$, and $\hat{\gamma}_{ij}\hat{p}_{ij} = 0$ otherwise. An unbiased estimator for $\Sigma_{\mathcal{E}}$ is given by

$$\widehat{\mathbf{\Sigma}} = \frac{1}{N} \sum_{k=1}^{N} \mathbf{y}^{(k)} \mathbf{y}^{(k)\top} \odot \widehat{\mathbf{\Gamma}}. \tag{4.8}$$

Even though this approach is very appropriate from the theoretical point of view, there is a major drawback that comes up when using it. This method fails to adapt to situations that occur frequently in our problem and in particular to the case the number of samples observed during a given time interval is very different from node to node.

An illustrative example of this issue is shown in Figure 4.3, where the node with the

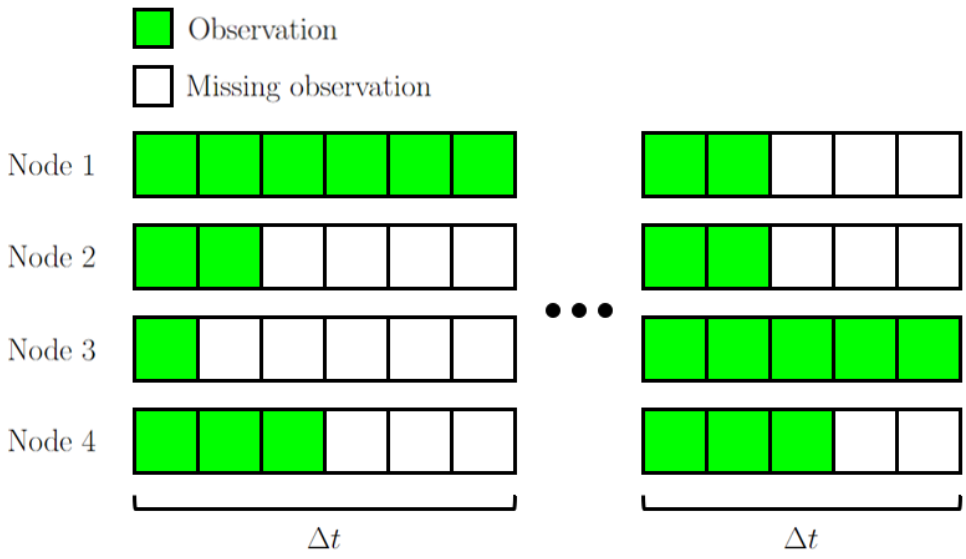


Figure 4.3: Illustrative example of missing observations approach.

highest number of observations in each time interval determines the number of missing observations in the other nodes. In this example there are only four nodes which means that, if we considered the missing observations approach explained, 45% of the samples would be considered to be missing. If this is extended to a more realistic scenario, where there are hundreds of nodes, it is easy to see that the vast majority of samples will be considered as missing and estimated data will outnumber actual observations. Intuitively this makes this approach not very desirable. Indeed, an experimental comparison between the computation of all possible squared differences method and the covariance matrix estimation method showed that the former outperforms the latter. Therefore, from now on, the method used to solve the issue of weight computing will be the one described in Section 4.2.2.1.

4.2.3 Graph learning algorithm

Up to this point we have introduced the topology inference and weight estimation used for graph construction. The complete graph learning algorithm using the concepts explained in the previous sections is summarized in Algorithm 1.

Algorithm 1 Tree graph construction algorithm

Input:

data: observations in the dataset
Δt: delta time parameter
Δs: delta space parameter
α: alpha parameter (weight computation)

Output:

A: tree graph adjacency matrix
W: complete graph adjacency matrix

procedure GRAPHLEARNING
  $T \leftarrow \left\lfloor \frac{t_{max}}{\Delta t} + 1 \right\rfloor$, $M \leftarrow \left\lceil \frac{x_{max}}{\Delta s} \right\rceil$, $N \leftarrow \left\lceil \frac{y_{max}}{\Delta s} \right\rceil$
  for each $obs \in$ data **do**
    $t \leftarrow \frac{obs_t}{\Delta t}$, $m \leftarrow \frac{obs_x}{\Delta s}$, $n \leftarrow \frac{obs_y}{\Delta s}$
    graph_data$[N \cdot m + n][t]$.add(obs)
  for each $v_1 \in \{1, \ldots, M \cdot N\}$ **do**
    for each $v_2 \in \{v_1 + 1, \ldots, M \cdot N\}$ **do**
      $w_{v_1,v_2} \leftarrow 0$
      $k \leftarrow 0$
      for each $t \in \{1, \ldots, T\}$ **do**
        for each $i \in \{1, \ldots,$length(graph_data$[v_1][t])\}$ **do**
          for each $j \in \{1, \ldots,$length(graph_data$[v_2][t])\}$ **do**
            $w_{v_1,v_2} \leftarrow w_{v_1,v_2} +$ (graph_data$[v_1][t][i] -$ graph_data$[v_2][t][j])^2$
            $k \leftarrow k + 1$
      if $k \neq 0$ **then**
        $w_{v_1,v_2} \leftarrow \left(\frac{w_{v_1,v_2}}{k} + 2\alpha\right)^{-1}$
  A $\leftarrow$ maximum_spanning_tree(**W**)

From Algorithm 1, note that the maximum spanning tree algorithm used to obtain **A**, the adjacency matrix of the learned tree graph, is not specified. Considering that **W**, the adjacency matrix with the weights for each pair of nodes corresponds to a complete graph, using Prim's algorithm to find the maximum spanning tree is faster, as explained in Chapter 2. The main reason is that, for dense graphs, Prim's algorithm is faster than Kruskal's algorithm. If the initial graph was sparse, then Kruskal's algorithm would be faster.

4.3 Interpolation

We now describe the method used to interpolate samples at desired target locations once the tree graph has been constructed. The interpolation schemes specifically for tree graphs are much simpler than those for more general graph structured data [4][27]. The main reason for this is that the graph structure is simpler, and the number of edges is the minimum possible to connect all the graph vertices: $|E| = |V| - 1$.

In order to interpolate a sample, we first need to know the location and time corresponding to the sample to be predicted. For the given location representing a point in space, we first find the square on the grid that contains the point and the resulting square corresponds to the graph node to which that sample is assigned. This graph node will be referred to as v_k, while the time interval is denoted t_k. The interpolation scheme will use observations from the sensor network captured within the same time interval and in different locations and the tree graph that needs to be previously constructed as explained in Section 4.2. Given $\mathbf{O}_{v_i} = \{o_{v_{i1}}, \ldots, o_{v_{iN_i}}\}$, the realizations observed in the node v_i and within the time interval t_k, and $\mathbf{A}$ the adjacency matrix of the tree graph model with entries a_{ij}, the interpolated value x_k at the node v_k in the time interval t_k will be computed as follows:

$$x_k = \frac{\sum_{i=1}^{M} \sum_{j=1}^{N_i} a_{v_k,v_i} \cdot o_{v_{ij}}}{\sum_{i=1}^{M} a_{v_k,v_i} \cdot N_i} \tag{4.9}$$

where $\{v_1, \ldots, v_M\} = \{v_i \mid a_{ik} \neq 0\}$, that is, all the 1-step neighbors of v_k. Note that (4.9) produces the interpolated value as the weighted mean of the neighbor nodes of v_k. It might happen that there are no observations available for any of the neighbor nodes of v_k in the time interval t_k. In that case, the 2-step neighbors will be considered next. If there are no available samples, the step will be increased by 1 unit until neighboring samples are found and the interpolation can be performed.

The overall edge weight considered in the case when interpolation comes from an n-step neighbor is the inverse of the sum of the reciprocals. This can be seen by considering

the case of a path graph with nodes $1, 2, 3, ..., n$, with corresponding edge weights are $a_{12}, a_{23}, \ldots a_{(n-1)n}$. Then, noting that the edge weights correspond to similarity (larger weight means greater similarity), so that greater similarity indicates shorter distance, we can see that the similarity of the end nodes along the path is the inverse of the sum of the reciprocals. This can be written as:

$$\frac{1}{a_{1n}} = \frac{1}{a_{12}} + \frac{1}{a_{23}} + \ldots + \frac{1}{a_{(n-1)n}}, \tag{4.10}$$

where we can see that the distances along the path are additive. The expression in (4.10) is equivalent to that the of the sum of resistances in an electric circuit, where similarity corresponds to conductance, while inverse similarity corresponds to resistance.

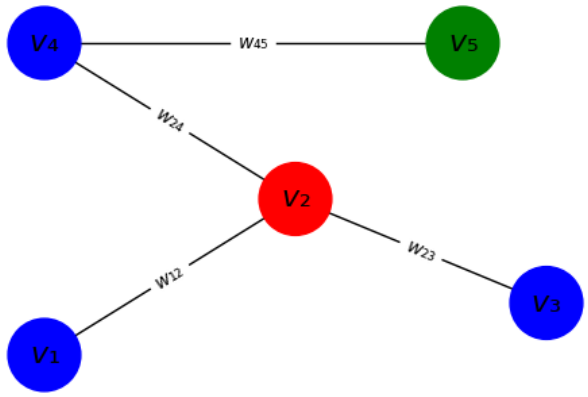


Figure 4.4: Simple graph used to exemplify the tree interpolation scheme.

Figure 4.4 shows a simple tree graph as an example of the interpolation that is performed. Suppose that we want to interpolate a sample in the node v_2 at the time interval t_1. To do so, observations from nodes v_1, v_3 and v_4 will be used, and weighted according to the respective edge weights w_{12}, w_{23} and w_{24} (the auxiliar observations from these nodes must belong to the same time interval t_1). If there are no any observations in any of these nodes, then we would use the observations from node v_5, with weight w_{25} defined by $\frac{1}{w_{25}} = \frac{1}{w_{24}} + \frac{1}{w_{45}}$, the inverse of the sum of the reciprocals of the traversed edges to go from v_2 to v_5.

It is important to note that this tree based interpolation method has a constraint, which is that interpolation of samples can only be performed in locations that the tree graph model constructed represents with its nodes. Additional areas can only be incorporated into the model if the mobile sensors visit those areas at some point.

4.4 Solution optimization

The graph learning method and interpolation scheme have been presented in the previous sections. In this section we present optimizations that can reduce the computational complexity, especially that of the graph learning part, which is the most computationally expensive. Optimizing the solution is important to make its implementation efficient and it can be very helpful to tune the tree model hyper-parameters.

4.4.1 Temporal complexity analysis

The temporal complexity of the algorithm is given by the following expression:

$$\Theta(|O| + |V|^2 \cdot T + |V|^2), \tag{4.11}$$

where $|O|$ is the number of observations, $|V|$ the number of nodes and T the number of time intervals defined. The first term in (4.11) corresponds to the cost of classifying all the data, which has a complexity of $\Theta(|O|)$, proportional to the number of observations in the dataset. The second term in (4.11) is the weight estimation complexity, $\Theta(|V|^2 \cdot T)$, because for each pair of vertices v_1, v_2, the weight of the edge that connects them is computed iterating all the time intervals $t_i, i \in \{1, \ldots, T\}$. Lastly, the third term in (4.11) corresponds to the maximum spanning tree implementation using Prim's algorithm, whose temporal cost is $\mathcal{O}(|V|^2)$. Note that since the time complexity can vary depending on the scenario, we provide an upper bound that reflects the complexity of the worst case scenario.

It is important to note that the magnitude $|V|$ is quite large if divide a rectangular region into a grid and assign each squared cell area to a node. The number of nodes will be

$$|V| = \left\lceil \frac{x_{max}}{\Delta s} \right\rceil \cdot \left\lceil \frac{y_{max}}{\Delta s} \right\rceil,$$

where Δs is the graph learning parameter defined in Section 4.2.1 and (x_{max}, y_{max}) are the coordinates of the right upper corner of the rectangular area, the point that is the most distant from the origin or reference point. As an example, consider the particular case of the air quality dataset gathered in the urban area of Zurich. For this case, we have that $x_{max} = 9\,000m$, $y_{max} = 12\,000m$. Considering a $\Delta s = 100m$, that is, each node covers a squared area of $100m \times 100m$ (a reasonable choice according to [28]), so that the number of nodes will be $|V| = 10\,800$ nodes. We can see that the number of nodes can increase rapidly in scenarios where the coverage area is large or the desired spatial resolution is high (small Δs).

4.4.2 Removal of candidate nodes without samples

From the temporal complexity analysis we learn that the factor that affects temporal complexity the most is the number of nodes. Therefore, if we can reduce the number of considered nodes, the overall complexity will be significantly improved according to Amdahl's law [29].

A first idea to reduce complexity comes from considering the spatial distribution of the data. The coverage area is defined as a rectangular area for simplicity and convenience but in practice some areas within the rectangular region will not be covered by mobile sensors. Thus, nodes (square areas) without any realizations can be removed from the node list. To account for that we will differentiate between candidate nodes and the actual nodes that contain measurements. Candidate nodes are named like this due to the possibility of being nodes of the tree graph model obtained from the graph learning method. But it

is not realistic to think that all candidate nodes will be part of the final graph, especially in a predictable or mobility environment.

As an example, in the Zurich air quality dataset, with $x_{max} = 9\,000m$, $y_{max} = 12\,000m$ and $\Delta s = 100m$, there are 10 800 candidate nodes. If all the realizations contained in the dataset are classified to their corresponding nodes, we can see how the data is distributed over the candidate nodes and how many nodes actually contain data. Figure 4.5 shows the node distribution and the frequency of observations in the described example and demonstrates that the actual nodes containing observations are a small subset of the candidate nodes. In this particular example, only 557 nodes contain samples, representing a 5.16% of the candidate nodes. Therefore, dropping all the nodes without realizations beforehand will contribute to a much faster weight computing, data processing and graph learning.

4.4.3 Maximum edge distance

Temporal complexity analysis from Section 4.4.1 shows that the most computationally expensive part of the algorithm is the weight estimation, with a cost of $\Theta(|V|^2 \cdot T)$. In Figure 4.6 we show the graph constructed with our algorithm for the Zurich air quality dataset with the following tree model hyper-parameters: $\Delta t = 30min$, $\Delta s = 150m$, $\alpha = 0.1$. Note that the weights are not displayed for convenience, but the graph is still an undirected weighted graph. Also note that the value of α is not relevant, since it only affects the value of the weights according to (4.3) and (4.4), but does not affect the order in which edges are selected to find the maximum spanning tree.

An important observation from Figure 4.6, is that there aren't any edges connecting nodes that are very far from each other. Since distant nodes are not connected, an optimization of the graph learning algorithm can be introduced, which consists in omitting the weights estimation for those pairs of nodes that are further than a certain threshold. This introduces a new parameter to the graph learning model, but it can potentially

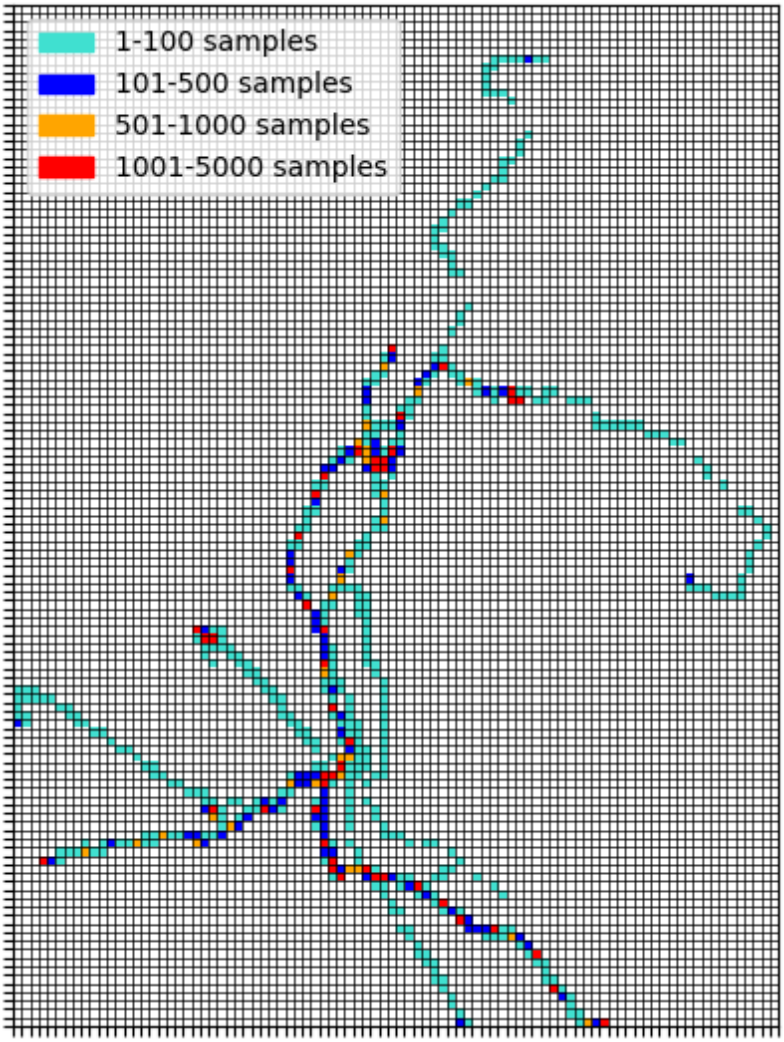


Figure 4.5: Node distribution and frequency of observations with $\Delta s = 100m$.

reduce the computational cost of weights estimation significantly, specially in cases where the area of coverage is very large or there are a lot of nodes. This new parameter will be called "maximum edge distance". As the name suggests, it represents the maximum distance considered between two nodes before estimating the weight of the edge that connects them. In other words, if two nodes are separated by more than the maximum edge distance, no edge is considered to be connecting these two nodes. This can reduce the execution time of the training algorithm significantly, because the weights estimation method needs to compute the weight between each pair of nodes in the graph. If the graph has $|V|$ nodes, then $|E| = \frac{|V|(|V|-1)}{2}$ is the number of edges of the complete graph.

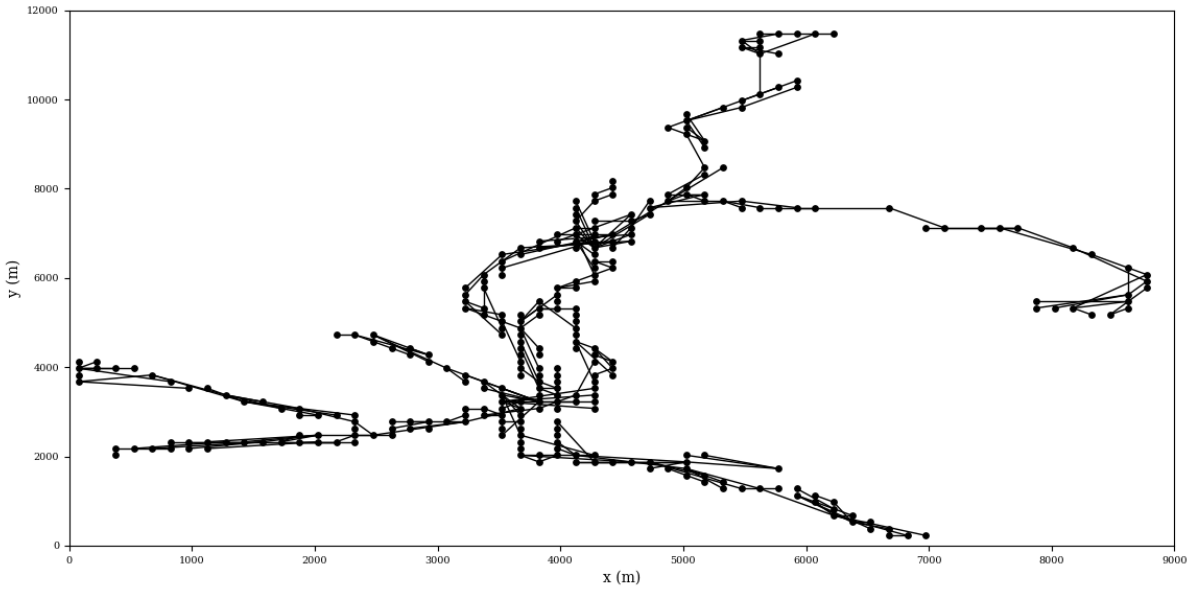


Figure 4.6: Example of tree graph obtained with $\Delta t = 30min$, $\Delta s = 150m$, $\alpha = 0.1$.

Reducing the number of candidate edges also represents a considerable improvement in memory and time when finding the maximum spanning tree.

A possible drawback of using a maximum edge distance is that a good choice is likely to be application dependent. The above example only shows that this approach works well for an air quality application, but we do not know if the same would happen with other types of data. Thus, in general, the value of this parameter will have to be found experimentally. The main condition that we need to guarantee is that we want the same tree graph as a result of the graph learning algorithm whether the max edge distance parameter is being used or not. Therefore, the optimal value for this parameter should be the minimum for which the graph obtained remains unchanged if the parameter is increased. In the worst case scenario, i.e., when nodes separated by a great distance are connected in the tree graph, the introduction of this parameter does not worsen the performance of the learning method. The introduction of this parameter to the model is exclusively meant to reduce the computational cost of all the candidate edges.

Chapter 5

Experiments and results

This chapter presents the experiments that have been designed and implemented to have a better understanding of the data, to optimize the tree model and tune its hyper-parameters and to fairly compare the performance of the proposed method with other state of the art interpolation methods.

5.1 Tree model validation

This section introduces the key concepts needed to perform the designed tree model validation, including the metrics used to evaluate the interpolation performance and the cross-validation techniques used to tune the model hyper-parameters and analyze their effect on the interpolation system designed.

5.1.1 Performance metrics

Interpolation methods have been widely used in many disciplines, and the literature introduces a broad variety of performance metrics used to evaluate interpolation methods. An extensive list of measurements used to assess the performance of interpolation methods is provided in [30] and [31], which present metrics for spatial interpolation and time series interpolation, respectively. Since the interpolation that we are performing is both in space and time, metrics that are commonly applied in both cases will be used here.

The output of the interpolation will consist of two vectors $\mathbf{p} = \{p_1, \ldots, p_N\}$ and $\mathbf{o} = \{o_1, \ldots, o_N\}$, representing the predicted values and the observations or samples,

respectively. A common performance metric used in interpolation is the mean squared error (MSE), which provides a measure of the variance of the residuals (prediction errors). The MSE is computed as follows:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(p_i - o_i)^2, \tag{5.1}$$

where N is the number of observations or samples, o_i are the observed values and p_i are the predicted or estimated values. Since the error is squared, a more common performance metric used in interpolation is the root mean squared error (RMSE), which provides a measure of the standard deviation of the residuals (prediction errors) [32]. RMSE is computed as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - o_i)^2}. \tag{5.2}$$

RMSE is more commonly used than MSE because RMSE is measured in the same units as the data being interpolated, while MSE is measured in squared units of the response variable. Nevertheless, in this case we also want to quantify the variability of the interpolation error. This can be done computing the variance of the squared error:

$$\sigma_e^2 = \frac{1}{N}\sum_{i=1}^{N}(e_i - \bar{e})^2, \tag{5.3}$$

where $e_i = (p_i - o_i)^2$ is the squared error and $\bar{e} = MSE$, computed as in (5.1). From this variance of squared error, we can get the standard deviation of the squared error by just applying the square root: $\sigma_e = \sqrt{\sigma_e^2}$, with σ_e representing the standard deviation of the squared error. This is useful to compare in terms of MSE the different state of the art interpolation techniques to the one presented in Chapter 4, because it allows us to introduce error bars that represent the squared error standard deviation of each interpolation method. Given the metrics presented in this section, we will be able to tune the tree graph model hyper-parameters, evaluate different interpolation methods

and compare their performances.

5.1.2 Cross-validation

Cross-validation is a set of model validation techniques for assessing how the results of a statistical analysis will generalize to an independent data set [33]. Basically, cross-validation is a resampling method that utilizes different subsets of the data to train and test a model on multiple iterations. It is mostly used when predicting and estimating how accurately a model will perform in practice [34].

As part of cross-validation, data samples are divided into complementary subsets, analysis is performed first on one subset (known as the training set) and validation is carried out on the other subset (known as the validation set or testing set). In most methods, cross-validation is performed several times using different partitions, and the results of the tests are combined (e.g., averaged) over the rounds to estimate the model's predictive performance. To summarize, cross-validation combines (averages) measures of fitness in prediction to derive a better estimation of model performance [35].

5.1.2.1 k-fold cross-validation

In k-fold cross-validation, the data is partitioned into k equal sized folds. Of the k folds, a single fold is retained as the validation data for testing the model, and the remaining $k-1$ folds are used as training data. The cross-validation process is then repeated k times, with each of the k folds used exactly once as the validation data [36].

Figure 5.1 shows the scheme of k-fold cross-validation for $k = 5$. The main advantages of using a k-fold cross-validation approach to evaluate a model performance are that training data is separated from the test data, which prevents the problem of overfitting [37], and that all the data is used for testing purposes through the different iterations involving this process, as it is depicted in Figure 5.1.

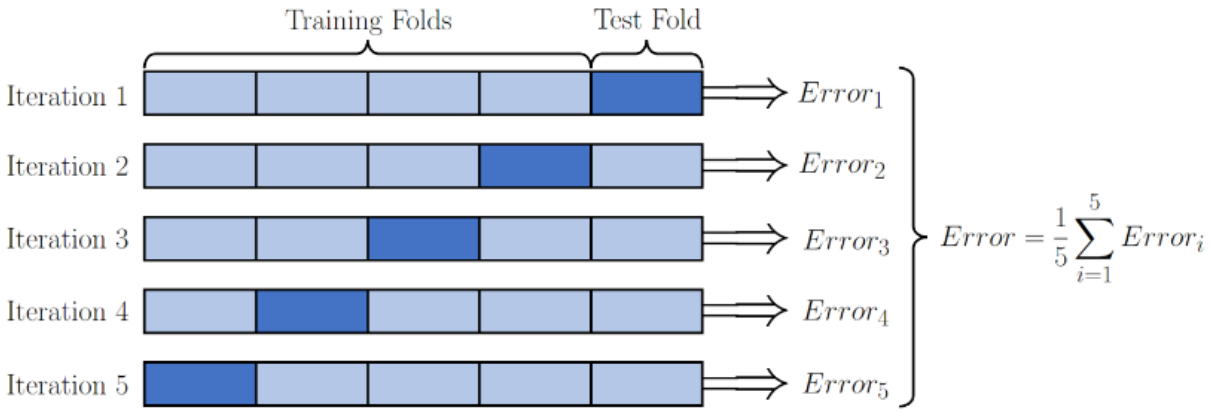


Figure 5.1: Illustrative example of k-fold cross-validation with $k = 5$.

5.1.2.2 k-fold nested cross-validation

When cross-validation is used simultaneously for selection of the best set of hyper-parameters and for error estimation, a nested cross-validation is required.

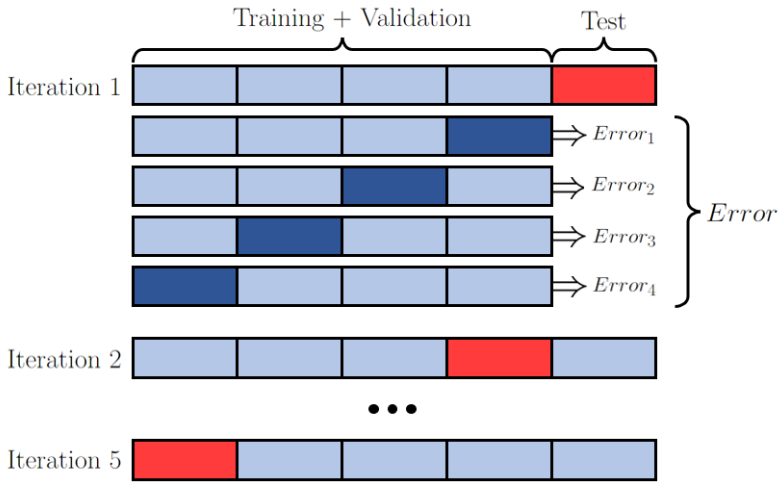


Figure 5.2: Nested cross-validation scheme.

Figure 5.2 shows the nested cross-validation scheme. Nested cross-validation is a generalization of the Train-Validation-Test protocol, where training is performed on the Train set for all hyper-parameter values, the combination of parameters that provide the best performance on Validation is selected and used to train a single model on Train+Validation whose performance is estimated on the Test set. Since the Test set is used only once by a single model, performance estimation has no bias due to the model selection process.

The final model is trained on all data using the best found values for the set of hyper-parameters considered. Nested cross-validation generalizes this protocol to cross-validate every step of this procedure: for each Test set, all folds serve as Validation, and this process is repeated for each fold serving as Test [38]. As shown in Figure 5.2, nested cross-validation is formed by an inner loop which follows a cross-validation scheme to obtain the error (RMSE) for each combination of hyper-parameters, and then the outer loop consists on learning a model using Training+Validation and the set of parameters that yielded to the best performance (lowest error) in the inner loop, and using the Test fold to evaluate the performance.

5.2 Hyper-parameter tuning

This section covers the hyper-parameter tuning of the tree model, which has been done using a k-fold nested cross-validation with $k = 10$ (see Section 5.1.2.2).

5.2.1 Hyper-parameter tuning: Δt

Data analysis is extremely important in order to understand how the data behaves and it can be very useful to extract some features that can be taken into consideration when optimizing the proposed solution hyper-parameters. Therefore, since the tree model considers time intervals of length Δt, the signal temporal autocorrelation will be analyzed in order to estimate a range of values for Δt to consider.

5.2.1.1 Temporal autocorrelation

Autocorrelation in time series is widely used to analyze the behavior of a signal over time. It can be used as a measure of how much influence past values of a signal have on its future values, e.g., to measure the influence of past air quality data to present air quality data. The formula used to compute the autocorrelation given a discrete signal $\boldsymbol{s}[n]$ is:

$$\mathbf{r}[k] = \sum_{n=-\infty}^{+\infty} s[n+k] \cdot s^*[n] \tag{5.4}$$

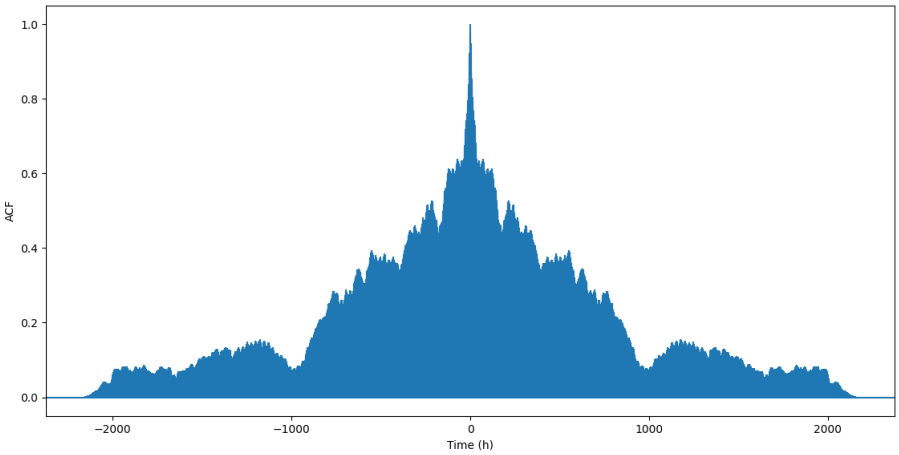
where $\mathbf{r}[k]$ is the autocorrelation function and $\mathbf{s}^*[n]$ is the conjugate of $\mathbf{s}[n]$, which in the case being $s^*[n] = s[n]\ \forall n$, due to the fact that $\mathbf{s}[n]$ is a real signal.

Before proceeding to compute the autocorrelation of the data as a function of time, it is important to state how the signal $\mathbf{s}[n]$ is defined from the data. Given that the observations are not evenly spaced over time, it does not make sense to use $\mathbf{s}[n] = \{o_1, o_2, ..., o_N\}$, with o_i being the i-th observation and N the number of observations. A more reasonable option to define $\mathbf{s}[n]$ is to select evenly spaced time intervals and compute the average of all the observations available within that time interval. This time interval needs to be short enough to obtain a high resolution autocorrelation, but at the same time it needs to be long enough to avoid having a lot of samples missing in $\mathbf{s}[n]$. For convenience, for the dataset used in this thesis, the time interval is set to 1 hour.

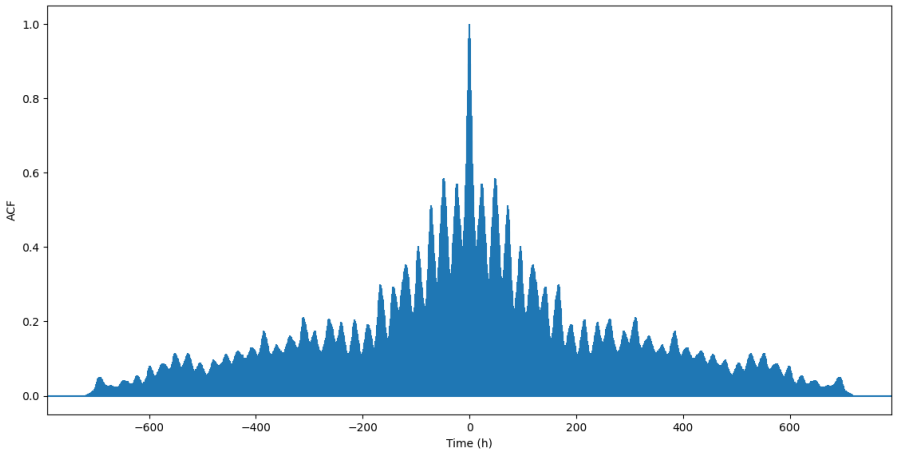
$$\mathbf{o}[n] = \{(t_1, o_1), ..., (t_N, o_N)\} \longrightarrow \mathbf{o'}[n] = \{(t'_1, o_1), ..., (t'_N, o_N)\},\ t'_i = \left\lceil \frac{t_i}{T} \right\rceil$$

$$\mathbf{s}[n] = \{s_1, s_2, ..., s_M\},\ \text{where}\ \ s_j = \bar{\boldsymbol{o}}_{\boldsymbol{j}}[n] \text{ and } \boldsymbol{o}_{\boldsymbol{j}}[n] = \{(t'_j, o_1), (t'_j, o_2), \ldots, (t'_j, o_{N_i})\},$$

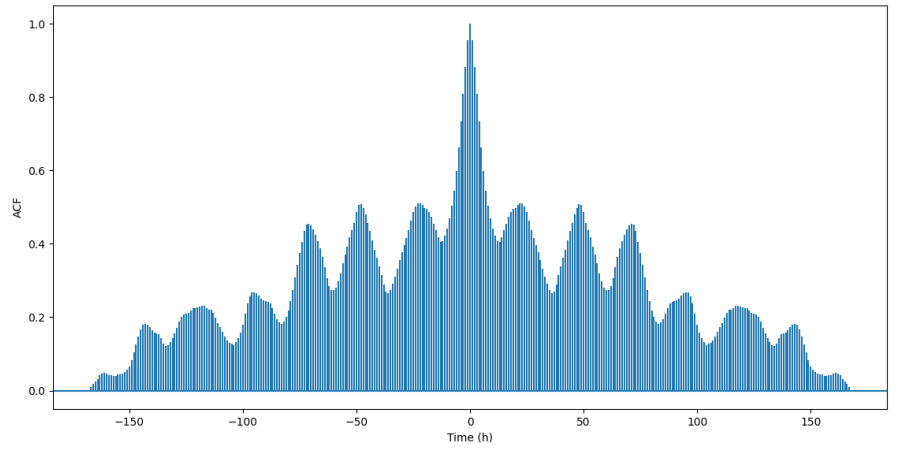
where T is the time interval defined, $\mathbf{o}[n]$ is the vector of observations, $M = t'_N$ is the number of samples of $\mathbf{s}[n]$ and $\mathbf{o_j}[n]$ is the vector containing all the observations in the time interval t'_j.



(a) Temporal autocorrelation for a 90 days period.



(b) Temporal autocorrelation for a 30 days period.



(c) Temporal autocorrelation for a 7 days period.

Figure 5.3: Temporal autocorrelation plots for different time periods.

The temporal autocorrelation for different time windows is shown in Figure 5.3. In Figure 5.3a, corresponding to the time period of 90 days, we can clearly see how the temporal autocorrelation decays over time. If we take a look at Figure 5.3b, corresponding to a time period of 30 days, we can appreciate multiple peaks in the correlation plot, but the overall analysis period is still too long to extract any relevant conclusions. Finally, if we compute the temporal autocorrelation for a 7 days period, as shown in Figure 5.3c, we can appreciate that these peaks previously spotted have a 24 hours periodicity. Moreover, it is quite relevant to notice that the temporal autocorrelation of the observed signal decays rapidly within the first few hours, reaching a relative minimum at around 12 hours. Therefore, this allows us to shorten the candidate values for the Δt hyper-parameter in that temporal interval. Also, this observed periodicity could be a factor to take into consideration towards future work.

5.2.1.2 Hyper-parameter selection and stability: Δt

Hyper-parameter selection is done using the k-fold nested cross-validation approach explained with $k = 10$. The candidate values for the Δt hyper-parameter are the following:

$$\mathbf{c}_{\Delta t}[n] = \{0.5, 1, 1.5, 2, 3, 4, 6, 8, 12\}$$

where $\mathbf{c}_{\Delta t}[n]$ refers to the vector that contains the candidate values for Δt expressed in hours. The upper bound of 12 hours is set because of the temporal autocorrelation analysis. The lower bound of 0.5 hours is found experimentally, because it is the lowest value for Δt that allows a valid tree graph construction. In other words, when the hyper-parameter is set to lower values, the algorithm fails to build a tree graph that consists of only one connected component.

The results for the nested cross-validation technique show that the optimal value for the hyper-parameter Δt is 0.5 hours. Moreover, this optimal value is also found to be

stable, meaning that it is always selected as the optimal value along all the iterations of the nested cross-validation process. This result is consistent with the temporal autocorrelation plots from Figure 5.3, in which it can be seen how the autocorrelation is greater at shorter time intervals. Thus, we would expect a lower optimal value for Δt if we had a higher data frequency, that is, more frequent measurements. This fact is also coherent with the expectation that, in the ideal case, we would be using simultaneous measurements to perform the graph learning and interpolation, as done in [9], which would be equivalent to setting Δt to a value very close to zero.

Aside from the nested cross-validation performed to select the optimal value for the hyper-parameter Δt, a k-fold cross-validation with $k = 10$ is performed for the different values of Δt contained in $\mathbf{c}_{\Delta t}[n]$. The main objective of this is observing how the performance varies as a function of this hyper-parameter. The other model hyper-parameters remain fixed and set to the optimal values found. These optimal values will be presented later, because they are not relevant to observe the RMSE as a function of Δt. The results obtained with this experiment are shown in Figure 5.4.

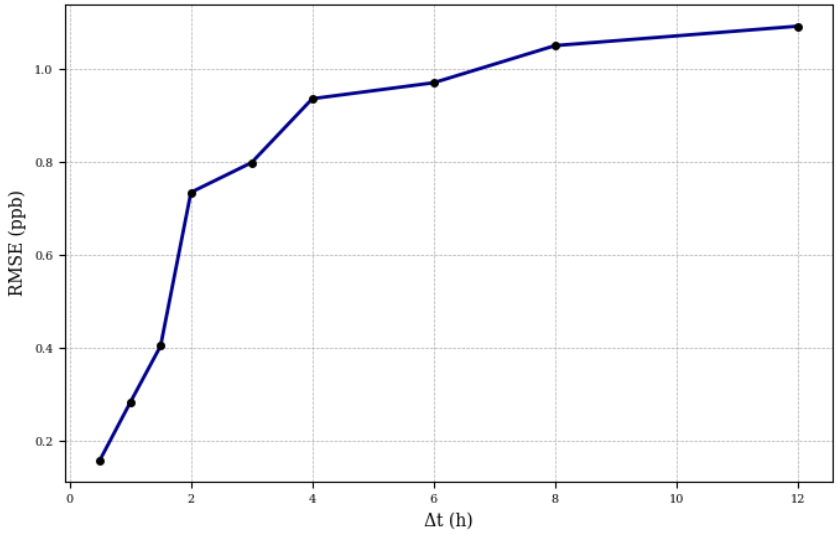


Figure 5.4: RMSE as a function of Δt.

As can be concluded from Figure 5.4, the lower the Δt, the lower the RMSE. Also, it is noticeable that the RMSE increases very rapidly with the value of Δt and then it levels

out. The evolution trend is comparable to a logarithmic trend.

5.2.2 Hyper-parameter tuning: Δs

Similarly to the Δt hyper-parameter, in order to tune Δs it is convenient to analyze the spatial autocorrelation in order to learn some features of the observed signal. Then, the selection of optimal values will be covered as well as the variation of the RMSE as a function of Δs.

5.2.2.1 Spatial autocorrelation

Spatial autocorrelation can be very useful to analyze the behavior of a signal over space and how it changes depending on the distance. It can be used as a measure of how much influence nearby values of a signal have on the actual signal at a certain location. The formula used to compute the autocorrelation given a discrete signal $\mathbf{s}[n]$ is the same as that in (5.4), which we used to compute the temporal autocorrelation. The only difference lies on the definition of $\mathbf{s}[n]$. In this case, instead of classifying the observations per time intervals, they are classified according to the distance to the reference point. Each position of the $\mathbf{s}[n]$ will represent a 1 meter interval.

$$\mathbf{o}[n] = \{(d_1, o_1), ..., (d_N, o_N)\} \longrightarrow \mathbf{o'}[n] = \{(d'_1, o_1), ..., (d'_N, o_N)\},\ d'_i = \text{round}(d_i)$$

$$\mathbf{s}[n] = \{s_1, s_2, ..., s_M\},\ \text{where}\ \ s_j = \bar{\mathbf{o}_\mathbf{j}}[n] \text{ and } \mathbf{o_j}[n] = \{(d'_j, o_1), (d'_j, o_2), \ldots, (d'_j, o_{N_i})\},$$

where $\mathbf{o}[n]$ is the vector of observations, d_i is the distance of the i-th observation with respect to the origin, $M = \max(d'_i)\ \forall i \in \{1, \ldots, N\}$ is the number of samples of $\mathbf{s}[n]$ and $\mathbf{o_j}[n]$ is the vector containing all the observations in the distance interval d'_j.

The spatial autocorrelation of the air quality dataset is shown in Figure 5.5, which has been created using samples taken within a 3 months period. There are a few interesting facts that we can deduce from Figure 5.5. Firstly, we notice the instant drop of the

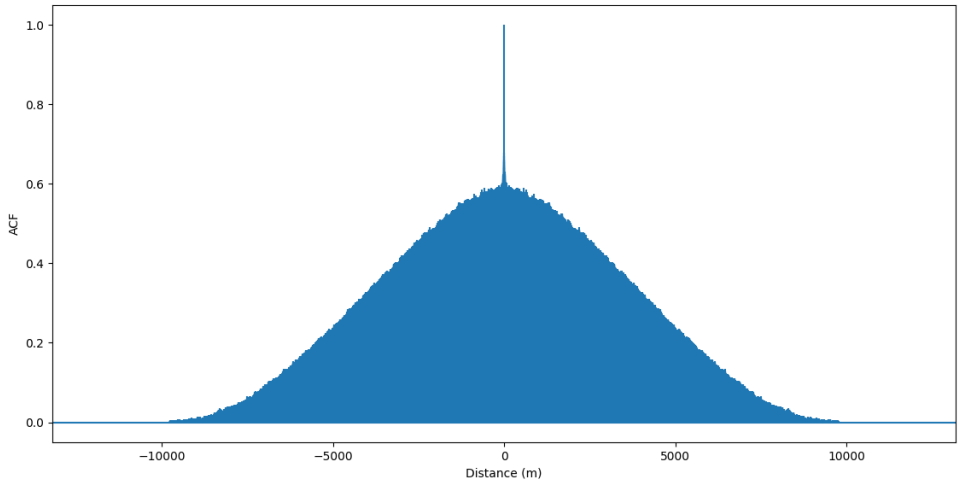


Figure 5.5: Spatial autocorrelation.

ACF (autocorrelation function) at zero. This allows us to conclude that the spatial autocorrelation is not as strong as the temporal one. Recall that the autocorrelation function always finds its maximum at zero displacement, irrespective of the input signal [39]. Despite the fact that the correlation is weaker, we can still observe a decaying correlation as the distance is increased. Also, this drop is smoother than in the temporal autocorrelation, which can be an indicator of certain flexibility when tuning and selecting the hyper-parameter Δs. Nevertheless, we can expect from these observed results that the optimal value for Δs will be small. From this, we can probably set the upper bound for Δs to 500 meters, since the signal autocorrelation starts to decrease more rapidly and we still want to keep a high spatial resolution.

5.2.2.2 Hyper-parameter selection and stability: Δs

The candidate values for the Δs hyper-parameter introduced as input in the nested cross-validation evaluation are the following:

$$\mathbf{c}_{\Delta s}[n] = \{50, 75, 100, 125, 150, 175, 200, 250, 300, 400, 500\},$$

where $\mathbf{c}_{\Delta s}[n]$ refers to the vector that contains the candidate values for Δs expressed in meters. The upper bound of 500 meters is set according to the reasoning presented in the previous section, that is, the spatial autocorrelation is higher between points that are closer in space and we want to preserve a high spatial resolution. The lower bound is set to 50 meters. This value is found experimentally, because it is the lowest value for Δs that permits a valid tree graph construction. Similarly to what happened with Δt, there is a limit beyond which is not possible to learn a one connected component tree graph due to the frequency of observations not being high enough. This limit is the lower bound defined for Δs.

The results for the nested cross-validation technique show that the optimal value for the hyper-parameter Δs is either 75 or 150 meters. This means that in some iterations of the nested cross-validation $\Delta s = 75$ meters leads to the lowest RMSE and in other iterations $\Delta s = 150$ meters performs better. Therefore, we can say that this hyper-parameter is not completely stable, even though it shows a high stability, since only two candidates are selected through the evaluation process and having a total of 11 candidate values.

Aside from the nested cross-validation performed to select the optimal values for the hyper-parameter Δs, a k-fold cross-validation with $k = 10$ is performed for the different values of Δs contained in $\mathbf{c}_{\Delta s}[n]$ to observe how the performance varies as a function of this hyper-parameter. Similarly to what has been done for Δt, the other model hyper-parameters remain fixed and set to the optimal values found ($\Delta t = 0.5$ hours, as presented in previous sections). The results obtained with this experiment are shown in Figure 5.6.

As can be concluded from Figure 5.6, the lowest RMSE is obtained with $\Delta s = 150$ meters, followed by $\Delta s = 75$ meters (both RMSE values are very close). This is consistent with the results obtained from the nested cross-validation approach, which found that a combination of these two values is optimal. Nevertheless, if we had to pick just one value for Δs, the decision would depend on different aspects. On the one hand, if we pick

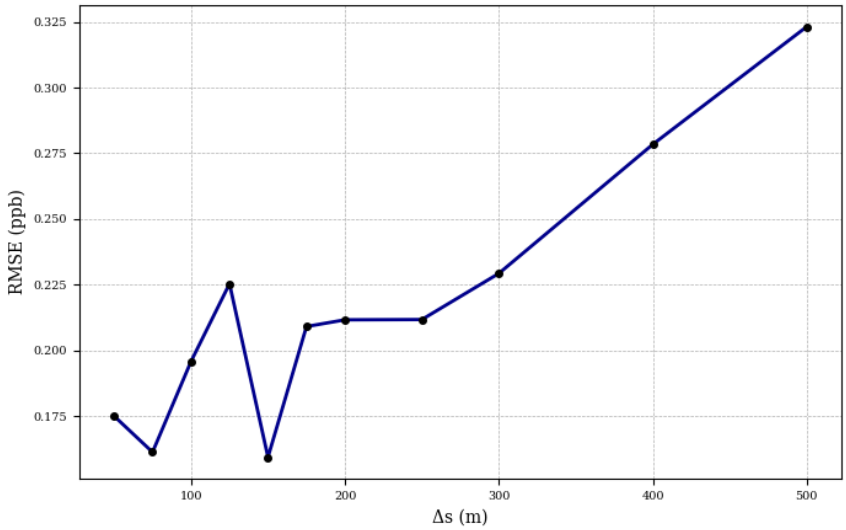


Figure 5.6: RMSE as a function of Δs.

$\Delta s = 150$ meters, we can say that the RMSE is slightly smaller but, more importantly, the time complexity is lower. This is because with a higher Δs value, the number of both candidate nodes and actual nodes in the graph is smaller, which reduces the number of edges and consequently the weights estimation and maximum spanning tree algorithm are faster. On the other hand, selecting $\Delta s = 75$ meters will provide a higher spatial resolution, at the expense of higher computational complexity.

5.2.3 Hyper-parameter tuning: α

Hyper-parameter α is a bit different compared to Δt and Δs, since it does not represent a physical magnitude such as time or space. It is a factor in the weights estimation formula expressed in 4.6. Recall that $\alpha > 0$ needs to be fulfilled. The main role of this hyper-parameter is to allow modeling the distribution of weights, in such a way that a small value of α will result in greater variability of the estimated weights than a very large value of α, which will predominate in the weights estimation formula and, consequently, this will cause all weights to be similar. Note that the value assigned to α does not affect the learnt tree graph edges, it only affects the value of its weights. In other words, the edges forming the final tree graph will be the same regardless of α. In addition, α prevents

edge weights to become infinite (or very large). Given that literature does not provide an optimal value nor a range of values for α, the candidates considered will be the following:

$$\mathbf{c}_\alpha[n] = \{0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.5, 1, 5, 10, 50, 100\},$$

where $\mathbf{c}_\alpha[n]$ refers to the vector that contains the candidate values for α. The candidates for this hyper-parameter cover a wide range of possibilities and focus mainly on the order of magnitude.

The results for the nested cross-validation technique show that the optimal value for the hyper-parameter is $\alpha = 0.1$. Moreover, this optimal value is also found to be stable, i.e. it is always selected as the optimal value along all the iterations of the nested cross-validation process.

Aside from the nested cross-validation performed to select the optimal value for the hyper-parameter α, a k-fold cross-validation with $k = 10$ is performed for the different values of α contained in $\mathbf{c}_\alpha[n]$ to observe how the performance varies as a function of this hyper-parameter. Similarly to what has been done in previous sections, the other model hyper-parameters remain fixed and set to the optimal values found ($\Delta t = 0.5$ hours, $\Delta s = 150$ meters). The results obtained with this experiment are shown in Figure 5.7.

Note that the x-axis in Figure 5.7 is in a logarithmic scale. It shows a very interesting result. On one side, we can see the consistency with the optimal value selection, with $\alpha = 0.1$ as the optimal value. Nevertheless, we can also appreciate that the RMSE is very similar for smaller values of α. Then, RMSE increases drastically when $0.1 < \alpha \leq 0.5$ and remains more or less constant after that. This result, which might seem surprising at first glance, actually make sense. If we think about the role of α in (4.6), we can see that as α increases, it starts to dominate in the formula. When this happens, all weights are similar and the graph becomes more similar to an unweighted graph, which contains less information than a weighted graph and, therefore, a worse interpolation is expected.

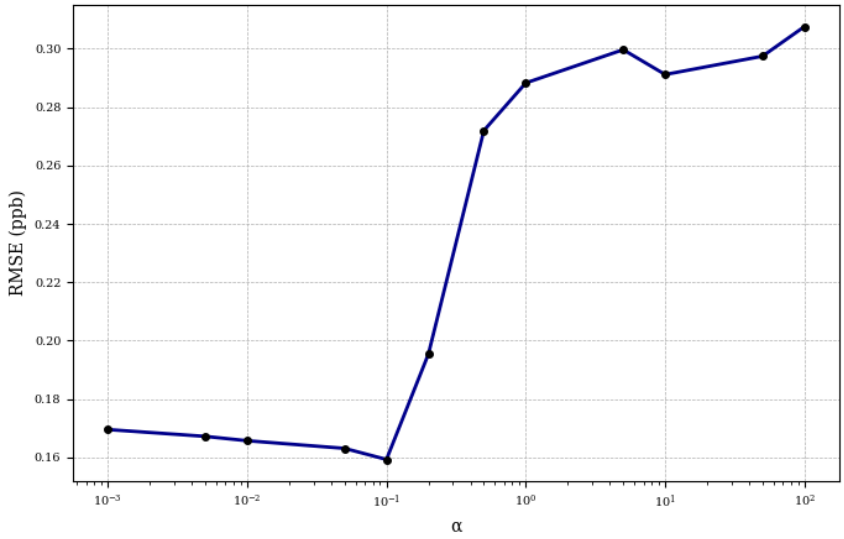


Figure 5.7: RMSE as a function of α.

This leads to a higher RMSE overall.

5.2.4 Hyper-parameter tuning: maximum edge distance

Finally, the last hyper-parameter to tune is the maximum edge distance. Recall that this parameter is introduced with the purpose of reducing the computational cost of the weights estimations rather than to improve the interpolation performance. The candidates for this hyper parameter are the following:

$$\mathbf{c}_{\text{max\_edge}}[n] = \{700, 800, 900, 1000, 1500, 2000, 3000, 4000, 5000, 7500, 10000\},$$

where $\mathbf{c}_{\text{max\_edge}}[n]$ refers to the vector that contains the candidate values for the maximum edge distance expressed in meters. The upper bound of 10 km is set because it represents the maximum distance present in the data. Thus, it is equivalent to the nonexistence of the maximum edge distance hyper-parameter. The lower bound of 700 meters is found experimentally, because it is the lowest value that allows a valid tree construction. When the hyper-parameter is set to lower values, the maximum edge distance considered is not enough to build the desired tree graph that consists of only one connected component.

The results for the nested cross-validation technique show that the optimal value for the hyper-parameter maximum edge distance is 1 km. Moreover, this optimal value is also found to be stable, meaning that it is always selected as the optimal value along all the iterations of the nested cross-validation process. In this case, it is relevant to note that, by definition, the optimal value for maximum edge distance is the lowest that provides the same performance than in the case where such hyper-parameter is not considered.

Aside from the nested cross-validation performed to select the optimal value for the hyper-parameter maximum edge distance, a k-fold cross-validation with $k = 10$ is performed for the different values contained in $\mathbf{c}_{\text{max\_edge}}[n]$. The results obtained with this experiment are shown in Figure 5.8.

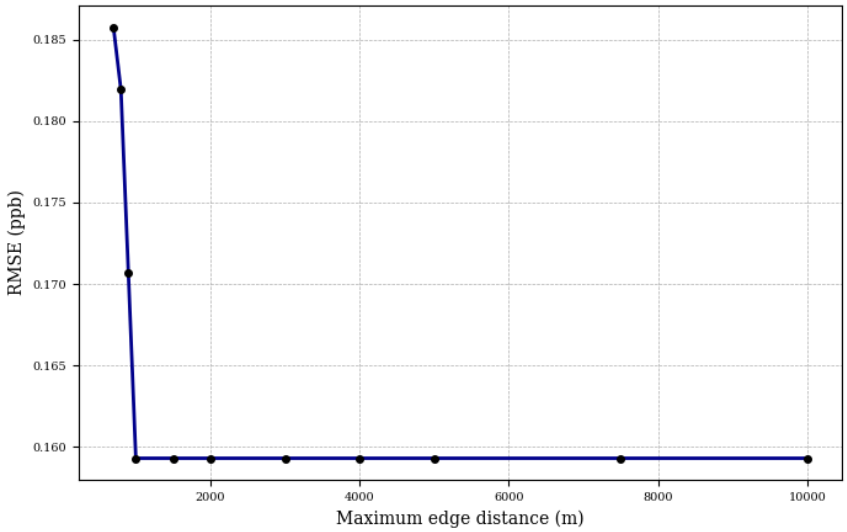


Figure 5.8: RMSE as a function of the maximum edge distance.

As can be seen in Figure 5.8, the RMSE remains constant once the threshold of maximum edge distance equal to 1 km is crossed. This is what we were expecting by the definition of this hyper-parameter. This fact is not as interesting as analyzing the edge weights computation time. Figure 5.9 shows the weights computation time as a function of the maximum edge distance.

Results in Figure 5.9 show that the execution time of the weights computation varies significantly depending on this hyper-parameter. In fact, we can conclude that intro-

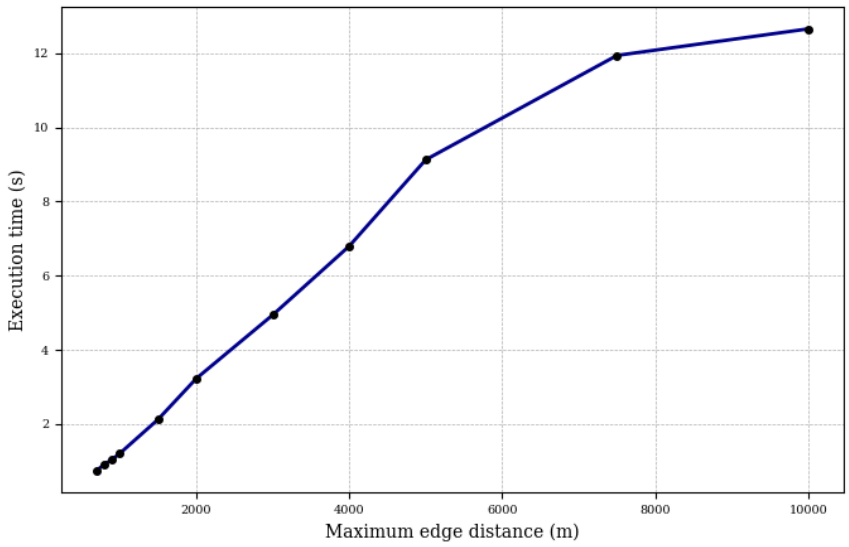


Figure 5.9: Weights computation time as a function of the maximum edge distance.

ducing this hyper-parameter provided a speed-up of 10.5 (10.5x faster) to the weights computation part of the algorithm, which is the heaviest part computationally speaking. Even though this improvement can be data or application dependent, the maximum edge distance is still a hyper-parameter to consider if the graph model is complex.

5.3 Comparison with state of the art interpolation methods

Once the tree graph method hyper-parameters have been tuned and their effect on the output prediction error has been analyzed, comparing it to more conventional and well-known interpolation methods can give a measure of how good this approach for interpolation is. We will compare it to inverse distance weighting (IDW) and Kriging [40]. Moreover, we also want to see if the trained models are consistent over time and not only when used in the same time period as the training data. Therefore, the experiment in which these interpolation methods will be compared will consist of training a model per each interpolation method with data captured during two weeks, and testing the interpolation performance

over the data captured during the next two weeks. The next sections introduce the basic concepts regarding these interpolation methods and, afterwards, the results are presented.

5.3.1 Inverse Distance Weighting

Inverse distance weighting (IDW) interpolation uses observations to interpolate a sample located in any position in space. As its name indicates, the predicted value will be the weighted mean of the observed samples, where the weights are the inverse of the distance between the point where we want to interpolate and the point where each observed sample is located. Mathematically, the interpolation is expressed as

$$x_p = \frac{\sum_{i=1}^{N} x_i \cdot w_{p,i}}{\sum_{i=1}^{N} w_{p,i}} \ , \quad \text{with} \quad w_{p,i} = \frac{1}{\text{dist}(x_p, x_i)},$$

where x_p is the predicted sample, x_i is the i-th sample in the vector of observations and $w_{p,i}$ is the weight given to the i-th observed sample when interpolating x_p. This weight is computed as the inverse of the distance between the location of x_p and the location of the i-th observed sample.

It is important to note that, in fact, IDW interpolation does not require any training and the computations involved are quite simple. Thus, it can be useful when we do not have enough data to train a tree graph or a Kriging model. As a drawback, its simplicity usually makes it not as good as other interpolation methods in terms of prediction error [41][42].

Besides that, we need to define how to select the set of observed samples used to interpolate. Since we are performing a spatio-temporal interpolation, we need to include the time variable to the interpolation method, while making it a fair comparison to the tree graph method presented. The best way to do so is by adding the constraint that the observed samples x_i used must belong to the same time interval t_j of length Δt that the sample that we want to interpolate x_p. Since there is a strong temporal autocorrelation

in the data, if the observed samples are not from the same time interval, the interpolation error would increase significantly.

5.3.2 Kriging

Kriging interpolation is a widely used interpolation method in the domain of spatial analysis and geostatistics. Kriging is similar to IDW in the sense that two points that are at the same distance from the prediction location are given the same weight. This is because isotropy is assumed, which means that there is uniformity in every direction. But instead of just taking the inverse of the distance, Kriging trains a variogram model that is fit using training data. Then, this variogram model is used to compute the weights and perform the interpolation [43].

The variogram can be estimated from sample data with the following formula:

$$\hat{\gamma}(\mathbf{h}) = \frac{1}{2N(\mathbf{h})} \sum_{i=1}^{N(\mathbf{h})} (z(\mathbf{x}_i) - z(\mathbf{x}_i + \mathbf{h}))^2, \tag{5.5}$$

where $z(\mathbf{x}_i)$ is the value of some property z (ozone ppb) at position $\mathbf{x}_i$ and $z(\mathbf{x}_i + \mathbf{h})$ is the value at $(\mathbf{x}_i + \mathbf{h})$. Then, $\mathbf{h}$ is referred to as the lag, and $N(\mathbf{h})$ is the number of pairs of observations separated by the lag $\mathbf{h}$. Note that this experimental variogram consists of some points that express a relation between a lag or distance $\mathbf{h}$ and the semivariance $\hat{\gamma}(\mathbf{h})$. At this point, we need to find a theoretical expression for this variogram, i.e., fit a curve to these obtained points. Some of the most common variogram models that provide a better fit are Gaussian, exponential and spherical models [44]. The expressions for this variogram models are the following:

- Gaussian model:

$$p \cdot \left(1 - e^{-\frac{d^2}{(4r/7)^2}}\right) + n$$

- Exponential model:

$$p \cdot \left(1 - e^{-\frac{d}{r/3}}\right) + n$$

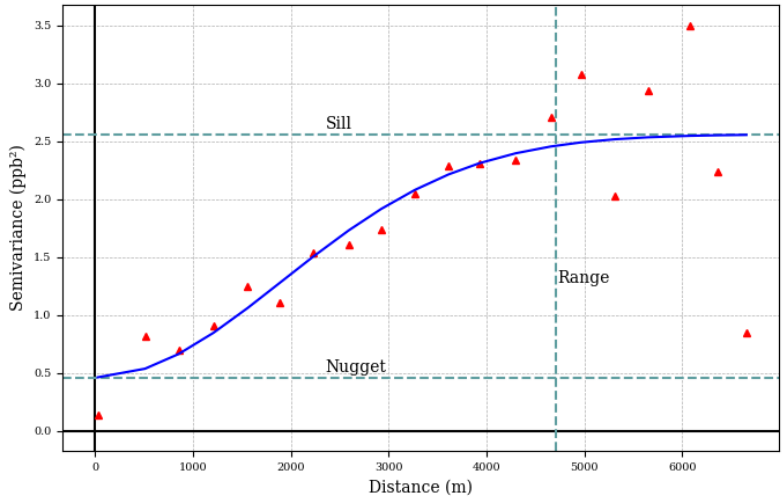
- Spherical model:

$$\begin{cases} p \cdot \left(\frac{3d}{2r} - \frac{d^3}{2r^3}\right) + n & d \leq r \\ p + n & d > r \end{cases}$$

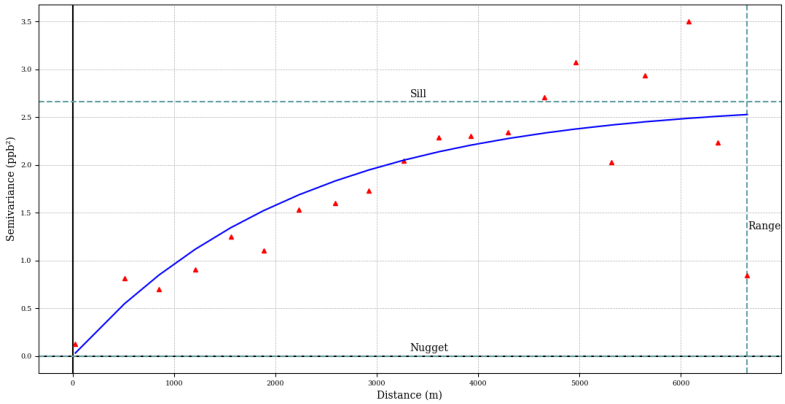
where p is the partial-sill, n is the nugget, r is the range and d is the distance. These are the well-known variogram parameters (except for the distance, which is the variogram's x-axis) and they are defined as follows:

- Nugget (n): semivariance value at which the variogram begins (at $d = 0$).
- Sill (s): ceiling of the variogram, where the variogram can no longer get higher than a certain value. The partial-sill (p) is defined as $p = s - n$.
- Range (r): distance in which the variogram first hits the sill.

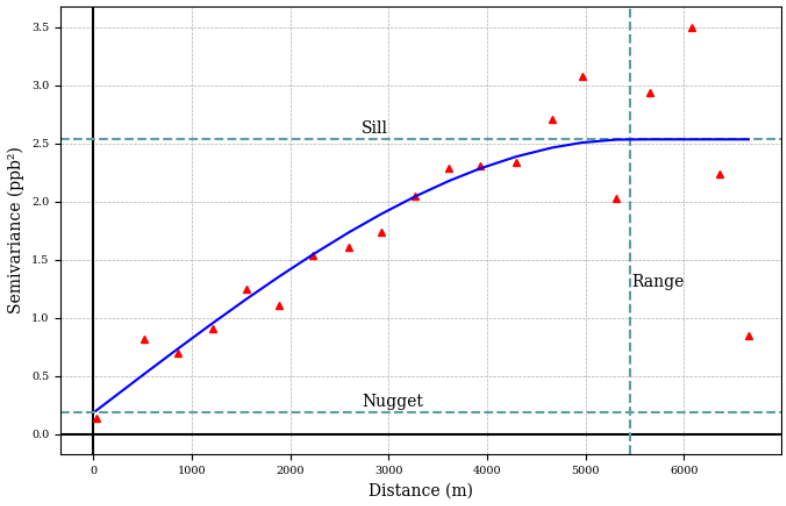
Examples of Kriging variograms for Gaussian, exponential and spherical models are shown in Figure 5.10. In these variogram plots, red triangles represent the semivariance points computed from the data according to 5.5, the dark blue line is the theoretical variogram found according to each model equation and the three Kriging variogram parameters are displayed in discontinuous light blue lines for clarity. In the particular case of the air quality dataset, the spherical variogram model provides the best fit, yielding to the lowest RMSE when interpolating, so it is the variogram model that is going to be used to compare to the other interpolation methods.



(a) Gaussian model.



(b) Exponential model.



(c) Spherical model.

Figure 5.10: Example of different Kriging variogram models.

Once the training is done and the variogram is learned, interpolation of the sample x_p can be performed using observed samples x_i with weights w_i. The idea of Kriging is that the estimator is unbiased (i) and the estimation variance is minimized (ii):

(i): For this to happen, $\sum_{\forall i} w_i = 1$ and the mean is stationary

(ii): $\sigma_\varepsilon^2 = E[x_p - \hat{x}_p]^2 = \mathrm{Var}(x_p - \hat{x}_p)$

For the mean to be stationary in a time-varying signal, we need to define time intervals in which this assumption holds. Once again, we introduce the time intervals of length Δt and we will use observed samples that belong to the same time window as the sample that we want to interpolate. Also, in order to minimize σ_ε^2, Lagrange multiplier method can be used to find the weights such that

$$L = \mathrm{Var}(x_p - \hat{x}_p) + 2\lambda(\sum_{\forall i} w_i - 1),$$

where L is the Lagrange function, λ is the Lagrange multiplier and $2\lambda(\sum_{\forall i} w_i - 1)$ is the part that guarantees $\sum_{\forall i} w_i = 1$ [45].

The only problem of this Kriging approach is that, if we do not consider time while training the variogram, we will be comparing pairs of observations very distant in time according to (5.5), which just takes into consideration the lag or distance between pairs of observations. Moreover, if we are assuming stationary mean, we also want this assumption to hold during training and, the only way to accomplish that is introducing the Δt parameter into the variogram learning process. Rearranging the variogram equation to consider time, we now have that:

$$\hat{\gamma}(\mathbf{h}) = \frac{1}{2N(\mathbf{h})} \sum_{i=1}^{N(\mathbf{h})} (z(\mathbf{x}_i, t_i) - z(\mathbf{x}_i + \mathbf{h}, t_i \pm t_\varepsilon))^2, \tag{5.6}$$

where t_i is the timestamp corresponding to the observation $z(\mathbf{x}_i)$ and $t_\varepsilon \in [-\Delta t, +\Delta t]$. In other words, we define a temporal sliding window of length Δt and we compare pairs of observations that are separated at most Δt.

5.3.3 Results

We now want to analyze the performance of IDW and Kriging interpolation and compare them to the tree graph method that is proposed in this thesis. Since the parameter Δt is introduced into all of the interpolation methods, all the performance results will be shown with respect to this temporal parameter. On the other hand, each interpolation model has been optimized on its own: the tree graph model hyper-parameter optimization was previously presented in Section 5.2, IDW does not require optimizations since the only parameter it has is Δt, and Kriging is optimal when selecting a spherical variogram model and the optimal variogram parameters (nugget, partial-sill and range) are found when fitting the curve expression to the variogram points following a minimum RMSE criterion.

Figure 5.11 shows the comparison of the three interpolation methods described in terms of the RMSE as a function of Δt in hours. This is the error obtained in the 2 weeks training and interpolation on the 2 following weeks experiment. We see that the lowest error is obtained by the designed tree graph method. We also see that Kriging performs better than IDW, which makes sense if we take into account that there is no training involved in the IDW method. What is also relevant from this plot is that, the lower values of Δt, the smaller the RMSE. We can see that, in general, as we increase Δt, the predictions are worse, so we are more interested in the performance when using lower values of Δt.

Figure 5.12 focuses on a range of smaller values of Δt. It shows the comparison between the three interpolation methods in terms of RMSE for values of Δt in the range between 30 minutes and 2 hours. It is also created with more points so we can actually appreciate the evolution of the estimation error. We still see that the tree graph is the one that performs the best, followed by Kriging. Actually, for $\Delta t = 30$ minutes, Kriging is quite close to the Tree graph method, but as Δt increases, the error difference between

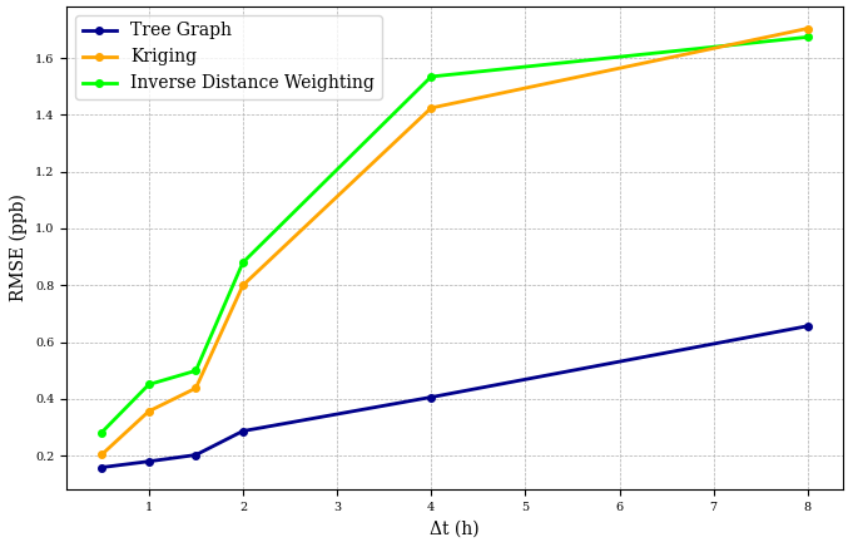


Figure 5.11: RMSE for different interpolation methods as a function of Δt.

Kriging and the tree approach also increases.

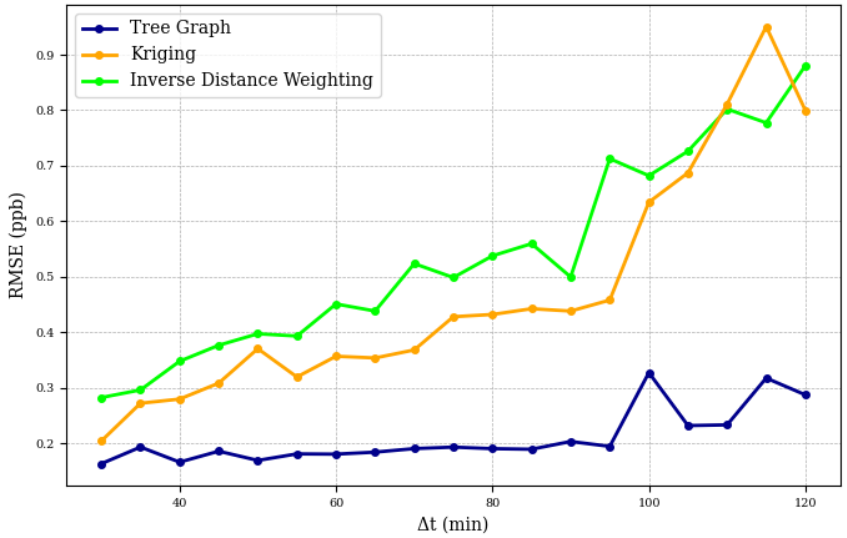


Figure 5.12: RMSE for different interpolation methods as a function of Δt.

Figure 5.13 presents the same comparison but in terms of MSE and it introduces the error bars that indicate the variance of the prediction error. On the one hand, we can observe that, the lower the Δt, the lower the variance of the prediction error, which is obvious considering that the magnitude of the error is also smaller. But what is more important is that the tree graph method shows almost no variance in the prediction error

compared to the other two interpolation methods. This fact could be seen as evidence that the tree graph interpolation is able to adapt to local changes while Kriging and the IDW methods fail to adapt locally.

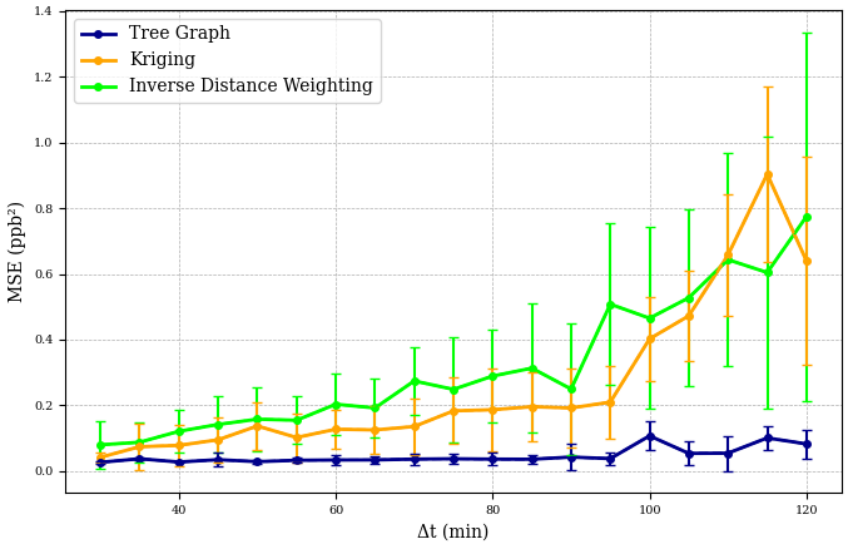
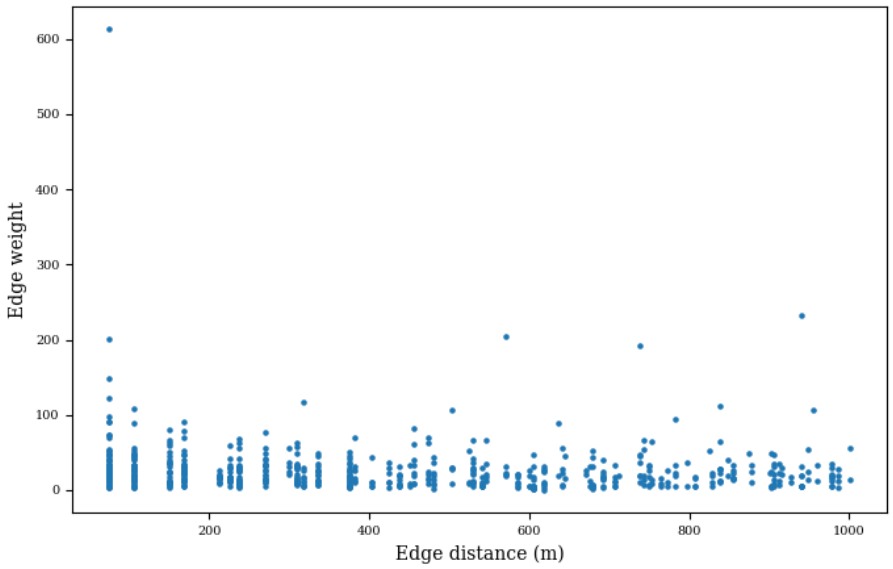
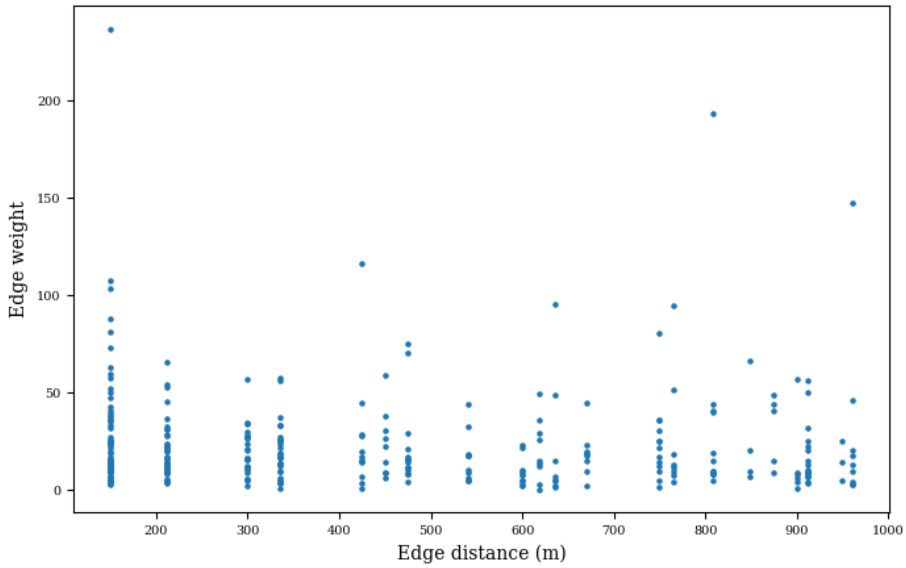


Figure 5.13: MSE with error bars for different interpolation methods.

To prove this point and to try to understand why the tree graph method performs better, we can look at the edge weights as a function the edge distance. Figure 5.14 shows the tree graph edge weights as a function of distance for both $\Delta s = 75$m and $\Delta s = 150$m. What we actually see in these plots is that, given a certain edge distance, there is a wide variety of edge weights, which indicates that the tree graph approach is adapting to local changes. On the other hand, it is important to recall that Kriging and IDW provide the exact same weight if the distance is the same (isotropy). Thus, they do not adapt to local changes and this is very likely to make them perform slightly worse than the tree graph.



(a) $\Delta s = 75$m



(b) $\Delta s = 150$m

Figure 5.14: Edge weights in the tree graph as a function of edge distance.

Finally, Figure 5.15 shows the comparison between the execution time of the different interpolation methods as a function of Δt. We can notice that the execution time for IDW is more or less constant because it requires no training. Then, as Δt increases, the execution time for the tree graph reduces, while it increases for Kriging.

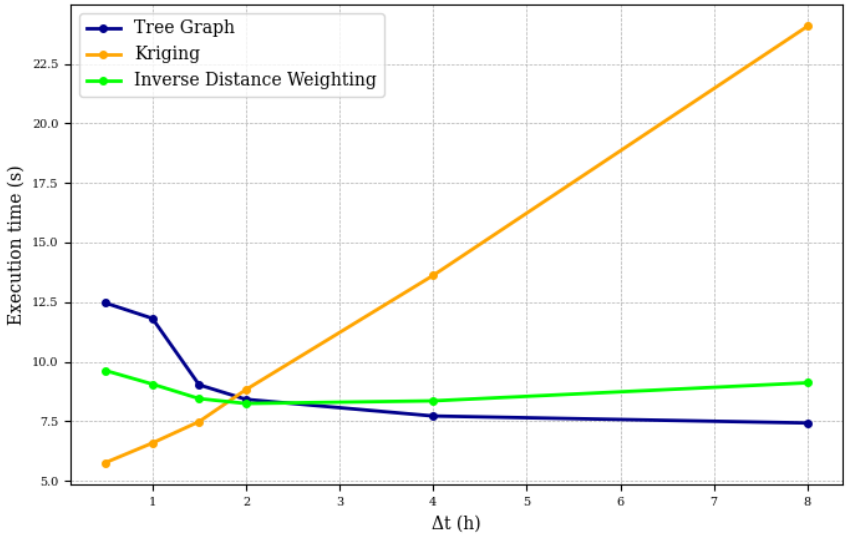


Figure 5.15: Execution time for different interpolation methods as a function of Δt.

Chapter 6

Conclusions

6.1 Conclusions

Several conclusions can be extracted from the work that this thesis presents. Firstly, the proposed graph-based spatio-temporal interpolation method outperformed Kriging and IDW interpolation techniques in terms of predictions error (RMSE), providing better estimations for observations. The graph edge weights are capable of capturing local information in space while Kriging or IDW approaches do not because they work under the assumption of isotropy, that is, uniformity in every direction. Nevertheless, a drawback of the graph interpolation method designed is that it only considers interpolation in locations represented by graph nodes. Besides that, the proposed tree graph method is more complex than Kriging and IDW, since it has more parameters and varying edge weights.

With respect to the tree graph model hyper-parameter values that are found to be optimal, we can conclude that the fact that Δt is found to be optimal for the lowest values is very consistent with the data analysis and the strong signal autocorrelation in time. In addition, the optimal values for Δs are also consistent with the order of magnitude that other investigations have found to be relevant in the field of air quality [46][47].

6.2 Future Work

Still, this is a novel approach and a lot of research can be done in the topic of mobile sensing networks from the graph signal processing perspective.

One question to be answered would be how can we incorporate interpolation at any location in space regardless of whether or not the location where we want to interpolate is represented by a node in the graph. Furthermore, studying the effect of environment variables with respect to the interpolated value and how to effectively integrate them to the tree graph model could be interesting. As an example, in the study of air quality, temperature, humidity, precipitations or wind speed can affect the measurements captured [48]. Additionally, the daily temporal autocorrelation periodicity found could be exploited to obtain better graph models or interpolation auxiliar samples. Moreover, extending this research to other applications or datasets captured as a part of a mobile wireless sensor network is necessary.

Besides that, other graph topologies different from a tree topology can be explored, as well as other graph interpolation techniques. The proposed tree graph model could also be improved by studying different definition of nodes as an alternative to the squares of side Δs considered in this research. For instance, nodes could represent circular areas in space with overlapping surfaces. Then, the observations located in the intersection between two nodes would be part of both nodes for training and learning the graph weights, and even for interpolation. In this case, it would be necessary to define a classification criterion for the observations located in the intersection area when computing the weights between two adjacent nodes. This approach could be useful to deal with very irregular datasets such as the one presented in this thesis, since it introduces more flexibility and it could help increase the number of samples per node and the spatio-temporal resolution as well.

References

[1] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, no. 5, pp. 808–828, 2018.

[2] M. O. Jackson, "Social and economic networks," in *Social and Economic Networks*, Princeton university press, 2010.

[3] A. Sandryhaila and J. M. Moura, "Discrete signal processing on graphs," *IEEE transactions on signal processing*, vol. 61, no. 7, pp. 1644–1656, 2013.

[4] S. K. Narang, A. Gadde, and A. Ortega, "Signal processing techniques for interpolation in graph structured data," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, IEEE, 2013, pp. 5445–5449.

[5] M. J. Menne, I. Durre, R. S. Vose, B. E. Gleason, and T. G. Houston, "An overview of the global historical climatology network-daily database," *Journal of atmospheric and oceanic technology*, vol. 29, no. 7, pp. 897–910, 2012.

[6] R. Hardy, P. Bates, and M. Anderson, "The importance of spatial resolution in hydraulic models for floodplain environments," *Journal of Hydrology*, vol. 216, no. 1-2, pp. 124–136, 1999.

[7] J. Rezazadeh, "Mobile wireles sensor networks overview," *International Journal of Computer Communications and Networks (IJCCN)*, vol. 2, no. 1, 2012.

[8] P. Ferrer-Cid, J. M. Barcelo-Ordinas, and J. Garcia-Vidal, "Graph learning techniques using structured data for iot air pollution monitoring platforms," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13 652–13 663, 2021.

[9] K.-S. Lu, E. Pavez, and A. Ortega, "On learning laplacians of tree structured graphs," in *2018 IEEE Data Science Workshop (DSW)*, IEEE, 2018, pp. 205–209.

[10] N. S.-N. Lam, "Spatial interpolation methods: A review," *The American Cartographer*, vol. 10, no. 2, pp. 129–150, 1983.

[11] M. Wu, J. Huang, N. Liu, R. Ma, Y. Wang, and L. Zhang, "A hybrid air pollution reconstruction by adaptive interpolation method," in *Proceedings of the 16th ACM conference on embedded networked sensor systems*, 2018, pp. 408–409.

[12] E. A. Bender and S. G. Williamson, *Lists, decisions and graphs*. S. Gill Williamson, 2010.

[13] R. C. Prim, "Shortest connection networks and some generalizations," *The Bell System Technical Journal*, vol. 36, no. 6, pp. 1389–1401, 1957.

[14] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[15] I. Amundson and X. D. Koutsoukos, "A survey on localization for mobile wireless sensor networks," in *International workshop on mobile entity localization and tracking in GPS-less environments*, Springer, 2009, pp. 235–254.

[16] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communications magazine*, vol. 40, no. 8, pp. 102–114, 2002.

[17] E. Natalizio and V. Loscrí, "Controlled mobility in mobile sensor networks: Advantages, issues and challenges," *Telecommunication Systems*, vol. 52, no. 4, pp. 2411–2418, 2013.

[18] V. Ramasamy, "Mobile wireless sensor networks: An overview," *Wireless Sensor Networks—Insights and Innovations*, 2017.

[19] B. Maag, D. Hasenfratz, O. Saukh, *et al.*, *Ultrafine Particle Dataset Collected by the OpenSense Zurich Mobile Sensor Network*, Sep. 2018. DOI: 10.5281/zenodo.3298842. [Online]. Available: https://doi.org/10.5281/zenodo.3298842.

[20] J. J. Li, B. Faltings, O. Saukh, D. Hasenfratz, and J. Beutel, "Sensing the air we breathe—the opensense zurich dataset," in *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[21] A. Ortega, *Introduction to Graph Signal Processing*. Cambridge University Press, 2022. DOI: 10.1017/9781108552349.

[22] G. Shen and A. Ortega, "Tree-based wavelets for image coding: Orthogonalization and tree selection," in *2009 Picture Coding Symposium*, IEEE, 2009, pp. 1–4.

[23] S. K. Narang and A. Ortega, "Perfect reconstruction two-channel wavelet filter banks for graph structured data," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2786–2799, 2012.

[24] B. Girault, A. Ortega, and S. S. Narayanan, "Irregularity-aware graph fourier transforms," *IEEE Transactions on Signal Processing*, vol. 66, no. 21, pp. 5746–5761, 2018.

[25] B. Lake and J. Tenenbaum, "Discovering structure by learning sparse graphs," 2010.

[26] E. Pavez and A. Ortega, "Covariance matrix estimation with non uniform and data dependent missing observations," *IEEE Transactions on Information Theory*, vol. 67, no. 2, pp. 1201–1215, 2020.

[27] P. Ferrer-Cid, J. M. Barcelo-Ordinas, and J. Garcia-Vidal, "Graph signal reconstruction techniques for iot air pollution monitoring platforms," *arXiv preprint arXiv:2201.00378*, 2022.

[28] D. Hasenfratz, O. Saukh, C. Walser, C. Hueglin, M. Fierz, and L. Thiele, "Pushing the spatio-temporal resolution limit of urban air pollution maps," in *2014 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2014, pp. 69–77. DOI: 10.1109/PerCom.2014.6813946.

[29] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, 1967, pp. 483–485.

[30] J. Li and A. D. Heap, "A review of comparative studies of spatial interpolation methods in environmental sciences: Performance and impact factors," *Ecological Informatics*, vol. 6, no. 3-4, pp. 228–241, 2011.

[31] M. Lepot, J.-B. Aubin, and F. H. Clemens, "Interpolation in time series: An introductive overview of existing methods, their performance criteria and uncertainty assessment," *Water*, vol. 9, no. 10, p. 796, 2017.

[32] A. G. Barnston, "Correspondence among the correlation, rmse, and heidke forecast verification measures; refinement of the heidke score," *Weather and Forecasting*, vol. 7, no. 4, pp. 699–709, 1992.

[33] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[34] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, Montreal, Canada, vol. 14, 1995, pp. 1137–1145.

[35] G. Seni and J. F. Elder, "Ensemble methods in data mining: Improving accuracy through combining predictions," *Synthesis lectures on data mining and knowledge discovery*, vol. 2, no. 1, pp. 1–126, 2010.

[36] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation.," *Encyclopedia of database systems*, vol. 5, pp. 532–538, 2009.

[37] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[38] I. Tsamardinos, A. Rakhshani, and V. Lagani, "Performance-estimation properties of cross-validation-based protocols with simultaneous hyper-parameter optimization," *INTERNATIONAL JOURNAL OF ARTIFICIAL INTELLIGENCE TOOLS*, vol. 24, Oct. 2015. DOI: 10.1142/S0218213015400230.

[39] R. P. Heilbronner, "The autocorrelation function: An image processing tool for fabric analysis," *Tectonophysics*, vol. 212, no. 3-4, pp. 351–370, 1992.

[40] A. Appice, A. Ciampi, F. Fumarola, and D. Malerba, "Missing sensor data interpolation," in *Data Mining Techniques in Sensor Networks*, Springer, 2014, pp. 49–71.

[41] G. Q. Tabios III and J. D. Salas, "A comparative analysis of techniques for spatial interpolation of precipitation 1," *JAWRA Journal of the American Water Resources Association*, vol. 21, no. 3, pp. 365–380, 1985.

[42] A. Kravchenko and D. G. Bullock, "A comparative study of interpolation methods for mapping soil properties," *Agronomy journal*, vol. 91, no. 3, pp. 393–400, 1999.

[43] M. A. Oliver and R. Webster, "Kriging: A method of interpolation for geographical information systems," *International Journal of Geographical Information System*, vol. 4, no. 3, pp. 313–332, 1990.

[44] P. K. Kitanidis, *Introduction to geostatistics: applications in hydrogeology*. Cambridge university press, 1997.

[45] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[46] D. Hasenfratz, O. Saukh, C. Walser, *et al.*, "Deriving high-resolution urban air pollution maps using mobile sensor nodes," *Pervasive and Mobile Computing*, vol. 16, pp. 268–285, 2015.

[47] C. Gariazzo, G. Carlino, C. Silibello, *et al.*, "Impact of different exposure models and spatial resolution on the long-term effects of air pollution," *Environmental Research*, vol. 192, p. 110 351, 2021.

[48] R. Ooka, M. Khiem, H. Hayami, H. Yoshikado, H. Huang, and Y. Kawamoto, "Influence of meteorological conditions on summer ozone levels in the central kanto area of japan," *Procedia Environmental Sciences*, vol. 4, pp. 138–150, 2011.

[49] B. Murphy, S. Müller, and R. Yurchak, *Geostat-framework/pykrige: V1.6.1*, version v1.6.1, Sep. 2021. DOI: 10.5281/zenodo.5380342. [Online]. Available: https://doi.org/10.5281/zenodo.5380342.

[50] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

Appendix A

Implementation

The implementation of the presented algorithms and the programs encapsulating them can be found on this GitHub project: `https://github.com/adriaguinovart/TFG`. The main files and their content are described next:

- `src/CrossValidation.cpp`: program containing the implementation of the cross validation used to evaluate the performance for each parameter combination in the tree graph model.
- `src/NestedCrossValidation.cpp`: program containing the implementation of the nested cross validation used to select the best tree graph model hyper-parameters.
- `src/KrigingForecast.py`: implementation of the Kriging variant presented in this thesis. This file includes code from the python library PyKrige [49], which was partially modified so that the implemented Kriging behaves as expected. It corresponds to the two weeks training and two weeks interpolation experiment (forecast).
- `src/InverseDistanceWeighting.cpp`: program containing the implementation of the inverse distance weighting interpolation method.
- `src/TreeForecast.cpp`: program containing the implementation of the tree forecast experiment (two weeks training and two weeks interpolation).
- `src/PlotNetwork.py`: program used to plot the graphs using the python library NetworkX [50].

- `src/PlotResults.py`: program used to create the results plots with the data obtained in the experiments.
- `src/utils/utils.h`: header file that defines some auxiliar methods used in different programs.
- `data/processedData.csv`: file gathering the processed data used to learn the interpolation models.