Algorithms for learning from spatial and mobility data

Maria Astefanoaei

Doctor of Philosophy
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2021

Abstract

Data from the numerous mobile devices, location-based applications, and collection sensors used currently can provide important insights about human and natural processes. These insights can inform decision making in designing and optimising infrastructure such as transportation or energy. However, extracting patterns related to spatial properties is challenging due to the large quantity of the data produced and the complexity of the processes it describes. We propose scalable, multi-resolution approximation and heuristic algorithms that make use of spatial proximity properties to solve fundamental data mining and optimisation problems with a better running time and accuracy. We observe that abstracting from individual data points and working with units of neighbouring points based on various measures on similarity, improve computational efficiency and diminish the effects of noise and overfitting. We consider applications in: mobility data compression, transit network planning, and solar power output prediction.

Firstly, in order to understand transportation needs, it is essential to have efficient ways to represent large amounts of travel data. In analysing spatial trajectories (for example taxis travelling in a city), one of the main challenges is computing distances between trajectories efficiently; due to their size and complexity this task is computationally expensive. We build data structures and algorithms to sketch trajectory data that make queries such as distance computation, nearest neighbour search and clustering, which are key to finding mobility patterns, more computationally efficient. We use locality sensitive hashing, a technique that associates similar objects to the same hash.

Secondly, to build efficient infrastructure it is necessary to satisfy travel demand by placing resources optimally. This is difficult due to external constraints (such as limits on budget) and the complexity of existing road networks that allow for a large number of candidate locations. For this purpose, we present heuristic algorithms for efficient transit network design with a case study on cycling lane placement. The heuristic is based on a new type of clustering by projection, that is both computationally efficient and gives good results in practice.

Lastly, we devise a novel method to forecast solar power output based on numerical weather predictions, clear sky predictions and persistence data. The ensemble of a multivariate linear regression model, support vector machines model, and an artificial neural network gives more accurate predictions than any of the individual models.

Analysing the performance of the models in a suite of frameworks reveals that building separate models for each self-contained area based on weather patterns gives a better accuracy than a single model that predicts the total. The ensemble can be further improved by giving performance-based weights to the individual models. This suggests that the models identify different patterns in the data, which motivated the choice of an ensemble architecture.

Lay Summary

Data with a spatial component is ubiquitous today: the location history stored on a mobile device, the number of cars passing by traffic counters, the observations recorded by weather sensors are all examples of data with associated location information. Analysing this type of data can help extract spatial patterns that inform decision making in many areas, such as infrastructure optimisation.

One fundamental task in analysing location data is computing distances between objects. These objects can be, for example, sequences of GPS locations, called trajectories, denoting the latitude and longitude of the places that someone travelled through in a given day. Given a set of such sequences, one problem of interest is to find similar trajectories, that follow the same path through a road network; in an urban context this could have applications such as discovering groups of commuters who move from one part of the city to another in the same time frame. To extract similar trajectories implies computing pairwise distances; however, exact computation is expensive, due to the number, size, and complexity of the trajectories. We devise new methods to represent trajectories that make distance computation and other similar tasks more efficient in terms of time and space used for computation.

Another application of analysing trajectory data is placing transportation resources efficiently. For example, data on cycling trips made in a city can inform how to place cycling lanes efficiently such that a large number of cyclists can use them, while respecting a limited construction budget. At the level of a city it is challenging to identify an optimal set of lanes because of the large number of trips to be analysed and the large number of candidate streets. We propose a transit network design solution for placing resources under given constraints that can handle large datasets and gives good results in practice.

We are looking at the task of predicting solar power based on numerical weather forecasts. It is known that the spatial distribution of power plants has an influence on the fluctuations in the total aggregated output. We show that having separate, local, models based on areas with similar weather conditions gives a better accuracy than a full model for the total output.

Acknowledgements

I gratefully thank my supervisor, Rik Sarkar, for giving me the chance to do the PhD and for his continuous guidance and support throughout. Thank you for all the time and effort you invested, for the patience and inspiration; I had and still have a lot to learn from you. Thank you to my second supervisor, Mirella Lapata, and to Paul Patras for the guidance during all my reviews and to my thesis examiners, Jane Hillston and Laura Nenzi for the valuable feedback on my thesis. I am very grateful to my colleagues, Panagiota Katsikouli, Abhirup Ghosh, Benedek Rozemberczki, and Lauren Watson, for the insightful conversations and shared ideas during group meetings. Thank you to everyone involved in the reading groups I was part of, Netsys, Applications of geometry and topology, Complex networks, and Algorithms, for the interesting talks, useful feedback and sense of community. Many thanks to Areti Manataki, whose passion for outreach work is inspiring.

I would also like to thank Aline Carneiro Viana and Asanobu Kitamoto for the chance to do internships in their respective labs. The opportunity to work with them gave me invaluable experience, academic, professional and cultural, that I will always cherish and that helped shape my future. Thank you to all my internship colleagues for instantly becoming friends and for making me feel at home. Special thanks to Clément Playout and Kassel Hingee, for the work on the material presented in Chapter 5 and for the beautiful friendship that resulted from this collaboration.

To the office mates I had in 3.50, Cheng, Ali, Anastasis, Ludo, Natalia, Yota, Pippa, Mousa, Ben, Ivana, Jason, Michael and Paul it was a pleasure to share this space with you. You are wonderful, hard working office mates and I am glad I got to know a bit about so many corners of the world through you. I will remember dearly Ben's explosive energy, Paul's inspiring passion for music, Michael's knowledge on absolutely everything, and all your cheerfulness that made 3.50 possibly the best office in the Forum.

Thanks to my team of thesis proofreaders: Yota, Ludo, Abhirup, Ben, Martino, Clément, Victor, Smaranda, for being so good at spotting mistakes and in general for your support and encouragement.

Many thanks to all my friends, the older ones and those I made during the PhD. Firstly, I cannot thank Ludo enough, my dear friend, office mate, flatmate and biggest supporter (even across the ocean), for being a constant positive presence during this time.

And to Yota for always having good advice and being a role model of determination. Thank you to Natalia and Ricardo, especially for taking care of me in the last few months, and sharing their home and many hobbies with me. Many thanks to Janie for being a great listener and for everything we shared, from painting evenings to trips, to Duygu for cosy evenings with Turkish food and all the encouragement calls, Nico for the inspiration to explore and learn, Martino for pasta making evenings, hugs, and linguistic facts, Joe, Kathryn, and Jenny for being amazing hosts and generally wonderful, Pablo for sharing all his knowledge, Ramon for his questions about life, Akash for all the banter, Todor for his contagious cheerfulness, Kai and Cole for the good spirits, and Rui, Stan, Andrew, Ursula, Juan, Alex, Cassia, Kate, Sam, Craig for lunches and coffee breaks, day-brightening smiles across the hallways, amazing trips, and all the time we spent at and outside work. Also thanks to friends from the Photography society and Beltane society for filling my free time beautifully with joy and learning. And thank you to all the friends in Romania, for keeping in touch and giving me a warm welcome every time we meet.

Most of all, thank you to my family, Lacramioara, Mariana, Fanica, whose impressive stubbornness was a true inspiration for finishing this work. Thank you for always being so supportive and caring.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Maria Astefanoaei*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Learning from data is the process of discovering and extracting patterns. Localised data is pervasive today, both at a granular and aggregated level. Over 5 billion people carry mobile phones[1], which are location sensing devices. They store data that can give insights about individual human mobility patterns and preferences, such as the mix of routine and exploration activities. Aggregated location data comes from a myriad of infrastructure systems, such as metro card readers, cycling counters, traffic sensing devices, cell towers, etc. Making sense of this abundance of location data has a wide reaching and direct impact - it can inform decision making in areas such as transportation, communication networks, crisis prevention and intervention, smart cities design, and others. Extracting patterns from localised data is often possible due to a certain amount of structure in human and natural processes, but not trivial, due to the inherent stochasticity and complexity that makes the patterns difficult to describe mathematically exactly.

We propose multi-resolution, scalable, approximation and heuristic algorithms for learning from large sets of geospatial data. The applications we consider are a range of machine learning tasks, such as nearest neighbour search, clustering, network design and timeseries prediction. The algorithms we present work in a bottom-up fashion, by first processing data locally. We propose different methods of abstracting from individual data points to form self-contained units based on their spatial proximity; working

[1]Source: `https://datareportal.com/global-digital-overview`

with these units reduces the computational running time and increases the accuracy of existing algorithms. The techniques we use are locality-sensitive hashing, geospatial clustering and spatial correlation and hierarchical forecasting. We first describe and motivate the choice of applications and corresponding data and in the following section we summarise the proposed algorithms for each application.

Considering the location of the process they describe, the types of data we work with are: dynamic and static. Dynamic data describes the locations of moving objects, while static data describes changing processes at fixed locations. The kind of dynamic data we analyse reflects human mobility, how people move in an urban area. The static data consists of observations of weather factors and energy power output. We motivate these particular choices from a sustainability perspective.

One of the pressing issues of the moment is climate change, caused by the buildup of greenhouse gas emissions. The two sectors that are responsible for the largest proportions of human-caused greenhouse gas emissions are transportation and electricity, each of them accounting for a half and a quarter of emissions, respectively. Both sectors are awash in quantifiable and relatively easy to source data, which makes them particularly suitable for leveraging machine learning techniques with the purpose of optimising services and diminishing negative effects on the environment.

Transportation can be decarbonised by optimising loading and routing. In an urban setting, more efficient transportation services can be achieved by analysing people's travel patterns. We consider human mobility data at the individual level - every trajectory comes from one person taking a self-contained trip. By *trajectory* we mean a series of latitude and longitude locations, which describe either the full path and its geometry, or just the origin and destination. This choice depends on the type of application the data is intended for. In some scenarios we are interested in the full path, which in the urban context would be representative for the streets and buildings a person is passing to reach their destination. In this case, most machine learning tasks for finding patterns, such as nearest neighbours or clustering, require a notion of *distance*. Computing distances between human mobility trajectories is a fundamental task that is made challenging by the size and complexity of the trajectories. To this end, we propose a *compression* technique for trajectories that makes distance computation and typical queries, such as nearest neighbours and clustering, scalable to the amount of data typically required in applications. In other cases, it is useful to abstract from the geometry of the path, and consider the trip from a utilitarian perspective - a person needed to get from one part of the city to another because they are going from home to work. The exact path they choose to take is not important, because it might change based on the resources that become available to them. *Aggregating* origin-destination trips can serve precisely this: to place resources efficiently so that a large number of people can fulfil their travel purposes. Here we are referring to transportation resources such as cycle lanes, bus routes, or metro lines. The optimal deployment of these resources can be described as *designing a transit network* for urban areas. This is a challenging problem due to multiple contradicting constraints, such as population coverage versus budget limitations. While analytical solutions are hardly scalable to the size of a city road network, heuristic machine learning algorithms are often found to give plausible solutions. Transportation is, therefore, one of the sectors where machine learning for location data can greatly optimise resource placement.

In terms of reducing emissions in the electricity sector, one area of focused interest is the transition to low carbon energy sources, of which solar, wind and hydropower are a substantial part. This transition is difficult because of the variable nature of renewable resources. Accurate power output forecasting algorithms are essential to making them a reliable, safe, and attractive alternative to high-carbon sources. In this thesis we focus on forecasting the electric power generated by solar photovoltaic systems. This depends on the level of incoming solar radiation, which in turn depends on atmospheric factors. Numerical weather forecasts have been successfully used as predictive variables and recent advances in climate modelling and weather forecasting can greatly improve solar power output prediction algorithms. However, some of the influencing weather factors, such as cloud cover, are difficult to predict accurately due to the turbulent nature of atmospheric processes. Difficult to predict weather phenomena can create fluctuations in the power output; however, it has been shown that aggregated output suffers less from large fluctuations, making it easier to predict, but the effect depends on spatial factors. We show how these factors influence a suite of machine learning models in order to guide the choice of framework for maximising performance.

Consequently, the contributions presented in this thesis focus on: compressing trajectories for efficient distance computation and extraction of travel patterns, designing a transit network by aggregating trips for optimised transportation, and improving the accuracy of output power prediction algorithms for low carbon sources of energy by considering spatial factors. We give more details about each in the next section.

1.2 Contributions overview

For geospatial data, locality is an intuitive property. We show that by grouping entities or subentities in independent, disjoint units based on their spatial proximity and using them as input alleviates the effects of noise, overfitting, and high dimensionality, leading to improvements in computational running time and accuracy. Depending on the application, we use locality-sensitive hashing, spatial clustering, and the spatial smoothing property and hierarchical forecasting.

Firstly, to understand transportation needs, efficient algorithms are needed for manipulating mobility data. Searching for similar GPS trajectories is a fundamental problem that faces challenges of large data volume and intrinsic complexity of trajectory comparison. We present a suite of sketches for trajectory data that drastically reduce the computation costs associated with near neighbour search, distance estimation, clustering and classification, and subtrajectory detection. Apart from summarising the dataset, our sketches have two uses. First, we obtain simple provable locality-sensitive hash families for both the Hausdorff and Fréchet distance measures, useful in near neighbour queries. Second, we build a data structure called MRTS (Multi Resolution Trajectory Sketch), which contains sketches of varying degrees of detail. The MRTS is a user-friendly, compact representation of the dataset that allows us to efficiently answer various other types of queries. Moreover, MRTS can be used in a dynamic setting with fast insertions of trajectories into the database. Experiments on real data show effective locality-sensitive hashing substantially improves near neighbour search time. Distances defined on the sketches show good correlation with Fréchet and Hausdorff distances.

Secondly, to encourage usage of low carbon transportation options (such as public transport, bicycles, ride-sharing) it is necessary to meet the travel demand optimally. For this purpose we present approximate algorithms to efficiently aggregate trips made in an urban setting with the purpose of understanding where transportation resources should be placed. We first formulate the problem as a bike lane problem - positioning lanes to maximise utility and meet given constraints (such as budget constraints); we show that this problem is NP-hard. We propose a fast heuristic algorithm that, given a road network, a set of locations of the origins and destinations of trips, and a set of constraints, identifies a subnetwork such that the number of trips that can use it is maximised. The algorithm is based on a novel multi-resolution clustering by projections technique. Experiments comparing the number of trips satisfied by a network generated with the proposed algorithm and the existent cycling network in London shows the efficiency of the technique.

Lastly, we present a novel method to forecast solar power output based on day ahead numerical weather predictions and show how spatial factors influence the accuracy of the predictions. The ensemble framework we propose is composed of a multivariable linear regression model, a support vector machines model and a gated recurrent unit neural network. Experiments on 2 years of solar power output data shows that the ensemble performs better than the individual models. We show that building separate models for areas with similar weather conditions has a better overall performance than a single model predicting the total output. Weighting the models differently for each area based on their training and validation performance improves the accuracy on the test set.

Machine learning tools are fundamental to understanding travel patterns, improving shared mobility, and modelling how weather factors influence low-carbon power output, all of which are essential towards sustainability. Local preprocessing and modelling improve the computation running time and accuracy of these tools. In the next section we describe how our contributions are presented in this thesis.

1.3 Structure of the thesis

In the next chapter we set the background and present work related to the methods and techniques that our proposed algorithms are based on: locality-sensitive hashing, geospatial clustering and spatial smoothing and hierarchical forecasting. We give details of how this methods are applied in relation to the applications motivated in the above section.

Chapter 3 presents compression algorithms for creating trajectory sketches. The sketches are useful for speeding up distance computation, which is at the base of many location data queries. The literature review focuses on distance metrics and methods for compression. The problem description introduces locality-sensitive hashing (LSH), which is the method we are using to build the sketches. We propose two distance metrics based on random disks deployment and show how they relate to two popular distance measures, Hausdorff and Fréchet . More specifically, we described the LSH families corresponding to each metric. We then present the construction of Multi Resolution Trajectory Sketches (MRTS) and show how they can speed up computation of typical location data queries, such as clustering. The section on experiments shows the correlation between Hausdorff, Fréchet and our proposed metrics, the speed up in nearest neighbours queries, robustness to data loss, time efficiency in distance computation and a comparison with an existing LSH method for trajectories.

Chapter 4 proposes heuristic algorithms for designing a transit network under certain constraints. The background section describes works related to network design and aggregating origin-destination trajectories for transit resources placement. We formulate the task in terms of placing bike lanes, as described in the problem description section. In the following section we present the construction algorithm that involves a multi-resolution clustering stage and a global merging stage; the steps are illustrated using a concrete example. The experiments section describes the datasets we use: the London road network, the Santander Cycles hire trips and the current bicycle lanes system in London, and the results of the comparison with our generated network.

Chapter 5 introduces a new ensemble framework for predicting solar power output from numerical weather predictions and analyses the impact of spatial factors on the accuracy of the predictions. The literature review gives a summary of current solar power prediction models and presents work on spatial correlation in relation to solar power prediction. We present in detail the datasets used for prediction, showing statistical properties and the lack of obvious patterns in the power output. We give an overview of the models used in the ensemble and explain the choice of features. In the experiments section we present the experimental frameworks and the error metrics. We show how the spatial factors influence the models and how this can be used for further accuracy improvements.

Lastly, we summarise our work in the Conclusions chapter and describe directions for future work.

Chapter 2

Background

We propose scalable approximation and heuristic algorithms that make use of spatial proximity properties of the data to improve computation efficiency and accuracy in the context of large datasets. We show how abstracting from individual data points to regions of neighbouring points can achieve performance improvements, by reducing the computational running time and the effect of noisy measurements and avoiding overfitting.

The techniques we use are locality-sensitive hashing, geospatial clustering and spatial correlation and hierarchical forecasting. locality-sensitive hashing is a hashing technique that generates similar hashes for objects similar, meaning that they are close according to some distance measure; this improves the efficiency of searching for similar objects and other similarity based tasks such as clustering. We show how this technique can be used in the context of spatial data and propose algorithms and data structures that are designed for GPS trajectories. We use geospatial clustering as an initial and intermediary step for a network design problem that we show to be NP-hard; working with spatially close groups instead of individual trajectories gives a heuristic-based solution with good results in practical applications. Finally, we show how time series prediction from multiple locations can be improved when the time series are grouped based on characteristics related to location, due to spatial correlation properties of the data. In the following sections we describe in detail each of the above techniques.

2.1 Locality-sensitive hashing

One of the most fundamental data-mining problems is finding similar data points, that are in proximity according to a given distance measure. The naive approach of computing the distance for every pair of points is prohibitively large for most real world datasets. locality-sensitive hashing (LSH) is a family of machine learning techniques that allow us to identify pairs of data points that are likely to be similar (close under a given distance measure), without considering every pair of points, therefore reducing the quadratic growth in computation time of pairwise computations. The range of applications for LSH is wide and includes finding similar audio signals, images, and text documents. We propose LSH algorithms and data structures for finding similar human mobility trajectories and patterns of movement.

General hashing is the process of transforming data of arbitrary size into fixed-sized values, referred to as *keys* or *buckets*. By indexing data with short hash keys, the look up and retrieval time is faster than by using the original data, becoming nearly constant time in the size of the dataset. Good hash functions are fast to compute and minimize the collision probability, which is the probability that two different objects are hashed to the same bucket. Conversely, the idea behind locality-sensitive hashing is to increase this probability, in the case of distinct but similar objects (see Figure 2.1). The locality-sensitive hash functions are designed such that two similar objects are more likely to arrive in the same bucket than two dissimilar objects. This allows for the search space of finding objects similar to a given query to be reduced to objects in the same bucket.

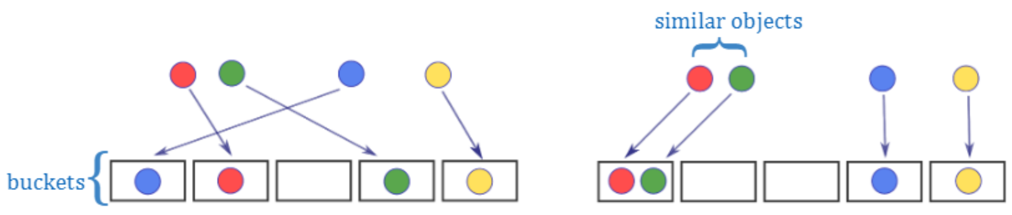


Figure 2.1: General hashing (left) and locality-sensitive hashing (right).

Formally, two objects are similar (or close) if the distance between them is at most r, and are considered dissimilar (or far), if the distance is at least $R = cr$, for $c > 1$. Define by p_1 the probability that two nearby objects are hashed to the same value (bucket) and p_2 the probability of far-away objects to collide. Any function with these properties is said to be part of the $(r, cr, p_1, p_2) - sensitive$ family F of functions.

Definition 1 (Locality-sensitive families)**.** *Let d be a distance measure defined on two objects T_1 and T_2. Given the values $r > 0$, $c < 1$, $0 \leq p_1, p_2 \leq 1$ a family $\mathcal{H}$ of hash functions satisfies the following conditions for every $f \in \mathcal{H}$:*

1. *if $d(T_1,T_2) \leq r$, then $Pr(f(T_1) = f(T_2)) \geq p_1$;*
2. *if $d(T_1,T_2) \geq cr$, then $Pr(f(T_1) = f(T_2)) \leq p_2$.*

Definition 2 (Sensitivity)**.** *A family of hash functions is (r, cr, p_1, p_2)-sensitive when the collision probabilities p_1, p_2 satisfy $p_1 > p_2$.*

Good locality-sensitive families have large p_1 and low p_2; the gap between these probabilities gives the sensitivity of the family and is measured by $\rho = \frac{\log 1/p_1}{\log 1/p_2}$. This gap can be widened through a process called *amplification*, which can be realised by *AND* and *OR-constructions*. Given a family of LSH functions F, an AND-construction builds a new family of functions F', where f' is the set of functions $\{f_1, \ldots, f_r\}$ from the family F. Then $f'(x) = f'(y)$ if $f_i(x) = f_i(y)$ for all $i \in \{1, r\}$. OR-constructions work similarly, with the difference that the previous statement is true for at least one of $i \in \{1, r\}$. The functions in any locality-sensitive family are chosen independently, therefore AND and OR-constructions give rise to (r, cr, p_1^r, p_2^r) and $(r, cr, (1-p_1)^r, (1-p_2)^r) - sensitive$ families, respectively. By choosing the number of amplifications carefully and cascading AND and OR constructions the low probability, p_1, gets closer to 0 and the high probability, p_2, gets closer to 1.

Based on a set of functions from a locality-sensitive hashing family, we can define the *sketch* of an object.

Definition 3 (Sketch)**.** *For a set of locality-sensitive hashing functions belonging to the same family, $\{f_1, \ldots, f_r\}$, the sketch of an object x is the sequence $\{f_1(x), \ldots, f_r(x)\}$.*

We propose algorithms and data structures to solve machine learning queries involving the similarity of trajectories with the use of sketches. Depending on the distance measure the task involves, we build different types of sketches. In general, not all distance measures admit a locality-sensitive hashing family. We introduce three distance measures for which we propose locality-sensitive hashing families.

Distance measures for trajectories

In what follows we consider distances between trajectories, which are simply ordered sequences of 2 dimensional points in the Euclidean plane. We consider two trajectories

$T_1 = \{u_1, u_2, \dots u_{m_1}\}$ and $T_2 = \{v_1, v_2, \dots v_{m_2}\}$, of length m_1 (number of vertices) and m_2, respectively. The distance between T_1 and T_2 is a function of the distances between pairs of points. Which pairs of points are considered in this computation depends on the chosen distance metric. A visualisation of two popular metrics can be seen in Figure 2.2.

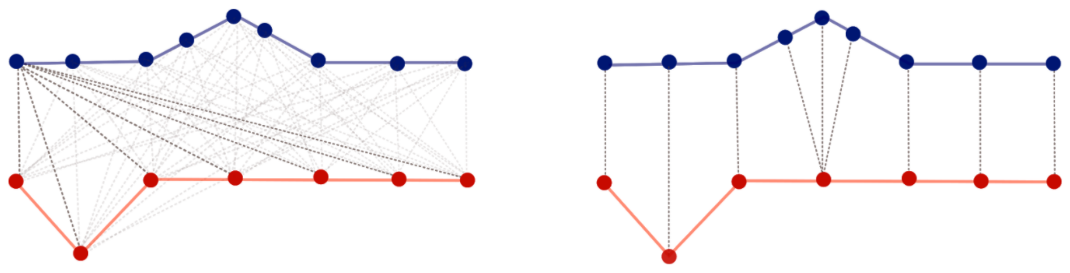


Figure 2.2: Hausdorff distance (left) is computed by considering the distances between every pair of points. Fréchet distance (right) finds pairings of points with minimum cost.

Hausdorff distance

Hausdorff distance was first introduced by Hausdorff in 1912 and is a common distance measure for curves; it is often applied in computer vision and graphics for image matching, because it gives a straightforward and intuitive way of comparing shapes. In its discrete version, it computes the maximum distance from every point on a trajectory to the other trajectory, formally:

Definition 4 (Hausdorff distance)**.** *The Hausdorff distance between T_1 and T_2 is:*

$$H(T_1, T_2) = \max\{\max_{u \in T_1} \min_{v \in T_2} d(u, v), \max_{v \in T_2} \min_{u \in T_1} d(u, v)\},$$

where d is any metric.

Here and in the following definitions $d(u, v)$ measures the Euclidean distance between the two points u and v.

The complexity of computing Hausdorff distance between the two trajectories is $O((m_1 + m_2)\log(m_1 + m_2))$ (Alt et al., 1995), using Voronoi diagrams (Huttenlocher et al., 1992). For the Hausdorff metric, Backurs and Sidiropoulos (2016) show that it admits embeddings with constant distortion and constant dimension. We propose locality-sensitive hashing families for the Hausdorff distance and prove the theoretical bounds for the collision probabilities p_1 and p_2.

Fréchet distance

The Fréchet distance is another common distance measure, that is generally believed to be more appropriate than Hausdorff for comparing curves in a plane (Alt et al., 1995). As opposed to the Hausdorff distance, it takes into account both the location and the ordering of the vertices in the curves. It is popularly called the "dog leash distance" - intuitively, it describes the length of the shortest leash that a person traversing a finite curve should have for their dog, if both the person and the dog can vary their speed. Formally, to define the Fréchet distance we use the concept of traversal (as in Driemel and Silvestri (2017)):

Definition 5 (Traversal)**.** *Given two trajectories T_1, T_2, of sizes m_1, m_2 respectively, a traversal $\tau = \{(i_1, j_1), (i_2, j_2), \cdots, (i_l, j_l)\}$ is a sequence of pairs of indices referring to a pairing of vertices from the two curves with the properties:*

1. $i_1 = 1,\ j_1 = 1,\ i_l = m_1$ *and* $j_l = m_2$
2. $\forall (i_k, j_k) \in \tau : (i_{k+1} - i_k) \in \{0, 1\} \wedge (j_{k+1} - j_k) \in \{0, 1\}$
3. $\forall (i_k, j_k) \in \tau : (i_{k+1} - i_k) + (j_{k+1} - j_k) \geq 1.$

An example of traversal is shown in Figure 2.2. Intuitively, a traversal is any pairing of vertices from the two curves in which no vertex is left out and all relations are either one-to-one or many-to-one. The *cost* of a traversal is the maximum distance between paired vertices. The discrete Fréchet distance is the minimum cost amongst all traversals.

Definition 6 (Discrete Fréchet distance)**.** *Let $\mathcal{T}$ be the set of all traversals between two trajectories T_1 and T_2. The discrete Fréchet distance $F(T_1, T_2)$ between them is:*

$$F(T_1, T_2) = \min_{\tau \in \mathcal{T}} \max_{(i_k, j_k) \in \tau} d(u_{i_k}, v_{j_k}),$$

where $d(u_{i_k}, v_{j_k})$ is the distance between the vertices with $u_{i_k} \in T_1$ and $v_{j_k} \in T_2$.

The running time for computing the Fréchet distance between the two trajectories T_1, T_2 is $O(m_1 m_2)$, which makes the naive algorithm for similar items search expensive. As discussed in Indyk et al. (2004), there is no trivial low dimensional embedding for the Fréchet distance. Driemel and Silvestri (2017) were the first to propose locality-sensitive hashing families for the Fréchet distance. Their method snaps trajectory vertices to vertices in grids, thus reducing the space of possible point coordinates to the coordinates of the grid vertices. The sequence of grid vertices corresponding to each trajectory serves as the hash. We propose a different method for building the hash families that uses randomness to further improve the collision probabilities and sizes of the hashes. We then use these hash families to build data structures that speed up the computation of typical machine learning queries for trajectories, such as clustering or nearest neighbours.

Dynamic time warping distance

Dynamic time warping (DTW) is a universal measure of similarity for timeseries that can vary in speed. Applications include matching of temporal sequences of video (Reyes et al., 2011), scoring audio alignments (Kaprykowsky and Rodet, 2006), clustering vessel trajectories (de Vries et al., 2012). Similarly to Fréchet distance, it considers all traversals between two curves, however, the cost of a traversal is computed by summing up the distances between pairs of vertices instead of taking the maximum. Formally, we defined the DTW distance as:

Definition 7 (Dynamic time warping distance)**.** *Let $\mathcal{T}$ be the set of all traversals between two trajectories T_1 and T_2. The dynamic time warping distance $DTW(T_1,T_2)$ between them is:*

$$DTW(T_1,T_2) = \min_{\tau\in\mathcal{T}} \sum_{(i_k,j_k)\in\tau} d(u_{i_k}, v_{j_k}),$$

where $d(u_{i_k}, v_{j_k})$ is the distance between the vertices with $u_{i_k} \in T_1$ and $v_{j_k} \in T_2$.

The computation time is the same as for the Fréchet distance, $O(m_1m_2)$. In general, for similarity search for DTW there are no data structures with provable guarantees, but several works propose heuristics (Chen et al., 2009; Rakthanmanon et al., 2012). While proving theoretical bounds for the hashing families for DTW distance is outside the scope of this work, we show experimentally that the type of families proposed for Fréchet distance are suitable in practice for DTW.

Distance measure for sequences

In our proposed method we compute sketches for trajectories based on the distance of interest. Comparing sketches instead of full trajectories drastically reduces the amount of computation for finding similar trajectories, partly because of the limited size of the sketches alphabet. We utilise two types of distance measures suited for sequence comparison.

Definition 8 (Hamming distance)**.** *The Hamming distance between two sequences is the number of positions where the symbols are different.*

Definition 9 (Edit distance)**.** *The edit distance between two sequences is the number of insertions, deletions, and substitutions needed to transform one sequence into the other.*

We show how to build the sketches and how the distances between them are computed in Chapter 3.

2.2 Geospatial clustering

A different technique that we use to preprocess data locally is *geospatial clustering*. In general, *clustering* is the task of grouping data points such that elements of a group are more similar to each other than to the elements in other groups. For geospatial clustering the data points have a physical location, which can be described by geophysical coordinates. The data points can refer to actual points, but also line segments, polylines, curves, etc. The definition of similarity is task-dependent. It can be described by common distance measures, such as Euclidean (for points) or Hausdorff (for line segments), or by more complex functions of distances; for example, in some context, one may consider that two curves are similar if the endpoints are geographically close, without any assumption about the rest of the curves.

Typically, spatial clustering algorithms can be split into *partition* and *fuzzy clustering*, depending on whether an object belongs to a single or multiple clusters. Based on the clustering method, we can further categorise spatial clustering algorithms as:

1. Hierarchical
2. Density-based
3. Model-based.

Hierarchical clustering algorithms work by either aggregation or division. In *agglomerative* hierarchical algorithms every data point (object) starts by being a cluster; in the next stages clusters are merged based on proximity. *Divisive* algorithms work in top-down approach, by separating the points. These approaches are often computationally expensive, especially in the presence of complex distance measures between entities (Nanni, 2005).

Density based algorithms work by finding areas where data points are concentrated and marking as noise points that are in sparse areas. One such popular algorithm is DBSCAN (Ester et al., 1996), that is widely used in spatial data mining applications and beyond, such as weather forecasting (Dey and Chakraborty, 2015).

In model based clustering algorithms the assumption is that the data comes from a mixture of probability distributions, each corresponding to a cluster (Fraley and Raftery, 2002). The algorithms then optimise the fit between data and the set of models using likelihood approaches. This clustering method has several advantages, such that it allows for clusters of different sizes and volume. However, results are heavily dependent on assumptions about the underlying model of the data.

We use geospatial clustering as a heuristic for a network design problem that we show to be NP-hard. We propose a new type of fast projection based clustering of trajectories (expressed as origin-destination pairs of locations) that reduces significantly the running time, while giving good results in practice.

2.3 Spatial correlation and hierarchical forecasting

Hierarchical forecasting of time series data refers to the strategy of grouping time series at multiple resolution levels. When time series have location information attached, exploiting the inherent aggregation structures of spatial data points can lead to improved prediction accuracy. For example, predicting temperature time series in a narrow area is more prone to errors than predicting the average for a large area, but not necessarily if the larger area covers a variety of weather conditions. For temperature, an effective spatial resolution depends on the location characteristics (varying between 5km to 18km for Mediterranean and European continental cities, as shown by Giunta et al. (2019)). A similar effect can be seen in the case of predicting solar power production.

One difficulty in predicting solar power production comes from the sudden changes in the output due to the rapid and stochastic movement of clouds. This can be circumvented by considering ensembles of power plants rather than individual ones. In certain cases the aggregated output is more smooth depending on the geographic distribution of the power plants: for a wider distribution the variability in the production is reduced (Graabak and Korps, 2016). This effect, known as the *smoothing effect*, comes from the delay in weather patterns that manifest themselves at different times at different sites (Ackermann and Sder, 2002). Over a large area the changing weather patterns bring a slower rate of change, as opposed to small areas where the fluctuations happen sequentially, giving a less smooth output curve. The smoothing effect was demonstrated by Wiemken et al. (2001) and Widen (2011) for solar power plants in Germany and Sweden, respectively. One main finding of the first study is that the standard deviation curve of the average daily power production decreases significantly for an ensemble of PV plants; the effect is driven by the spatial distribution, rather than the number of sites. Of course, the scale of the effect depends on the temporal and spatial resolutions. The experiments were run on 10 years of radiation data from 6 stations (with distances between 200km and 680km). The spatial cross-correlation of the hourly data shows an exponential decay as a function of interstation distance. Spatial correlation and the smoothing effect have important consequences for both the placement of power plants and for forecasting algorithms.

Fonseca Junior et al. (2014) showed that the smoothing effect due to spatial correlation has a similar impact on the accuracy of predictions of a SVM forecasting model as applying a dimensionality reduction method on the input data. Similarly to the models we propose, their models use numerical weather forecast at the location of each power plant. The hourly power output was predicted for 4 regions in Japan; these follow the traditional geopolitical regional division. The authors show that applying principal component analysis (PCA) to remove redundant information in the weather data gives a 10% improvement in accuracy for the proposed prediction model, but the improvement is dependent on the type of region. One important factor is the diversity of the climate in that region.

In this work we propose an ensemble model that predicts the day ahead total power output time series for a region with 15 solar power plants. We show that building separate models for two subregions with similar weather conditions gives a higher accuracy than having a model that directly predicts the total. Moreover, weighting the models based on their performance on the training sets corresponding to the two areas further improves the accuracy of the ensemble.

Chapter 3

Multi-resolution sketches and locality sensitive hashing for fast trajectory processing

3.1 Introduction

Location analytics is a fundamental aspect of insights from GPS, sensor and mobile phone data, with applications in various domains ranging from social sciences, communication networks to smart cities and transport. Effective analysis depends on fast response to basic queries, for example, finding the nearest neighbour (or all near neighbours) of a given trajectory. Ability to efficiently update the query database – e.g., inserting a new trajectory or adding new points to an existing trajectory – helps one to handle streams of trajectory data continuously updated in real time.

Trajectories are usually compared using Hausdorff distance (Atallah, 1983), Fréchet distance (Eiter and Mannila, 1994), or the dynamic time warping (DTW) distance (Sakoe and Chiba, 1978). The last two are most popular as they incorporate the intrinsic sequential nature of the trajectories in the comparison. However, these distances are computationally challenging to apply to large datasets. Dynamic programming algo-

A version of this chapter is published in Astefanoaei et al. (2018).

rithms to compute Fréchet and DTW between two trajectories of length m require $O(m^2)$ time per comparison, which makes them prohibitive in large datasets.

Nearest neighbour searching on points in the *exact* version require either linear query time or space exponential in dimension, popularly referred to as the *curse of dimensionality*. Recently it was proven (Alman and Williams, 2015) that a query time sublinear in n (number of instances) for the batched Hamming nearest neighbour problem would refute the Strong Exponential Time Hypothesis (SETH). With such barriers, researchers turned to *approximate* nearest neighbour search. Formally, a c-approximate nearest neighbour algorithm will return an item whose distance to the query is at most c times the true nearest neighbour of the query item. Indyk and Motwani (1998) were the first to develop Locality-sensitive hashing (LSH) for this approximation problem, and it was subsequently improved upon in various works to obtain a query time sublinear in n, and polynomial dependence in d in all parameters (see for example: Gionis et al. (1999); Indyk (2001); Charikar (2002); Panigrahy (2006); Andoni and Razenshteyn (2015)). The main idea in LSH is to construct a hash function that ensures that similar items are usually hashed to the same buckets, while dissimilar items are hashed to different buckets.

However, unlike point datasets, where one can exploit the bounded doubling dimension of the underlying metric space (Arya et al., 2008; Backurs and Sidiropoulos, 2016), compressing trajectory data while preserving these distances is difficult in general, when no constraints are imposed on trajectories: the space of trajectories under Fréchet distance does not have a bounded doubling dimension and does not easily admit a low dimensional embedding (Driemel et al., 2016).

LSH for trajectories. Indyk (2002) describes a version of locality sensitive hashing for product spaces that apply to Fréchet distances. However, this approach has an immense space requirement, which makes it impractical in GPS datasets. A recent work by Driemel and Silvestri (2017) gives simpler mechanisms for locality sensitive hashing of trajectories. This method works by snapping the vertices of trajectories to grids, so that trajectories with Fréchet distance much smaller than the grid size are likely to snap to the same vertices. The vertex order on the grid then serves as a locality sensitive hash.

Apart from near neighbour queries, other important research problems in trajectory datasets include quickly estimating the distance between two trajectories in the dataset, clustering trajectories, or detecting whether one trajectory is similar to a sub-trajectory of another, etc. The above works do not focus on such problems. Furthermore, it is highly desirable to obtain a data structure that can support fast insertions of new trajectories into the dataset, and also one where the user can specify a desired degree of detail.

Our contributions. We devise a hierarchy of sketches for trajectories, that not only perform the basic tasks of locality sensitive hashing and near neighbour search, but can also be used to directly estimate distances between trajectories, cluster trajectories, detect subtrajectories, etc. Our sketches and hash families have the added advantage of being simpler than existing techniques.

Our *basic* sketch is simply a bit vector representing the intersection of the trajectory with a randomly deployed set of disks in the plane (see Figure 3.1); a bit element is 1 if the trajectory intersects the corresponding disk and 0 otherwise. This bit vector can act as a simple locality sensitive hash with provable probability guarantee. The main insight is that when disks are deployed randomly, similar trajectories are likely to intersect a similar set of disks. A simple measure of distance using this sketch is the Hamming distance giving the number of bits where two sketches differ. The *ordered* version of this sketch is a sequence representing the order in which the trajectory enters and exits the disks. These ideas are shown in Figure 3.1. Measures such as edit distance (Levenshtein, 1966) can be used here to find how similar two sketches are. In the *hierarchic* version, we deploy disks of a different size at each level of the hierarchy, with the “highest” level consisting of the largest disks.

In real data analysis, our scales of interest, and therefore the size of the hierarchy are bounded by the application and the platform. For example, GPS localisation measurements have errors up to a few meters, and thus *closeness* of GPS data is not significant at scales smaller than this size. Similarly, beyond a certain distance, for example hundreds of kilometres, data points can simply be treated as *far*, since applications such as near neighbour search are not useful beyond such distances. Let us call these lower and upper bounds L and U.

We make use of these bounds to get theoretically rigorous results of practical importance. With trajectories d distance apart localised in a region of area A we can show the following:

- there is a constant factor approximation of the distance between them in time

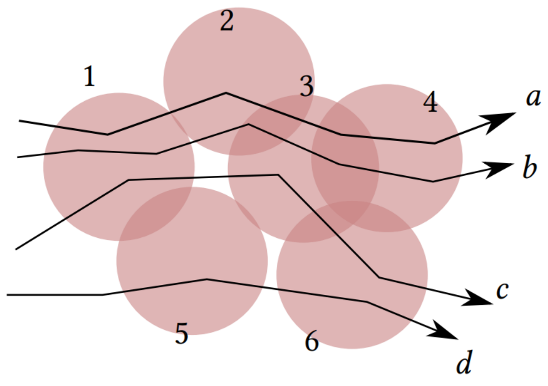


Figure 3.1: Trajectory sketches with respect to randomly deployed disks. Similar trajectories like a and b are likely to pass through same set/sequence of disks. For trajectory a, the basic unordered sketch is 111100, the ordered sketch recording entry and exists is $1,2,3,2,4,3,4\ldots.$

$O(\min(m, \log(U/d)\log(A/d^2)))$

- a new trajectory can be inserted into the sketch database in $O(m\log(A/L^2))$ time
- an $O(m)$ approximation to the nearest neighbour query can be computed in $O(m\log^2 n)$ time.

Experimental results Experimental results on two real datasets show that the distance metric based on the basic unordered sketch is strongly correlated with Hausdorff distance, and the distance based on the ordered sketch is correlated with Fréchet and DTW distances. The correlation is shown by comparing the Hausdorff and Fréchet distances with our proposed metrics for a set of trajectories with varying distances. The correlations hold even when a large proportion of the data is missing, for example due to sensing/communication failure or lack of resources. The sketch-based distances can be computed orders of magnitude faster than Hausdorff, Fréchet , or DTW.

We found that a flat structure – about 50 disks of $2km$ radius for areas between $10km^2$ and $14km^2$ – gives good practical results. Observe that with bit vector size of 50, Hamming distance or matching of hash can be made extremely fast. We used this feature for data pruning in nearest neighbour search. Using the sketch, we identified trajectories of same or similar sketch for comparison, and discarded everything else. Then we applied Hausdorff and Fréchet distance computation on the small selected set. The sketch achieved a pruning ratio of about 80% (fraction of discarded items), while achieving accuracy of 80% – where the true nearest neighbour was found in the selected set. As a result of the high pruning ratio, on large datasets, this process runs an order of magnitude faster.

In other experiments we studied the effect of size and number of disks used. We also computed the error in distance to the nearest neighbour in the multi-resolution version of the sketch and found this to be usually low.

The rest of the paper is organised as follows. In Section 2 we present related works. In Section 3 we describe the type of queries that we are considering. In Sections 4 and 5 we describe our sketches and prove that they provide an LSH family (thus solving the near neighbour problems). In Section 6 we describe the MRTS (Multi Resolution Trajectory Sketch) data structure and show how to handle insertions, distance estimation, classification and clustering queries. Section 7 contains our detailed experimental results.

3.2 Related work

Distances between curves are challenging to compute. For inputs of length m, Fréchet distance $F(\cdot)$ and DTW distance $DTW(\cdot)$ can be computed in $O(m^2)$ (Alt and Godau, 1995; Sakoe and Chiba, 1978). Recent results show that assuming strong exponential time hypothesis, strongly subquadratic time algorithms are impossible for Fréchet (Bringmann, 2014) . Thus, in datasets of n trajectories, simple near neighbour search will run in $O(nm^2)$ time. Indyk's approach to LSH using product metrics (Indyk, 2002) achieves an approximation of $O(\log m + \log\log n)$ and nearest neighbour query time of $O(m^{O(1)} \log n)$ using $O(|X|^{\sqrt{m}}(m^{\sqrt{m}}n)^2)$ storage, where X is the domain in question – loosely, the area of the region containing the trajectories. This large data structure to be constructed and stored, and the resultant query time polynomial in m are the main challenges of using this method in GPS datasets with large m.

The recent work by Driemel and Silvestri (2017) uses more intuitive methods to achieve locality sensitive hashing for Fréchet and DTW. The first strategy proposed is to snap trajectories to a grid placed with a random translation. The *hash* of a trajectory in this system is given by the sequence of vertices its snapped version traverses. For two trajectories P and Q, the probability of having identical hash depends on how the grid cell size δ compares to length m and the Fréchet distance $d_F(P,Q)$. In the plane, the probability of identical hash can be shown to be $\left(1-\frac{4md_F(P,Q)}{\delta}\right)$. In this scheme, it is possible to construct an LSH with approximation factor linear in m for parameters suitably chosen for a pair of trajectories. A different scheme in (Driemel and Silvestri, 2017) proposes to apply a fixed sequence of random perturbations to the vertices of a trajectory P before snapping them to nearest grid points. The query undergoes a different perturbation per vertex before snapping. This scheme achieves a constant approximation factor, but the probability of hashes matching for two similar trajectories can be shown only to be more than $\approx 2^{-4m}$, which is impractically small for datasets with long GPS trajectories. Since this scheme is meant mainly for answering near neighbour queries, it does not provide answers to various other types of queries one may be interested in, e.g., quickly estimating the distance between two trajectories, clustering, subtrajectory detection, etc.

Other works in managing large trajectory datasets include methods of simplification, where the goal is to construct an approximate curve that is close to the original. The Douglas-Peucker algorithm (Douglas and Peucker, 1973) is possibly the most popular algorithm of this type, which works well in many practical cases. A recent work uses topological persistence to simplify trajectories online, and find the significant turns at different resolutions (Katsikouli et al., 2014). Compact sketches based on topology of the domain have been developed by Ghosh et al. (2018). These methods treat trajectories individually, and do not provide any guarantees on distance computation or search in datasets. The power of large cluster computers has been exploited by Xie et al. (2017) by developing indexing for segmented trajectories. In-network mining for popular subtrajectories is considered by Katsikouli et al. (2018).

In contrast to these works, our focus was to speedup trajectory processing by building highly compressed summaries. This enabled us to rapidly estimate distances between trajectories and prune large fractions of data when searching for similar trajectories.

3.3 Problem description and background

We let A denote the area of the region (typically a square or a rectangle) of interest that contains all the trajectory data. Let n denote the number of trajectories in the dataset. When comparing two trajectories, we will use the term “distance between them” to mean either the Hausdorff or the Fréchet distance (defined in Chapter 2).

We define a trajectory T of length m as a sequence of ordered embedded vertices $T = \{v_0, v_1, \dots v_m\}$. We do not include other information than the order of the vertices, so essentially we treat trajectories as curves. The discrete representation simplifies the application of algorithms directly to the data without need for interpolation.

We assume an upper bound U and a lower bound L on the distances of interest. This means that we only care about the actual distance between a pair of trajectories if their distance is between L and U. For other pairs of trajectories, i.e., trajectories that are farther than U apart or within L of each other , we just want to detect if this happens, so if they are either farther or within the distance. We partition the interval $[L,U]$ into subintervals, depending on the degree of detail desired by the user.

Let a be a user-specified constant larger than 1 (otherwise we set $a = 2$) and let $l = \log_a(U/L)$. Define $r_i = U/a^i$ for $i \in \{0,\dots,l\}$. Thus $r_0 = U$, and the r_i's decrease geometrically until $r_l = L$. These r_i will serve as partitions of $[L,U]$ – the smaller the r_i, the greater the accuracy.

We provide a data structure that performs the following operations efficiently for a given trajectory T (see Table 3.1 for the full set of notations):

1. (c,r)**-near Neighbour queries**: if there exists a trajectory in the dataset within distance r, return a trajectory in the dataset that is within cr of T. Our methods allow a linear approximation $c = O(m)$ answer with high probability in $O(m \log n)$ time.

2. c**-approximate nearest neighbour queries**: return a trajectory whose distance to T is within c times the distance of T to its nearest trajectory in the dataset. We can answer the c-approximate nearest neighbour queries in $O(m \log^2 n)$ time.

3. **Clustering/classification**: we can partition the dataset of n trajectories into $P_1, \cdots, P_l$, where trajectories in the set P_i are within r_i and at least r_{i+1} away from the query trajectory T. The entire partitioning takes $O(l \min(m, \log(A/L^2)))$ time. To compute only P_i for a given i, our data structure takes $O(\min(m, \log(A/L^2)))$ time.

4. **Distance estimation**: Given another trajectory T', we can compute a constant factor approximation of the distance d between T and T' in $O(\min(m, \log(U/L)\log(A/d^2))$ time, where A is the area of the region where the trajectories are deployed.

5. **Subtrajectory detection**: Given another trajectory T', we can identify if it is

similar to a subtrajectory of trajectory T in time $O((\min(m, \log(A/L^2)))^2)$. Notice that a subcurve of a curve may have a large Fréchet or Hausdorff distance to the original curve, and so such queries are not merely distance queries.

6. Finally, the data structure handles **fast insertions** of trajectories to the dataset. Inserting a new trajectory of length m into our data structure takes $O(m\log(A/L^2)\log(U/L))$ time (Lemma 3).

Symbol	Description
T	trajectory (sequence of GPS locations)
m	length of trajectory
n	number of trajectories
A	area of the map
$H(T_1, T_2)$	Hausdorff distance between trajectories T_1 and T_2
$F(T_1, T_2)$	Fréchet distance between trajectories T_1 and T_2
D	number of disks
r	radius of a disk
$S_{D,r}(T), S(T)$	binary sketch
$OS_{D,r}(T), OS(T)$	ordered sketch
$d_{D,r}$	distance between sketches (binary or ordered)
U, L	upper and lower bounds on the degree of detail
ℓ	number of layers
L_i	layer i
D_i	number of disks on layer i
r_i	radius of disks on layer i
A_{dif}	area of the symmetric difference
EST_{kF}	estimator of kF

Table 3.1: Table of notations

Locality-sensitive hashing. Solutions for approximate nearest neighbours often employ Locality-sensitive hashing (LSH) algorithms which ensure that the probability of two objects being attributed to the same hash is high for similar objects and low for substantially different objects. A family of hash functions is a collection of mappings that are all defined on the same sets: mappings from the set of all objects, which is of arbitrary size to a set of fixed size (the set of “buckets”). See Chapter 2 for formal definitions for locality-sensitive hashing families and the related concepts.

Distance Measures. We define locality-sensitive hashing families for the Hausdorff and Fréchet distances and prove theoretical bounds on the collision probabilities; we show experimentally that a similar method gives good results in practice for DTW.

3.3.1 LSH families and the near-neighbour problem

Given a (r, cr, p_1, p_2)-sensitive hashing family, there is a standard framework for solving (c, r)-near neighbour queries. Using this framework, one can also solve the c-approximate nearest neighbour queries. We briefly describe it here, and refer the reader to more comprehensive descriptions (Indyk and Motwani, 1998; Driemel and Silvestri, 2017) for details. First, we construct a new family $\mathcal{H}'$ of hash functions by concatenating $\ell = \max\{1, \log_{p_2} 1/n\}$ hash functions. This decreases the collision probability to at most $1/n$. We then choose $k = (1/p_1^{\ell})$ hash functions from the family $\mathcal{H}'$, and insert each trajectory into k hash tables. This completes the preprocessing phase. Once the query arrives, we search among all trajectories that collide with the query in the k hash tables, and compute the distance of the query trajectory to such trajectories. We can stop as soon as a trajectory within cr distance is found, or in the reporting version, report all trajectories within cr. The space and query time of such a data structure is governed by the parameter $\rho = \log p_1 / \log p_2$. The space used is $O(n^{1+\rho} + nm)$ and the query time is $O((m \log m) n^{\rho})$ for Hausdorff distance and $O(m^2 n^{\rho})$ for Fréchet distance.

For the LSH families we derive in this paper far away trajectories never hash to the same bucket. In this case, the query time is $O(m)$. These query times are for the algorithm to work with a constant probability. To get (c, r)-near neighbours with probability at least $1 - 1/n$, we can repeat the above process $\log n$ times, leading to an extra logarithmic overhead in the space and query time. Given a data structure to solve the (c, r)-near neighbour problem, one can use the concept of ring trees as described by Indyk and Motwani (1998) to solve the c-approximate nearest neighbour problem.

3.4 Compact sketches and LSH for near neighbours

We propose a simple strategy that allows us to build LSH schemes for Hausdorff and Fréchet distances. The sketches are based on randomly deploying disks on a map and considering their intersections with trajectories. We then define metrics on these sketches that are related to the distances between the corresponding trajectories.

3.4.1 Binary sketches for Hausdorff distance

Our approach is based on the observation that if two trajectories T_1, T_2 are such that the Hausdorff distance $H(T_1, T_2)$ is small, then they both go through approximately the same neighbourhoods on the map.

Fix a radius $r > 0$, and let $D = \Theta(A/r^2)$. We define our *sketch* $S(T)$ of a trajectory T as the record of its intersection with D random disks on a map:

Definition 10 ((D,r)-Binary Sketch)**.** *The Binary Sketch of a trajectory T is the binary vector $S_{D,r}(T) = e_1 \dots e_D$ of length D, defined in terms of a set of D random but fixed disks of radius r. The vector element $e_i = 1$ if the disk i intersects trajectory T, otherwise it is* 0.

We define the measure of distance between the sketches as the number of bits that are different:

Definition 11 ((D,r)- Binary Sketch Distance)**.** *The Binary Sketch Distance between two trajectories T_1, T_2 is the Hamming distance (Hamming, 1950) between their (D,r)-Binary Sketches. That is, the number of indices with different entries: $d_{D,r}(T_1,T_2) = \left|\{i : e^1_i \neq e^2_i\}\right|$.*

In other words, this is the L_1 norm of the binary difference of the two vectors and it represents the number of disks that intersect exactly one of the trajectories.

3.4.1.1 Data structures to speed up sketch computation

Let T be a trajectory of length m. In $O(mD)$ time we can test each disk for every point in the trajectory, and retrieve the set of disks intersected by the trajectory, i.e., the position of 1s in the sketch vector. However, when D is large (recall that $D = \Theta(A/r^2)$), we can actually do better.

Notice that for a given point p on the trajectory, we want to quickly determine which of the D disks of radius r contain p. This is essentially a range searching problem: Let Q denote the centres of the D disks, and B denote a ball of radius r around p. A *circular range query* for B determines which points in Q lie inside B. The disks corresponding to those points are exactly those that contain p (the centre of B). Using known range searching data structures from computational geometry (Matousek, 1992; Aggarwal et al., 1990), this can be accomplished in $O(\log D + k)$ time, where k is the number of disks containing p. In summary:

Observation 1. *Given a set of D disks of radius r and a trajectory T of length m, the set of k disks that intersect T can be computed in $O(m \log D + k)$ time.*

3.4.1.1.1 Computing the binary sketch distance. For large r such that $D = O(m)$, one can compute the Hamming distance between sketches of length D in the straightforward manner. For small r, D is larger than the number of disks intersected by the trajectory (which will never be more than m, but could be significantly smaller), resulting in sparse bit vectors. When D is large enough that the vectors are very sparse, we do not store the entire sketch but just the indices of the disks intersected by the trajectory. Assume that no trajectory intersects more than I disks, where $I << D$, and $I = O(m)$. Then we can actually compute the binary sketch distance in $O(I \log I)$ time using a set intersection query, where the two sets are the sets of disks intersected by each trajectory. Thus, computing the distance takes at most $O(\min(D, m \log m))$ time, and is typically significantly smaller; e.g. for uniformly sampled trajectories, this can be done in $O((m/r)\log(m/r))$ time.

3.4.1.2 Binary sketches provide an LSH family

Given two trajectories in an area of size A and a disk of radius r, the LSH family is defined in the following theorem:

Theorem 1. *Let T_1, T_2 be two trajectories of lengths m_1 and m_2 respectively, intersecting at least one disk. Let $m = \min(m_1, m_2)$ and $c > 1$ a constant. Then the following are true:*

1. *If $H(T_1, T_2) < \frac{2r}{c}$ then $P(S(T_1) = S(T_2)) > 1 - \frac{8\pi r^2 Dm}{cA}$*
2. *If $H(T_1, T_2) > 2r$ then $P(S(T_1) = S(T_2)) \leq 1 - \frac{2\pi r^2}{A}$.*[1]

[1]Thank you to Haotian Wang for the correction.

In order to prove the theorem, we need a geometric lemma.

Lemma 1. *The probability that a randomly placed disk separates two vertices at distance h is bounded by $4\pi rh/A$.*

Proof. The disk separates the vertices when the centre lies in the symmetric difference of the disks of radius r centred at the two vertices. Let us call this the *symmetric difference area.* This is shown pictorially in Figure 3.2.

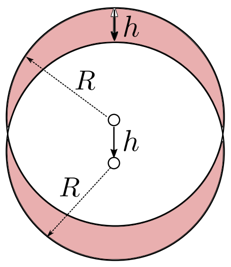


Figure 3.2: *Symmetric difference area:* A disk of radius R separates the two vertices if and only if its centre lies in the symmetric difference of disks at those vertices given by the shaded area.

This area is bounded by $4\pi rh$. Thus the probability that a particular disk x separates the vertices, is at most $4\pi rh/A$.

□

Proof of Theorem 1. (1) For every $u \in T_1$ there is a $v \in T_2$ such that $d(u,v) \leq H(T_1,T_2) = H$, and vice versa. Let us refer to this set of pairs as the *closest pairs*. Any disk of radius r that contains u but not any vertex from T_2, must also not contain v, and therefore must have its centre in the symmetric difference area for u and v. This applies for all closest pairs. Thus, any disk i such that $d_i^1 \neq d_i^2$ must lie in the symmetric difference of at least one closest pair. There are m closest pairs and any closest pair is separated by a distance at most H. By union bound, the probability that the centre of a disk is in a symmetric difference area is bounded by $(4\pi mrH)/A$. Substituting $H < 2r/c$ gives the desired bound. To prove (2), we note that if the distance is larger than $2r$, then there is at least one pair where the distance is larger than $2r$. The probability that the sketches are the same is the probability that no disk intersects either of the points in the pair, which is at most $1 - \frac{2\pi r^2}{A}$.

Observe that setting $D = \Theta(A/r^2)$ and $c = O(m)$ gives us a constant probability of collision, and thus by the standard framework of (c,r)-near neighbour data structure, we obtain the desired space and query bounds.

3.4.2 Ordered sketches for Fréchet distance

As opposed to Hausdorff, the Fréchet distance between trajectories takes into account the ordering of the vertices. Therefore, we propose sketches that maintain the order of the intersections with the disks.

Definition 12 (Ordered Sketch)**.** *Let T be a trajectory and $\mathcal{D}$ the set of disks spread on the map. We define by the Ordered Sketch the sequence of indices of the disks that T enters and exits. We denote it by $OS(T)$.*

Figure 3.1 shows a trajectory a passing through disks $1,2,3,4$. The trajectory first enters disk 1, exits 1, enters disk 2, enters disk 3 and so on.

We define a measure of distance (dissimilarity) and also a measure of similarity between two ordered sketches, and therefore, between their corresponding trajectories. These will help us answer distance related queries when we describe our Multi Resolution Trajectory Sketch (MRTS) data structure.

Definition 13 ((D,r)-Ordered Sketch Distance)**.** *The Ordered Sketch Distance between two trajectories T_1,T_2 denoted by is the edit distance between their (D,r) Ordered Sketches. That is, the number of insertions, deletions and substitutions required to transform one ordered sketch into another.*

Definition 14 ((D,r)-LCS measure)**.** *The LCSM (Longest Common Subsequence Measure) between two trajectories T_1,T_2 is the length of the LCS between their (D,r) Ordered sketches.*

3.4.2.1 Ordered sketches provide an LSH family

We first find the probability of collision of two trajectories in terms of their Fréchet distance.

Lemma 2. *Given two trajectories T_1,T_2 with m_1,m_2 vertices, $m = \min(m_1,m_2)$ and D disks of radius r the probability that they hash to the same sequence is bounded by:*

$$P(OS(T_1) = OS(T_2)) \geq 1 - \frac{4\pi rDm}{A} F(T_1,T_2).$$

Proof. Suppose the trajectories do not hash to the same sequence. From Lemma 3 in (Driemel and Silvestri, 2017) we know that an optimal traversal T will separate the trajectories in at most m components, each of them a star. Intuitively a star component has one vertex on one trajectory (traversal is static on this curve), and some vertices on the other trajectory (where the traversal is happening). We consider E_k to be the event that a disk separates a pair of vertices in the component k.

Since the component is a star, there must exist vertex v such that v is connected to all other vertices in the component. All the lengths are less than $F(T_1,T_2)$. The probability that any pair is separated is less than $4\pi rF(T_1,T_2)/A$ (this is just the symmetric difference area - similar to what we had before). There are at most m components in the traversal and D disks, therefore, by union bound, the probability that the hashes differ is bounded by $\frac{4\pi rDm}{A}F(T_1,T_2)$. □

This provides us with the following locality sensitive hash family.

Theorem 2. *Let T_1,T_2 be two trajectories of lengths m_1,m_2, intersecting at least one disk. Let A be the area of the region, r the radius of the disk, D the number of disks, $m = \min(m_1,m_2)$ and c a constant. Then the following are true:*

1. *If $F(T_1,T_2) < \frac{2r}{c}$ then $P(OS(T_1) = OS(T_2)) > 1 - \frac{8\pi r^2 Dm}{cA}$*
2. *If $F(T_1,T_2) > 2r$ then $P(OS(T_1) = OS(T_2)) \leq 1 - \frac{2\pi r^2}{A}$.*

Proof. For the second part, if the distance is larger than $2r$, then there is at least one pair where the distance is larger than $2r$. The probability that the sketches are the same is the probability that no disk intersects either of the points in the pair, which is at most $1 - \frac{2\pi r^2}{A}$. For the first part we use Lemma 2.

□

3.4.2.2 Choice of parameters

In applications, we assume the user chooses the values for $2r$ and $\frac{2r}{c}$, which represent the distances at which two trajectories are considered far or close, respectively. m depends on the length of trajectories in the dataset. The size of the area where the trajectories are located is also given. The number of disks is chosen such that they cover most of the area with high probability - then every trajectory will intersect a disk with high probability. Therefore setting $D = c(A/r^2)$ makes the probability of any given point not being covered $(1-\pi r^2/A)^{c(A/r^2)}$, which goes to $\frac{1}{e^c}$ as c increases. In experiments we show that a small number of disks suffices to cover a large portion of a given area. Given these values ($2r$, c, A, D, m) one can compute the collision probabilities, p_1 and p_2, from the above theorems. The size of the area and the number of disks can be changed to increase or decrease p_1 and p_2. A practical example is presented in the Experiments section.

3.4.3 Summary of LSH and near neighbour results

We have shown how the binary sketches and the ordered sketches provide a locality sensitive hash family for the Hausdorff and the Fréchet distances, respectively. In both families, setting the number of disks $D = \Theta(A/r^2)$ and $c = O(m)$ gives us a constant collision probability of nearby trajectories, and we have low collision probability for far-away trajectories. From the framework described in Section 3.3.1, we obtain that

Theorem 3. *There exists a data structure using binary and ordered sketches, that takes space $O(n)$ memory words, and returns an $O(m)$ approximation (in the Hausdorff metric using the binary sketch, and in the Fréchet metric using the ordered sketch) to the*

1. *(c,r)-near neighbour problem in $O(m\log n)$ time.*
2. *c-approximate nearest neighbour problem in $O(m\log^2 n)$ time.*

Now we turn to the other kinds of queries, and the data structure we use to solve them.

3.5 MRTS: Multi-Resolution Trajectory Sketch

In this section we describe the MRTS (Multi Resolution Trajectory Sketch), a layered data structure that we will use to solve distance estimation, classification, clustering and subtrajectory detection queries.

3.5.1 Description

Recall that U and L are upper and lower bounds on the degree of detail desired by the user, and $r_i = U/a^i$ for $0 \le i \le l$, with $l = \log_a(U/L)$ and $r_l = L$. Define $D_i = \Theta(A/r_i^2)$

for $0 \leq i \leq l$.

Our layered data structure has l layers, denoted by L_i, $0 \leq i \leq l$. Layer L_i stores

1. The binary sketches of all the n trajectories, for D_i random disks of radius r_i each. When $D_i = \Omega(m)$, we store the set of disks intersecting a trajectory as opposed to the bit vector of length D_i.

2. The ordered sketches of all the n trajectories, for D_i random disks of radius r_i each.

3. In addition, there are forward and backward pointers between a trajectory in the dataset and its corresponding sketches at each layer. Each sketch also stores a counter with the number of trajectories pointing to it.

Insertions. When a new trajectory T is to be inserted in the dataset, we use the data structure referred to in Observation 1. Computing the sketch on layer i requires $O(m \log D_i + k_i)$ time, where k_i is the number of disks of radius r_i that intersect T. Since $D_i \leq D_l$, overall, this costs us at most $O(m \log D_l l + k)$, where k is the total number of disks intersecting T. Plugging in the values of D_l and l, we get
Lemma 3. *A new trajectory T can be inserted into MRTS in $O(m \log(A/L^2) \log(U/L) + k)$ time, where k is the total number of disks intersecting the trajectory.*

Space. Any point p on a trajectory T lies in $O(1)$ out of the D_i disks for every i. This is because D_i is the number of disks required to cover the region A and the disks are deployed uniformly at random in the domain. Thus the sketch of T on layer i requires at most $O(m)$ space, and so in total the sketches for the trajectory T in the entire MRTS data structure uses $O(ml)$ space. Since there are n trajectories,
Observation 2. *The MRTS uses at most $O(nm \log(U/L))$ words of memory in space.*

Notice that $O(nm)$ words of space are required to store the dataset. Moreover, the bound above is pessimistic in the sense that we do not consider the fact that for densely sampled trajectories, many points of the trajectory will lie in the same disk. In fact, one can show that for uniformly sampled trajectories, $O(r)$ points lie in a disk of radius r. This reduces the space usage of our data structure to $O(\frac{nm}{L} \log(U/L))$ memory words.

3.5.2 MRTS and distance queries

Now we show how to handle other types of queries mentioned in our problem description. For the sake of brevity, we will restrict ourselves to the Fréchet case, which is arguably harder than the Hausdorff case, and more interesting. We point out that all of our results generalise to the Hausdorff case with the same (or better) bounds. However, before we describe our query procedure, we need some geometric lemmas.

Lemma 4. *Consider two disks of radius r, and let h be the distance between their centres. Let A_{dif} denote the area of their symmetric difference (see Figure 3.2). Then 1) if $h > 2r$, $A_{dif} = 2\pi r^2$, and 2) if $h \leq 2r$, $A_{dif} \geq 2rh$.*

Proof. Part 1) is obvious, as the disks are disjoint. For part 2), the area of the symmetric "lens", or the intersection of the two disks, is given by $2r^2\cos^{-1}(h/2r) - (h/2)\sqrt{4r^2-h^2}$. The area of the symmetric difference is therefore

$$\begin{aligned}
A_{dif} &= 2(\pi r^2 - 2r^2\cos^{-1}(h/2r) + (h/2)\sqrt{4r^2-h^2}) \\
&= 2r^2(\pi - 2(\pi/2 - (h/2r) - O((h/2r)^3))) + (h/2r)\sqrt{1-(h/2r)^2} \\
&\geq 2r^2(h/r + O((h/2r)^3) \\
&\geq 2rh.
\end{aligned}$$

□

This gives us the following crucial observation:

Observation 3. *Let T_1 and T_2 be two trajectories such that their Fréchet distance is F. Then the probability that a randomly chosen disk of radius r intersects one trajectory but not the other is at least $2rF/A$.*

This is because if we look at the component realising the Fréchet distance (the maximum length in the traversal), there must be a pair of vertices $u \in T_1$ and $v \in T_2$ that are F apart. A disk separates these two vertices with probability equal to the measure of their symmetric difference area, which by the above lemma is at least $2rF/A$.

Lastly, we show that the ordered sketches already provide an upper bound on the Fréchet distance.

Lemma 5. *Let T_1 and T_2 be two trajectories with Fréchet distance F. Consider their ordered sketches with D disks of radius r, and assume all vertices overlap at least one disk. If the ordered sketches coincide, then $F < 2r$.*

Proof. If the ordered sketches coincide then for every vertex in T_1 there is at least one vertex in T_2 such that the distance between them is lower than $2r$, because they should both belong to the same set of disks. There is a traversal that groups every pair of such vertices, giving a cost of at most $2r$. This means that an optimal traversal has a lower cost, therefore the Fréchet distance is less than $2r$. □

3.5.2.1 Querying the MRTS

We now describe how we query the MRTS for different kinds of queries.

Distance Estimation. Given the indices of two trajectories T_1 and T_2 in the dataset, we want to approximate the Fréchet distance between them. Note that precomputing the distance of a newly inserted trajectory to all the existing trajectories in the dataset trivially solves this problem in $O(1)$ time, but the cost is the high insertion time, at least $O(m^2n)$. The MRTS has low insertion time, and is still flexible enough to answer such queries faster.

Let us consider the optimal traversal of the two trajectories that realises their Fréchet distance F. As observed, this traversal can be broken down into at most m components, each of which is a star. In some of these components the distance is $O(F)$, whereas in others it is significantly lower. Let k be the number of components in which the maximum distance between the vertices of the component (the centre of the star and the other vertices on the second trajectory) is $\Theta(F)$. We will first estimate the product kF.

The main reason in estimating kF is that the symmetric difference area of the two trajectories depends linearly on k. If $k = 1$, then the two trajectories travel very close together until they reach this component; in this case the area of the symmetric difference is $O(rF)$. On the other extreme, two trajectories could be travelling at a constant distance F from each other (thus giving the same Fréchet distance between the trajectories), but the area of the symmetric difference is $O(rFk)$.

Let ℓ_i denote the fraction of disks on layer i that intersect one trajectory but not the other. Clearly, ℓ_i is an unbiased estimator for the symmetric difference area (for disks of radius r_i). We compute $A_{dif} = \sum_{i=1}^{h} \ell_i A/r_i$, and let $\mathrm{Est}_{kF} = A_{dif}/h$. We can show that this is an unbiased estimator of the true value of kF, and by increasing the number of disks at each layer, we can get higher concentration for this estimator. The expected value of ℓ_i on the other hand equals $O(r_i kF/A)$, as the probability that one disk separates the two trajectories equals $r_i kF/A$.

Query Algorithm: Report r_j, where j is the smallest value of i such that $\ell_i \geq r_i \mathrm{EST}_{kF}/A$.

We conjecture that this value of r_j reported is a constant factor approximation of the Fréchet distance between the trajectories. Note that if one just wants an upper bound, the r_i corresponding to the last layer i such that $S_1^i = S_2^i$ (the sketches coincide) gives us an upper bound, as by Lemma 5, we know that the Fréchet distance between the two trajectories is at most $2r_i$, since the sketches coincide.

This procedure requires us to examine the sketches at all levels in the worst case, requiring $O(\min(m, \log(U/L)\log(A/d^2))$ time. To get an upper bound we can stop at the first layer where we discover the sketches are different.

Classification/clustering. Our classification procedure is simple: for a given interval $[r_{i+1}, r_i]$, we define the set P_i to be all the trajectories that have the same sketch on layer i of the MRTS as the query trajectory.

Using the pointers, we can report the trajectories in P_i, or, if one just needs the count, we can read the counter value of the bucket corresponding to the sketch of the query trajectory. Running this procedure for all $1 \leq i \leq l$ gives us the partition of the dataset. Procedures such as those proposed by Mirzasoleiman et al. (2016) can be used to select representative elements as centres for exemplar based clustering. Clustering of trajectories can be applied to anonymise locations traces (Zeng et al., 2017).

Subtrajectory Detection. Given the query "Is trajectory T_1 r_i-close to a subtrajectory of T_2?", we first insert T_1 and T_2 in the MRTS if they haven't been inserted yet. Then we locate the sketches of T_1 and T_2 on layer $i-1$. We compute the LCS between these sketches, and answer yes if the sketch for T_1 appears as a subsequence, and no otherwise. To get accurate results with high probability, we can increase the number of disks on layer $i-1$, and take the majority answer.

3.6 Experiments

We tested the performance of binary and ordered sketches on two real datasets. The experiments show that:

- there is a correlation between the proposed distance measures and Hausdorff (strong correlation), Fréchet and DTW (Section 3.6.1);
- the correlation holds even with a small number of disks or with incomplete data (Sections 3.6.1, 3.6.3);
- the sketches achieve high pruning ratio with good accuracy, less storage space and better running time in nearest neighbours queries (Sections 3.6.2, 3.6.4, 3.6.5).

We ran all experiments on both the CRAWDAD dataset of taxis in Rome (Bracciale et al., 2014) and the ECML/PKDD dataset of Taxi Trajectories in Porto (Moreira-Matias et al., 2013).

Data preparation

The ECML/PKDD (Porto) dataset (Moreira-Matias et al., 2013) describes the mobility of 442 taxis driving in the city of Porto (Portugal) for 1 year and is composed of 1.7 million data points. The GPS locations (latitude, longitude) are recorded every 15 seconds with mobile data terminals installed in the vehicles. Each trajectory represents a complete taxi trip taken by a passenger. The lengths vary from $1km$ to $15km$. We randomly selected 1000 trajectories in a $5km \times 10km$ area of the map.

The CRAWDAD (Rome) dataset (Bracciale et al., 2014) contains the GPS locations of 320 taxi drivers working in the city centre of Rome (Italy). Each trace is for one driver for one day at about 7 second intervals, giving roughly 22 million data points. To obtain trajectories similar to individual trips, as in the Porto dataset, we split each trajectory into trips, such that the length of each is at most a 3 times the distance between source and destination. This gives trajectories with lengths roughly between $2km$ and $5km$. We randomly selected 1000 trajectories in a $7km \times 2km$ area of the map.

Choice of parameters

For all experiments with a fixed number of disks we picked uniformly at random 50 disks of radius $2km$. In general, the choice of radius depends on the scale of the problem. Suppose we consider two $1km$ trajectories to be close if their distance is less than $10m$. We choose enough disks to cover the map. Based on results in Section 3.4, we want to maximise $1 - \frac{8\pi r^2}{c}$ while $\frac{2r}{c} = 10m$. For $r = 2km$, the probability $1 - \frac{8\pi r^2}{c}$ becomes 0.75, which is a lower bound for the probability that the sketches are the same when the trajectories are less than $10m$ apart.

3.6.1 Correlation with Hausdorff, Fréchet and DTW

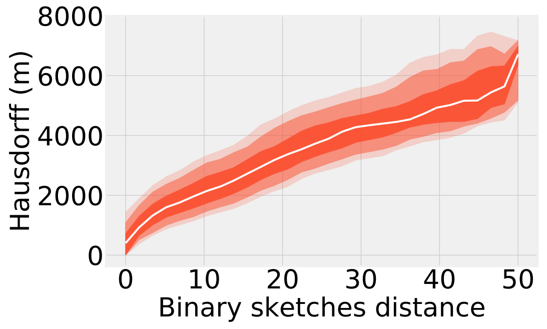
We checked how the proposed distance measures compare to Hausdorff, Fréchet and DTW. Figures 3.3 (a), (b), (c), (e), (f) show the results for the Porto and Rome datasets. For each pair of trajectories we compute the distance: with the binary unordered sketches for Hausdorff and with the ordered version for Fréchet and DTW. We grouped the possible values in 30 bins and show in the plots the median value (white line) and the 5-95, 10-90, and 25-75 percentiles (the shaded areas). We see that the sketch distance measure bears a clear correlation to the traditional measures. For Hausdorff the correlation is high (≈ 0.8). For Fréchet and DTW, while the correlation is lower, there is a clear distinction between close and distant trajectories, showing the utility of locality sensitive hashes.

3.6.1.1 Effect of radius and number of disks

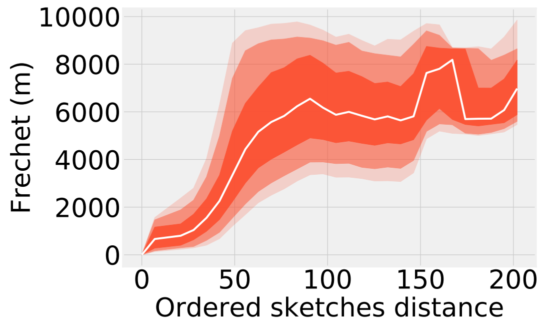
This correlation naturally raises a question of how it is influenced by parameters such as the number and radii of disks. Figure 3.4 (a) shows the effect of disk radius; each curve corresponds to results with a fixed number of disks and a varying radius between $500m$ and $4km$. The shaded area corresponds to the 5 to 95 percentiles based on 30 reruns. For each trial we computed the Pearson correlation coefficient between our distance measure and Hausdorff distance.

In Figure 3.4(a) a radius size of around $2.5km$ achieves the maximum correlation, beyond which the disks become less discriminatory and the correlation decreases. We also observe that too small a radius does not perform as well – to use a smaller radius we would need to consider more disks.

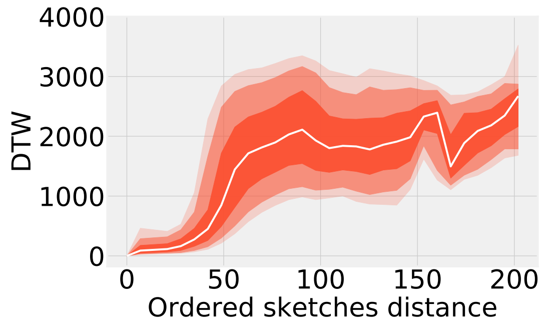
The correlation also increases with the number of disks. A higher number also assures low variability in performance between trials. Even with a low number, such as 10, we get a good correlation of ≈ 0.7 or more as long as the radius is in the right range ($1.5km$- $3km$). Beyond a certain number, increasing the number of disks does not improve performance. For a good correlation we need to choose enough disks to get a cover of the map, but having more disks offers a more fine grained distance measure



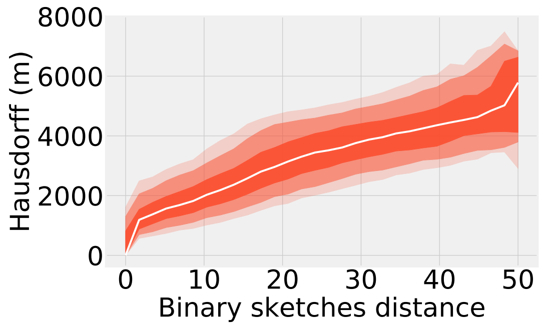
(a) Porto - high correlation with Hausdorff



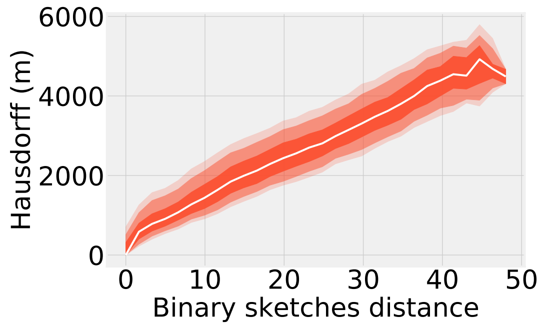
(b) Porto - correlation with Fréchet



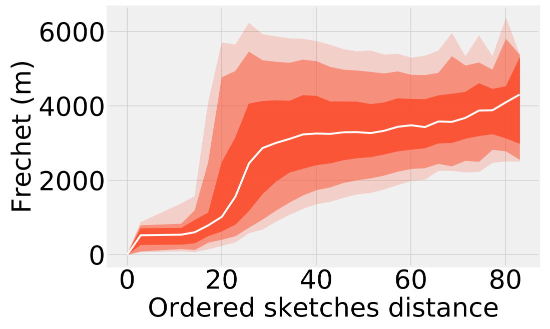
(c) Porto - correlation with DTW



(d) Porto - 90% of locations removed

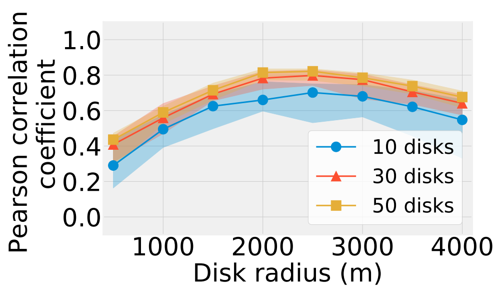


(e) Rome - high correlation with Hausdorff

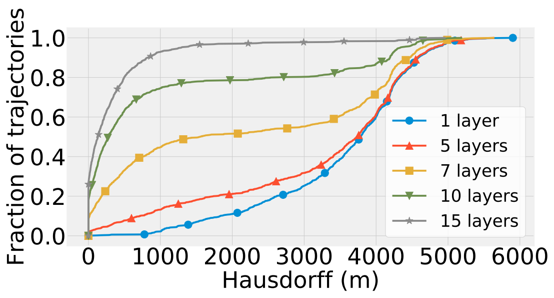


(f) Rome - correlation with Fréchet

Figure 3.3: Correlation of binary and ordered sketch distances with Hausdorff, Fréchet , and DTW for the Porto and Rome datasets (the 5-95, 10-90, and 25-75 percentiles).



(a)



(b)

Figure 3.4: (a) Correlation coefficient with Hausdorff distance depending on the disk radius and the number of disks. Variance decreases when increasing the number of disks.
(b) CDF of the Hausdorff distance error between true and found nearest neighbours. With more layers the error decreases.

at the cost of greater storage and computation, as we discuss in relation to nearest neighbours queries in Section 3.6.2.

3.6.2 Nearest neighbour search

For a variable number of disks (10, 30, 50) and fixed $r = 2km$, we computed the nearest neighbour for each trajectory in the dataset using Hausdorff and Fréchet distances. To prune the search space using our sketches we selected only those trajectories that had sketch distance equal to 0, 1, 2 etc. We measured in what proportion of cases the true nearest neighbour is in the remaining set (accuracy), and how large the eliminated set is compared to the total dataset (pruning ratio). In Figure 3.5 we show how the pruning ratio changes with accuracy.

There is little variation in the accuracy for a specific number of disks. Interestingly, changing the number of disks does not influence the results too much for nearest neighbour search with LSH; with a small number we can achieve a high pruning ratio and high accuracy. However, a larger number of disks provides better resolution when using the sketches as an estimate of distance.

Figures 3.5 show that accuracy can be as high as about 80% at a pruning ratio of over 80% for both Hausdorff and Fréchet distances, even with a small number of disks. Thus, this method allows us to restrict our attention to a small fraction of the

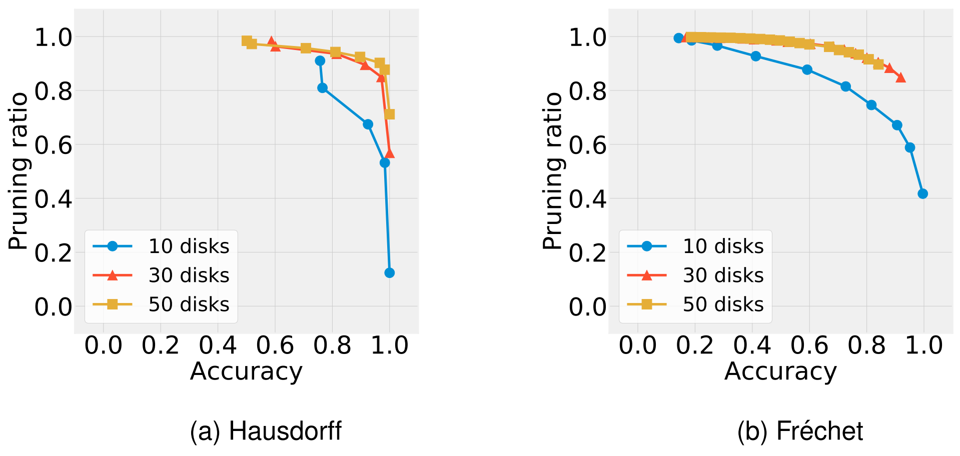


Figure 3.5: The performance for nearest neighbours queries increases with the number of disks. A small number (30) suffices.

trajectories and still obtain the correct nearest neighbour in most cases.

Multi-resolution trajectory sketches

In Figure 3.4 (b) we show how the distance error to the nearest neighbour changes with the different number of layers in the multi-resolution sketches. We considered 1, 5, 7, 10 and 15 layers. At the first level the disks have radius $r = 2km$; this decreases by a factor $\varepsilon = 0.1$ at each level. The number of disks at level i is given by $A/r_i^2 \log(A/r_i^2)$, where r_i is the radius at level i.

To find the nearest neighbours given a query trajectory we compare the sketches at every layer, starting from the top, and only advance to the next one if the sketches match. When we reach the last layer, the set of the remaining trajectories is searched for the nearest neighbour. The difference in distance between the true nearest neighbour and the one we found gives the error. Figure 3.4 shows the CDF of the error.

At the first level the number of disks is low (13). Performance increases quickly with the number of layers. In practice, the performance can be further improved by checking for approximate matches between sketches instead of exact matches (corresponding to distance 0).

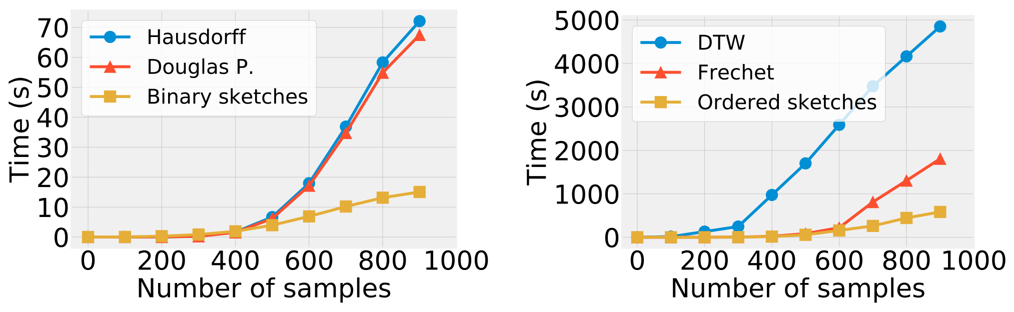
3.6.3 Robustness to data loss and privacy of locations

Trajectory data can be subject to sensing errors and noise and often has missing data. In other cases, a lower frequency of location sampling is preferred to maintain the user's privacy in between check-ins (Niedermayer et al., 2013). The disk distance performs well even when large parts of the trajectories are missing. Interestingly, when only 10% of the data points in each trajectory are retained, the disk distance is still highly correlated with the Hausdorff distance, as can be seen in Figure 3.3 (d). In this experiment we computed the disk distance on only a 10% sample of the location points in each trajectory, and plotted against their true initial Hausdorff distance.

Beyond the natural robustness to unreliable sensing, this result implies efficiency of storage and computations as only a small random sample needs to be stored for useful comparison.

3.6.4 Time efficiency

We compared the running times of our method using pruning with Hausdorff, Fréchet and DTW when computing distance matrices.



(a) Binary sketches

(b) Ordered sketches

Figure 3.6: Better performance in terms of running time.

In the first experiment shown in Figure 3.6 (a) we used the standard Hausdorff implementation between point sets and computed the time for sampled sets of up to 1000 trajectories. Using the Douglas-Peucker algorithm to first simplify the trajectories did not bring a large improvement in the computation time. The time taken to compute all pairwise sketch distances is more than 5 times faster than computing Hausdorff distance for 1000 trajectories. This also includes the preprocessing time of picking the disks and computing the sketches.

For the second experiment in Figure 3.6 (b) we computed Fréchet and DTW distances efficiently with Python libraries. The results show that the sketch distance computation is 2 times faster than Fréchet for 1000 trajectories when using 50 disks - enough to get a 90% accuracy for nearest neighbours search with 80% pruning ratio, as can be seen in Figure 3.5.

3.6.5 Comparison with existing LSH method

We compared the performance of the ordered sketches with the grid-based sketches proposed by Driemel and Silvestri (2017). From the LSH scheme they propose, the basic one has an approximation factor linear in the number of vertices that a trajectory has, similarly to our proposed scheme. In their work the hashes are constructed in the following way:

- consider a grid that covers all trajectories;
- each vertex in the trajectory is replaced by the closest node in the grid;
- from the resulting sequence of nodes the consecutive duplicates are removed.

For a particular grid, this results in a hash function whose image is a subset of sequences of nodes from the grid. The family of hash functions is constructed by considering shifted versions of the grid, parametrized by a random variable that is at most the size of the grid in any dimension.

The work does not explicitly discuss the choice of grid size in a practical setting; therefore, in experiments we considered various sizes of grids, ranging from 100m to 15km. We use the same values as diameters of disks for our scheme. The set of sizes includes a much larger range of values than the ones that are optimal for our proposed scheme for the dataset in use (the optimal range is between 1km and 4km according to Figure 3.4 (a)). For each size s we built a grid that covers the map, where the distance between any two neighbouring nodes is equal to s. We denote the number of resulting nodes by n_{grid}. To compare with our proposed method, we consider $n_{grid}/2$ disks of diameter s (in ordered sketches we note both the entrance inside a disk and the exit). Notice that we are using the ordered sketches (not the fixed size binary sketches) because the grid approach gives a LSH scheme for Fréchet . Both alphabets for constructing the hashes are composed of n_{grid} elements; the size of the hashes themselves depends on the length of the trajectory - this could in theory be as large as the number of vertices in the trajectory in both cases. Two grid hash functions are equal if the sequences are the same. Two ordered sketches are considered the same if the edit distance is at most 2.

In each experiment we consider a grid of size s with n_{grid} nodes and a set of $n_{grid}/2$ disks of diameter s. Based on these values, the grid and disk hashes are computed for each trajectory. If the hash values of two trajectories are equal according to the corresponding scheme, then they are kept as possible nearest neighbours. Given a certain trajectory, we count how many trajectories have the same hash; the size of this set gives the pruning ration. The number of instances when the true nearest neighbour is in this set gives us the accuracy. Figure 4.9 (a) shows the results comparing the performance in terms of pruning ratio and accuracy. The two methods perform approximately the same with slightly better results for the disks version: for 90% accuracy, the pruning ratio is close to 0.8. However, in terms of size we notice a significant improvement. For each experiment and each trajectory we compare the length of the hashes for the two schemes. In Figure 4.9 we notice that almost all trajectories have a disk hash shorter than the grid hash and often the size is twice as small, reaching even a fraction of less than 0.25. This can partly be explained by the fact that in the case of grid hashes each vertex of a trajectory has a correspondent in the grid (unless repeated); for the disk hashes a trajectory could travel inside a disk without exiting or intersecting other disks, therefore without adding another element to the hash.

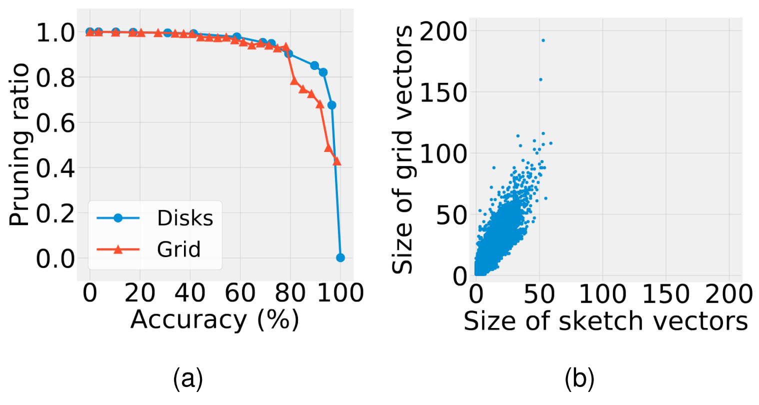


Figure 3.7: Better performance in terms of (a) accuracy and (b) storage size.

3.7 Conclusion

We presented a randomised sketching scheme to compactly represent trajectories, and established its effectiveness in both theory and practice of trajectory processing. The theoretical results give practical guarantees and the experimental results show the applicability of the scheme to real-world scenarios. Such scenarios include analysis of spatial trajectories in player performance (Gudmundsson and Horton, 2017) or ridesharing matching (Shang et al., 2014). Planar networks and trajectories are common in robotics, biological systems, seismology and many other domains, where applications of the scheme can be useful in reducing the computation time in clustering and similarity search. Adaptation of the randomised scheme in higher dimension has interesting applications, for example in recommender systems, when searching for nearest neighbours for high-dimensional feature vectors. Another interesting direction is to verify if similar theoretical guarantees hold for other distances, such as DTW.

Chapter 4

Demand driven transit network design with constraints: case study on bicycle lanes planning

4.1 Introduction

As a response to increased traffic congestion and the need to reduce carbon emissions, cities consider ways to modernise, build and extend transit systems. Transit network design solutions can benefit from analysing the large amount of crowd-sourced location data available, which provides valuable insights into population mobility needs. Designing efficient metro lines, bicycle paths, or bus routes brings a number of conflicting constraints into play: from the user's perspective, an efficient transit network is easily accessible and time-efficient, while from the provider's side there is a limitation on the budget, amongst others. Methodologically, the transit network design problem is complex. As described by Newell (1979) in a seminal work, it is a non-convex optimisation problem and large scale solutions are often based on heuristics.

A common approach in data driven transit network design is to cluster trajectories, which are ordered sets of GPS locations (latitude and longitude) from people travelling in an urban setting. This task is by itself difficult because of the size, complexity and diversity of trajectories. Clustering algorithms often take into account the similarity between trajectories (Evans et al., 2012; Jiang et al., 2016). Explicitly computing distances between long sequences of locations is expensive; while there exist linear and near-linear time approximations for distances in the plane, such as Hausdorff or Fréchet (Driemel et al., 2012; Taha and Hanbury, 2015; Astefanoaei et al., 2018), the number of pairwise computations is prohibitively large. Expensive computations can be avoided by considering only the origins and destinations of trajectories, in this way abstracting from their geometry. The literature on clustering origin-destination pairs is extensive; often the goal is to find trips whose starting and ending locations are similar. We notice that, in the context of transit network design, it is useful to group trips that follow a similar direction, but are not necessarily geographically close. For example, a bus route would not only serve the travellers who enter and exit at the start and end of the route, but a larger number of people along the way. Other types of clustering algorithms identify popular segments in a road network (Jiang et al., 2016); in the context of cycling mobility, these segments could be candidates for bicycle lanes construction. Since popular roads tend to concentrate in the centre of a city, designing a transit network based on popular paths can result in inefficient coverage. For these reasons we consider a more broad definition of clusters that avoids expensive trajectory similarity computation, allows grouping trips that do not necessarily share nearby origin and destination locations and offers a more broad coverage than popular segments. Based on these clusters we identify paths that are the base of the transit network and then further modify the network to fit the constraints. We use a cycling network as a case study.

Our contributions. Given the GPS locations of origins and destinations of user trips, our proposed algorithm finds a transit network of bounded total cost such that every covered pair of locations is within a given distance from the network and the route through the new transit network is not much longer than the shortest route through the initial road network. We formulate the problem in terms of building a network of cycle lanes and prove that it is NP-hard.

The heuristic algorithm we propose has three stages:

1. *Multi resolution local clustering* of the trips that can be served by the same path;
2. *Global merging* of the discovered clusters into a network of paths;
3. *Network simplification* based on the given cost constraints.

The first and third stage of the algorithm are both NP-hard. Geometrically, the proposed clustering task is similar to covering segments in the plane with rectangles of fixed width. This is equivalent to geometric set coverage, a classical problem that is known to be NP-hard, but has a $\log(n)$ greedy approximation scheme, which is the best-possible approximation unless $P = NP$ (Feige, 1998). We propose a fast heuristic algorithm for the coverage problem that takes into account given design constraints and that is the base of our transit network construction scheme. For the *network simplification* stage we propose three greedy strategies and show that even the simplest one, which has running time linear with the number of edges in the network, gives good results in practice.

We ran experiments on the London street network and the Santander Cycle hire data. In one experiment we show that only a small sample of 500 trips is enough to build a network that covers approximately 60% of 5000 unseen trips. In other experiments we compare the coverage ratio of a proposed network of paths generated with our algorithm with the Cycleways bicycle paths existing network. The comparisons show that the number of covered trips is approximately 20% more when compared with a real network of the same size, using the most basic simplification strategy.

The remainder of this chapter is organised as follows. In Section 4.2 we summarised related works. In Section 4.3 we formulate the bike lane problem and show that it is NP-hard. In Section 4.4 we describe the proposed algorithm and illustrate it with an example. Section 4.5 presents the data, experimental set-up and results.

4.2 Related work

Designing a transit network is a complex task involving the interplay of geographical, economical and social factors. It is often modelled as a non-linear multi-objective problem that cannot be solved without abstracting from some of the factors and imposing major simplifications. In the literature, when the transit network design problem is formulated as an optimisation problem, the usual quantities of interest are: trip coverage, population coverage, costs, trip length (in either time or distance), number of allowed routes (Newell, 1979; Petrelli, 2004). Any of these quantities can be either optimised or constrained, which leads to the large array of intrinsically different problems. Maximising the trip coverage requires data related to population mobility, while optimising population coverage requires information about important locations, such as hospitals, universities and so on. Moreover, depending on the type of transit network, further constraints can be added such as network topology, number of stations and their placement (for bus, metro and rail networks, e.g. Laporte and Pascoal

(2015)), or continuity and connectivity (for cycling paths, e.g. Bao et al. (2017)).

The solutions to the network design problem are split into analytical (Wirasinghe, 1980), mathematical programming (Ceder and Israeli, 1998) and heuristic (Pinelli et al., 2016). Exact analytical or mathematical programming solutions are proposed in the case when the road network is reduced to a smaller graph, based on prior knowledge or constraints such as network topology (for example when designing a metro system and the shape is known (Laporte and Pascoal, 2015; Laporte et al., 2011)). A number of heuristic solutions do not have such constraints, but are not designed for a large-scale, street-level context. The work of Pinelli et al. (2016) proposes a data-driven transit network design algorithm based on Call Detail Records (CDR) data, which means the resolution of the solution is at the level of cell tower locations. Mauttone et al. (2017) build a multi-commodity network flow mixed-integer mathematical program for solving a similar problem, but the experiments show that it is not scalable to a large network.

A different formulation for the transit network design problem is finding *k primary corridors* given a set of GPS trajectories. The aim is to find k paths in a road network, such that: either the distance to the initial dataset of GPS trajectories is minimised (Jiang et al., 2016), or the intra-cluster pairwise distances are minimised (Evans et al., 2012) . Other works take into account budget limitations and continuity constraints (Yu et al., 2018). However, these works consider whole trajectories, which is computationally expensive. In designing routes that are accessible and useful to a large number of users, we are more interested in the origins and destinations of their trips, rather than the actual trajectory they choose to take. Generally, there is a large number of paths with similar lengths in a road network; there is evidence that people change their travel behaviour based on the available resources (Verplanken and Roy, 2016).

Clustering origin destination pairs is used in aggregating and mapping mobility flow patterns. Zhu and Guo (2014) developed a hierarchical clustering method based on the nearest neighbours of the origins and destinations. They propose a scalable algorithm to group trips that have similar origins and similar destinations. Kumar et al. (2016) use density based clustering algorithms such as DBSCAN for all the data points to identify *hotspots* and then extract the valid trip clusters. These and a larger number of works on origin destination pairs place the problem in the Euclidean space rather than a road network, which would be more informative when designing a transit network. Moreover, clustering is often understood as grouping trips with geographically close endpoints. Thinking in the context of building a system of cycling paths, we notice that this type of clustering is not necessarily the most insightful - cyclists can enter and leave at any point along the path. These works also do not consider the geographical spread of clusters (in terms of length and width), which is relevant because paths should be easily accessible - in a wide cluster locations at the extremity of the cluster are far away from the median.

In contrast to the above, this work considers a fast, scalable method for clustering origin destination pairs. We take into account the entire road network and real user trips.

4.3 Bike path problem description

Suppose we are given a set of user trips defined by the locations of the departure (origin) and destination and the task is to find paths in the road network that the users can follow to arrive to their destination under certain constraints. We describe the Demand Driven Network Generation (DNG) algorithm in terms of designing a network of bike paths. A similar approach can be applied to other types of transportation (such as buses or trains).

In designing an efficient transit network, we consider a series of physical metrics that capture the system's performance. As identified by Daganzo (2010); Estrada et al. (2011); Lee (2012) and others, these metrics relate to user and agency costs. User costs are a function of the time spent in the transit network, which can be split into waiting time (to access the network and the means of transport) and in-vehicle time. In this work we look at the spatial component of designing a transit network; speed and frequency of vehicles depend on the application. Therefore, we adapt the user costs to take into account distance rather than time. In the context of building bikes paths, we measure how accessible the network is to users, so the distance from the user starting and end points to the closest paths. From the user perspective, Lee (2012); Ceder (2001) propose to measure how competitive a transit network is by computing the ratio between a function of the distance through the transit network and the distance through the road network otherwise, usually the length of the shortest path between the origin and destination of the trip. We use the same idea in defining a constraint on the maximum stretch of a user path. In terms of agency costs, the relevant metric for the spatial transit network is the infrastructure length.

Our objective is to maximise the number of users who can use the paths efficiently, given constraints related to the above metrics: the origin and destination locations of their trip are close to the bike paths, the trip through the paths are not much longer than the trip through the road network, the path construction budget is limited. Therefore, in designing the network we define the following constraints:

- **construction constraint** - there is a cost associated with building a bike path on each road segment; this depends on the properties of the road, but for the moment, for simplicity, we assume that the cost is the length of the segment (other costs could include the actual construction cost or the level of traffic congestion); therefore we limit the total length (total cost) of paths to be built to T;
- **transit distance constraint** - the bike paths should be close to the origin and destination of a trip; we allow transitions between the origin-destination locations and the bike paths as long as the Euclidean distance is not longer than a fixed constant d_t;
- **total distance constraint** - for each trip, the route following the bike paths should not be much longer than the shortest route through the road network; we fix the stretch factor of the path to a constant s.

Given the above constraints, let us describe the setting for the bike path problem.

Let S be a set of origin-destination pairs of latitude-longitude coordinates defined as $S = \{((lat_{o_i}, lon_{o_i}), (lat_{d_i}, lon_{d_i})), i = 1, \ldots, n\}$, where n is the number of trips, and $G = (V, E)$ a road network, where the vertex set V is the set of intersections with known coordinates and the edge set E is the set of road segments. The *bike path problem* is to find a subgraph G' of G such that the total length (cost) of the road segments is below a threshold T and the number of (G', d_t, s)*-covered* trips is maximised.

Definition 15 ((G', d_t, s)-covered)**.** *We say a trip is (G', d_t, s)-covered if the endpoints are within distance d_t to the road network G' and the shortest path distance through the network has a stretch of at most s.*

Definition 16 (Stretch factor)**.** *We say the stretch factor of a path p is s if it is at most s times longer than the shortest path between the same nodes.*

The stretch factor is commonly used on graphs to measure how much distances are distorted after a transformation. As in the areas of geometric spanners and weighted graphs that approximate Euclidean distances, we are interested in finding a subgraph of the road network such that distances between points (in our case origins and destinations) are minimally distorted.

For any path p in G, we say that a trip is *covered by p* if it is $(G_p, d_t, s)-$covered by the subgraph G_p induced by the path p in the network. Note that some trips can be partially covered, but we do not consider them as contributing to the final set.

With the notations from Table 4.1, *the bike path problem* is formally defined as:

Bike path problem. *Given S, $G(V,E)$, d_t, s, T, find a subgraph $G' \subset G$ with $L(G') < T$ such that the size of the set $C = \{x \in S | x \text{ is } (G', d_t, s) - \text{covered}\}$ is maximal.*

Notation	Description
S	origin-destination pairs
$G(V,E)$	road network
d_t	transit distance
s	road network route stretch
T	threshold on total length (cost)
$L(G')$	total length (cost) of edges in G'
C	the set of covered trips

Table 4.1: Table of notations

We show that the bike path problem described above is NP-hard, as it can be reduced from the Knapsack problem (Mathews, 1896).

Theorem 4. *The bike path problem is NP-hard.*

Proof. Suppose that the transition distance d_t is 0 (which is equivalent to all trips having endpoints in the network) and that each trip can be covered by exactly one edge. Then, for each edge there is an associated set of trips that is covered; this gives the utility of the edge. The cost is a function of the length, in this case the length itself. Then the bike path problem is equivalent to finding the set of edges that maximises the total utility (total number of trips covered) given the constraint on the cost (the total length of the edges is less than T). This is exactly the 0-1 Knapsack problem (Mathews, 1896), known to be NP-hard (Karp, 1972). □

Given the above result, we propose a heuristic algorithm that we describe in the following section. In the Experiments section we show how the generated network compares to the bike routes network in London.

4.4 Demand driven network generation (DNG) algorithm

To better understand how the origin destination trips are distributed geographically we abstract from the network itself and first consider the problem in the Euclidean space, where the trips are simply line segments between the origin and the destination. We cluster the segments based on geometric properties and then combine the medians of the clusters to create a network of paths. This network is simplified such that the total cost is below the given threshold. Therefore, the algorithm we propose has three stages:

1. **Multi-resolution local clustering.** The trips that can be served by the same paths in the network are grouped together.
2. **Global merging.** The clusters are merged to create a network of paths.
3. **Network simplification.** The edges are scored based on a measure of utility. Those with low scores are removed to reach the construction constraint.

The algorithm is flexible in terms of efficiency constraints; the parameters can be fixed according to the available resources and desired accuracy. We give more details about the three stages in the next section, followed by an example that illustrates all the steps. Implementation details are given in the Experiments section.

4.4.1 Multi-resolution local clustering

In the first stage we group trips that are likely to be served by the same path. These are trips that are either geographically close, or that follow a similar trajectory.

For now we disregard the road network and the stretch factor constraint. We work in the Euclidean space, where the origin-destination pairs are segments and the paths we design are polygonal chains. The trips that are covered by a path are at distance at most d_t from the path, measured at both endpoints. Suppose the rectangle in Figure 4.1 has width $2d_t$ and length L, the maximum length of any segment in the map. The trips covered by the rectangle can all be served by the median (the red line), because all endpoints are at distance at most d_t. Notice that the segments do not have to be close to each other; the path covers even trips that have do not pass through the same regions.

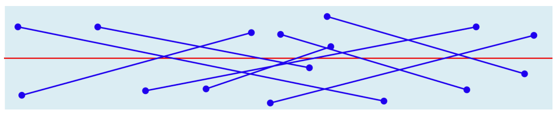
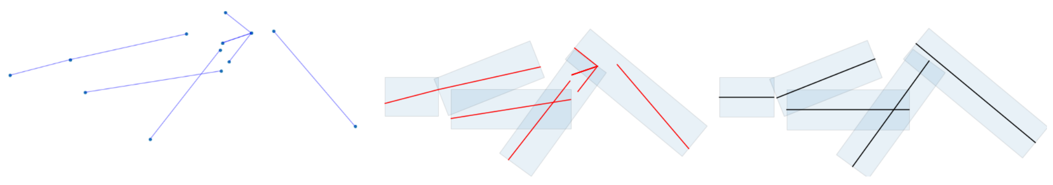


Figure 4.1: Rectangle of width $2d_t$ covering segments with endpoints in distinct areas.

Clustering the trips becomes then a problem of covering all segments with rectangles of width $2d_t$ and length L, while minimising the number of rectangles (see Figure 4.2). We call this the *rectangle coverage problem* and prove that it is NP-hard.



(a) A set of trips (b) Covering rectangles (c) The medians

Figure 4.2: A suboptimal solution to the rectangle coverage problem. The optimal solution would cover the 2 leftmost segments with one rectangle. The medians (Figure (c)) are used to build the road network.

Definition 17. ***The rectangle coverage problem*** *is defined as the problem of finding the minimum number of rectangles of width $2d_t$ and length L (the maximum length of the segments) with arbitrary orientations, that can cover a given set of segments.*

Theorem 5. *The rectangle coverage problem is NP-hard.*

Proof. Suppose all segments have length 0, so they are points in the 2D plane. The rectangles can have any orientation, therefore the problem is equivalent to covering the points with disks of a fixed radius. The discrete unit disk cover problem is NP-hard, as it is a geometric version of the set cover problem (Das et al., 2011). □

For the general set cover problem, the greedy algorithm gives a $O(\log n)$ approximation, tight to a constant factor, where n is the maximum between the number of points and the number of sets. In the geometric case, special attention has been given to the discrete unit disk cover (DUDC) problem, namely covering all points in the space with the minimum number of unit size disks. DUDC has better approximation schemes than the general set cover problem, because it takes advantage of geometric properties. Mustafa and Ray (2010) propose a local search algorithm that gives a PTAS

(Polynomial Time Approximation Scheme) and later Agarwal and Pan (2014) give a nearly-linear time approximation scheme for DUDC. Other works look at the problem of covering segments (as opposed to points) with disks (Basappa, 2018) or axis-aligned squares (Acharyya et al., 2019). However, the proposed schemes implement complex data structures that have prohibitively large constants in the running time (Bus et al., 2018, 2017). Moreover, it is unclear how and if the same schemes can be adapted to the problem of covering segments with rectangles with arbitrary orientations.

We propose a greedy algorithm for the rectangle cover problem with three subroutines for choosing the maximum coverage rectangle at each iteration: a $O(n^4)$-time exact solution, a $O(n^2 \log(n))$-time heuristic and a $O(n^2)$-time heuristic. For the following results we assume that any given segment has length at least $2d_t$ (the total allowed walking distance to and from a path).

4.4.1.1 Greedy coverage

The greedy solution we propose for the rectangle coverage problem is to iteratively find the rectangle that covers the maximum number of segments. Let L be the maximum length of a line segment on the bounded area of the map. Assuming that each rectangle has a maximum length L, we cover all segments with rectangles of dimension $L \times 2d_t$. We discuss the exact and heuristic algorithm for maximum coverage with rectangles of arbitrary orientations.

$O(n^4)$ algorithm for finding the maximum coverage rectangle

We show that the maximum coverage rectangle can be found by checking every $L \times 2d_t$ rectangle that intersects a triplet of non-collinear points from the set of endpoints of the segments (with no differentiation between origin and destination locations). We say that a rectangle *intersects* a set of points if the points are on the sides of the rectangle. Any rectangle covering a set of points that does not intersect a triplet of points can be translated so that it intersects a triplet, and still covers the set, as proved in Theorem 6. Lemma 6 shows that the set of points the new rectangle covers is at least as large as the previous set. Therefore, the maximum coverage rectangle can be found by checking each triplet of points induced rectangle, which gives a $O(n^4)$ algorithm, proved in Theorem 7.

Theorem 6. *Let H be the convex hull of a given set of segments with at least 3 non-collinear points. Suppose there exists a $L \times 2d_t$ rectangle $\mathcal{R}$ that covers H. Then there exists a $L \times 2d_t$ rectangle $\mathcal{R}'$ that intersects H in 3 points not all on the same side.*

Proof. Consider any rectangle $\mathcal{R}$ of size $L \times 2d_t$ that covers H and let A, B, C, D be the left/top/right/down-most vertices of H in the directions given by the sides of $\mathcal{R}$, as in Figure 4.3.

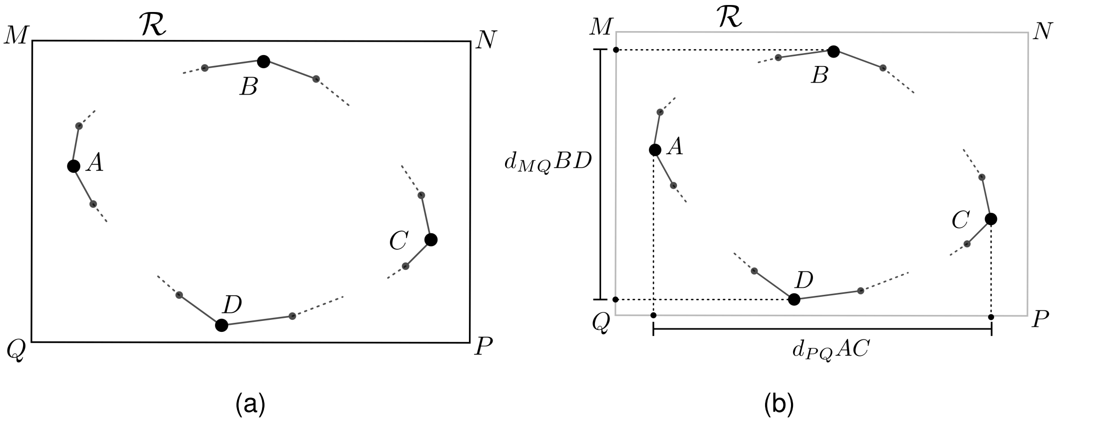


Figure 4.3: (a) The convex hull H and an enclosing rectangle $\mathcal{R}$ of size $L \times 2d_t$. (b) $d_{MQ}BD$ is the length of the projection of BD on MQ; $d_{PQ}AC$ similarly defined.

Let $d_{PQ}AC$ be the distance between the projections of A and C on PQ and similarly define $d_{MQ}BD$ (see Figure 4.3 (b)). If either $d_{PQ}AC = L$ or $d_{MQ}BD = 2d_t$, then $\mathcal{R}$ can be translated to intersect 3 of A, B, C, D (top-down for $d_{PQ}AC = L$ and right-left for $d_{MQ}BD = 2d_t$).

Consider the case when $d_{PQ}AC < L$ and $d_{MQ}BD < 2d_t$. Translate $\mathcal{R}$ until it intersects A and B, as in Figure 4.4 (a). Denote by A' and B' the vertices to the right of A and B and θ_1 and θ_2 the angles that AA' and BB' make with the rectangle; suppose $\theta_1 < \theta_2$. Then we can rotate $\mathcal{R}$ until it intersects A', keeping A and B on $\mathcal{R}$ and maintaining the covering property. See Fig. 4.4.

After the translation and rotation, $\mathcal{R}$ intersects H at A, A', B which are 3 vertices of the convex hull with 2 on adjacent sides of $\mathcal{R}$.

□

Lemma 6. *By translating and/or rotating a rectangle covering a set of points as in the Theorem 6, the new rectangle will cover at least as many points as before.*

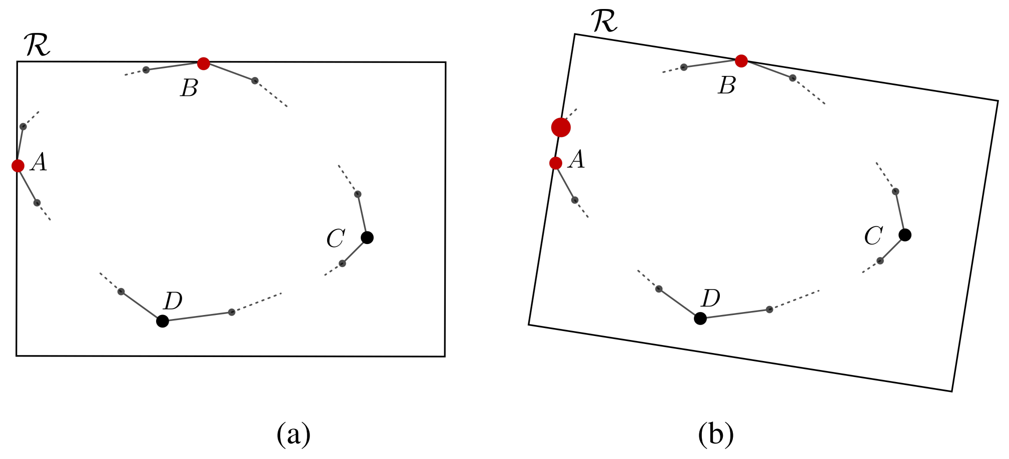


Figure 4.4: (a)$\mathcal{R}$ is first translated to intersect the convex hull H at A and B. (b) Then it is rotated to intersect A', the vertex to the right of A.

Proof. By the translation and rotation operations from Theorem 6 the convex hull of the set of points remains covered. Therefore there are at least as many segments that are now covered. □

Theorem 7. *Any maximum coverage rectangle can be moved so that it intersects 3 of the points it covers.*

Proof. Suppose a maximum coverage rectangle intersects 0, 1, or 2 of the endpoints of the segments it covers, but not 3. In Theorem 6 we show how to translate this rectangle such that the set of points is still covered and it intersects 3 points. By Lemma 6 the new set of points is at least as large as before. Therefore, the rectangle is still the maximum coverage rectangle. □

By Theorem 7, to find the maximum coverage rectangle of a set of points it is enough to check the rectangles given by every combination of 3 points. Next we show that the number of rectangles we need to consider is constant in the number of triplets of points.

Lemma 7. *Let A,B,C be 3 points. If the points are non-collinear and $d_{AB}C, d_{BC}A, d_{AC}B \notin \{L, 2d_t\}$ there is at most 1 rectangle of size $L \times 2d_t$ that intersects the points. Otherwise, any rectangle intersecting the points is valid.*

Proof. In the first case, if there is a rectangle that intersects all of A,B,C then either the 3 points are each on one side of the rectangle, or 2 on one side and 1 on the other (otherwise they would have to be collinear). If the points are each on a different side, then there is no rotation or translation that we can do to the rectangle. If they are on 2 sides, they would have to be on adjacent sides, otherwise the condition on distances is not valid. Again, no rotation or translation would be possible.

In the second case, there are possibly infinitely many rectangles intersecting the points. Suppose not all of them cover the same number of points and consider the one that covers the most. If all the points are collinear, this rectangle will be found when the 2 extreme points will be part of a triplet. Otherwise, it will be found when any 3 non-collinear points of the convex hull will be considered as the triplet. □

Theorem 8. *Let S be a set of n segments. The maximum coverage rectangle of size $L \times 2d_t$ can be found in $O(n^4)$.*

Proof. Let P be the set of endpoints of segments in S. Take every combination of 3 points from P and for each of them find the $L \times 2d_t$ rectangle that intersects them, if any. Count the number of segments it covers in $O(n)$. One of the rectangles will coincide with the maximum coverage rectangle. Since the number of rectangles is at most 1 and the number of point triplets is $\binom{2n}{3}$, then the algorithm runs in $O(n^4)$. □

Lemma 8. *The greedy algorithm choosing the maximum coverage rectangle at each step runs in $O(n^5)$.*

Proof. In the worst case each rectangle covers 1 segment, so the maximum coverage rectangle algorithm would have to be ran $O(n)$ times, giving the $O(n^5)$ running time. □

Similarly to the set cover problem, the greedy algorithm above gives a $\log(n)$ approximation.

$O(n^2 \log(n))$ algorithm for finding the maximum coverage rectangle

We propose a $O(n^2 \log(n))$ algorithm to find a rectangle that covers the maximum number of points with rectangles of width $2d_t$. The algorithm works by projecting segments onto lines and finding the intervals that contain the most projections. We call these intervals *projection windows*, defined below.

Definition 18 (Projection window)**.** *Given a set of segments S and a line ℓ we define the projection window as the shortest segment on the line that contains the projections of all segments from S onto ℓ (see Figure 4.5).*

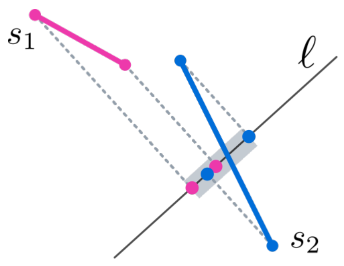


Figure 4.5: The grey segment is the projection window of segments s_1, s_2 on line ℓ.

The algorithm chooses the projection window that contains the most projections.

Lemma 9. *If a set S of segments can be covered by a $L \times 2d_t$ rectangle, then there exists a line ℓ such that the length of the projection window is at most $2d_t$.*

Taking a line perpendicular to the median of the rectangle gives the result. We also observe that the shortest projection window is given by the line perpendicular to the narrowest rectangle that can cover the segments. Now let us consider lines that are not perpendicular to the rectangles.

Lemma 10. *Suppose a set of segments S can be covered by a $L \times 2d_t$ rectangle. For any such rectangle and any line ℓ that makes an angle of at most α with the perpendicular on the rectangle, the length of the projection window is at most $L \sin\alpha + 2d_t \cos\alpha$ (see Figure 4.6).*

Proof. The longest projection window is obtained when the endpoints of the segments are on the sides of the rectangle. Suppose there are only 2 segments of length L that are $2d_t$ apart as in Figure 4.6. The result follows from trigonometrical calculations.

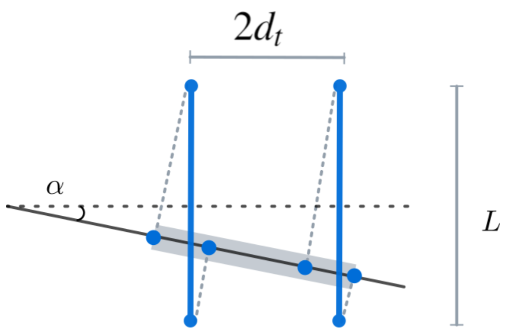


Figure 4.6: The line makes an angle α with the rectangle.

$\square$

Suppose we are in the worst case where there are n segments of length L (the maximum length of a route) split in 2 clusters of overlapping segments at distance $2d_t$ from each other. Only a line that is perpendicular to the n segments can discover that they all fit in a tube of width $2d_t$. If there is a line at angle at most α from the perpendicular, then we would need a window of length $L\sin\alpha + 2d_t\cos\alpha$, to discover this cluster (when α goes to 0, the length goes to $2d_t$). Suppose we allow an error δ in the width of the tube. Making $\delta = L\sin\alpha + 2d_t\cos\alpha - 2d_t$, means we can choose the lines to be at angle at least α from each other and all the clusters will fit in a tube of width at most $2d_t + \delta$.

Lemma 11. *Given a set of n points, checking if there exists a $L \times 2d_t$ rectangle that covers them can be done in $O(n\log(n))$.*

Proof. Similarly to Theorem 6, we consider the convex hull of the points and a rectangle $L \times 2d_t$ that intersects at least 3 of the vertices - an edge and its antipodal vertex. We can check if all the points are covered by the rectangle in constant time. We keep rotating the rectangle such that each edge is on a side of the rectangle. If none of the rotations gives a full coverage, then there exists no covering rectangle. Since there are at most n edges of the convex hull, and it takes $O(n\log(n))$ to build the convex hull, the total running time is $O(n\log(n))$. $\square$

For each $2d_t + \delta$ window we check if any of the sets they cover can fit in a $L \times 2d_t$ rectangle. Based on the previous results, we propose the following greedy algorithm for finding a set of rectangles that covers the maximum number of segments:

1. Consider m lines centred at the middle of the the set of segments at angle $\alpha = 360°/2m$ from each other. Compute $\delta = L\sin\alpha + 2d_t\cos\alpha - 2d_t$.

2. Project all segments on the m lines.

3. For each line consider the windows of length $2d_t + \delta$ starting from every projected point.

4. For each window find the set of segments that can fit in a rectangle of width $2d_t$. This will give the coverage count for each window.

5. Find the window with the most covered segments.

To illustrate the algorithm consider the set of segments in Fig.4.7a and the 2 perpendicular lines. In Figure 4.7b(b) we see all the intervals of length $2d_t + \delta$ starting from every projected point on the horizontal line. The grey cluster is the one that covers the maximum number of pairs. The pink and blue segments belong to one cluster and the green segment makes a separate cluster.

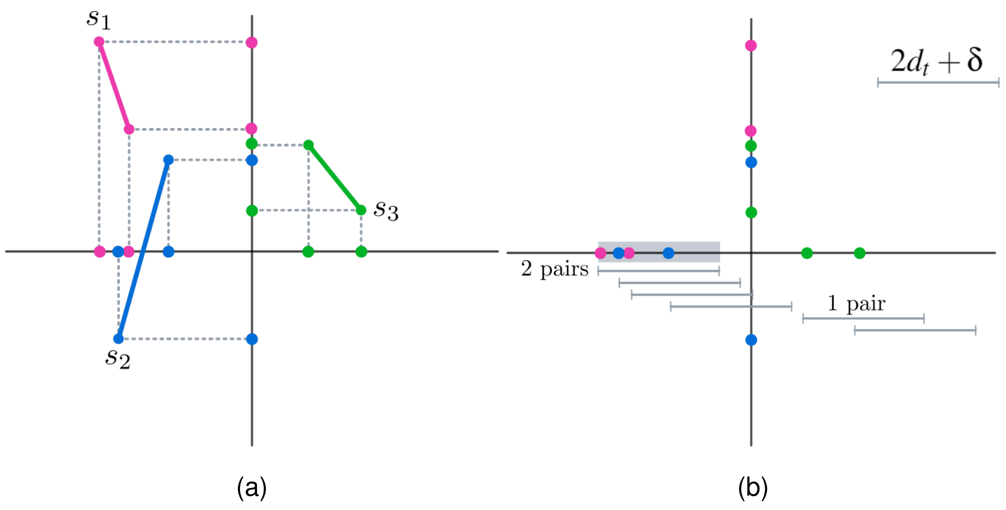


Figure 4.7: An example for 2 lines and 3 segments

$O(n^2)$ algorithm for finding the maximum coverage rectangle

In practice, to speed up the implementation, we ignore the δ error and consider only projection windows of length $2d_t$. In this way, it is no longer necessary to check whether the segments corresponding to the projection window fit in a $L \times 2d_t$ rectangle. This gives a $O(n^2)$-time heuristic algorithm. The medians of the clusters will then be matched to the road network, by finding the shortest path between the nodes closest to the endpoints.

Because of the δ error, the approximation ratio for the greedy set coverage problem, is slightly worse than $\log(n)$ in our case. The error in approximation depends on the number of projection lines chosen and the rectangles length L (d_t given). We discuss how the error can be minimised in the next section. Before that, we give details about solving the rectangle coverage problem in the road network.

We give an alternative algorithm that works directly on the road network, as opposed to clustering trips in the Euclidean space. As before, $G=(V,E)$ is the road network and S the set of pairs of origin-destination locations. For each location we find the closest node in the network. Let V_s be the set of pairs of nodes in the graph corresponding to the set of trips. We first define $d_t - neighbourhood$.

Definition 19 ($d_t - neighbourhood$)**.** *Given a path in a graph, the $d_t - neighbourhood$ is the set of nodes such that the distance from any node to the path is within a distance d_t.*

Suppose we know the shortest paths between any 2 nodes in G. If some pairs have multiple shortest paths, then the nodes can be perturbed slightly so that there will be only one shortest path. Alternatively, other metrics (time of travel between the two nodes, congestion, number of turns etc.) can be used as a tie breaker. For each shortest path we calculate its $d_t - neighbourhood$ and count the number of trips it covers. Then we greedily pick the neighbourhood that covers the highest numbers of trips.

This algorithm eliminates the errors coming from matching the medians of the clusters to the road networks. However, in practice and in our experiments, the $O(n^2)$ algorithm involving projections on lines gives good results and is much faster. To make the transition to the road network, for each rectangle in the plane we consider its median and the nodes in the network that are closest to the median. The shortest path between the nodes is the path corresponding to the rectangle. In a large graph this may have the issue that long shortest paths have a much larger length in the graph than in the plane. To circumvent this and other issues we introduce *multi-resolution clustering*, described in the next section.

4.4.1.2 Multi-resolution clustering

The previous section describes an algorithm for covering segments with rectangles of width $2d_t$, but with no bound on the length, other than the one imposed by the boundaries of the map. This brings a few issues:

- longer rectangles will contain the most trips, but will not necessarily find the dense clusters of trips;
- longer rectangles might have empty portions where a path is not necessary;
- projecting long rectangles on the lines gives a larger error;
- when transitioning to the road network, long paths will have a large distortion.

Our proposed solution to these issues is a multi-resolution framework. We split the trips based on a grid and perform the projection algorithm in each of the cells of the grid. We then consider translations of the grid and grids with larger cells until all trips are covered. For every grid we merge the clusters and add them to the existing network, as we describe in the next section.

4.4.2 Global merging

All trips that were covered so far belonged to a cluster and were served by one particular path, corresponding to the rectangle they were covered by. Other trips can be covered by a combination of paths, as long as the length respects the total distance constraint.

After considering the grid with the smallest resolution, a group of clusters emerges with their corresponding paths. The paths are connected if they intersect. All newly covered trips are removed from the set and the process repeated for translated and larger grids until there are no more trips to cover. With each grid we update the routing network. Once the final routing network is obtained, we match it to the actual road network, by considering shortest paths between the nodes closest to the endpoints of an edge.

4.4.3 Network simplification

When the generated network does not respect the construction constraint (has a total edge cost above T) a subset of the edges is removed. We propose a greedy strategy to remove edges based on their cost and utility. Let us first define two measures of utility.

Definition 20 (Restricted edge betweenness centrality)**.** *Let u_t, v_t be the nodes in the network that are closest to the endpoints of a trip t. We denote by P_S the set of all shortest paths for all trips in set S. Let e be an edge of any path in P_S. The restricted edge betweenness centrality of e is the number of shortest paths from P_s that an edge is part of.*

Note that this is different than the usual definition for edge betweenness centrality because it does not consider the shortest paths between any pair of nodes, but only for a restricted set (given by the trips).

Definition 21 (Edge importance)**.** *Let u_t, v_t be the nodes in the network that are closest to the endpoints of a trip t. We denote by P_S the set of all shortest paths for all trips in set S. Let e be an edge of any path in P_S. The edge importance for e is the number of trips that can no longer be satisfied if it is removed.*

We use either measure of utility of an edge to compute its score, defined below.

Definition 22 (Edge score)**.** *Let e be an edge in the road network with utility u_e and cost c_e. The score of edge e is $\frac{u_e}{c_e}$.*

In the definition of the edge score the utility is given by either measure of importance (restricted edge betweenness centrality or the edge importance). The cost of an edge can be chosen to be the length of the edge or simply 1, in the case when the score is given only by the utility of the edge. The greedy strategy is to remove edges with the lowest score until the construction constraint is met. An alternative solution is to consider this as an inverse knapsack problem and solve using one of the known pseudo-polynomial time algorithms (Andonov et al., 2000).

4.4.4 Network expansion

Consider the case when there exists a transit network in place and the task is to find the next edges to be added to the network to maximise coverage. First, the trips already covered are removed. Then at each iteration (for each grid), after discovering the clusters, the network paths corresponding to their medians are added to the existing network. The covered trips are eliminated from the working set and we continue with the next iteration.

4.4.5 DNG algorithm illustrated

To illustrate how the DNG algorithm works we considered a subset of 200 origin destination trips made with the Santander Cycles hire scheme bicycles (Figure 4.8). The area occupied by the trips is approximately $9km \times 15km$. We chose the transit distance to be $d_t = 750m$ and the stretch factor 1.5. This means that a trip is covered if both the origin and destination are at least $750m$ from a path and if the length of the route using the path is at most 1.5 times the length of the route through the street network.

Split trips by grid. The first step is to generate grids and separate the segments in the corresponding cells (Figure 4.9). In this example we chose only one grid with cells of size half the size of the map. For simplicity we do not consider translations here. Any segment that is not covered in an iteration of the algorithm is considered in the following iterations, in larger grids (or, as in this case, in the whole map).

Project trips onto lines. In each cell we project the corresponding segments on 20 equally spaced, concentric lines (Figure 4.10). On every line we count the number of projected points in each interval of length $2d_t = 1.5km$. All segments that have the projections in one such interval can be covered by a rectangle of width $2d_t$. This allows

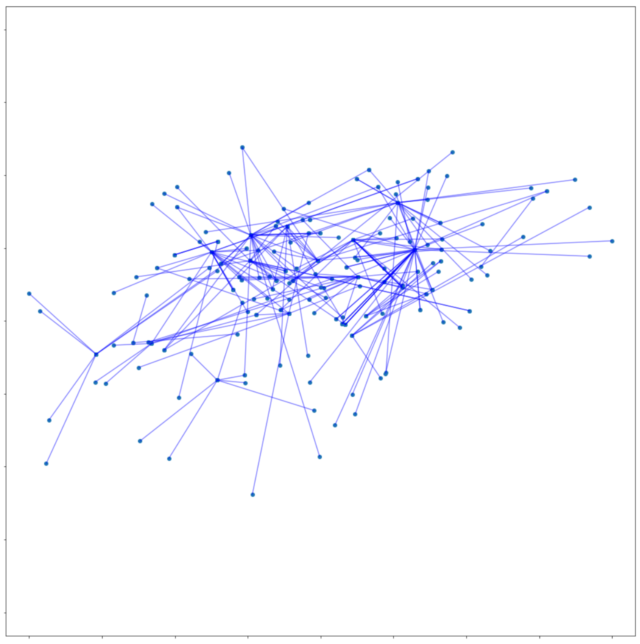


Figure 4.8: A subset of 200 bike trips in the centre of London.

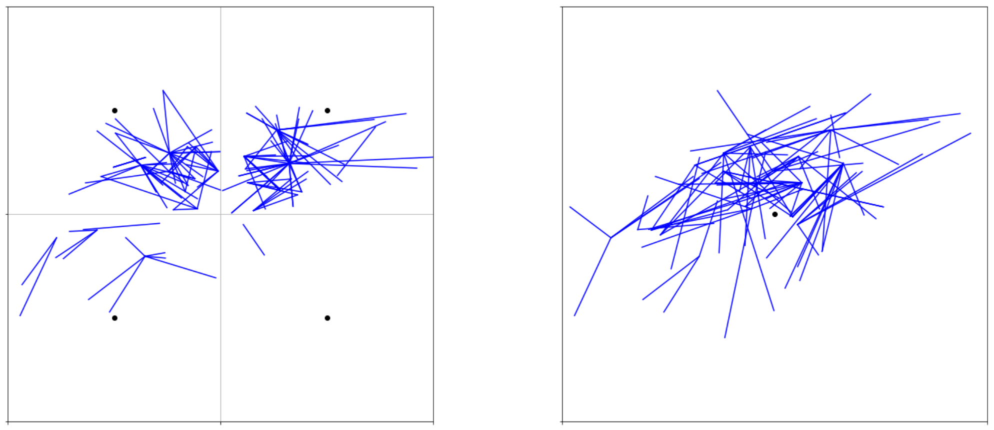


Figure 4.9: The grid on the left corresponds to the first iteration of the algorithm. On the right there are all remaining segments that were not covered in the first iteration.

us to find covering rectangles in which the median is never further than $d_t = 750m$ from any origin or destination. In terms of cycling paths, this means that any user wanting to use a route (the median) will have to ride for at most d_t on a path that is not a cycling route.

Find the densest clusters. In each grid cell we iteratively pick the intervals of length

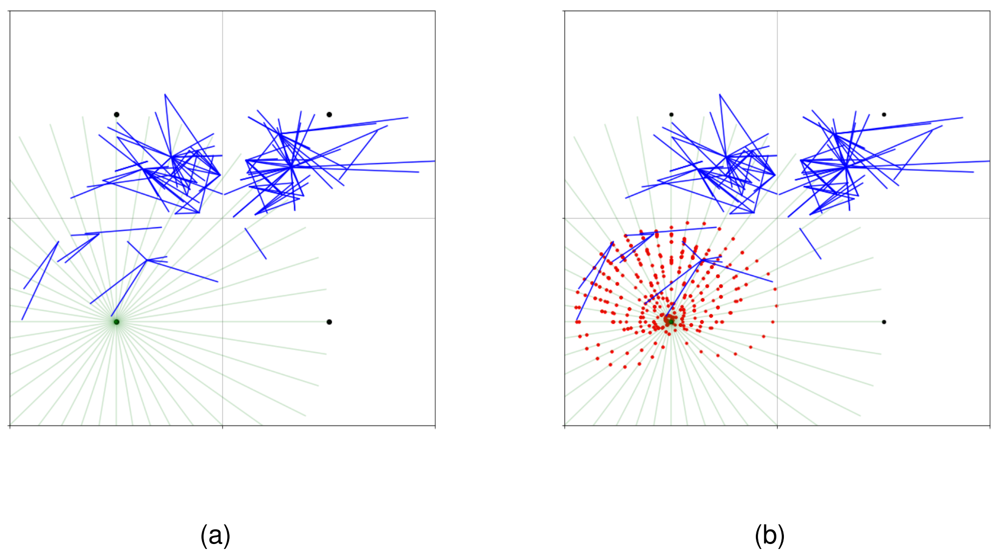


Figure 4.10: (a) 20 projection lines centred at the centre in every cell where there is at least one segment. (b) The projections of every segment in the cell on the projection lines.

$2d_t$ that contain the most projection points (in this example at least 6 projection points - we choose rectangles that cover at least 3 trips). For every such interval we build the covering rectangle, with the median crossing through the middle of the interval.

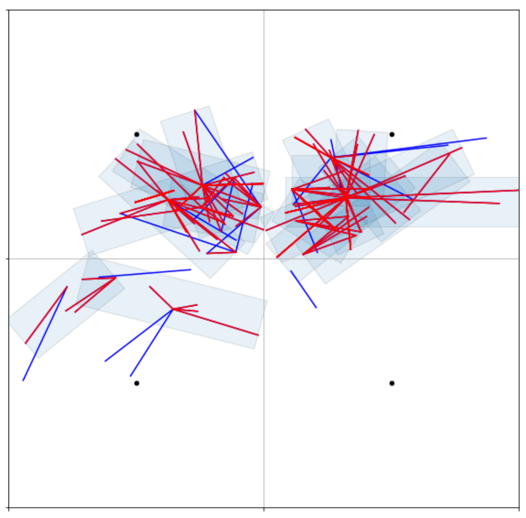


Figure 4.11: The covering rectangles chosen by the greedy approach.

Build routing graph. We build a planar graph using the medians as edges. The nodes are the endpoints of the medians and every intersection of medians (Figure 4.12). A set of trips is fully covered by the rectangles discovered in the previous step. Other trips can be covered by considering transitions between paths. We verify the total distance constraint (stretch factor of any path should be at most $s = 1.5$) for all trips, in or outside the grid considered at this iteration. We repeat the previous steps for the next grid with all trips that were not covered which will give a set of rectangles and their corresponding medians. We update the graph with the new edges.

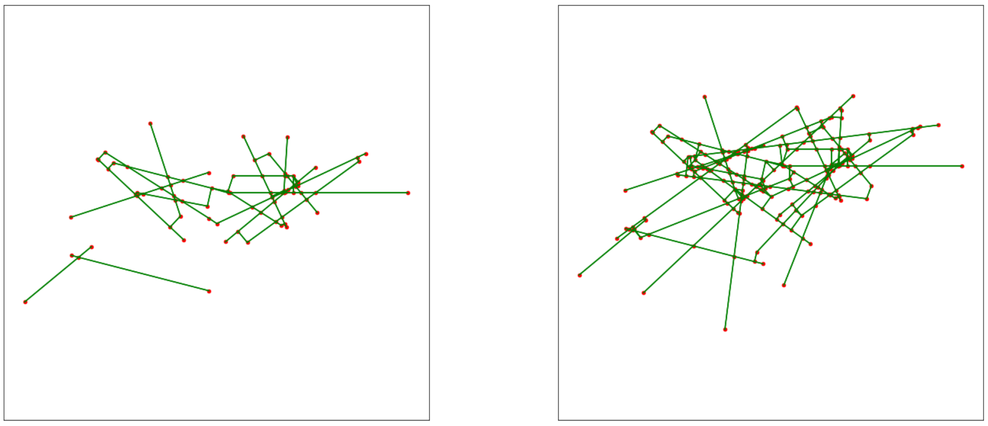
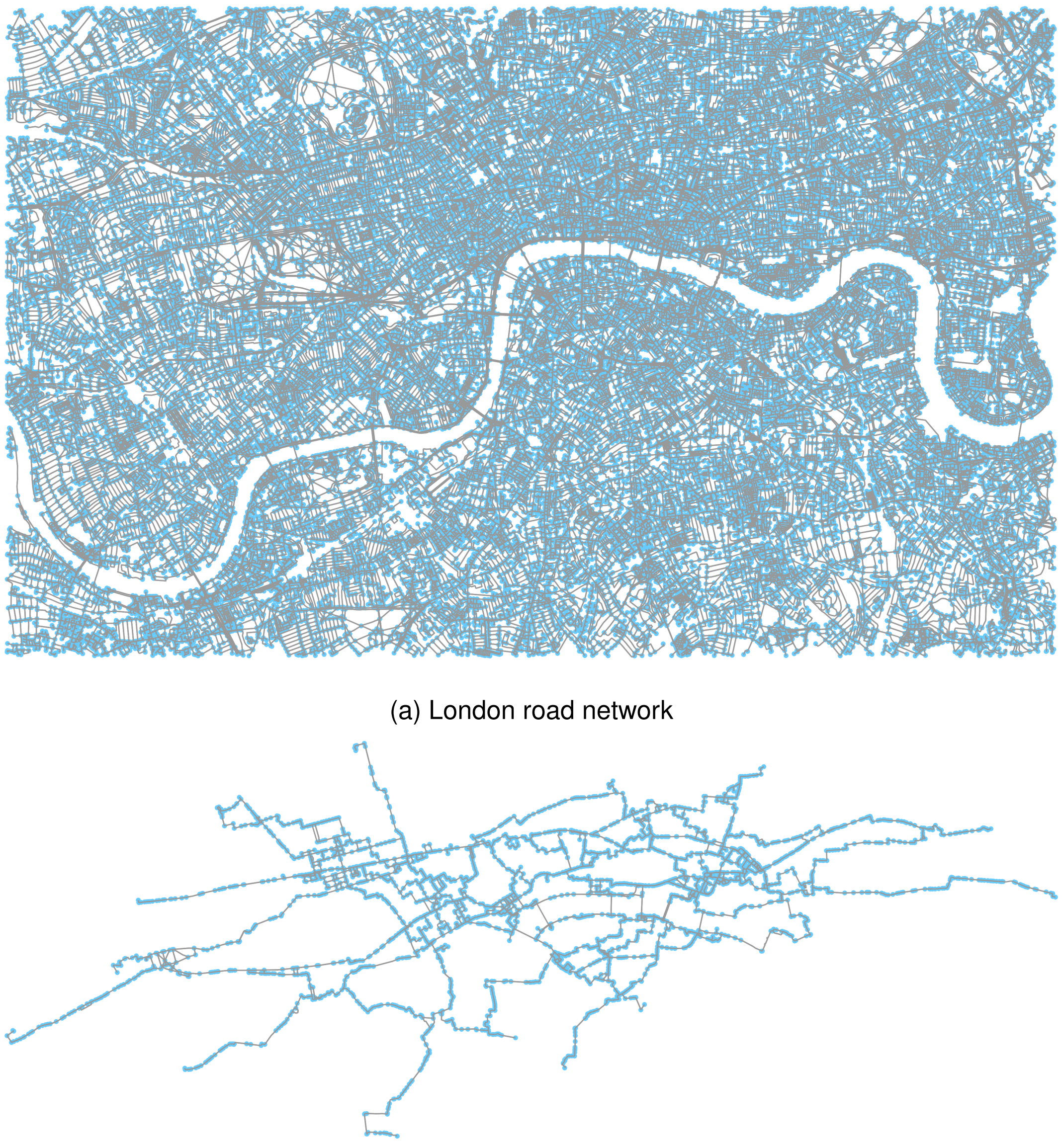


Figure 4.12: The routing graph after the first (left) and second (right) iteration.

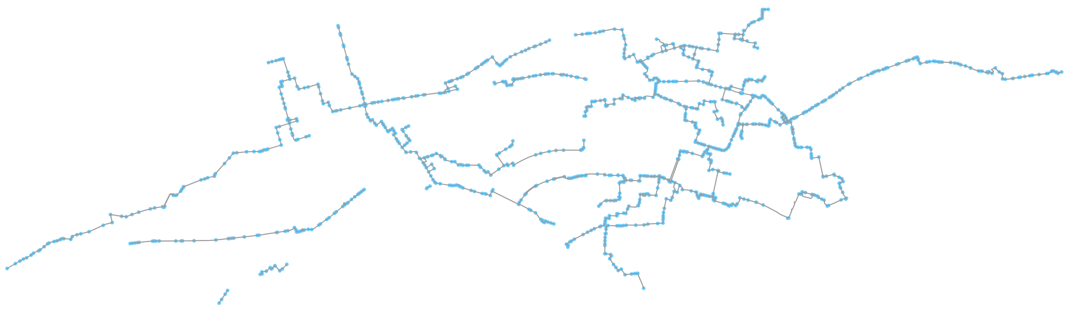
Match the routing graph with the road network. Once the routing graph is finalised, we find the corresponding nodes and edges in the complete road network. For every edge in the routing graph we identify the nearest nodes in the road network; the edge is then represented by the shortest path between the nodes (Figure 4.13). Due to the graph metric not being the same as the Euclidean distance, a subset of trips will not be covered in the matched network, while another subset will. To avoid this we can directly match every edge found in the previous step to the road network. In this way the routing graph is always a subgraph of the road network and the intermediate subsets of covered trips are included in the final set. However, in experiments we notice a negligible difference between the two approaches, therefore we use the first approach for simplicity.

Simplify network. If the total edge length of the matched network is larger than the allowed length, we remove edges iteratively, according to the strategies presented in



(a) London road network

(b) Transit network



(c) Simplified transit network

Figure 4.13: London road network and the networks found with the DNG algorithm.

Section 4.4.3. Here we used the simplest strategy, based on the edge betweenness centrality (the number of shortest paths passing through every edge); we eliminate all edges with edge betweenness centrality 1, which gives the subgraph shown in Figure 4.13 (c).

4.5 Experiments

Our experiments show that the cycle paths network generated with the DNG algorithm cover 20% more trips than the current cycle paths network in London. We give details about the datasets used and the implementation in the next section.

4.5.1 Dataset description

The cycle paths network generated with the DNG algorithm is constructed based on a subset of Santander Cycle hire journeys provided publicly by Transport for London (TfL)[1]. We compare its coverage rate with that of the existing cycle routes network in London, which we build by matching the cycle routes provided by TfL to the OpenStreetMap (OSM) London road network. The three datasets are described below.

Santander Cycles journeys

Santander Cycles is a public hire scheme in London. TfL publishes anonymous user datasets that contain the name of the origin and destination docking stations for each rental. We extracted the data for one week, 21^{st}-27^{th} of August 2019, a total of $242,193$ trips. The origin and destination docking stations were then matched with their respective latitude and longitude locations. Figure 4.14 shows the usage of docking stations (with aggregated origin and destination locations). We notice a number of hotspots concentrated in the city centre; they are usually close to a tube station, consistent with the assumption that bike rentals alleviate the "last mile problem" (DeMaio, 2009), that describes the movement of people or goods to a final destination. The assumption is further supported by the distribution of lengths: about 40% of the trips are less than 2km, as can be seen in the top plot in Figure 4.15. In this work we are looking to

[1]Powered by tfl open data: `https://cycling.data.tfl.gov.uk/` Accessed:2019-12-20.

partially cover trips, therefore we do not consider those with lengths less than 1km. The final lengths distribution can be seen in the bottom plot in Figure 4.15.

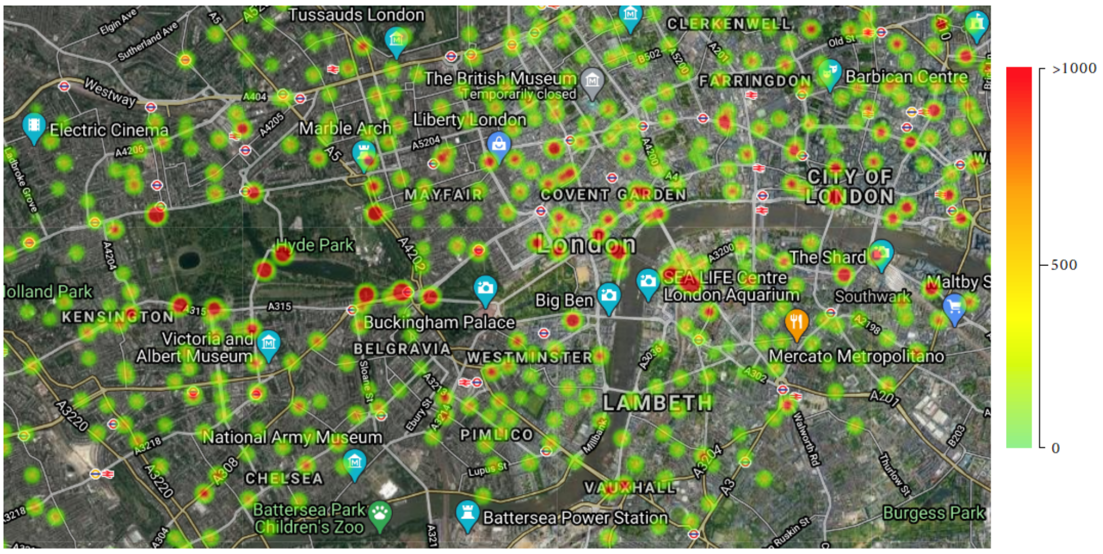


Figure 4.14: The aggregated usage of docking station (count for both arrival and destination).

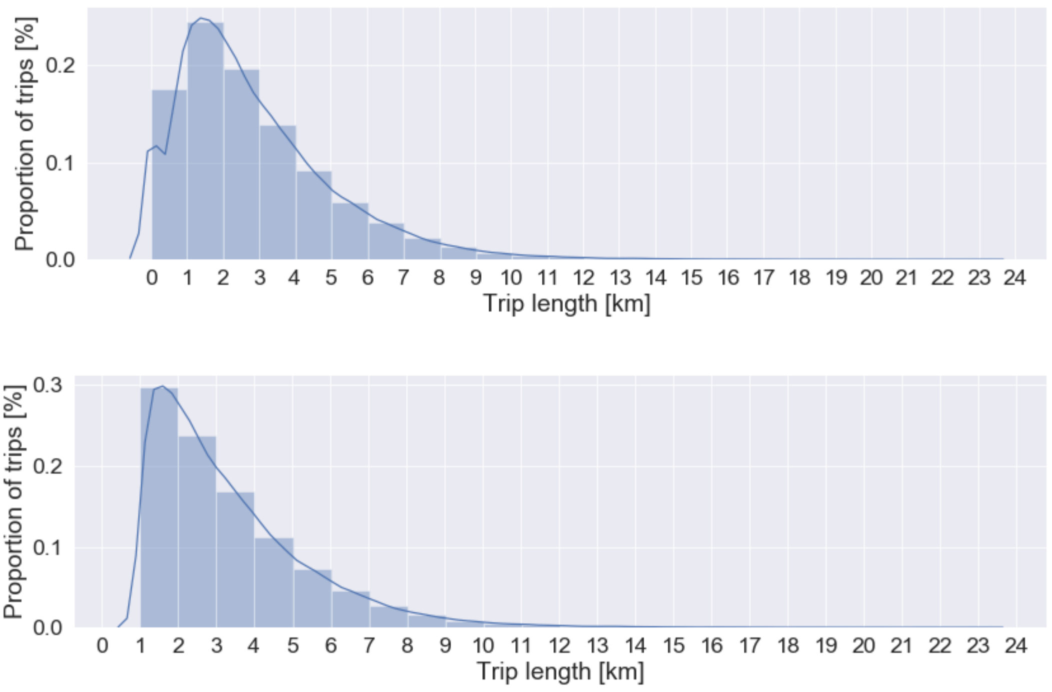


Figure 4.15: Trip lengths distribution before and after removing the trips shorter than $1km$. Most trips are relatively short.

London road network

We selected a portion of the London road network based on the bounding box determined by the hire journeys (coordinates: 51.542138°N, 51.454752°S, -0.002275°E, -0.229116°W). The network, shown in Figure 4.16, was extracted with the Python package OSMNnx (Boeing, 2017) that works with OpenStreetMaps APIs to retrieve and analyse street networks. The resulting London subgraph is composed of 72531 nodes and 183807 edges and has a total edge length of 7993km. The network was used together with the cycle routes data to build the London cycle network.

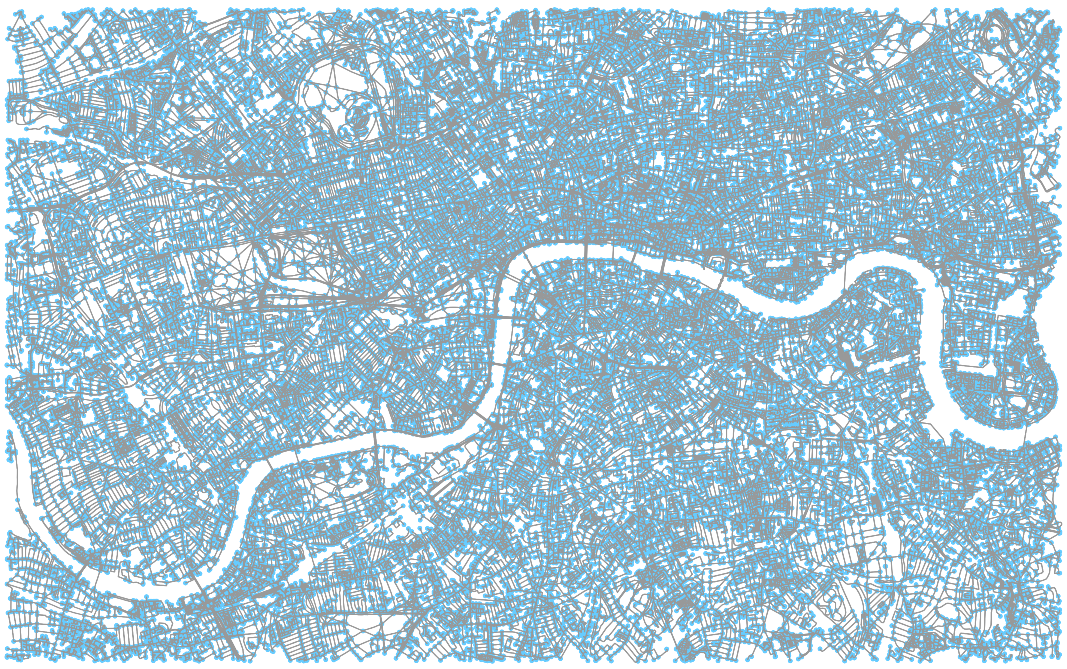


Figure 4.16: London road network

Cycle routes

The routes provided by TfL, also called Cycleways (shown in Figure 4.17), are paths generally segregated from the rest of the traffic; the network is currently being expanded and is estimated to reach 450km by 2024 [1].

The GeoJSON file that contains the data describes the routes' geometry. They are split into line segments, where each segment is a sequence of latitude longitude coordinates,

[1]As mentioned on: `https://tfl.gov.uk/modes/cycling/routes-and-maps/cycleways`

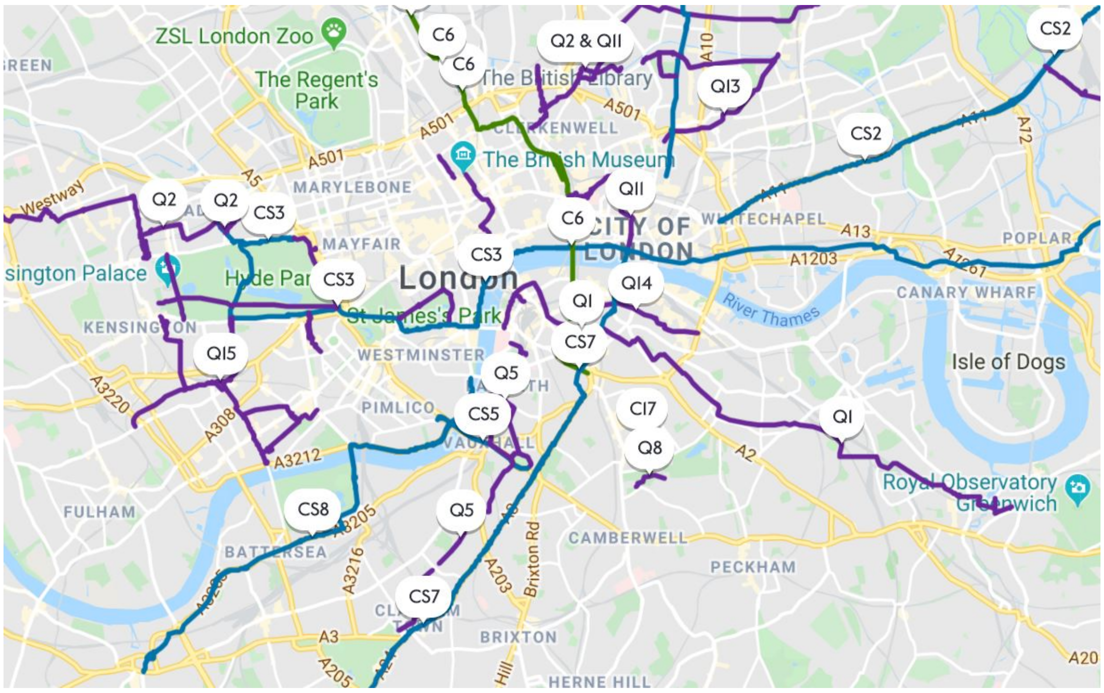


Figure 4.17: Opened London Cycleways and the individual names (e.g. CS2)

which need to be matched to the road network. Map matching is generally not a trivial problem (Newson and Krumm, 2009) due to errors in measurements and multiple candidate matches. In our particular case, the difficulty comes from the missing correspondence between the points describing the lines and the road network. Simply using built-in functions for building the graph misses road network nodes along the paths (see Figure 4.18); these are key in our problem because they allow the entrance and exit all along the paths.

Our approach was to select the nodes closest to the endpoints of each line and compute the shortest path between them in the London road network. In this way we make sure that every node found along the routes is included. The final network (in Figure 4.19) is the induced subgraph determined by the set of nodes in the shortest paths; it has 2842 nodes and 5326 edges and a total edge length of approximately 212km. In our experiments we build a network of similar length and compare the number of trips covered.

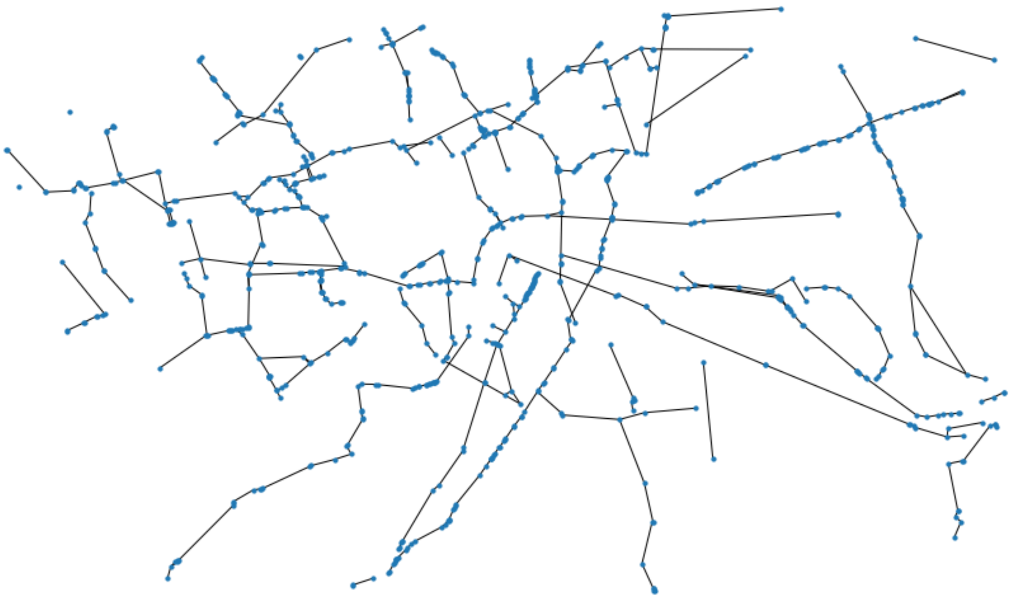


Figure 4.18: Cycleways network (open and planned routes) based on shape

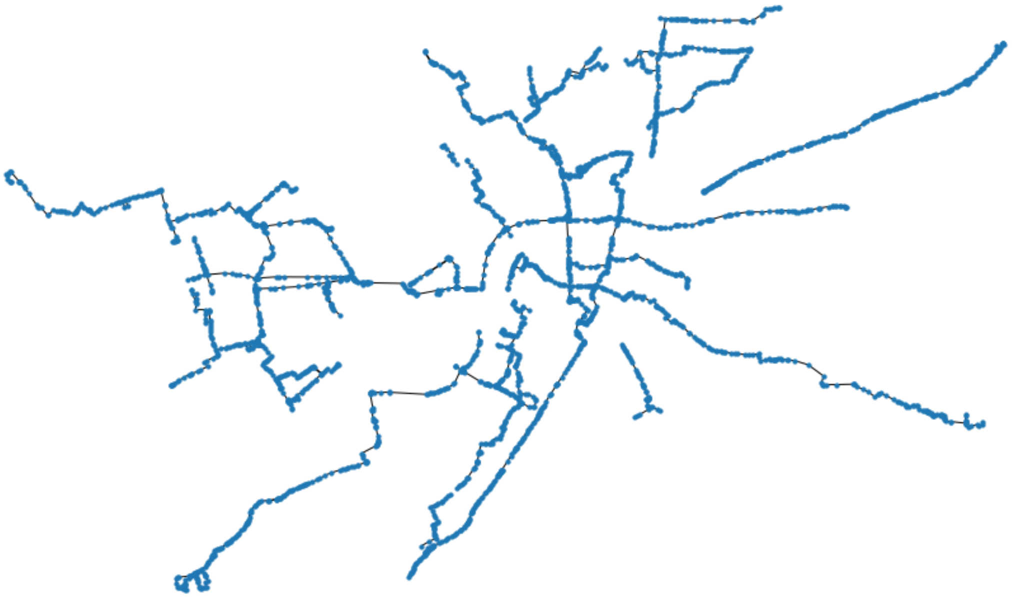


Figure 4.19: Cycleways network (open routes) based on shortest paths

4.5.2 Experimental setup

In the following experiments we set the maximum travel distance to reach a path d_t to be $750m$ and the stretch factor s to be 1.5. The number of projection lines is 50. For the first experiment we consider a set of grids where the smallest cell size is $3.75km$

(5 grids) and we do not consider translations. We impose that every rectangle should cover at least 3 trips. In the comparison experiments the smallest cell size is $2.5km$ and we do consider the grid translations. In all experiments the algorithm stops when it reaches at least 90% coverage.

4.5.3 Trip coverage

The algorithm needs a small number of trips to build a network with good coverage. To show this we selected random samples of different sizes from the set of all trips. Based on each sample we built a network of paths. Then we computed the coverage given by each network for a new sample of 5000 trips. Figure 4.20 shows the number of trips covered for each network (denoted by *Covered*); the figure also shows the number of trips for which the endpoints are close to a path, a route through the network exists, but the stretch constraint is not respected (denoted by *Path exists*). With a small set of 100 trips the constructed network covered more than 1500 out of the 5000 unseen trips. The paths built based on 1000 trips cover more than 75% of new trips. By having a good coverage on a set of trips that was not seen, we show that the algorithm learns the structure of the data. Due to setting a minimum coverage for each rectangle, some trips remain uncovered because there are no other nearby trips and are therefore outliers in the dataset.

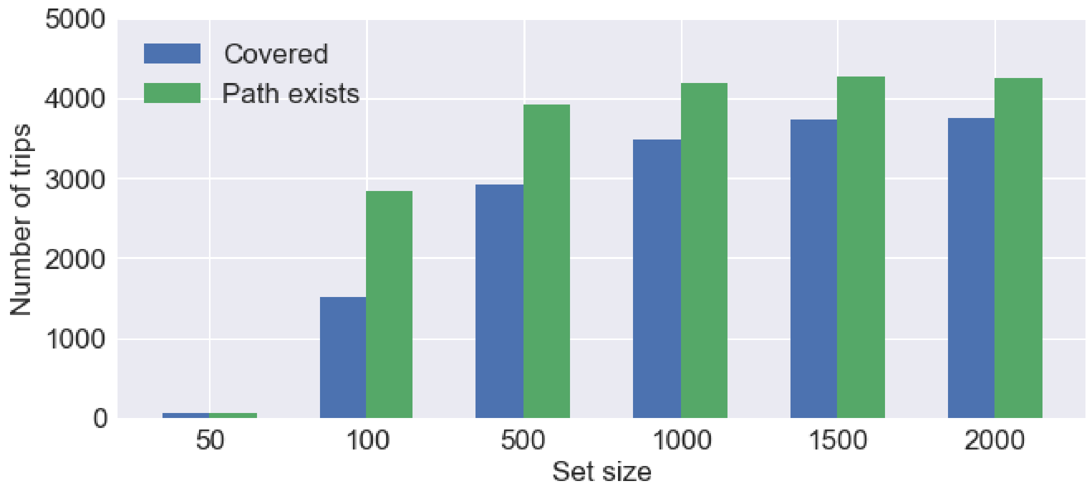


Figure 4.20: The number of trips out of 5000 previously unseen trips covered by a network built based on 50, 100, 500, 1000, 1500, and 2000 trips.

4.5.4 Comparison with current cycling network

For this experiment we extracted a random sample of 500 trips from the total set. Based on this we generated a network that was then simplified to reach the same total length as the Cycleways network, the current bike lane system in London. We then considered sets of different sizes containing trips that were previously not seen. For each set we computed 3 quantities: the number of trips that are close to a lane, the number of such trips where a path exists through the network and the number of trips for which the stretch constraint is also valid. Figure 4.22 shows that the proposed network consistently covers more trips than the current one, usually with 20% more trips. In this setup we used the most basic network simplification technique, where we considered the cost of each edge to be 1 and the utility the number of trips that use it.

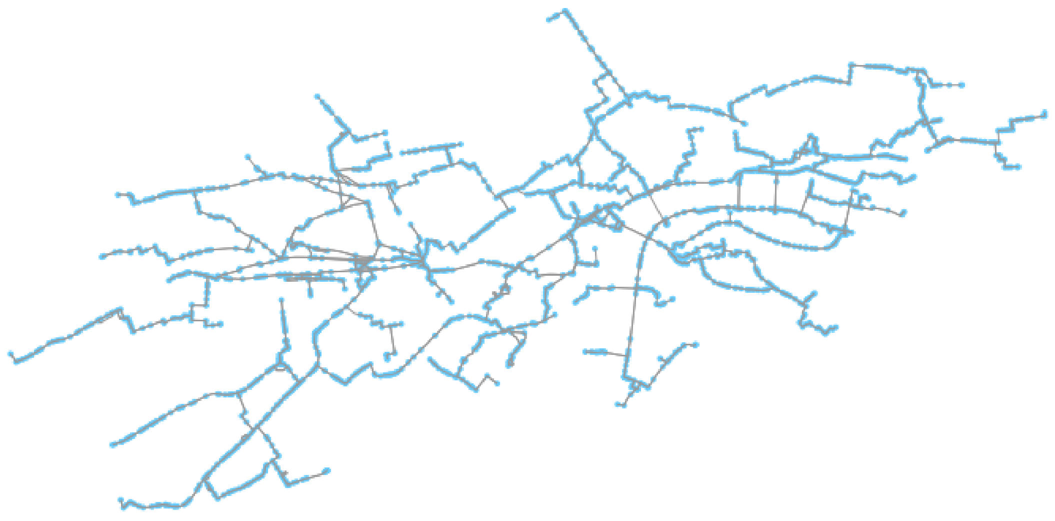


Figure 4.21: The resulting simplified network.

Figure 4.22 shows that the Cycleways network provides more lanes that are close to the origin and destination of the trips than our proposed network. That can be partly due to the nature of the data - it is possible that bike docking stations are placed closer to existent bike paths. Regardless of this, one reason why the final number of covered trips is less than in our proposed network is the lack of continuity between paths in the Cycleways network (comparison of blue and green bars in Figure 4.22). Improving the connectivity of the cycling network is widely recognised as a main objective of cycling infrastructure planning (Furth and Noursalehi, 2015). Our proposed method can be used as a tool to improve the current network and find the street segments likely to increase the number of trips covered, without introducing large detours.

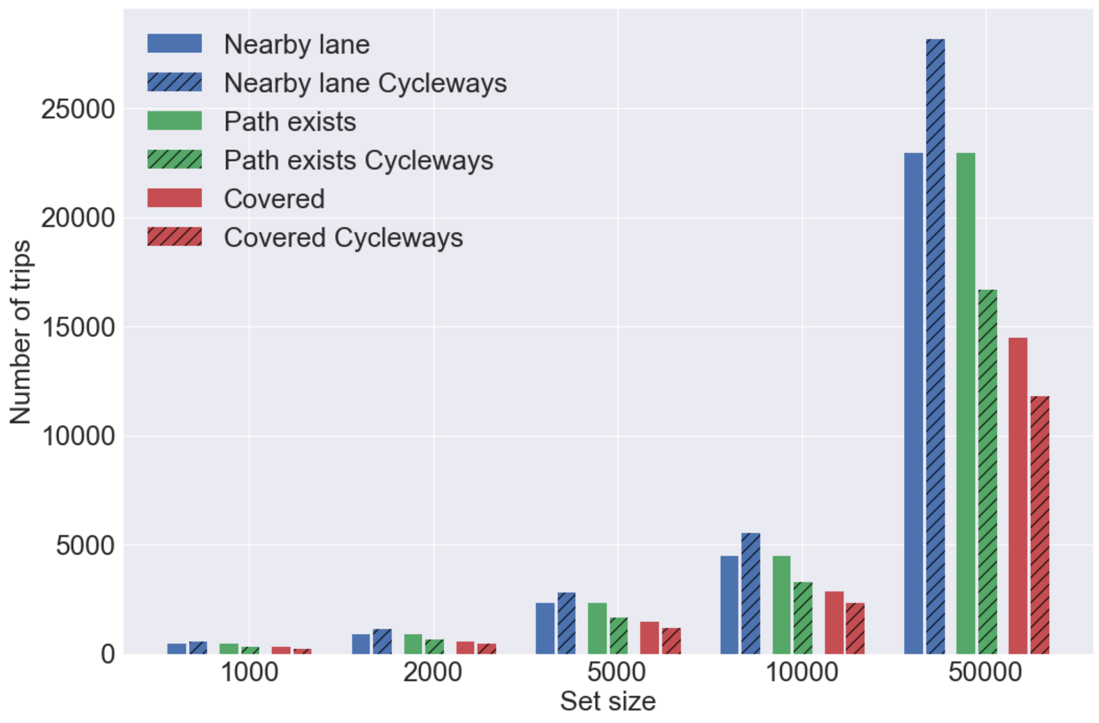


Figure 4.22: Comparison on the number of trips covered with the Cycleways network.

Chapter 5

Regional factors in an ensemble framework for photovoltaic power prediction

5.1 Introduction

World energy consumption is predicted to increase by 50% by 2050[1] due to urbanisation and increased standards of living. Given the high levels of greenhouse gases emissions caused by traditional energy sources, demand for sustainable alternatives is on the rise, with renewable sources being the fastest growing. Including renewable sources in the current energy system is difficult because power output forecasting, crucial to the optimal planning and running of renewable power plants, is challenging under variable weather conditions.

In this work we focus on forecasting solar photovoltaic (PV) power production. As one of the most popular sources of renewable energy, it is found in residential, commercial, off-grid and utility-scale domains and accounts for more than 2% of the global power consumption[2]. The generated power is the result of the complex interaction of atmospheric factors, which by themselves are difficult to model due to the chaotic nature of the atmosphere (Zeytounian, 1991). Some of the challenges specific to solar power forecasting are the rapid, stochastic formation and movement of clouds, which can

[1]Source: `https://www.eia.gov/todayinenergy/detail.php?id=41433`
[2]Source: `https://ourworldindata.org/renewable-energy`

lead to sudden, large fluctuations in the power output, and quantifying the impact of fog and snow on the energy production. It is critical to be able to predict the intensity and duration of fluctuations. This type of fluctuations make the PV power time series non-stationary; the power production is noncontinuous with a behaviour that resembles a long memory process (Perpin and Lorenzo, 2011). These factors pose challenges in PV power production forecasting.

Machine learning has the potential to mitigate some of the challenges related to PV power production forecasts. The increasing available volume of both energy and weather data enables the usage of new technologies and data-intensive algorithms for more accurate forecasts. One advantage of machine learning techniques is that the interacting factors do not need to be modelled explicitly; while less transparent than physical models, they often give high accuracy in practice. A further advantage is the possibility to handle the noise inherent to weather forecasts indirectly.

The accuracy of forecasting models is highly dependent on the spatio-temporal context (Perez et al., 2016). As we describe in the next section, prediction models are needed for a variety of time frames and spatial resolutions. In this work we focus on next day prediction for power output with a resolution of 30 minutes, which is a typical setting useful in practical applications. With the temporal dimension fixed, we assess the impact of the spatial component. It has been shown that aggregating the output of systems of power plants leads to the smoothing of the power timeseries, due to spatial correlation (Graabak and Korps, 2016); this implies that predicting the output of a system of power plants gives more accurate results than predicting individual outputs. However, the strength of the effect depends on the spatial distribution of the plants (Wiemken et al., 2001). Fonseca Junior et al. (2014) study the smoothing effect in relation to support vector machine models for prediction in a few different regions. They show that reducing the spatial correlation in the input data improves the accuracy of the prediction models. We investigate how this applies to other machine learning models and analyse the impact of other spatial factors on the accuracy of predictions for a suite of frameworks.

Our contributions. Firstly, we propose a day-ahead PV power prediction model for systems of solar power plants in regions of Hokkaido, Japan. The predictions are made based on numerical weather forecasts, clear sky forecasts and a persistence model. The ensemble architecture combines the outputs of a recurrent neural network, a support vector machine model, and a multivariable linear model, which exploit different patterns in the data. The ensemble gives a higher accuracy than the individual models in terms of root mean square error. Secondly, we investigate the influence of spatial factors on the accuracy of the ensemble and individual models. We show that having separate local models for self contained areas (based on similar weather conditions) gives higher accuracy than a single model that predicts the total output. Moreover, models including the weather forecasts from the neighbouring region perform better than the ones restricted to the area of prediction. This is particularly interesting given that the effect is noticed in both temporal and non-temporal models. We show that the ensemble model can be further improved by learning the weights of the individual models based on their performance on the training step. The models have varying performance depending on the area of prediction, which justifies both building the ensemble with different types of models and splitting the larger area into regions with homogeneous weather conditions.

We continue by summarising recent related works in Section 5.2, where we give an overview of different types of forecasting models. Section 5.3 describes and analyses the input and output data, and presents the format used by our model. In Section 5.4 we describe the ensemble model and the feature selection process. Lastly, the experiments set-up, results and analysis are presented in Section 5.5.

5.2 Related work

Contrary to conventional energy sources, which most often can be precisely planned, renewable sources have a variable, dynamic nature, largely dependent on weather conditions (Shivashankar et al., 2016; Sobri et al., 2018). Integrating these sources into the current energy system at a large scale without destabilising it poses a series of challenges, amongst which: compensating for the intermittent production and building accurate forecast methods for reliable planning. Having better production predictions lowers the running costs of renewable energy systems because it reduces the need for alternative sources of energy. Depending on the geography and climate of the deployment area and the properties of the system, making accurate predictions can be difficult - while there are models that forecast individual weather conditions at various degrees of accuracy, it is the complex interplay between these factors that translates into the amount of energy produced.

Photovoltaic power output. In this thesis we focus on the production of solar photovoltaic (PV) electric power. A photovoltaic system is a power system comprised of solar cells that convert the energy of light into electricity through the photovoltaic effect, a chemical and physical process. The electrical efficiency of a photovoltaic system depends on the physical properties of its cells. These properties, the installation arrangements and meteorological factors determine the electric power output. Therefore, the difficulty in predicting the power output is at least as high as for predicting the individual meteorological factors. Out of these, cloud cover has a large influence on the changes in the output - a passing cloud can cause a change of more than 60% of the peak output in a few minutes (Mills et al., 2011). We refer to the level of changes in the power series describing the power output as the *variability* of the output. Intuitively, a smooth time series curve is easier to predict than a highly variable one. The temporal resolution of the series has an influence on the level of variability, with shorter time steps increasing the variability. Within a PV plant or system of plants the aggregated output is often less variable than the individual output, depending on the geographical distribution. This is known as the *smoothing effect* caused by *spatial correlation*.

Weather factors. Reliable weather forecasts are critical for accurate power output predictions. Mathematical grid models for numerical weather predictions use systems of differential equations that govern atmospheric motion and evolution (Kalnay, 2003), which describe basic conservation laws. There are several issues that arise: the analytical solution for these equations is impossible and therefore approximate solutions are needed; the quality of the solution depends on the quality of the measurements of the initial conditions (which are difficult to obtain, especially in areas such as oceans); the resolution of the grid (usually between 1-5km for regional models) is larger than the scale of a weather process, such as cloud formation (a typical cumulus cloud is less than 1 km). In the case of cloud coverage, satellites and ground-based sensors can give information for short term predictions. However, for longer time scales cloud movement is difficult to predict; clouds change shape, speed or dissipate depending on the atmospheric conditions and all these factors influence the level of solar irradiation that reaches the power plants. The challenging nature of forecasting weather factors accurately, especially cloud formation and movement, reflects why predicting PV power output is also difficult.

Accurate PV power forecasts are needed at a wide variety of temporal horizons, from a few seconds (also called *nowcasting*) to a few days. Short term predictions are usually for time intervals up to a few hours, medium term for next day predictions, while long term refers to a time frame of a few days. Depending on the resolution of the data and its type, different approaches are at hand; they are most often split into: statistical, physical, and hybrid (which combines the previous two). The next paragraphs give a brief overview of some of these methods.

5.2.1 Statistical models

Linear models

The *persistence model* (PM) is a fairly common reference tool for solar power prediction (Diagne et al., 2013). It predicts that the output is the same at time $t+1$ as it was as time t, where the time frame can range anywhere from minutes to days or years. While the results for short term forecasting (at the scale of minutes or 1h) can be fairly accurate, predictions are generally inaccurate for a horizon larger than 1h. As it is a common benchmark in the literature, we will use a persistence model to compare our model's predictions.

Autoregressive moving average (ARMA) and *Autoregressive integrated moving average* (ARIMA) models are popular in prediction of timeseries because of their ability to utilise statistical properties of the data. They are a combination of two elementary models: autoregressive (AR) and moving average (MA). The AR part refers to regression on pasts values, while the MA part models the error as a combination of errors from the past. ARMA models can only be applied to time series that are stationary (with consistent statistical properties over time); ARIMA is an extension that allows modelling of non stationary series. Both models rely solely on past data, which led to the creation of *autoregressive moving average models with exogenous inputs* (ARMAX) (Li et al., 2014) which allow the inclusion of external factors, such as temperature and humidity in our case. Autoregressive models perform better than persistence models, with ARMAX achieving highest accuracy. However, they are also mostly used for short forecast horizons, on the scale of a few hours (Li et al., 2014).

Linear regression (LR) models describe the correlation between input variables and output by fitting a linear equation. Multivariable linear regression (MLR) is a generalisation of simple linear regression, where there is still one output variable, but multiple explanatory variables. Stand-alone linear regression models have difficulties in modelling the complex relationships between environmental and physical factors and power production (Abuella and Chowdhury, 2015; Deo and Şahin, 2017).

Linear models have the advantage of being interpretable, making the feature selection process more transparent. This is particularly relevant when dealing with multiple sources of exogenous input (in the PV power forecast case - the weather factors), that have varying impact on the predictions. Nonetheless, in practice, non-linear models often give better performance (Sobri et al., 2018).

In this work we use a Multivariable linear regression model as part of the ensemble.

Multivariable linear regression (MLR)

The simplest model in the ensemble is the multivariable linear regression (MLR) model, which finds a linear relationship between the input variables and the power output at each point in time. Using the notation defined in the previous section, the prediction given by the *MLR* model for region *R* for day *d* at time *t* is:

$$P_{MLR,R}(d,t) = \beta_0 + \beta_1 X_p(d,t) + \beta_2 X_{CS}(d,t) + \sum_{\substack{(x_i,y_i)\in R \\ j\in\{1,\dots,6\}}} \beta_{3,i,j} X_{NW}(x_i,y_i,d,t,f_j). \tag{5.1}$$

Non-linear models

Support vector machines (SVMs) are supervised learning models that can solve non-linear classification and regression tasks. They are able to find more complex patterns than linear models and are faster, can prevent overfitting, and can have better convergence than neural networks, which can be trapped into a local minimum. In the context of solar power prediction, SVMs have been used successfully in conjunction with other heuristics and prepossessing of the data. For example, Shi et al. (Shi et al., 2012) first cluster days based on the weather conditions into sunny, foggy, rainy and cloudy days and they build a separate SVM model for each type of day. Then, according to the weather forecast for the following day, they predict the solar output using the corresponding model.

We give more details about the implementation of SVMs in our ensemble.

SVM models are non-probabilistic binary linear classifiers that can be extended to solve regression problems. In the case of two-class classification, the model solves the optimisation problem of maximising the *margin*, which is the distance between the decision boundary and the data points belonging to different classes that are closest to each other. Thus, it finds the hyperplane that best separates the data under the condition that the training examples are classified correctly. For regression, the same ideas apply, with the slightly different condition that the predictions for the training instances are within an error from the true values. When the input data shows a complex nonlinear relationship with the output, it is mapped using a kernel function into a higher dimensional space, where a linear discriminator can be found.

The regression function has the form $f_\omega(\boldsymbol{x}) = \omega \cdot \phi(\boldsymbol{x}) + b$, where ϕ maps the inputs to a high-dimensional space. The function f_ω is optimal when it minimises the regularised ε-insensitive loss function $L = \sum_{i=1}^{n}(o_i - f_\omega(\boldsymbol{x_i}) + \frac{1}{2}||\omega||^2$. Considering each timestamp from the training set independently as $\{(\boldsymbol{x_1}, o_1), \dots, (\boldsymbol{x_n}, o_n)\}$ the regression optimisation problem is formally formulated as:

$$\begin{aligned}
\text{minimise} \quad & \frac{1}{2}||\omega||^2 \\
\text{subject to} \quad & o_i - (\omega \cdot \phi(\boldsymbol{x_i}) + b) \leq \varepsilon, \\
& (\omega \cdot \phi(\boldsymbol{x_i}) + b) - o_i \leq \varepsilon, \\
& i = 1, \dots, n,
\end{aligned} \tag{5.2}$$

where the notations are:

- o_i - the output for training instance i
- n - the number of training instances (in our case the *number of days* $\times$ *number of measurements per day*)
- ε - the width of the error-insensitive tube, which describes the maximum deviation from the actual output.

In this case it is assumed that the function f exists such that all data points are in the $f \pm \varepsilon$ tube. This is extended to allow a number of errors (data points outside the tube) by introducing a *soft margin*. Given a positive constant C - known as the regularisation parameter - and two positive constants ξ, ξ' - slack variables - that measure the deviation from the tube, the optimisation problem becomes:

$$\begin{aligned}
\text{minimise} \quad & \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{n}(\xi + \xi') \\
\text{subject to} \quad & o_i - (\omega \cdot \phi(\boldsymbol{x_i}) + b) \leq \varepsilon + \xi, \\
& (\omega \cdot \phi(\boldsymbol{x_i}) + b) - o_i \leq \varepsilon + \xi', \\
& \xi, \xi' \geq 0, i = 1, \ldots, n.
\end{aligned} \tag{5.3}$$

This is known as *Support Vector Regression* and can be transformed into its dual quadratic programming problem using Lagrange multipliers:

$$\begin{aligned}
& \text{minimise} \frac{1}{2}(\boldsymbol{\alpha} - \boldsymbol{\alpha^*})^T K(\boldsymbol{\alpha} - \boldsymbol{\alpha^*}) + \varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} o_i(\alpha_i - \alpha_i^*) \\
& \text{subject to} \quad \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0, \\
& \qquad\qquad 0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \ldots, n,
\end{aligned} \tag{5.4}$$

where $\boldsymbol{\alpha}, \boldsymbol{\alpha^*}$ are the Lagrange multipliers and K is a $n \times n$ symmetric matrix given by the chosen kernel function.

Radial basis functions (RBF) are a popular choice for the SVM kernel function. In the implementation used in our model it is defined as:

$$K_{i,j} = e^{-\gamma|\boldsymbol{I_i} - \boldsymbol{I_j}|^2}$$

where $\boldsymbol{I_i}$ and $\boldsymbol{I_j}$ are the i^{th} and j^{th} vectors of measurements and $1/\gamma$ is the number of features of each input vector. RBF kernels give good results in practice and are easier to calibrate than other kernels.

Artificial neural networks (ANNs) are perhaps best suited to model non-linear, dynamic data for a wide range of time horizons, especially in the case of noisy, uncertain input, which is the case with weather forecasts. One important advantage is their capacity to model data with high volatility, without imposing restrictions on the type of data or on its statistical properties, such as known distribution. Recurrent Neural Networks (RNN) achieve state of the art performance in many temporal modelling problems (De Mulder et al., 2015; Lipton, 2015). The more advanced RNN layers possess an additional memory unit that can identify correlation between inputs and future outputs in temporally-indexed sequences of values. The RNN models proposed for PV power forecasting differ in their architecture and input data. Abdel-Nasser and Mahmoud (Abdel-Nasser and Mahmoud, 2019) propose a series of long short-term memory (LSTM) RNN models with different architectures to predict hour-ahead power output. Like all RNNs, LSTMs model the temporal changes, but they have the added advantage that they avoid long term dependency problems such as vanishing or exploding gradients (Bengio et al., 1994). Gated recurrent unit (GRU) RNNs are similar to LSTMs, but have fewer parameters, which makes them more computationally efficient. They outperform LSTMs on certain tasks (Jozefowicz et al., 2015) and are better suited in cases when there is less training data. Wang et al. (Wang et al., 2018) show the performance of GRUs in the context of short-term PV power predictions.

The most complex model in the ensemble is a Gated recurrent units network, described below.

Gated recurrent units network (GRU)

Artificial neural networks (ANN) are widely used in power output forecasting due to their ability to extract patterns from highly complex, multidimensional data such as weather forecasts. Generally, the stages in building an ANN model are: (1) splitting the data into training, validation and test; (2) choice of architecture and parameters; (3) training and validation, in which the learnt weights of the network change until a certain level of accuracy is reached; (4) testing on unseen data.

Recurrent neural networks (RNN) are a type of network that is adapted to work with sequential data. Standard RNNs are difficult to train because they keep track of the weights from every node that contributed to the calculation of the output; this gives the exploding and the vanishing gradient problems (Bengio et al., 1994). We used a particular RNN layer design called Gated Recurrent Units (GRU) (Chung et al., 2014) that is represented in Figure 5.1 (more details about the notation are given in later paragraphs). GRU networks address the exploding and the vanishing gradient problems by adding update and reset operations that choose which information should be passed on to the next layer.

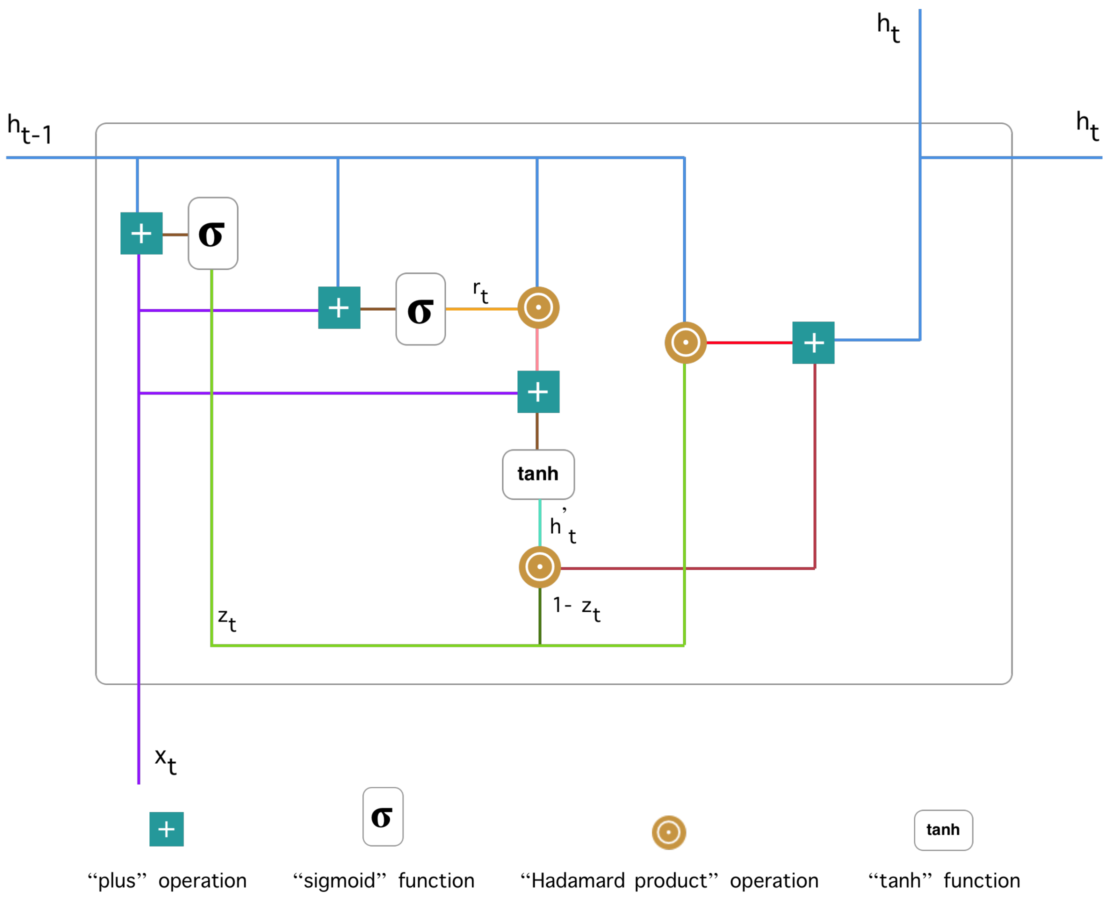


Figure 5.1: Gated recurrent unit[1]

We briefly describe how the activation of a GRU works. The state at time step t, denoted as the hidden state h_t, acts as the memory of the network, holding information about the data that was seen in the previous $t-1$ time steps. Looking at Figure 5.1, we see that the information that arrives at each unit is the value of the previous hidden state, h_{t-1}, and the input at the current state x_t. The update and reset gates are values between 0 and 1 computed using the input, previous state and learnt weight vectors. They allow the hidden state to either ignore the previous state or update it with a new hidden state. At each time step (corresponding to a hidden state) the following computations take place:

1. update gate: $z_t = \sigma(W^{(z_t)} x_t + U^{(z_t)} h_{t-1})$ - both the current input and previous state vector are multiplied with their corresponding weight vectors; a sigmoid function is applied such that the result is between 0 and 1;

2. reset gate: $r_t = \sigma(W^{(r_t)} x_t + U^{(r_t)} h_{t-1})$ - the same operations as before, but with

[1]Picture from: `https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9e`

different learnt weight vectors $W^{(r_t)}$ and $U^{(r_t)}$;

3. current memory content: $h'_t = \tanh(Wx_t + U(r \odot h_{t-1}))$ - if r is close to 0, then the hidden unit will ignore the previous state; a tanh function is applied to squeeze the value between -1 and 1;

4. final memory: $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$ - the reset gate chooses how much to keep from the previous state and how much to rewrite with the current memory content.

Intuitively, the two gates help capture dependencies at different scales. Short-term dependencies tend to be learnt when the reset gates are activated (r close to 1), while long-term dependencies are captured by units where the update gates are more active (Cho et al., 2014).

Other layer designs that we considered included the simplest, original RNN layer design and the more elaborate Long Short-Term Memory (LSTM) layer design (Schmidhuber and Hochreiter, 1997). We use a GRU layer design due to the better performance in our experiments; it is also better suited for the amount of historical data that was available.

5.2.2 Physical models

Physical models describe explicitly the relationship between power production and environment, based on information such as a detailed physical description of the atmosphere (including gases and aerosols), local topography and plant properties. There are three types of physical models, depending on the source of the input data: *numerical weather predictions*, *sky imagery*, and *satellite images*. Numerical weather prediction models use differential equations that describe the dynamics of the atmosphere (Mathiesen and Kleissl, 2011). Sky imagers provide a hemispheric view of the sky, from which clouds, their position and motion are determined (Yang et al., 2014); they are particularly useful in very short term power prediction. Similarly, satellite images are used to model cloud motion (Perez et al., 2010). These models have the advantage that they are deterministic physical models and they do not need long historical data. One of the disadvantages is that they are designed specifically to a particular power plant and location and they need detailed information about the characteristics and parameters of all underlying factors of power production (Ulbricht et al., 2014).

5.2.3 Hybrid models

Hybrid models combine the approaches of statistical and physical models. Statistical models can become hybrid if they use the output of physical model as input for the prediction. Similarly, physical models can become hybrid if they correct systematical errors using statistical tools. The popularity of hybrid models is due to their capability to take advantage of the strong points of different techniques. There is a large body of evidence that they outperform stand-alone models (Zhang, 2003; Ogliari et al., 2017).

Ensemble learning via stacking

Ensemble learning refers to the method of combining multiple learning algorithms to improve the predictive performance of the individual models. In this work we compare an average ensemble with a weighted average ensemble, that combines the models based on their performance on the validation dataset. The approach is called *stacked generalization* or *stacking* Wolpert et al. (2013). First, each type of model is trained on a subset of the data, the training set; secondly, we learn the weights of each model in the ensemble based on the error on a separate subset of the data, the validation set. Finally, the ensemble model combines the individual models with the respective learnt weights to predict the output on a new dataset, the test set. *Grid search* is one typical method to find the weights of the individual models. It considers a subset of all parameter combinations. We discuss the details of this approach in the experiments section.

5.3 Data representation and analysis

In this section we give details about the sources of data used in our models. The input consists of weather forecasts and forecasts of power output in the absence of clouds. We also incorporated persistence data, the location, individual capacity, and the azimuth and tilt angles of the panels. A visual inspection of the power output timeseries shows there are no obvious temporal or spatial patterns. In all the following plots the values for the output and the dates have been removed at the request of the data provider.

5.3.1 Photovoltaic power output

The PV power output data was provided by Tokyo Electric Power Company Holdings (TEPCO) and Hokkaido Electric Power Company (HEPCO) as part of the “PV in HOKKAIDO” contest for Predicting Power Output of Solar Power Plants in Hokkaido, Japan. The output was aggregated from power plants in two regions of the island; the regions have different climatic features, geographical distribution of power plants, total capacities and maximum output, as seen in Figure 5.2. We will refer to the two regions as S_1, for the southwest region, and S_2, for the eastern one. The final aim is to predict the output for the entire island.

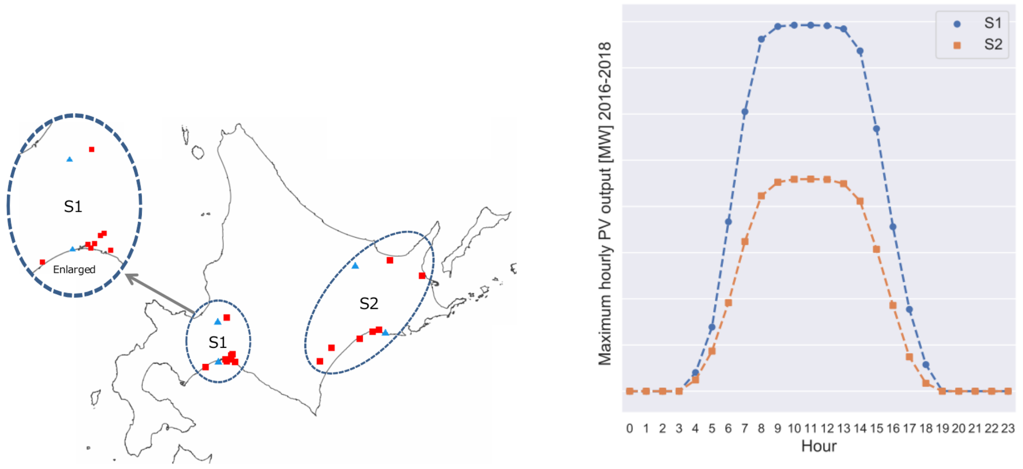


Figure 5.2: The geographical regions (power plants represented by red squares) and the respective maximum power output for each region.

The PV power output is aggregated from 15 solar power plants across the island between January 2016 and December 2018, with measurements taken every 30 minutes. Due to a blackout in September 2018, 2 weeks of data are missing. We investigate the existence of spatial-temporal patterns by analysing the power output visually, and comparing the distributions by region, month, hour and maximum achieved values.

A comparison of the time series data reveals no obvious temporal or spatial regularity, as is clear from Figure 5.3. The top figure shows data for the same week from the 3 years with a variety of weather conditions: clear, mostly sunny, partly cloudy, mostly cloudy and overcast. While the PV power outputs between the two regions are more highly correlated (particularly on clear days, disregarding the difference in amplitude, see Figure 5.3 bottom), there is still a large discordance in the daily patterns, caused in part by the movement of clouds from one side of the island to the other and differences in amounts of fog and snow.

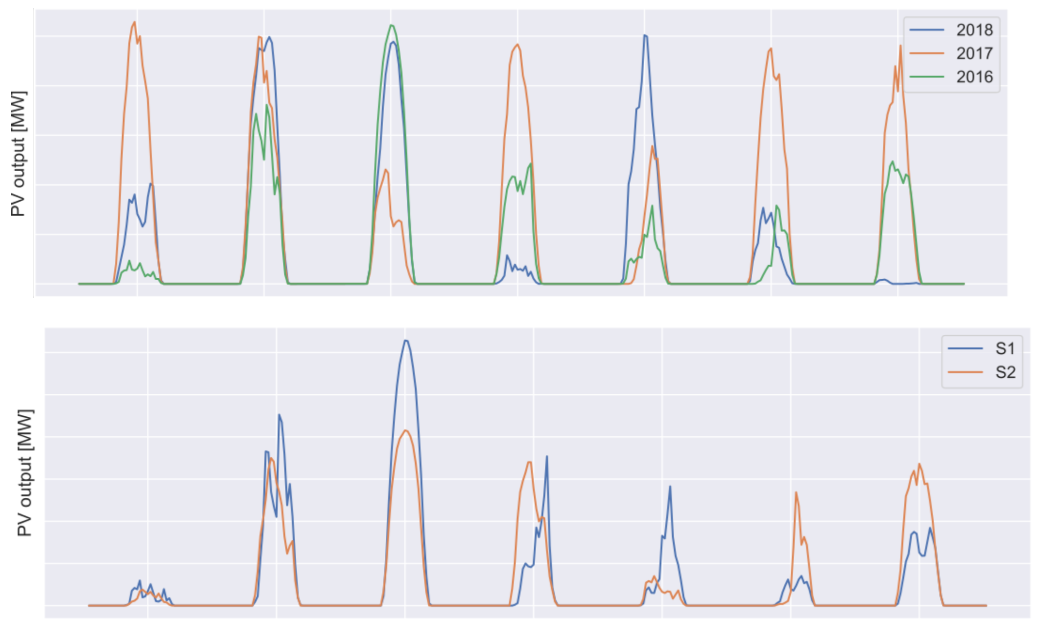


Figure 5.3: Power output timeseries comparison across years (top) and regions (bottom) for a variety of weather conditions for one week.

The average hourly output, as expected, follows a bell shaped curve (Figures 5.4 and 5.5). Across the two regions, besides the difference in the amplitude of the output, we notice a much larger variance in the distribution of hourly output for region S_1 (see Figure 5.5). Correspondingly, an inspection of the regional output timeseries reveals different levels of smoothness (with the output in S_2 being more smooth). This can partly be explained by the different geographical distributions of power plants in the two regions: most of the power plants in S_1 (including the highest capacity one) are concentrated in a small area, while the power plants in S_2 are scattered in a larger area. This is related to the smoothing effect, discussed in Section 2.3: for sufficiently large distances dispersion of power plants leads to a decrease in fluctuations of the aggregated output. We will use this insights to interpret the results of our experiments.

Lastly, looking at the total maximum monthly power output, we notice some seasonal similarities. Figure 5.6 (a) displays the maximum monthly values for the 3 years of data; February, March, April, May and December have close maximum values, while the highest variation between the 3 years is recorded in January. The maximum power output is usually registered between 10am and 1pm (Figure 5.6 (b). However, the

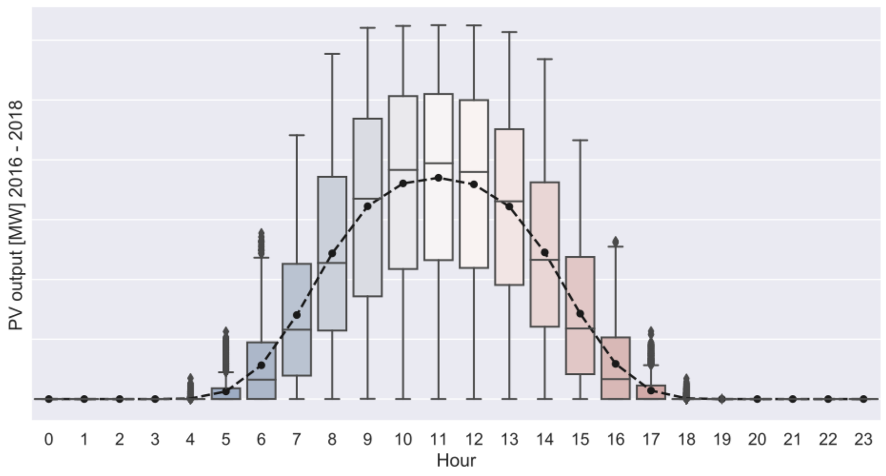


Figure 5.4: Total output between 2016 and 2018. The box plots show the 5-95 percentiles for each hour and the dotted line connects the respective means. The average hourly output perfectly follows a bell curve.

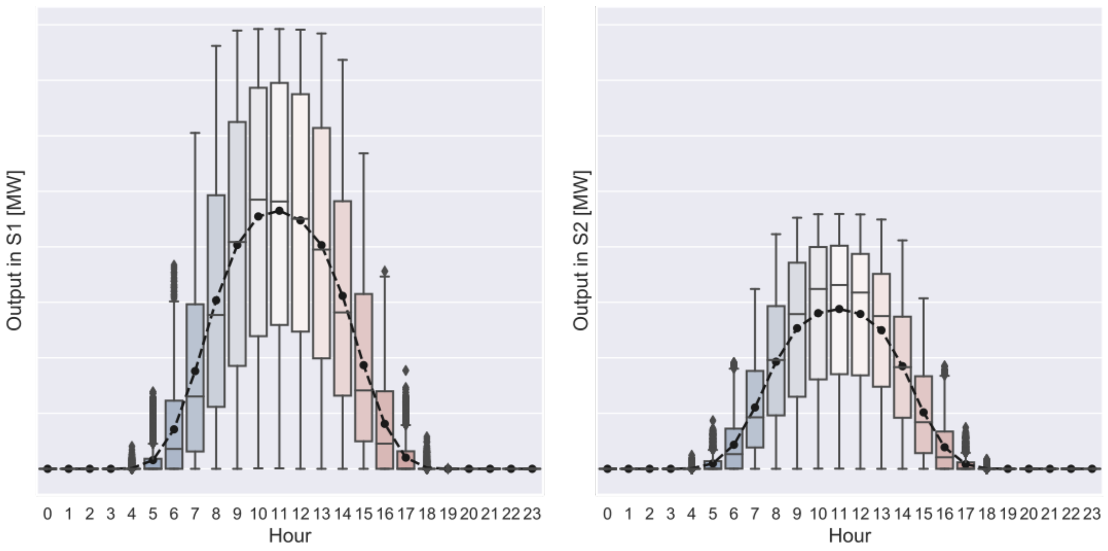
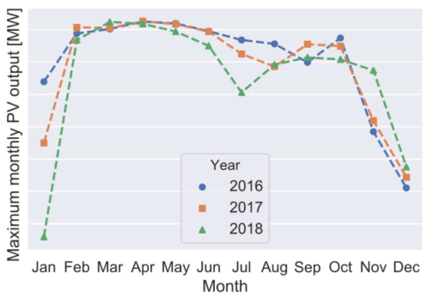
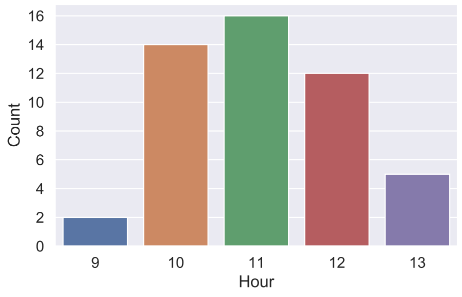


Figure 5.5: The hourly average power output in the two regions.

distributions of the values from 11am, 11:30am, 12pm and 12:30pm (Figure 5.7) do not show the same pattern of seasonal similarity - there is a large variation in the values even between the months with similarly large maximum values. The similarity in maximum power output between some of the months could be useful in designing specialised seasonal forecast methods, with the observation that the maximum is registered at different hours even in similar months. However this is out of the scope of this work.



(a)



(b)

Figure 5.6: The total maximum power recorded every month for the 3 years and the count for the hours when the maximum was recorded. Predicting the maximum each day is challenging.

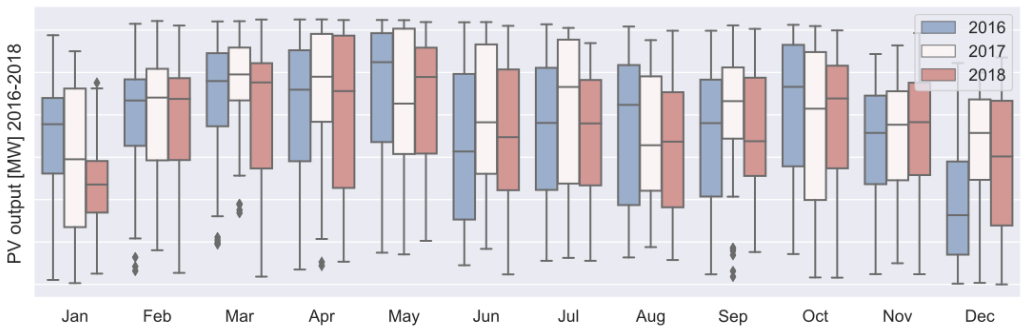


Figure 5.7: The monthly and yearly statistics for the power recorded between 11am and 12pm show high variability for total power produced at a fixed time.

The analysis of temporal and spatial properties of the PV power output shows high variability and no conclusive regional, seasonal, or hourly patterns. In general, changes in solar power output can be large, appear suddenly and often, due to the direct influence of fluctuations in solar irradiance caused by passing clouds (as opposed to wind power generation where the production is more continuous). As we will show through our model, numerical forecasts for temperature, wind direction and cloud cover can predict if and when these fluctuations will appear. For clear days, when such fluctuations are absent, the output is relatively easy to predict using clear sky models. We next describe the numerical weather forecast data and then the clear sky models data.

5.3.2 Numerical weather forecasts

We use weather forecasts given by the meso-scale models (MSM) of the Japan Meteorological Agency [1]. *Meso* refers to the regional scale of the model – which varies between a few kilometres and a few thousand kilometres, as opposed to the *global* scale. MSMs are 3-dimensional models based on *primitive equations*: the momentum equation, thermodynamic equation and equation for conservation of mass. In our case, the MSMs predict meteorological parameters at 5km grid points over the next 39 hours, every 3h, for a rectangular flat area of $4080km \times 3300km$ covering Japan and its surroundings. In our models we use the forecasts made at 6pm to predict the output for the following day; this is a typical requirement in a practical setting. The forecasts have a temporal horizon of 10 hours to 26 hours which corresponds to a prediction interval from 4am to 20pm the next day. [2].

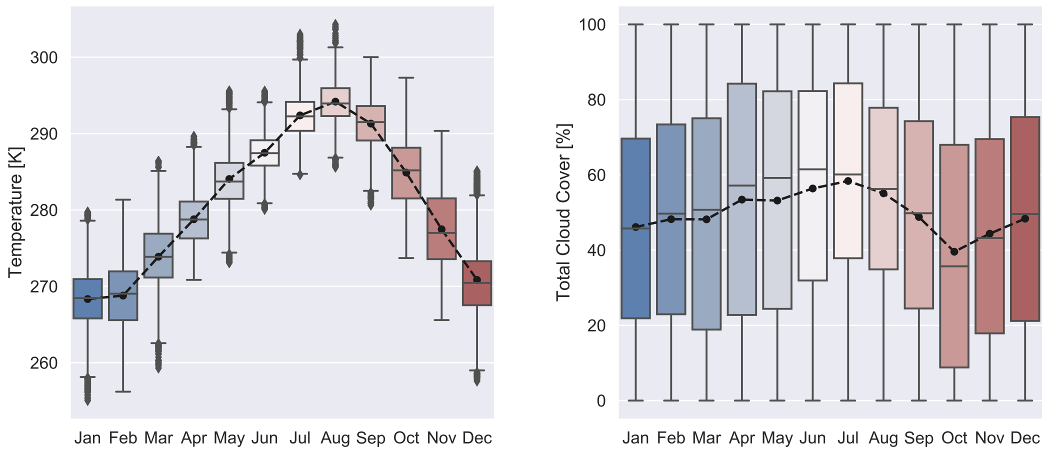
We considered a subset of the weather properties that are forecast for every hour and extracted the values at the closest grid point to each power plant – in total 15 locations. For any given prediction day we used forecasts created at 6pm the previous day. The weather features in our models are:

- cloud cover (%) - low, medium, high and total cloud cover
- pressure (PA) and pressure reduced to mean sea-level pressure (PA)
- relative humidity (%)
- temperature (K)
- total precipitation (kg/m^2)
- u-component and v-component of wind (m/s).

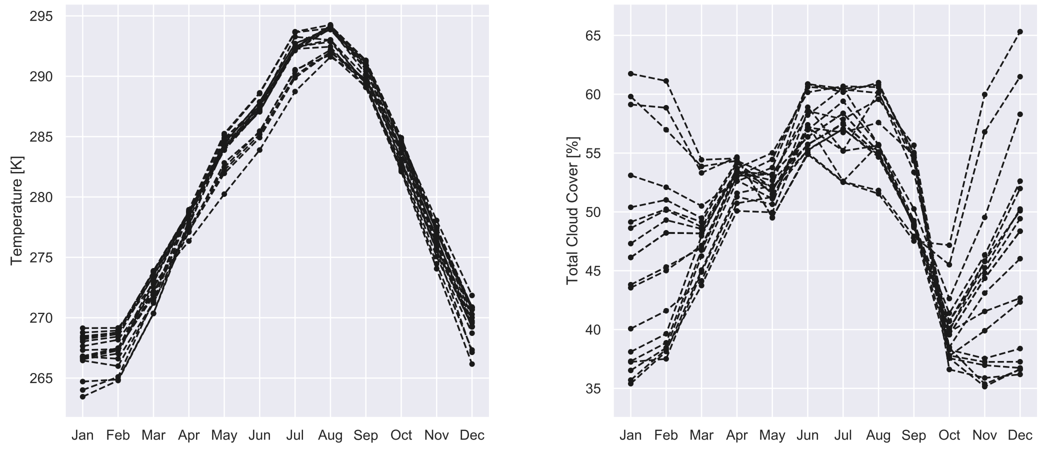
The meteorological forecasts are different in terms of accuracy, due to greater variability both at the temporal and spatial levels. Figure 5.8 compares the monthly values for one location (a) and all 15 locations (b) between temperature and total cloud cover. The total cloud cover is defined as the percentage of the surface obscured by all clouds for the entire atmospheric column at a particular grid point.

[1]Japan Meteorological Agency Numerical Weather Prediction Activities at: `https://www.jma.go.jp/jma/en/Activities/nwp.html`, accessed: 13 March 2019

[2]The latest MSM forecasts can be downloaded from `http://apps.diasjp.net/gpv/cgi-bin/uncgi.cgi/GPVdate2b.sh`



(a) Comparison of values for temperature and total cloud cover at one location.



(b) Comparison of mean monthly values for temperature and total cloud cover at all 15 location.

Figure 5.8: Spatio-temporal monthly variation of temperature and total cloud cover

5.3.3 Clear sky model predictions

Clear sky models predict the *global horizontal irradiance* (GHI) in the absence of visible clouds. The GHI is the sum of direct and diffused radiation reaching Earth's surface and is highly correlated with the amount of solar power produced by photovoltaic panels at that location. The models incorporate information about extraterrestrial solar irradiance at normal incidence and the atmospheric absorption and scattering of solar radiation (known as the *Linke turbidity index*) at a specific location, taking into account the altitude. In the absence of weather observations or numerical weather forecasts, the models produce similar forecasts for each year.

We used the open-source Python module `PVLIB` (Holmgren et al., 2018), more specifically the implementation of the Ineichen and Perez clear sky model (Reno et al., 2012), to forecast the clear sky GHI quantity at each power plant at 30 minute intervals for the 3 years of data. We then converted the timeseries into predictions of power output assuming a default solar panel size, type and AC/DC converter. Weighted summation of predictions for each power plant lead to regional predictions for the power output in the absence of all weather effects. The weights were chosen according to the AC nominal capacity of each power plant. The predictions were then scaled to match the 99^{th} quantile of the power output.

Figure 5.9 shows the forecast for one week compared with the actual output in the corresponding region. The predictions are fairly accurate for days with no clouds; the mismatch between the maximum values is due to the scaling factor being different across seasons - the 99^{th} quantile for the power output is most likely recorded during the summer months, and will be larger than the maximum recorded in the winter (see Figure 5.7). Adding temperature and wind forecasts to the clear sky model could result in more accurate forecasts for clear days. In our experiments we noticed that clear sky predictions had the highest impact when compared to any of the weather factors. Notice that clear sky predictions also delimit when days start and end.

5.3.4 Input data representation

Denote by d and t a specific day and time and by x and y the latitude and longitude of the location of a power plant. Let O be the power output, CS the clear sky model and NW the numerical weather forecast. We define:

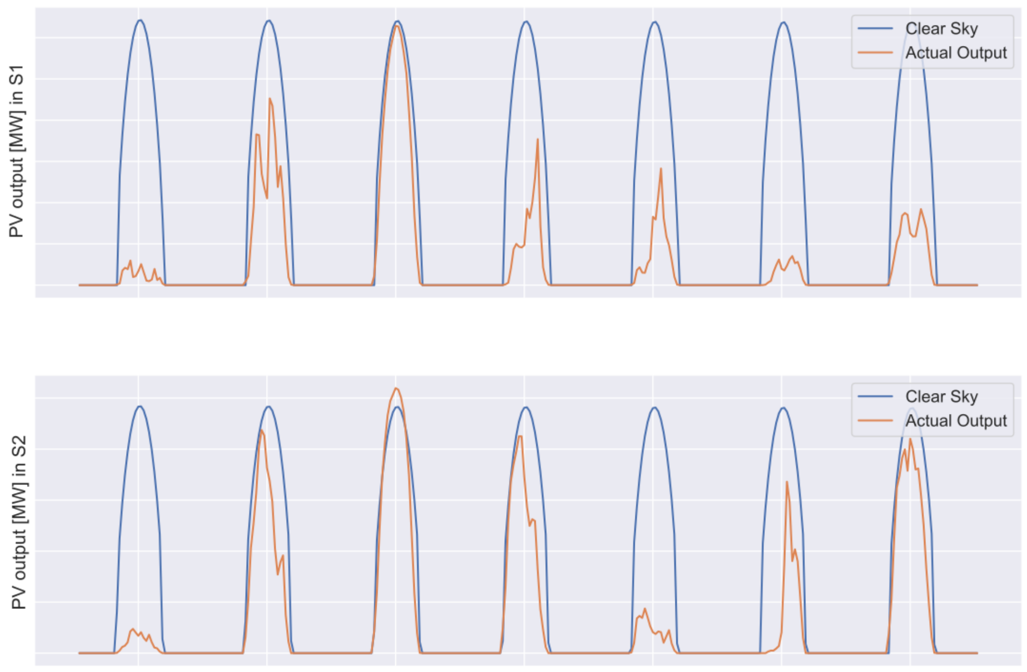


Figure 5.9: Clear sky power output predictions for one week in the two regions.

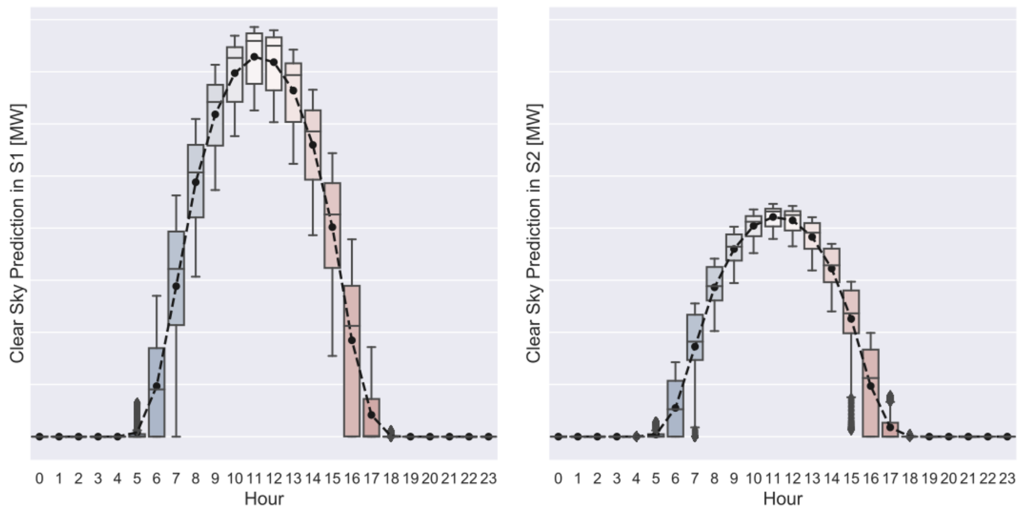


Figure 5.10: Comparison of the hourly distributions of the clear sky predictions in the two regions.

- persistence model predictions (aggregated for a region)

$$X_P(d,t) = O(d-1,t)$$

- clear sky model predictions (including the properties of power plant at location

$(x,y))$

$$X_{CS}(x,y,d,t) = CS(d,t,p(x,y))$$

- numerical weather forecasts (where $f1,\dots,f_6$ are the weather variables)

$$X_{NW}(x,y,d,t) = \begin{pmatrix} NW(x,y,d,t,f_1) \\ \vdots \\ NW(x,y,d,t,f_6) \end{pmatrix}$$

Say the prediction is made for the next day between time t_1 and t_T. Then we have the following vector notations:

$$\mathbf{X}_P(d) = (X_P(d,t_1)\dots X_P(d,t_T))$$

$$\mathbf{X}_{CS}(d) = (X_{CS}(d,t_1)\dots X_{CS}(d,t_T))$$

$$\mathbf{X}_{NW}(x,y,d) = X_{NW}(x,y,d,t_1)\dots X_{NW}(x,y,d,t_T).$$

Denote by M the PV power prediction model. The next day prediction of the PV power output for region $R \in \{T, S_1, S_2\}$ is:

$$P_R(d) = M \begin{pmatrix} \mathbf{X}_P(d) \\ \mathbf{X}_{CS}(d) \\ \mathbf{X}_{NW}(x_1,y_1,d) \\ \vdots \\ \mathbf{X}_{NW}(x_N,y_N,d) \end{pmatrix}$$

for all $(x_i,y_i) \in R$. We denote the matrix of all input for region R and day d simply by $I_{R,d}$, therefore $P_R(d) = M(I_{R,d})$.

5.4 PV power output prediction model

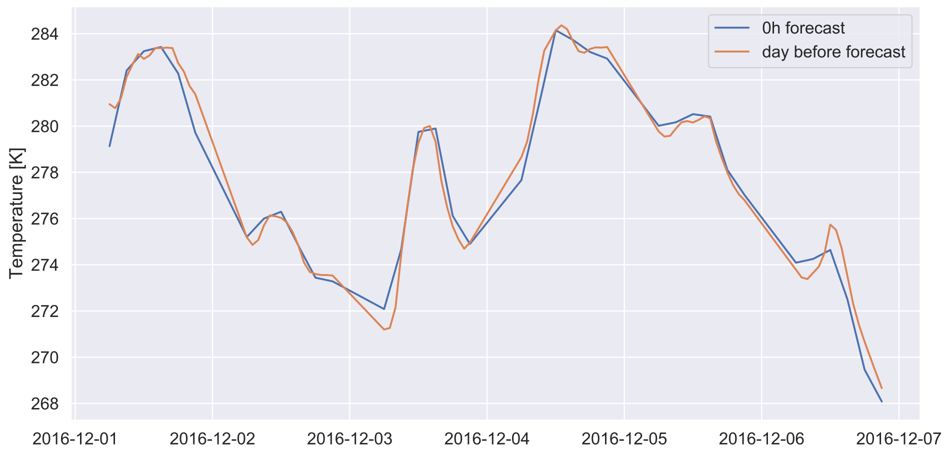
Our proposed model is an ensemble of three types of models: *multivariable linear regression (MLR)*, *support vector machines (SVM)* and *gated recurrent units network (GRU)*. The concept behind ensemble learning is to train a suite of base learners and combine their individual predictions. The final values are the average of the predictions from the individual models. We observed that each of the models performed well, but combining the results gives the highest accuracy, which suggests that they exploit different patterns in the data. We give details about the way the ensemble is constructed.

5.4.1 Feature selection

As described in Section 5.3, the complete feature set is composed of the persistence model predictions, the clear sky model predictions, and the weather forecasts at the location of each power plant. For some of our experiments we eliminated a set of the weather forecasts, keeping only the most relevant ones. The selection was made by separately training a linear regression, a gradient boosting model, an SVM, and a GRU network on the full set of features. For the first three models the features were chosen according to their weights; for the neural network we did a *sequential backward selection* that sequentially removes the features that give the smallest decrease in accuracy, these being the least relevant ones. There was a significant overlap between the types of features that the models selected.

The clear sky model predictions were classified as the most important by all models; this is to be expected, because on clear days the predictions match the power output almost exactly. Amongst the weather forecasts, the variables that were most commonly rated as important were wind and temperature. Perhaps surprisingly, very few of the cloud features were ranked as important. One reason is the difficulty to predict cloud coverage accurately, especially the day before and at hourly intervals. This is partly because cloud formation is the result of the complex interaction of a large number of parameters. Moreover, forecasting models are often based on correctly identifying cloud coverage at multiple altitudes from satellite imagery, which can be difficult especially in the presence of dense high clouds. Indeed, comparing the day ahead forecasts for temperature and cloud cover to the 0h forecast, we notice large discrepancies in the accuracy (see Figure 5.11). 0h forecasts are simulation results for the following 3 hours based on current observations at certain locations, for each point on the MSM grid.

To understand the extent to which the cloud forecasts are less accurate than the other weather variables, we computed the errors between the forecasts and the 0h forecasts for each variable. We rescaled the values such that they are between 0 and 1 and then computed the mean squared error between 0h forecast and day ahead forecasts. This



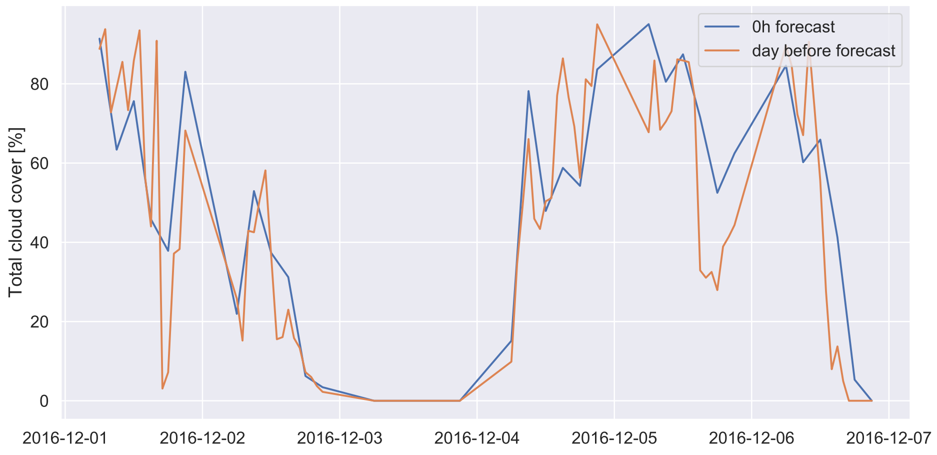


Figure 5.11: Comparison between 0h forecast and day before forecast for temperature and total cloud cover.

comparison suffers from the fact that the 0h forecast is not an actual observation, but the result of a simulation. However it helps identify the general trend in the forecast errors. In Figure 5.12 we can see that all variables related to cloud cover have a much larger error than all other variables. Cloud cover variables are therefore most difficult to predict, yet the ones that can improve prediction the most.

5.4.2 Grid search weighted average ensemble

The weight of each of the models in the ensemble corresponds to the expected performance of that model. To find the coefficients of the individual models we use grid

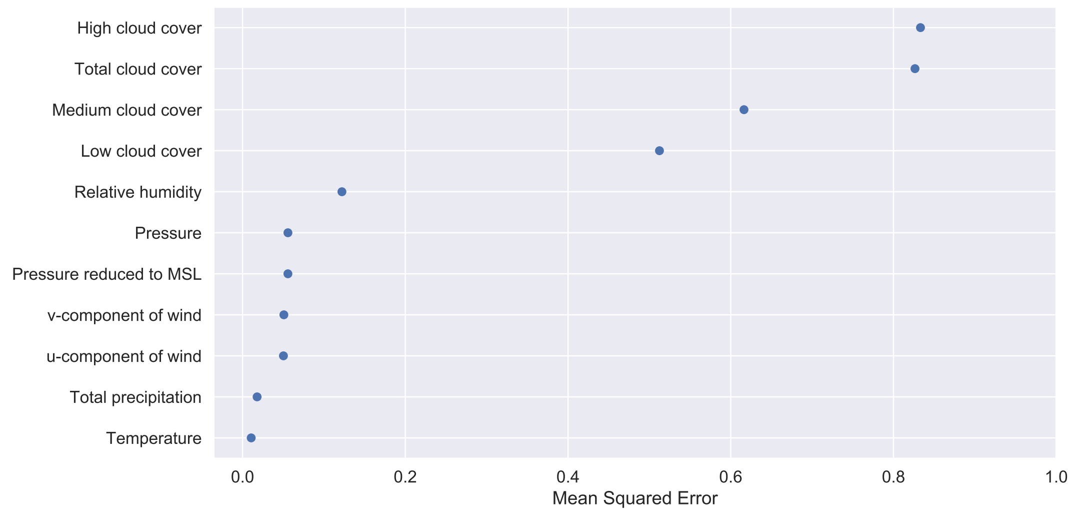


Figure 5.12: The mean squared error for each normalised weather factor. Cloud cover forecasts have substantially larger errors.

search, described in Section 5.2. In general, the weights can be any values strictly larger than 0 and lower than 1 such that they sum up to 1. We choose values between 0.1 and 1 with a step size of 0.05 and test all combinations of weights on the validation dataset.

5.5 Experimental setup and results

We first present the suite of experimental setups based on the prediction region and the set of features, we continue with the error metrics, the results and an analysis. Firstly, we compare the performance of the ensemble with the individual models. The ensemble achieves the highest accuracy, higher than the baseline models and on par with results from the literature from nearby regions. More importantly, our experiments show that the spatial distribution of the plants has an impact on the forecast accuracy, but this depends largely on the type of model. MLR is mostly unaffected by the choice of features, while SVM and GRU have contrasting behaviours. First let us describe the experimental setups we considered.

We denote the output for the three regions of prediction by S_1, S_2, T, where T refers to the total (S_1 and S_2 together) and run experiments with the following settings:

- one model for predicting T, using the full set of features: M_T

- one model for predicting T, using the set of highest ranked features: $M_{T,F}$
- two models predicting $S1$ and $S2$, using the full set of features for both: M_R; the outputs are then summed up to compare with the actual output
- two models predicting $S1$ and $S2$, using the set of highest ranked features for both: $M_{R,F}$
- two models predicting $S1$ and $S2$, using only the features from the respective region: $M_{R,S}$

Output	Feature set	Name
T	full set	M_T
T	highest ranked	$M_{T,F}$
S_1, S_2	full set	M_S
S_1, S_2	highest ranked	$M_{S,F}$
S_1, S_2	regional	$M_{S,R}$

Table 5.1: Model names

Implementation details

We use the `sklearn` Python library to implement the MLR models and SVMs with a radial basis function kernel. The GRU model is implemented in `Pytorch`. The number of layers in the neural network and the size of each layer were selected using dichotomous search. The network is composed of 2 layers with ReLU activation, Adam optimiser, 0.01 learning rate, 20% dropout, batch size of 64 and the training is for 100 epochs.

5.5.1 Dataset split

We trained the model on 2 years of data (2016 and 2017) and report the results on one year of data (2018). In this way the results are not influenced by seasonal weather patterns. Out of the 2 years of training data, we used 10% for validation; to simulate the task of predicting whole days, the data is split into days, rather than individual measurements, resulting in a set of approximately 30 days for validation.

5.5.2 Evaluation metrics

To evaluate the accuracy of the predictions we compute three error metrics. We denote by C_t the total nominal capacity of the power plants and by C_m the average obtained power for the test year. P_i is the prediction at timestamp i and O_i is the actual output. The sum is computed on all timestamps in the test year. The errors we measured are:

- root mean squared error normalised by the total capacity:

$$NRMSE_1 = 100 \times \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - O_i)^2}}{C_t}$$

- root mean squared error normalised by the average obtained power:

$$NRMSE_2 = 100 \times \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - O_i)^2}}{C_m}$$

- root mean squared error:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - O_i)^2}.$$

The *skill score* is a measure of performance with regards to a baseline, most often the persistence model. It is computed as $1 - \frac{\sum_{i=1}^{N}(P_i - O_i)^2}{\sum_{i=1}^{N}(P_{i,b} - O_i)^2}$, where $P_{i,b}$ is the prediction of the baseline.

We optimise the ensemble with respect to the root mean squared error and present the results in the next section.

5.5.3 Performance of the ensemble model

The results from Table 5.4 show that the highest accuracy was achieved by the ensemble model that combines predictions for S_1 and S_2 based on the set of most important features, while the lowest accuracy was given by the model that predicts the output for the two regions only based on the local features. All models perform better than the baselines.

Our most accurate model has a skill score of 0.77 for the one year of data, which is similar to the scores reported by Fonseca et al. (Fonseca Junior et al., 2014, 2015) for regional PV power forecasts in several regions in Japan. However, as pointed out in

Model	$NRMSE_1$	$NRMSE_2$	$RMSE$
M_T	52.36	6.69	27.25
$M_{T,F}$	51.70	6.60	26.91
M_S	51.91	6.63	27.02
$M_{S,F}$	**50.95**	**6.51**	**26.52**
$M_{S,R}$	52.42	6.69	27.28
P	104.11	13.30	54.19
CS	140.26	17.92	73.00

Table 5.2: Error metrics for the ensemble models, the persistence model (*P*) and the clear sky model (*CS*)

their work, the same model can give slightly different skill scores depending on the particularities of the regions.

To better understand the differences in accuracy we look at the individual performance of the models that make up the ensemble, shown in Table 5.3. The main findings are summarised below:

- The ensemble gives the highest overall performance; this is achieved in the framework where we consider the highest ranked features and the prediction is done separately for the two regions ($M_{S,F}$).
- The ensemble has the highest accuracy in all the different settings ($M_{T,F}$, M_S, $M_{S,F}$, $M_{S,R}$), except for the case where we consider all features (M_T); the accuracy is slightly lower than GRU because of the poor performance of SVM in this setting; this can be explained by the higher amount of correlated variables in the full set of features when compared to the top features.
- The low performance in the framework where only the features specific to one area are considered ($M_{S,R}$) suggests that the models utilise information from the neighbouring area.

We denote each model by the type and framework, for example $M_{GRU,T}$ is the GRU neural network model trained to predict the total output on the full set of features. Looking at Table 5.3, we notice that the framework that gave the highest accuracy for the ensemble is the one where the deep model has a low performance. This is compensated by the performance of the SVM model in this framework, where we

Model	Error	M_T	$M_{T,F}$	M_S	$M_{S,F}$	$M_{S,R}$
Ensemble	$NRMSE_1$	52.36	51.70	51.91	**50.95**	52.42
	$NRMSE_2$	6.69	6.60	6.63	**6.51**	6.69
	$RMSE$	27.25	26.91	27.02	**26.52**	27.28
GRU	$NRMSE_1$	**52.15**	52.39	52.52	52.46	53.81
	$NRMSE_2$	**6.66**	6.69	6.71	6.70	6.87
	$RMSE$	**27.14**	27.27	27.33	27.30	28.01
MLR	$NMRSE_1$	59.5	60.61	**59.48**	60.60	61.15
	$NRMSE_2$	7.60	7.74	**7.60**	7.74	7.814
	$RMSE$	30.98	31.54	**30.95**	31.54	31.82
SVM	$NMRSE_1$	59.11	55.82	57.35	**54.88**	58.20
	$NRMSE_2$	7.55	7.13	7.32	**7.01**	7.43
	$RMSE$	30.77	29.05	29.85	**28.56**	30.29

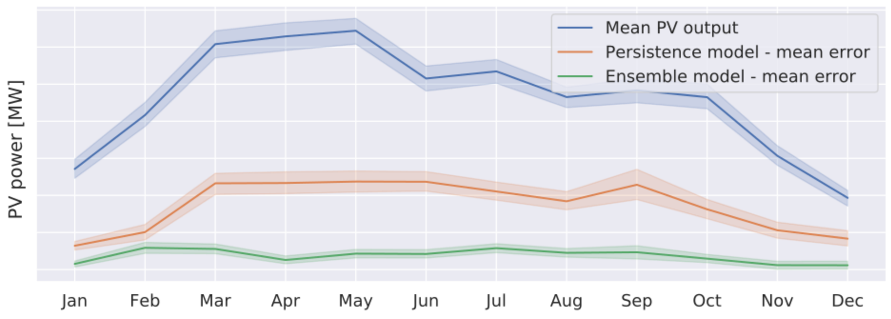
Table 5.3: The performance of the ensemble, GRU, MLR, and SVM models for all settings. Blue and red cells mark lowest and highest errors, respectively, for each model across the different frameworks.

considered only the highest ranking features and predicted the output for S_1 and S_2. The difference in accuracy between $M_{SVM,T}$ and $M_{SVM,S,F}$ comes from the amount of correlation in the feature vectors. The complete set of features contains similar weather forecasts from locations of power plants that are close together, especially from region S_1. The radial basis function is a popular choice of kernel for SVMs, but, given that it takes into account only the distances between points, it tends to be impacted by repeated/correlated variables. This is confirmed by comparing $M_{SVM,T}$ with $M_{SVM,F}$ and $M_{SVM,S,R}$ with $M_{SVM,S,F}$. For the latter case, even if the set of features is restricted to the ones corresponding to the area of prediction, it is precisely this set that contains the most correlated feature vectors. On the other hand, neural networks can possibly use correlation as a property of the data. While generally having a large set of features can be detrimental because of overfitting, having a large set of correlated features does not necessarily have the same impact; unnecessary features will be given a small weight. We notice that the accuracy of $M_{GRU,T}$ is similar to $M_{GRU,T,F}$, which confirms that the set of highest ranked features was well chosen.

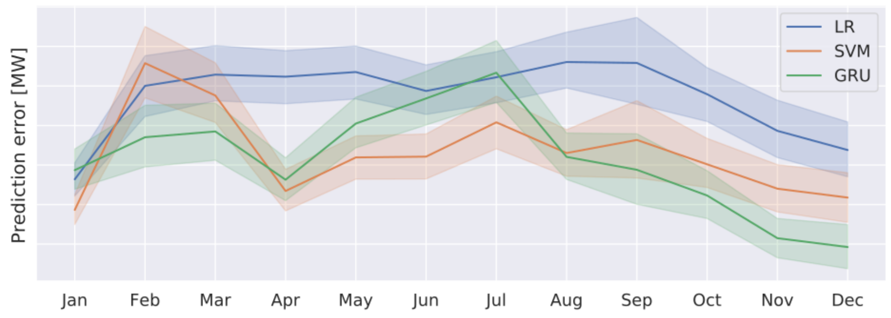
5.5.4 Seasonal and monthly patterns

Throughout the test year the mean prediction error for the ensemble model remains fairly stable, as can be seen in Figure 5.13. When considering each model individually, we notice a peak in the prediction error in February, the month with the highest snowfall, and another peak in July, probably due to fog specific to the summer months, especially in the eastern area of the island (corresponding to region S_2). The forecasting model would benefit from including data on snowfall and fog.

An interesting observation from Figure 5.13 (b) is that the SVM model performs better than GRU between April and August, despite GRU being the model with a temporal component.



(a) Comparison between the mean input and the mean error for the persistence model and the ensemble in the best performing framework ($M_{S,F}$).



(b) Comparison of the mean error between the individual models.

Figure 5.13: The mean prediction errors and their 5-95 percentiles by month

Looking in more detail at the daily curves values, we notice that GRU produces curves that generally match the output in amplitude, but are smoother, while SVM and MLR are better at identifying fluctuations, but not necessarily their intensity. This suggests that a different scheme for combining the output (such as weighted average or bagging) from the several models could be beneficial.

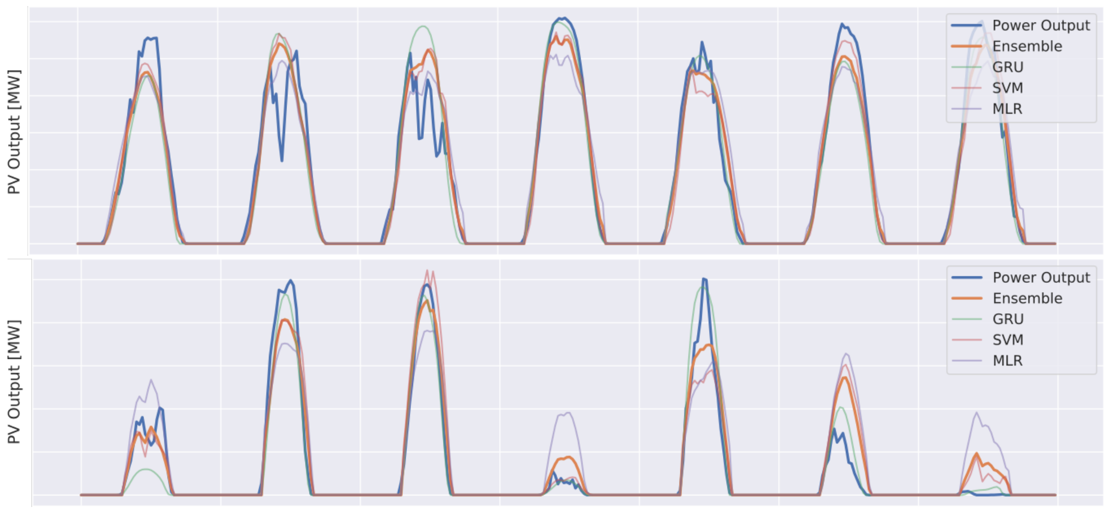


Figure 5.14: PV power prediction for one week in June and one week in December.

5.5.5 Comparison with the weighted ensemble model

The proposed model can be further improved by choosing the weights of the individual models based on their performance on the training and validation sets. In Table 5.4 we show the comparison with the best performing ensemble, $M_{S,F}$, which predicts the output for the separate regions using top features from both areas. We denote the new weighted model as $M_{S,F,W}$. It is composed of the 6 same models as $M_{S,F}$ (GRU, MLR, SVM for each of the 2 regions), but weighted according to their performance.

Model	$NRMSE_1$	$NRMSE_2$	$RMSE$
$M_{S,F}$	50.95	6.51	26.52
$M_{S,F,W}$	**49.41**	**6.31**	**25.7**

Table 5.4: Comparison between the ensemble model and the weighted ensemble model.

To find the weights we perform grid search on all combinations of values between 0.1 and 1 with a step size of 0.05. Starting with all values set to 0.1, we keep each set that gives an error lower than the current minimum. We record the corresponding error on the validation set, which gives the curve shown in Figure 5.15. We use the *elbow method* (Thorndike, 1953) to determine the set of weights that could perform well on the test set. Choosing the global minimum for the validation set error has the risk of overfitting the model. The elbow method is a heuristic that finds the cut-off point corresponding to a set of parameters for which the error starts decreasing at a lower rate. We choose this point to be 125, which gives the corresponding weights, for regions S_1, S_2 respectively :

- GRU - 0.2, 0.25;
- SVM - 0.25, 0.1;
- MLR - 0.1, 0.1.

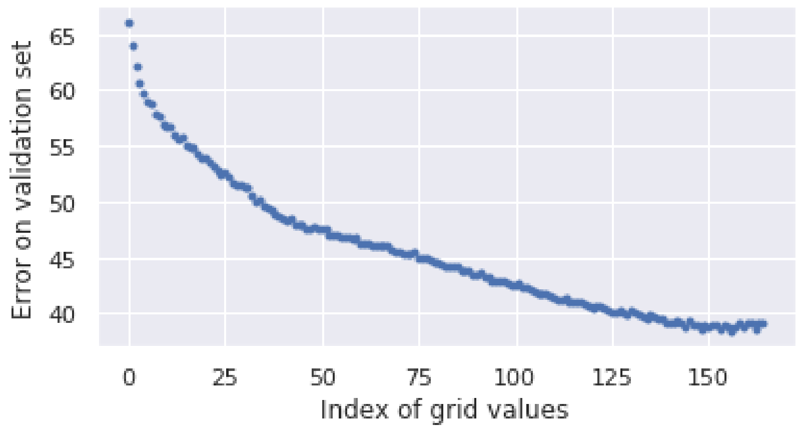


Figure 5.15: $NRMSE_1$ error on the validation set. We apply the elbow method to identify the set of weights that give high accuracy on the rest set without overfitting. At index 125 the improvement of accuracy starts decreasing.

The different weights for the different areas suggest that the models identify different patterns in the data and justifies using the different types of models. A next area of investigation is whether the accuracy of the ensemble can be improved by considering different weights based on the weather profile of the next day forecast.

Chapter 6

Conclusions and future work

Our work consists of scalable algorithms for solving fundamental problems in learning from location data. We show that by grouping data points into independent, disjoint units based on properties related to their spatial proximity and using them as input diminishes the effects of noise, overfitting, and high dimensionality, leading to improvements in computational running time and accuracy. Firstly, we presented a novel method to compress human mobility trajectories that is useful in making distance estimation and similarity queries computationally efficient. Secondly, we developed algorithms for designing transit networks based on a dataset of origin destination trajectories under given constraints. Finally, we show the effect of the spatial component on the accuracy of an ensemble and individual machine learning models for prediction of solar power output. In this section we give more details about each topic and give directions for future work.

Sketches for trajectories

In Chapter 3 we presented a randomised sketching scheme to compactly represent trajectories. The sketches allow efficient algorithms for a suite of central trajectory queries and tasks: near neighbour search, distance estimation, clustering, etc. We established the effectiveness of the sketching technique in both theory and practice of trajectory processing. We formally describe the locality sensitive hashing schemes for Hausdorff and Fréchet distances and prove the probabilistic guarantees. The experiments show strong correlation with Hausdorff and correlation with Fréchet and DTW.

In practice, highly compressed sketches achieve a pruning ratio of 80% accuracy of 80% for nearest neighbours queries.

The type of trajectories we are considering are common in robotics, biological systems and many other domains. One direction of future work is to adapt the basic randomised sketching scheme in these areas. For example, in movement ecology, clustering of trajectories is used to analyse processes such as spatial displacement in foraging behaviour. Cleasby et al. (2019) show that Fréchet distance and dynamic time warping are particularly useful when clustering trajectories with the purpose of observing changes in movement. They also observe that the volume of animal movement data is growing rapidly, which makes efficient computation techniques increasingly useful in this area.

A different direction of work is to consider the temporal dimension; in this case the trajectories are 3 dimensional objects and the disks are replaced by spheres. Clustering spatio-temporal trajectories is useful for example in identifying groups of commuters who take the a similar route in the same time frame.

Transit network design

Chapter 4 presents a heuristic algorithm for the transit network design problem. We define it in the context of building bike lanes, but it can be built upon for other modes of transportation such as bus routes or metro lines. Given a road network, a set of locations for the origins and destinations of bike trips in a city, and constraints on the construction cost, distance travelled to a path and stretch on the new path, the *bike lane problem* is to find a series of paths that would maximise the number of trips that would benefit from using the system. We show that the problem is NP-hard and propose a series of solutions, amongst which a fast heuristic algorithm. Compared with the existent cycling system in London, the proposed algorithm covers at least 20% more trips with the simplest heuristic. We expect a greater increase with the more complex heuristics. The algorithm can also be used to generate new segments for the extension of an existent network.

A point of improvement is to merge overlapping rectangles, which would result in fewer streets selected in the same area. Another point is that the data we used comes from cycle hires - while it is representative for a large number of trips, is restricted to certain locations (the hire stations).

We note that this is a purely mathematical solution to the bike lane problem - in practice there are many other constraints that we have not yet considered. For example, not all segments in the road network are candidates for bike lanes; this is easy to circumvent by only keeping candidate edges in the initial network.

A direction for future work is to consider paths that follow a straight line, as opposed to shortest paths in the road network which can have many turns, since there is evidence that cyclists prefer fewer turns (Hood et al., 2011). To this end the network matching part of the framework that finds shortest paths in the road network for each tube median could incorporate a fewest-turns-and-shortest path finding algorithm, such as the one proposed by (Zhou et al., 2014).

We mentioned that this work can be extended for other modes of transportation, such as buses or trains. In this case there appear several other factors: the transit network should be split into a fixed number of routes, the design should take into account placement of stations and the temporal dimension of the trajectories.

Regional factors in solar power prediction

PV power forecasting is a challenging problem due to the complex factors involved in power production and their variable nature. Machine learning models have been successful in achieving high accuracy for next day prediction based on numerical weather forecasts. In this work we propose a new ensemble framework with three types of models: multivariable linear regression, support vector machine and gated recurrent units network. The ensemble gives a better accuracy than the stand-alone models in terms of the root mean square error. By analysing a suite of model frameworks we observe that for the ensemble building models for regions with homogeneous weather conditions gives a higher accuracy than having one model for the total output. Moreover, predictions for an area benefit from weather forecasts from neighbouring areas. Weighting the models differently based on the performance on training and validation sets further improves the accuracy.

In terms of improving the current model, in future experiments we will consider including snow and fog forecasts. The weighting mechanism could include information about weather forecasts, by first clustering the next day predictions and then choosing the weights based on the type of weather.

In the case of solar power plants there are clear spatio-temporal dependencies between different locations. One method that can utilise the spatial correlation of the input data from different sites is predicting the output with graph neural networks (Gori et al., 2005). Instead of sets of forecasts for each weather variable for each day, the input would consist of graphs, where for every site there is a corresponding node that contains the forecast data; an edge between nodes represents how the forecasts are correlated. Wu et al. (2020) propose a graph neural network framework that learns the dependencies between nodes. Experiments on a solar power prediction task in a single step setting show improvements over previous methods. In future work we could investigate whether a similar method gives better performance for next day predictions.

Bibliography

Abdel-Nasser, M. and Mahmoud, K. (2019). Accurate photovoltaic power forecasting models using deep LSTM-RNN. *Neural Computing and Applications*, 31(7):2727–2740.

Abuella, M. and Chowdhury, B. (2015). Solar power forecasting using artificial neural networks. In *2015 North American Power Symposium (NAPS)*, pages 1–5.

Acharyya, A., Nandy, S. C., Pandit, S., and Roy, S. (2019). Covering segments with unit squares. *Computational Geometry*, 79:1 – 13.

Ackermann, T. and Sder, L. (2002). An overview of wind energy-status 2002. *Renewable and Sustainable Energy Reviews*, 6(1):67 – 127.

Agarwal, P. K. and Pan, J. (2014). Near-linear algorithms for geometric hitting sets and set covers. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG14, pages 271–279, New York, NY, USA. Association for Computing Machinery.

Aggarwal, A., Hansen, M., and Leighton, T. (1990). Solving query-retrieval problems by compacting Voronoi diagrams. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 331–340. ACM.

Alman, J. and Williams, R. (2015). Probabilistic polynomials and Hamming nearest neighbors. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 136–150. IEEE.

Alt, H., Behrends, B., and Blömer, J. (1995). Approximate matching of polygonal shapes. *Annals of Mathematics and Artificial Intelligence*, 13(3-4):251–265.

Alt, H. and Godau, M. (1995). Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91.

Andoni, A. and Razenshteyn, I. (2015). Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801. ACM.

Andonov, R., Poirriez, V., and Rajopadhye, S. (2000). Unbounded knapsack problem: Dynamic programming revisited. *European Journal of Operational Research*, 123(2):394–407.

Arya, S., Mount, D. M., Vigneron, A., and Xia, J. (2008). Space-time tradeoffs for proximity searching in doubling spaces. In *European Symposium on Algorithms*, pages 112–123. Springer.

Astefanoaei, M., Cesaretti, P., Katsikouli, P., Goswami, M., and Sarkar, R. (2018). Multi-resolution sketches and locality sensitive hashing for fast trajectory processing. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL 18, pages 279–288, New York, NY, USA. Association for Computing Machinery.

Atallah, M. J. (1983). A linear time algorithm for the Hausdorff distance between convex polygons.

Backurs, A. and Sidiropoulos, A. (2016). Constant-distortion embeddings of Hausdorff metrics into constant-dimensional lp spaces. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 60. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Bao, J., He, T., Ruan, S., Li, Y., and Zheng, Y. (2017). Planning bike lanes based on sharing-bikes' trajectories. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1377–1386.

Basappa, M. (2018). Line segment disk cover. In Panda, B. and Goswami, P. P., editors, *Algorithms and Discrete Applied Mathematics*, pages 81–92, Cham. Springer International Publishing.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Boeing, G. (2017). Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers Environment and Urban Systems*, 65:126–139.

Bracciale, L., Bonola, M., Loreti, P., Bianchi, G., Amici, R., and Rabuffi, A. (2014). CRAWDAD dataset roma/taxi (v. 2014-07-17). Downloaded from `http://crawdad.org/roma/taxi/20140717`.

Bringmann, K. (2014). Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 661–670. IEEE.

Bus, N., Garg, S., Mustafa, N. H., and Ray, S. (2017). Limits of local search: Quality and efficiency. *Discrete & Computational Geometry*, 57(3):607–624.

Bus, N., Mustafa, N. H., and Ray, S. (2018). Practical and efficient algorithms for the geometric hitting set problem. *Discrete Applied Mathematics*, 240:25 – 32. Linear Optimization.

Ceder, A. (2001). Operational objective functions in designing public transport routes. *Journal of advanced transportation*, 35(2):125–144.

Ceder, A. and Israeli, Y. (1998). User and operator perspectives in transit network design. *Transportation Research Record*, 1623(1):3–7.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM.

Chen, Y., Chen, G., Chen, K., and Ooi, B. C. (2009). Efficient processing of warping time series join of motion capture data. In *2009 IEEE 25th International Conference on Data Engineering*, pages 1048–1059. IEEE.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Cleasby, I. R., Wakefield, E. D., Morrissey, B. J., Bodey, T. W., Votier, S. C., Bearhop, S., and Hamer, K. C. (2019). Using time-series similarity measures to compare animal movement trajectories in ecology. *Behavioral Ecology and Sociobiology*, 73(11):151.

Daganzo, C. F. (2010). Structure of competitive transit networks. *Transportation Research Part B: Methodological*, 44(4):434–446.

Das, G., Fraser, R., Lpez-Ortiz, A., and Nickerson, B. (2011). On the discrete unit disk cover problem. volume 22, pages 146–157.

De Mulder, W., Bethard, S., and Moens, M.-F. (2015). A survey on the application of recurrent neural networks to statistical language modeling. *Comput. Speech Lang.*, 30(1):61–98.

de Vries, G. K. D. et al. (2012). *Kernel methods for vessel trajectories*. SIKS.

DeMaio, P. (2009). Bike-sharing: History, impacts, models of provision, and future. *Journal of Public Transportation*, 12.

Deo, R. C. and Şahin, M. (2017). Forecasting long-term global solar radiation with an ANN algorithm coupled with satellite-derived (MODIS) land surface temperature (LST) for regional locations in Queensland. *Renewable and Sustainable Energy Reviews*, 72:828–848.

Dey, R. and Chakraborty, S. (2015). Convex-hull & DBSCAN clustering to predict future weather. In *2015 International Conference and Workshop on Computing and Communication (IEMCON)*, pages 1–8. IEEE.

Diagne, M., David, M., Lauret, P., Boland, J., and Schmutz, N. (2013). Review of solar irradiance forecasting methods and a proposition for small-scale insular grids. *Renewable and Sustainable Energy Reviews*, 27:65 – 76.

Douglas, D. and Peucker, T. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 11(2):112–122.

Driemel, A., Har-Peled, S., and Wenk, C. (2012). Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127.

Driemel, A., Krivošija, A., and Sohler, C. (2016). Clustering time series under the Fréchet distance. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 766–785. Society for Industrial and Applied Mathematics.

Driemel, A. and Silvestri, F. (2017). Locality-Sensitive Hashing of Curves. In *33rd International Symposium on Computational Geometry (SoCG 2017)*, volume 77, pages 37:1–37:16.

Eiter, T. and Mannila, H. (1994). Computing discrete Fréchet distance. Technical report, Citeseer.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231.

Estrada, M., Roca-Riu, M., Badia, H., Robusté, F., and Daganzo, C. F. (2011). Design and implementation of efficient transit networks: procedure, case study and validity test. *Procedia-Social and Behavioral Sciences*, 17:113–135.

Evans, M. R., Oliver, D., Shekhar, S., and Harvey, F. (2012). Summarizing trajectories into k-primary corridors: A summary of results. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL 12, pages 454–457, New York, NY, USA. Association for Computing Machinery.

Feige, U. (1998). A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652.

Fonseca Junior, J. G. d. S., Oozeki, T., Ohtake, H., Takashima, T., and Ogimoto, K. (2014). Regional forecasts and smoothing effect of photovoltaic power generation in Japan: An approach with principal component analysis. *Renewable Energy*, 68:403 – 413.

Fonseca Junior, J. G. d. S., Oozeki, T., Ohtake, H., Takashima, T., and Ogimoto, K. (2015). Regional forecasts of photovoltaic power generation according to different data availability scenarios: a study of four methods. *Progress in Photovoltaics: Research and Applications*, 23(10):1203–1218.

Fraley, C. and Raftery, A. E. (2002). Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*, 97(458):611–631.

Furth, P. and Noursalehi, P. (2015). Evaluating the connectivity of a bicycling network. Technical report.

Ghosh, A., Rozemberczki, B., Ramamoorthy, S., and Sarkar, R. (2018). Topological signatures for fast mobility analysis. In *26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL)*. ACM.

Gionis, A., Indyk, P., Motwani, R., et al. (1999). Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529.

Giunta, G., Salerno, R., Ceppi, A., Ercolani, G., and Mancini, M. (2019). Effects of model horizontal grid resolution on short-and medium-term daily temperature forecasts for energy consumption application in European cities. *Advances in Meteorology*, 2019.

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

Graabak, I. and Korps, M. (2016). Variability characteristics of european wind and solar power resourcesa review. *Energies*, 9:449.

Gudmundsson, J. and Horton, M. (2017). Spatio-temporal analysis of team sports. *ACM Computing Surveys (CSUR)*, 50(2):1–34.

Hamming, R. W. (1950). Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160.

Holmgren, W. F., Hansen, C. W., and Mikofski, M. A. (2018). PVLIB python: a python package for modeling solar energy systems. *Journal of Open Source Software*. DOI: 10.21105/joss.00884.

Hood, J., Sall, E., and Charlton, B. (2011). A GPS-based bicycle route choice model for San Francisco, California. *Transportation letters*, 3(1):63–75.

Huttenlocher, D. P., Kedem, K., and Kleinberg, J. M. (1992). On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane. In *Proceedings of the eighth annual symposium on Computational geometry*, pages 110–119.

Indyk, P. (2001). On approximate nearest neighbors under L norm. *Journal of Computer and System Sciences*, 63(4):627–638.

Indyk, P. (2002). Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 102–106. ACM.

Indyk, P., Matoušek, J., and Sidiropoulos, A. (2004). Low-distortion embeddings of finite metric spaces. *Handbook of discrete and computational geometry*, 37:46.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. *Proc. 30th ACM Symposium on Theory of Computing (STOC)*, 8:321–350.

Jiang, Z., Evans, M., Oliver, D., and Shekhar, S. (2016). Identifying k primary corridors from urban bicycle GPS trajectories on a road network. *Information Systems*, 57:142 – 159.

Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350.

Kalnay, E. (2003). *Atmospheric modeling, data assimilation and predictability*. Cambridge university press.

Kaprykowsky, H. and Rodet, X. (2006). Globally optimal short-time dynamic time warping, application to score to audio alignment. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 5, pages V–V. IEEE.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Katsikouli, P., Astefanoaei, M., and Sarkar, R. (2018). Distributed mining of popular paths in road networks. In *DCOSS 2018-International Conference on Distributed Computing in Sensor Systems*, pages 1–8. IEEE.

Katsikouli, P., Sarkar, R., and Gao, J. (2014). Persistence based online signal and trajectory simplification for mobile devices. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 371–380. ACM.

Kumar, D., Wu, H., Lu, Y., Krishnaswamy, S., and Palaniswami, M. (2016). Understanding urban mobility via taxi trip clustering. In *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, volume 1, pages 318–324.

Laporte, G., Mesa, J., Ortega, F., and Perea, F. (2011). Planning rapid transit networks. *Socio-Economic Planning Sciences*, 45(3):95 – 104.

Laporte, G. and Pascoal, M. M. (2015). Path based algorithms for metro network design. *Computers and Operations Research*, 62:78 – 94.

Lee, Y.-J. (2012). Comparative measures for transit network performance analysis. In *Journal of the Transportation Research Forum*, volume 47.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Li, Y., Su, Y., and Shu, L. (2014). An ARMAX model for forecasting the power output of a grid connected photovoltaic system. *Renewable Energy*, 66:78 – 89.

Lipton, Z. C. (2015). A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019.

Mathews, G. B. (1896). On the partition of numbers. *Proceedings of the London Mathematical Society*, 1(1):486–490.

Mathiesen, P. and Kleissl, J. (2011). Evaluation of numerical weather prediction for intra-day solar forecasting in the continental United States. *Solar Energy*, 85(5):967 – 977.

Matousek, J. (1992). Reporting points in halfspaces. *Computational Geometry*, 2(3):169–186.

Mauttone, A., Mercadante, G., Rabaza, M., and Toledo, F. (2017). Bicycle network design: model and solution algorithm. *Transportation Research Procedia*, 27:969 – 976. 20th EURO Working Group on Transportation Meeting, EWGT 2017, 4-6 September 2017, Budapest, Hungary.

Mills, A., Ahlstrom, M., Brower, M., Ellis, A., George, R., Hoff, T., Kroposki, B., Lenox, C., Miller, N., Milligan, M., et al. (2011). Dark shadows. *IEEE Power and Energy Magazine*, 9(3):33–41.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. (2016). Distributed submodular maximization. *The Journal of Machine Learning Research*, 17(1):8330–8373.

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., and Damas, L. (2013). Predicting taxi-passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402.

Mustafa, N. and Ray, S. (2010). Improved results on geometric hitting set problems. *Discrete and Computational Geometry*, 44:883–895.

Nanni, M. (2005). Speeding-up hierarchical agglomerative clustering in presence of expensive metrics. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 378–387. Springer.

Newell, G. (1979). Some issues relating to the optimal design of bus routes. *Transportation Science*, 13(1):20–35.

Newson, P. and Krumm, J. (2009). Hidden Markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, pages 336–343, New York, NY, USA. ACM.

Niedermayer, J., Züfle, A., Emrich, T., Renz, M., Mamoulis, N., Chen, L., and Kriegel, H.-P. (2013). Probabilistic nearest neighbor queries on uncertain moving object trajectories. *Proc. VLDB Endow.*, 7(3):205–216.

Ogliari, E., Dolara, A., Manzolini, G., and Leva, S. (2017). Physical and hybrid methods comparison for the day ahead PV output power forecast. *Renewable Energy*, 113:11 – 21.

Panigrahy, R. (2006). Entropy based nearest neighbor search in high dimensions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1186–1195. Society for Industrial and Applied Mathematics.

Perez, R., David, M., Hoff, T. E., Jamaly, M., Kivalov, S., Kleissl, J., Lauret, P., Perez, M., et al. (2016). Spatial and temporal variability of solar energy. *Foundations and Trends® in Renewable Energy*, 1(1):1–44.

Perez, R., Kivalov, S., Schlemmer, J., Hemker, K., Renn, D., and Hoff, T. E. (2010). Validation of short and medium term operational solar radiation forecasts in the US. *Solar Energy*, 84(12):2161 – 2172.

Perpin, O. and Lorenzo, E. (2011). Analysis and synthesis of the variability of irradiance and PV power time series with the wavelet transform. *Solar Energy*, 85(1):188 – 197.

Petrelli, M. (2004). A transit network design model for urban areas. *WIT Transactions on The Built Environment*, 75.

Pinelli, F., Nair, R., Calabrese, F., Berlingerio, M., Di Lorenzo, G., and Sbodio, M. L.

(2016). Data-driven transit network design from mobile phone trajectories. *IEEE Transactions on Intelligent Transportation Systems*, 17(6):1724–1733.

Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270.

Reno, M. J., Hansen, C. W., and Stein, J. S. (2012). Global horizontal irradiance clear sky models: Implementation and analysis. *SANDIA report SAND2012-2389*.

Reyes, M., Dominguez, G., and Escalera, S. (2011). Featureweighting in dynamic timewarping for gesture recognition in depth data. In *2011 IEEE international conference on computer vision workshops (ICCV Workshops)*, pages 1182–1188. IEEE.

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49.

Schmidhuber, J. and Hochreiter, S. (1997). Long short-term memory. *Neural Comput*, 9(8):1735–1780.

Shang, S., Ding, R., Zheng, K., Jensen, C. S., Kalnis, P., and Zhou, X. (2014). Personalized trajectory matching in spatial networks. *The VLDB Journal*, 23(3):449–468.

Shi, J., Lee, W.-J., Liu, Y., Yang, Y., and Wang, P. (2012). Forecasting power output of photovoltaic systems based on weather classification and support vector machines. *IEEE Transactions on Industry Applications*, 48(3):1064–1069.

Shivashankar, S., Mekhilef, S., Mokhlis, H., and Karimi, M. (2016). Mitigating methods of power fluctuation of photovoltaic (PV) sources - A review. *Renewable and Sustainable Energy Reviews*, 59:1170 – 1184.

Sobri, S., Koohi-Kamali, S., and Abd Rahim, N. (2018). Solar photovoltaic generation forecasting methods: A review. *Energy Conversion and Management*, 156:459–497.

Taha, A. A. and Hanbury, A. (2015). An efficient algorithm for calculating the exact Hausdorff distance. *IEEE transactions on pattern analysis and machine intelligence*, 37(11):2153–2163.

Thorndike, R. L. (1953). Who belongs in the family? *Psychometrika*, 18(4):267–276.

Ulbricht, R., Hahmann, M., Donker, H., and Lehner, W. (2014). Systematical evaluation of solar energy supply forecasts. In Woon, W. L., Aung, Z., and Madnick, S., editors, *Data Analytics for Renewable Energy Integration*, pages 108–121, Cham. Springer International Publishing.

Verplanken, B. and Roy, D. (2016). Empowering interventions to promote sustainable lifestyles: Testing the habit discontinuity hypothesis in a field experiment. *Journal of Environmental Psychology*, 45:127–134.

Wang, Y., Liao, W., and Chang, Y. (2018). Gated recurrent unit network-based short-term photovoltaic forecasting. *Energies*, 11:2163.

Widen, J. (2011). Correlations between large-scale solar and wind power in a future scenario for Sweden. *IEEE Transactions on Sustainable Energy*, 2(2):177–184.

Wiemken, E., Beyer, H., Heydenreich, W., and Kiefer, K. (2001). Power characteristics of PV ensembles: experiences from the combined power production of 100 grid connected PV systems distributed over the area of Germany. *Solar energy*, 70(6):513–518.

Wirasinghe, S. (1980). Nearly optimal parameters for a rail/feeder-bus system on a rectangular grid. *Transportation Research Part A: General*, 14(1):33 – 40.

Wolpert, D. H., Bieniawski, S. R., and Rajnarayan, D. G. (2013). Probability collectives in optimization. In *Handbook of Statistics*, volume 31, pages 61–99. Elsevier.

Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., and Zhang, C. (2020). Connecting the dots: Multivariate time series forecasting with graph neural networks. *arXiv preprint arXiv:2005.11650*.

Xie, D., Li, F., and Phillips, J. M. (2017). Distributed trajectory similarity search. *Proceedings of the VLDB Endowment*, 10(11):1478–1489.

Yang, H., Kurtz, B., Nguyen, D., Urquhart, B., Chow, C. W., Ghonima, M., and Kleissl, J. (2014). Solar irradiance forecasting using a ground-based sky imager developed at UC San Diego. *Solar Energy*, 103:502 – 524.

Yu, J., Wang, Z., Lu, B., and Sun, H. (2018). Mining k primary corridors from vehicle GPS trajectories on a road network based on traffic flow. In *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 79–85.

Zeng, J., Telang, G., Johnson, M. P., Sarkar, R., Gao, J., Arkin, E. M., and Mitchell,

J. S. (2017). Mobile r-gather: Distributed and geographic clustering for location anonymity. In *Proceedings of the 18th ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM.

Zeytounian, R. K. (1991). *Meteorological fluid dynamics: asymptotic modelling, stability and chaotic atmospheric motion*, volume 5. Springer Science & Business Media.

Zhang, G. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159 – 175.

Zhou, Y., Wang, W., He, D., and Wang, Z. (2014). A fewest-turn-and-shortest path algorithm based on breadth-first search. *Geo-spatial Information Science*, 17(4):201–207.

Zhu, X. and Guo, D. (2014). Mapping large spatial flow data with hierarchical clustering. *Transactions in GIS*, 18(3):421–435.