Sustainable Communication Networks
University of Bremen
Prof. Dr. Anna Förster

Master Thesis

TRAILS Mobility model Optimization

of

Leonardo Andres Sarmiento Moscoso

Matriculation Number: 3110474

Bremen, 12th of September 2019

Supervised by:
Prof. Dr. Anna Förster
MSc Vishnupriya Kuppusamy
Dr.-Ing. Asanga Udugama

I assure, that this work has been done solely by me without any further help from others except for the offical attendance by the Chair of Sustainable Communication Networks. The literature used is listed completely in the bibliography.

Bremen, 12th of September 2019

(Leonardo Andres Sarmiento Moscoso)

CONTENTS

ABSTRACT

According to state of the art research, simulation of mobile networks is preferred over real test beds, specially in the evaluation of communication protocols used in Opportunistic Networks (OppNet) or Mobile Ad hoc NETworks (MANET). The main reason behind it is the difficulty to perform experiments in real scenarios. Nonetheless, in order to define mobility patterns of users in a simulation, a mobility model is required. Trace-based models can be used for this purpose, but they are difficult to obtain and they are not flexible or scalable. Another option is TRAce-based ProbabILiStic Mobility Model (TRAILS). TRAILS mimics the spatial dependency, the geographic restrictions, and the temporal dependency from real scenarios. Additionally TRAILS can be scaled in number of mobile users and simulation time.

In this thesis an optimized tool was implemented to generate TRAILS graphs from real traces. A module in OMNet++ was developed to import TRAILS graphs and evaluate mobile networks in a simulated environment. Additionally, TRAILS was validated by comparing the simulated mobility patterns with mobility patterns of real traces. The performance of the implemented algorithms were evaluated in terms of computation time and memory consumption.

TRAILS is capable of representing the interaction between users of real scenarios with a higher accuracy if the record time of traces is shorter. Additionally, it was observed that a simulation with TRAILS requires less computation time than a simulation with real traces, and that a TRAILS graph uses less data memory than traces.

CHAPTER 1

Introduction

There is a increasing interest in the development of mobile network protocols [1], and there are two possible methodologies to evaluate the performance of new protocols. The first methodology is to use of real test-beds, and the second is through simulation. Test beds can mirror real scenarios closer than simulations, but simulations are easier to scale and to repeat, than experiments with test beds [2]. In consequence, simulators are preferred in the evaluation of OppNet and MANET protocols. [3]

In order to simulate a mobile network protocol, a simulator requires a mobility model to generate mobility patterns for each mobile node [4]. Mobility models have a major effect in the protocol performance of wireless networks. Therefore, a mobility model should mimic the movement patterns of the targeted real scenario, otherwise the observations made from the simulation may be incorrect [5]. In other words, it is possible to evaluate a mobile wireless network protocol with a reasonable accuracy using a simulator, only if the mobility model used in the simulation represents the mobility patterns of expected real users. [6]

Mobile network protocols have a large range of applications. For example, wireless sensor networks to monitor sea birds [7], workers in disaster recovery scenes [8], or defense applications such as tactical missions [9]. In all of the mentioned applications mobility plays a major role [10]. Mobility models are not only useful for mobile networks, they can also be used in traffic forecasting, mobile virus spread, and urban planning. [2]

Mobility models can be trace-based models or synthetic models. Trace-based models represent the movement of real users on specific scenarios, but they are difficult to obtain, and recorded scenarios available in public databases are limited. Synthetic models are flexible and scalable, but depending on the model they may result different from real scenarios. [11]

Usually, mobility models are employed to represent human mobility which is characterized by different aspects. People are normally more attracted to stay in popular places and tend to visit the same places every day, but they also take irregular trips [12]. People are categorized as a social network with a high level of clustering. Human movement is heavily motivated by the necessity to interact and cooperate. Human movement is not completely random and it can be predicted to some extent [13]. In consequence, if the objective of the model is to represent a social network such as humans, a synthetic mobility model cannot be based purely on random movements. [14]

There are three characteristics that a synthetic mobility model should capture from traces in order to mimic the behavior of real mobile users. Those characteristics are spatial dependency, temporal dependency, and geographic restrictions [15]. Pure-random synthetic models such as Random WayPoint (RWP) [16] do not capture any of these characteristics. On the other hand, there are mobility models that capture one or more of the mentioned characteristics. For example, Community-based Mobility Model (CMM) [17] captures spatial dependency by exploiting the social relation between users from real scenarios. Another example is TRAILS model [18] proposed in the paper entitled “TRAILS - a Trace-Based

Probabilistic Mobility Model" which captures all three characteristics [18].

TRAILS captures spatial dependency from real scenarios by forming communities of mobile nodes in the same places real nodes spend time. Mobile users in TRAILS mimic geographic restrictions by traveling through the same trajectories as users in recorded traces. Additionally, TRAILS portrays a similar temporal dependency as real scenarios because mobile users move between different locations with the same speed in simulations with TRAILS and in real traces [18]. To capture the behavior of real mobile users, TRAILS [18] builds mobility patterns by extracting information from real traces about the places where users spend most of their time, and the routes that users choose to travel to those places.

This thesis was designed to fulfill four goals. The first goal is the development of an optimized tool to generate TRAILS' mobility graphs from real scenarios. The second goal is the implementation of TRAILS in a simulator. The third goal is the validation of TRAILS in relation to real scenarios. Finally, the fourth goal is the analysis of the model performance in different aspects such as computation time and memory consumption. Additionally, we seek to evaluate how TRAILS is affected when the number of mobile users and simulation time vary.

The methodology used in this thesis is literature review and statistical analysis. Characteristics that should be captured by a mobility model such as TRAILS were reviewed. Metrics were defined to measure the similarities and differences between TRAILS and real scenarios. Additionally, relevant algorithms were outlined in order to implement the TRAILS mobility model. In addition of scientific sources, quantitative approaches were used to test different hypothesis related with the performance of TRAILS. Statistical tests were performed with sampling distributions of mobility metrics of four different real scenarios and its corresponding TRAILS simulations. Finally, different metrics of the TRAILS model were analyzed.

This thesis has the following structure: the first chapter is an introduction; the second chapter explains the concepts behind mobile networks and mobility models, it presents a comparison of different mobility models, and it describes a mobility model based on traces called TRAILS; the third chapter describes the algorithms involved in the transformation of traces into TRAILS graphs; the fourth chapter explains the strategies used to reduce the computation time in the generation of TRAILS graphs; the fifth chapter presents the steps followed in the implementation of a TRAILS graph generator and a TRAILS simulator; the sixth chapter presents a comparison between TRAILS and traces, it analyzes the behavior of users in a TRAILS mobility model, and it shows the performance of the implemented algorithms; the seventh chapter describes conclusions and recommendations for future research.

It is argued in this thesis that the implemented TRAILS generator and simulator presented a favorable performance in terms of computation time and memory consumption. Although, it was determined that TRAILS cannot mimic real-traces perfectly, TRAILS is scaled in simulation time and number of mobile users in a way that cannot be achieved with trace-based mobility models.

CHAPTER 2

Related work

This thesis is built on previous research that published in a paper entitled "TRAILS - A Trace-Based Probabilistic Mobility Model". The mentioned paper presents a mobility model called TRAILS and describes the concepts behind it [18]. In this chapter, an overview about mobile networks is presented. Then, the key factors of mobility models are reviewed, and finally TRAILS is described.

2.1 Mobile Networks

There are two large categories of mobile networks. The first category is MANET and the second one is OppNet. In the past decade many researchers have studied the effects of mobility models in both categories. [19]

MANET is a set of mobile users forming a network without any pre-defined infrastructure [20]. On the other hand, OppNet is a mobile network which does not guaranty an end to end communication path between a sender and a receiver [21]. In consequence, mobility models have a major role in the design of a mobile network protocol. For this reason, the evaluation of that kind of protocol should be performed under realistic conditions such as a mobility model capable of representing a real scenario in a reasonable way [1].

In order to measure the performance of mobile network protocols the following metrics are used: **throughput** which expresses the average number of bits successfully received per second. [22]; **end to end delay** which describes the mean value of the time interval from a message being transmitted by a source node until the message is received by the destination node. [23]; and **packet delivery ratio** which represents the mean value of the relation between the total number of packets transmitted and the total number of packets successfully received. [24]

In a wireless network the transmission of data is more sensitive to failures due to the deterioration of link quality in established links [7]. Additionally, mobility causes changes in the routes leading to higher packet delay, and neighbor discovery mechanisms should be executed more frequently, because of changes in the network topology. Therefore, the metrics used to evaluate mobile network protocols are very sensitive to the characteristics of the mobility model. [25]

Mobile network protocols are classified in two categories based on the strategies used to forward packets through a network. The first category is called Memory-less protocols, and the second is called Memory-full protocols. Memory-less protocols have very simple rules in order to forward messages. On the other hand, in Memory-full protocols such as RAPID [27] or epidemic [26], a mobile node make use of the contact history with other nodes to forward messages. Therefore, mobility models are also required to properly design Memory-full protocols.

Simulation is preferred in the early stages of development or research of mobile network protocols, because a simulation is faster and more affordable than experiments in real environments [4]. However, there are some OppNet's simulators, each one of them has different characteristics. The ONE simulator is an integrated solution that can simulate delay-tolerant wireless protocols with synthetic and trace-based mobility models [28]. Adython simulates OppNets only with trace-based mobility models [29]. Opportunistic protocol simulator (OPS) [30]is an extension of the discrete event simulator OMNet++ [31]. OPS has the capacity to simulate OppNet's protocols with synthetic and trace-based mobility models. During this research TRAILS was implemented to work with OPS. [30]

2.2 Mobility models

There are two types of mobility, weak mobility and strong mobility. Weak mobility is characterized by unusual joins of new nodes in a network, or existing nodes experimenting a failure in their hardware, For example exhausting their batteries. On the other hand, Strong mobility is characterized by concurrent user's joins and node's failures as well as physical displacement of nodes. A mobility model is used to describe Strong mobility. [32]

A mobility model is defined as a set of rules used to generate trajectories for mobile users. Mobility models are required to capture the variations of a network topology provoked by the movement of users [4]. Mobility models describe the behavior or the movement of a user in relation with its position. The input of a mobility model is a mobility graph and it contains all the information that a user needs in order to behave according to a specific mobility model.

Mobility models are divided in two categories, Synthetic models and Trace-based models. Trace-based models are a set of trajectories from mobile users in a real environment, and synthetic models seek to represent the behavior of mobile users by rules defined randomly and parameters extracted from traces. [11]

2.2.1 Trace-based mobility models

Trace-based mobility models are a collection of information regarding trajectories of several users from real scenarios. [11]

Usually in trace-based models each user follows a trace composed by a set of trace-points where a trace-point is defined as an object with two attributes, geographical position, and time stamp. Additionally, the position and the time stamp have the same reference system for the whole model and the trace-points of each trace are ordered in time. [33]

Trace-Based models are expensive and difficult to obtain because they require the participation of real mobile users with devices to record their trajectories. However, the following technologies can be used to record real traces. [7]

- **Global Navigation Satellite System (GNSS)** [34]: gives the absolute latitude and longitude of a mobile user [35]. GNSS devices are high energy consuming and they are frequently disconnected in indoor environments. [36]
- **Radio Frequency Identification (RFID)**: static RFID transmitters can share their pre-saved position at a short range of one to two meters.

- **Received signal strength indicator (RSSI)**: RSSI can be used to determine the distance between two mobile nodes [38]. However, with the combination of triangulation [39] or trilateration [40] techniques it is possible to estimate the position of mobile nodes.
- **Acelerometers**: are able to detect translational and rotational movement, but the accuracy is affected by accumulation of errors. [41]
- **Hybrid**: techniques based on Bayesian probability in combination with more than one of the previous methodologies have proved to be very effective in terms of accuracy in the estimation of the position of mobile nodes. [42]

There are data bases of public access with trace-sets of real scenarios such as CRAWDAD [43], UNC-FORTH [44], and MobiLib [45]. However, the scenarios in those data-bases are limited, and each trace-set has its own format which means that we would need a different processing algorithm to use a different trace-set. [7] An example of a graph of a trace-based model is shown in figure 2.1.

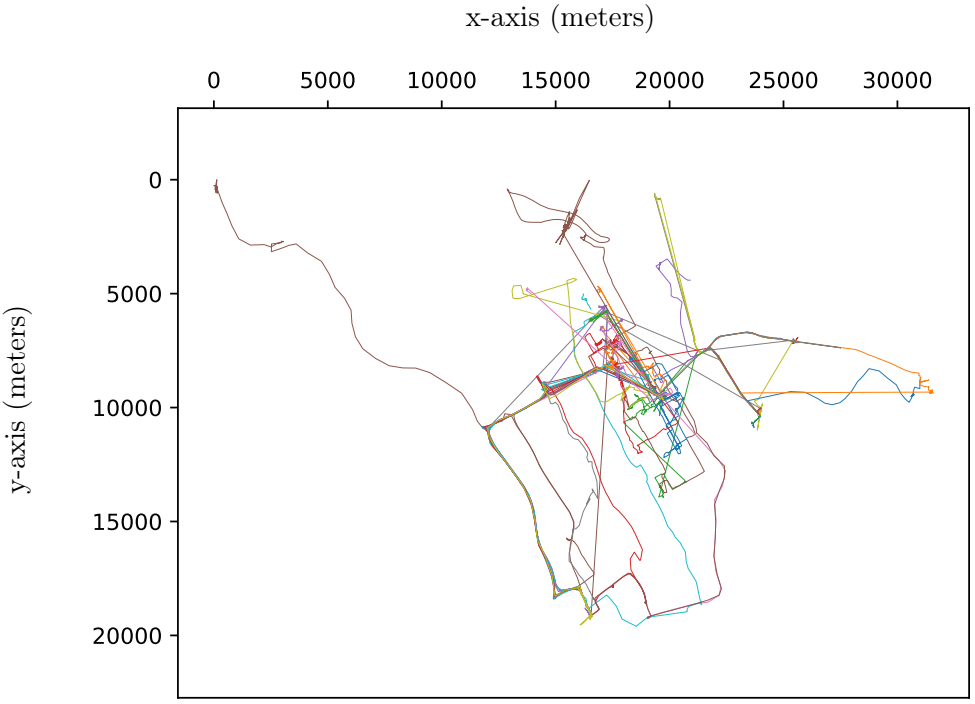


Figure 2.1: Traces graph of 39 pedestrians of 22.7 hours in New York [33]

2.2.2 Synthetic mobility models

A Synthetic mobility model is a collection of rules to control the physical displacement of mobile users in a simulation. [11]

A Synthetic model should contain different characteristics from a real targeted scenario in order to avoid misleading performance results in a simulation of mobile network protocols [5]. **Temporal dependency** which is the velocity that the mobile nodes use to travel [5]. **Spatial dependency** which is the social relation between mobile nodes [5]. **Geographic restrictions.** which are pre-defined trajectories, and obstacles to limit the displacement of mobile users. [5]

There are several synthetic mobility models such as RWP [16] where mobile nodes move through a surface without restrictions, with random directions, and with random velocities; CMM [17] in which mobility nodes form groups periodically based on a interaction matrix that represents the social relation between each pair of nodes; and TRAILS [18] that creates flexible simulations by extracting information about popular locations and pre-defined paths from real scenarios.

RWP is a purely randomic mobility model that does not capture any of the mentioned characteristics from real scenarios [16]. However, CMM does capture spatial dependency by using the social relation between real users in a simulation [17]. TRAILS captures geographic restrictions and temporal dependency by using real predefined paths, and it mimics spatial dependency by extracting information about common locations from real scenarios. [18]
Table 2.1 summarizes the characteristics of different synthetic mobility models.

Mobility model	Temporal dependency	Spatial dependency	Geographic restriction
RWP [16]	No	No	No
CMM [17]	Yes	No	No
TRAILS [18]	Yes	Yes	Yes

Table 2.1: Mobility models characteristics captured from real scenarios

2.3 TRAce-based ProbabILiStic Mobility Model (TRAILS)

TRAILS is a mobility model in which a user performs movements extracted from recorded traces, but it chooses the destinations randomly. The TRAILS model can be simulated with a larger number of users and for a longer time than the original traces. Therefore, TRAILS is a full scalable, and flexible mobility model.[18]

In TRAILS a graph generator creates a mobility graph with the information extracted from traces. Additionally, a simulator contains the rules that a mobile user follows with respect the mobility model.

2.3.1 Graph

The TRAILS graph is a mobility graph composed by a list of Points of interest (POIs) and a list of links between POIs. According to TRAILS a **Point of interest (POI)** is defined as a place in which one or more users spend a time interval higher than a Temporal range (TR) in it. For example, a super market, an office, or a house. [18]

A POI is represented as an object with the following attributes.

1. **Position** Cartesian coordinates of the POI's location.
2. **Spatial range (SR)** It is the POI's diameter. It is constant and it has the same value for all the POIs in the TRAILS graph.
3. **Time list** List of stay times, where a stay time is the time interval spent by a user inside the circle described by the position and diameter of the POI.

TRAILS defines a **Link** as a transition between two POIs. A link is represented with an object with the following attributes.

1. **Initial POI** Point of interest in which the link starts.
2. **Final POI** Point of interest in which the link ends.
3. **Trajectory** Set of consecutive trace-points between Initial POI and Final POI.

An example of a TRAILS graph is described in Figure 2.2. The links have arrows to represent direction, and the POIs are represented with the symbol X.

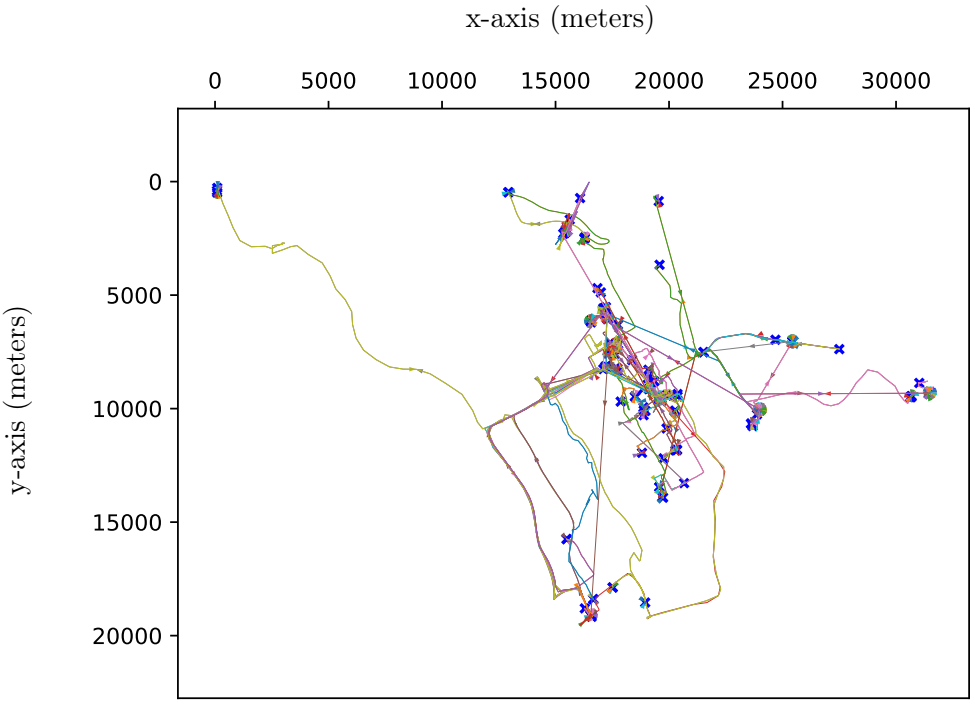


Figure 2.2: TRAILS graph of pedestrians in New York [33]

2.3.2 User behavior

According to TRAILS a simulated user has the following behavior, as it is shown in figure 2.3.

1. It chooses a random POI (A) as its new POI (N).
2. It starts in the position of N.
3. It selects a random stay time (H) from the Time list of N.
4. It selects a random link (Q) that has N as initial POI from the links list.
5. It waits a time interval equal to H in the position of N.
6. It follows the trace-points of Q until it reaches the position of its Final POI (B).
7. The user is assigned to B as its new POI (N)
8. The user repeats the process from step 3.

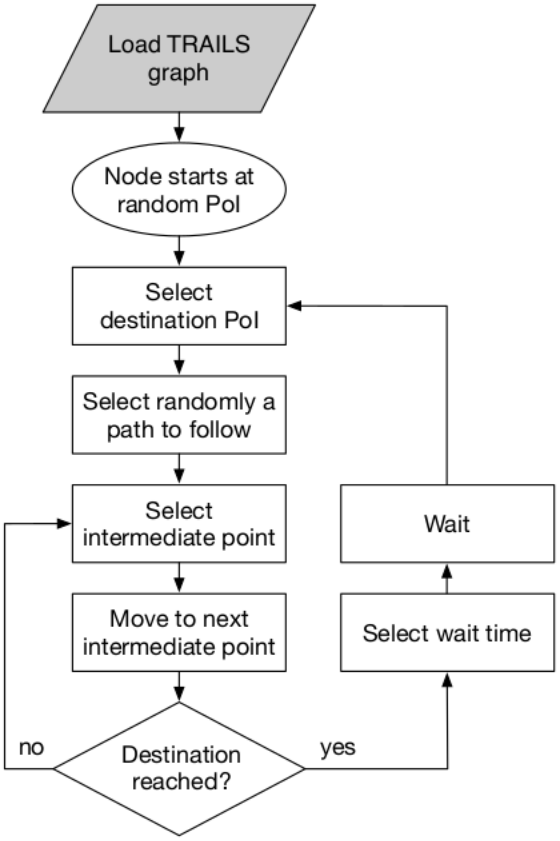


Figure 2.3: User behavior according to TRAILS [18]

CHAPTER 3

TRAILS processing algorithms

In order to use the TRAILS model it is necessary to generate a TRAILS graph. This chapter describes the algorithms used to process traces, and to create a TRAILS graph from the processed traces.

3.1 Traces processing

TRAILS graphs are extracted from recorded traces. Therefore, before generating a TRAILS graph, it is necessary to implement algorithms to import traces, transform coordinates, and filter measurement noise.
In order to reduce computation time without changing the final result, the Traces processing algorithm additionally removes redundant trace-points as is explained in section 4.1

3.1.1 Coordinates

There are traces formats in which the spatial position is described in degrees (longitude and latitude), and the TRAILS generator works with Cartesian coordinates in meters. Therefore, it is necessary to transform the coordinates of the traces. The Haversine formula computes the distance between two points (J and H) in meters. In the Haversine formula the input points are described by longitude and latitude in degrees. [46]
The pre-processing algorithm finds the minimum longitude and minimum latitude (minLon, minLat) of all trace points in the trace graph, then it computes the Cartesian coordinates with the relations 3.2, 3.3.

$$distance = Haversine(longitude[i], latitude[i], longitude[h], latitude[h]) \tag{3.1}$$

$$Xcoordinate = Haversine(minLon, latitude[i], longitude[i], latitude[i]) \tag{3.2}$$

$$Ycoordinate = Haversine(longitude[i], minLat, longitude[i], latitude[i]) \tag{3.3}$$

3.1.2 GNSS noise

GNSS devices present errors on its position measurements. Position errors (trace noise) can be perceived as movements that a user did not execute (false trace points) as shown in figures 3.1, 3.2.
The pre-processing algorithm uses two methodologies to detect and to remove false trace points.

The first methodology measures the distances of a trace point (tp[i]) with respect to its previous trace point (tp[i-1]) and its next trace point (tp[i+1]), then, if any of both distances is larger than a threshold (MaxD), tp[i] is classified as a false trace point.
The Distance threshold to filter trace-points (MaxD) is defined by the maximum distance that a user can travel in the same direction. In other words, MaxD depends on the limits of the area used to record the traces.

The second methodology measures the velocities of a trace point (tp[i]) with respect to its previous trace point (tp[i-1]) and its next trace point (tp[i+1]), then if any of both velocities is larger than a threshold (MaxV), tp[i] is classified as a false trace point.
The Velocity threshold to filter trace-points (MaxV) is defined by the maximum velocity that a user can travel. For example, if the users are cars inside a city, then MaxV should be approximately 60m/s (216Km/h), because is unlikely that a car in a city would travel at a higher speed.

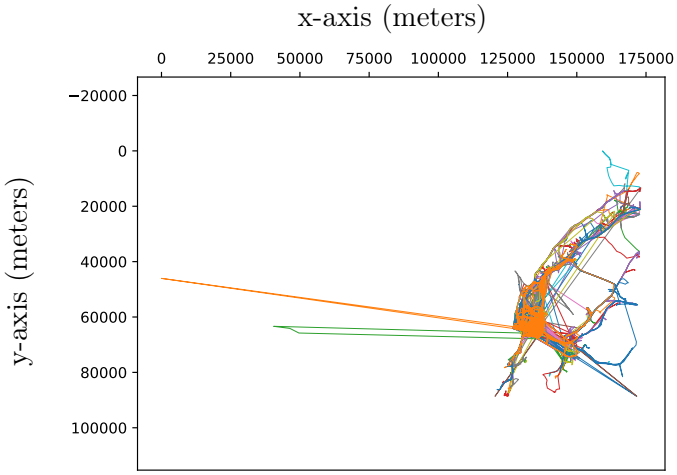


Figure 3.1: Traces graph of 22 taxis during 30 days in San Francisco with false trace-points[47]

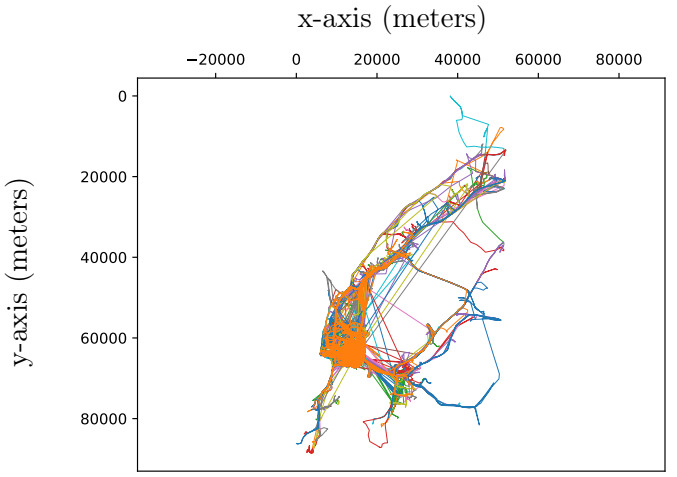


Figure 3.2: Traces graph of 22 taxis during 30 days in San Francisco without false trace-points[47]

3.2 Extraction of points of interest

The POIs are extracted from traces by grouping **STPs**. A Set of consecutive trace-points (STP) of a POI (Z) is defined as a segment of a trace that represents a user's activity for a time interval larger than TR, and at distance smaller than SR/2 in relation with the position of Z. The figure 3.3 shows a POI (Blue circle) and 2 STPs (Red segments).

The STPs are grouped with two different methods depending on the availability of POIs in the graph. This chapter describes those methods and it summarizes the extraction algorithm.

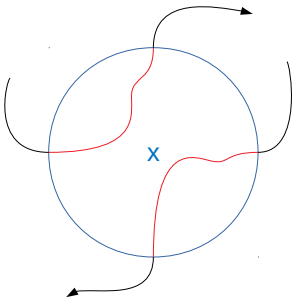


Figure 3.3: Representation of 2 STPs in a POI

3.2.1 Related STP Method (RSM)

Related STP method (RSM) takes as input a List of trace-points (Tp) and a List of POIs (Lp), then it searches a STP (B) of a POI (X), where X is part of Lp and one or more trace-points of B are part of Tp. If the B is found, RSM returns the B and X.
The algorithm behind RSM is described with the following steps.

1. Read the first trace point (tp[i], i=0) of Tp.
2. By using Neighbor Cell Algorithm (NCL) described in section 4.3, find a POI (X) where the distance between the position of X and the position tp[i] is smaller than SR/2.
3. If X was found go to step 5. On the other case, go to step 4.
4. If tp[i] is the last trace-point in Tp, Return Null. On the other case, increase i by 1, and go to step 2.
5. Read the trace-point tp[i+h], with h=0;
6. If the distance between the position of X and the position of the consecutive trace-point tp[i+h+1] is smaller than $SR/2$, go to step 7. On the other case, go to step 8. Distances are compared with Comparison of multiple distances algorithm (COMD) described in section 4.2.
7. Increase h by 1 and go to step 6.
8. Calculate the time interval (stay time) between tp[i] and tp[i+h].
9. If the stay time is larger than TR, create a STP (B) with the trace segment between tp[i] and tp[i+h], and go to step 10. On the other case, go to step 4.
10. Return B and X.

3.2.2 Independent STP Method (ISM)

Independent STP method (ISM) takes as input an initial trace-point (tp[i]) and it searches a STP without taking into account the POIs in the graph. If the STP is found, RSM returns the STP.
The algorithm behind ISM is described with the following steps.

1. Start with a trace point tp[i+h], h=0.
2. Create a list of trace-points (A) with the trace point tp[i+h].
3. Add a consecutive trace point tp[i+h+1] to A and find the smallest enclosing circle around the trace-points in A.
4. If the new circle's diameter is smaller than SR go to step 5. On the other case, go to step 6.
5. Increase h by 1 and go to step 3.
6. Calculate the time interval (stay time) between tp[i] and tp[i+h].
7. If the stay time is larger than TR, create a STP with the trace-points of A, and go to step 8. On the other case, exit without returning the STP.
8. Return the STP.

ISM uses the Smallest enclosing circle algorithm (SEC) proposed by Emo Welzl, to find the smallest circle around a list of trace-points [48]. SEC has as input a set of points and gives as output the center and diameter of the smallest circle enclosing of that set of points.

3.2.3 Extraction algorithm

The extraction algorithm uses the SR and TR as inputs, and returns a list with all points of interest (POIs list) in a traces graph.
The POIs are found by following the next steps per each trace.

1. Start with the first trace-point (tp[i],i=0) of a user's trace;
2. Search a valid STP (A) starting from tp[i] by using ISM.
3. If A is found go to step 4. On the other case, increase i by 1, and go to step 2.
4. By using RSM, search a valid STP (B) that is related to a POI (X), and that one or more trace-points of B are part of A. Lp is formed by all POIs in the graph (POIs list), and Tp is formed by the trace-points of A.
5. If B and X are found, add time interval of B to the Time list of X, and go to step 6. On the other case, go to step 7.
6. Compute i=h+1, where h is the index of the last trace-point of B, and go to step 2.
7. Create a POI (Z) with the geometric center of the smallest enclosing circle around the trace-points of A, add the time interval of A to the Time list of Z, and add Z to the POIs list (Lp).
8. Compute i=h+1, where h is the index of the last trace-point of A, and go to step 2.

In figure 3.4 is shown an example of the extraction algorithm. The example describes the following steps.

1. STP A is found with the ISM method.
2. POI (X) is created with the geometric center of A, and the time interval of A is added to the Time list of X.
3. STP B is found with the ISM method.
4. STP C and POI (X) are found with the RSM method, and the time interval of C is added to the Time list of X. (C has some of the trace-points of B).

The extraction algorithm creates a new POI (Y) with a STP (D) only if non of the trace-points of D can form a new STP that can be part of any of the existing POIs. Therefore, the extraction algorithm avoids the creation of redundant POIs.

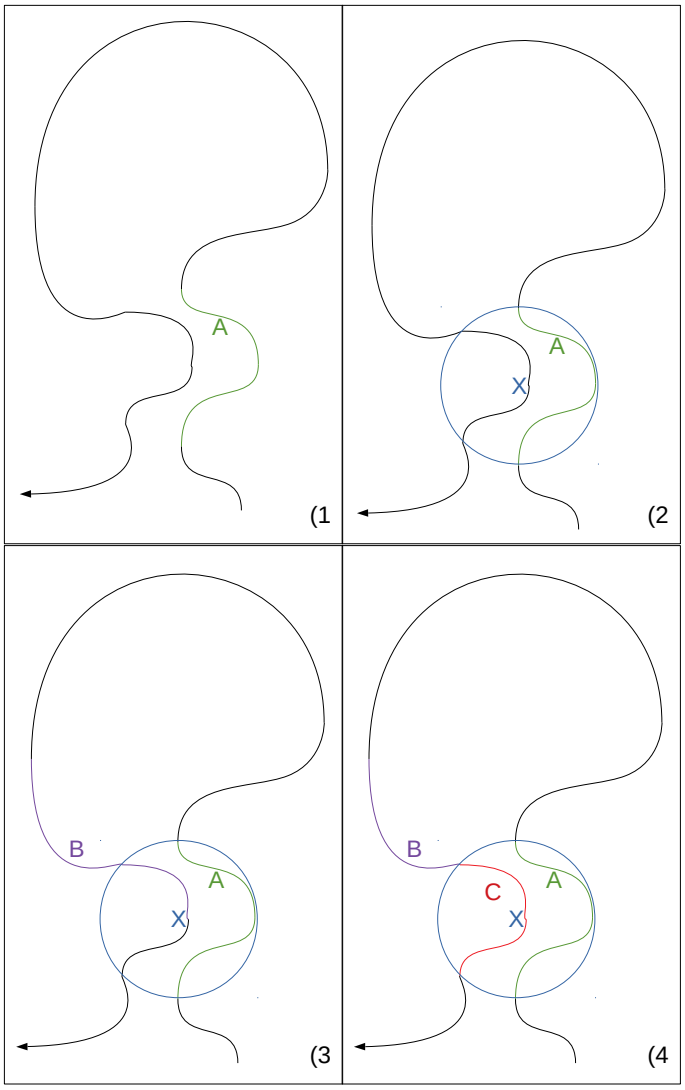


Figure 3.4: Example of the extraction algorithm

3.3 Connection of points of interest

POIs are connected with each other by links. A link is formed by a set of consecutive trace-points from the recorded traces. To extract a link the next steps are followed. First, find a set of trace-points (Q) between 2 POIs. Q is limited by the trace-points tp[i+1], and tp[h-1]. Where, tp[i] is the last trace-point of a STP that belongs to an initial POI (A), and tp[h] is the first trace-point of a STP that belongs to a final POI B. Then, create a link, add Q, A, and B as its attributes. Finally, add the link to the links list.

There are links that produce an undesired behavior in a TRAILS simulation. Those links are called No-return links and unrealistic links.

3.3.1 No-return Links

According to Figure 3.5, if a user (H) is in the POI A, and then it takes the link Y, H is able to return to A by taking the path formed by links X and W. Ina different scenario, if H starts in A, but it takes link Z, then there is no possible path that H can follow that leads back to A. In the example presented in Figure 3.5, Z is categorized as a **no-return link**. In other words, if a link Q has an initial POI C and a final POI D, and there is no path from D to C, then Q is a **no-return link**.

Based in the example of figure 3.5, no-return links generate mobility constraints in the users of the TRAILS mobility model. However, TRAILS can be simulated with no time constraint. Therefore, TRAILS handles no-return links in 3 different ways. **Bidirectional mode** which creates artificial links in the opposite direction for all existing links. **Smart Bidirectional mode** which creates artificial links in the opposite direction for only no-return links. **Cyclic mode** which removes all no-return links.

No-return links are identified by the TRAILS processing algorithm with the Breath-First Search algorithm. [49]. Anyway, No-return links are present in a TRAILS graph because TRAILS graphs are built from traces with a finite recording time, and usually a user's trace does not end where it started.

As is shown in Figure 3.6, the TRAILS graph changes significantly, if the no-return links are removed. On the other hand, the TRAILS graph presented in Figure 3.7 does not change much, if the no-return links are removed. In conclusion, traces with long recording periods can be less affected by no-return links.

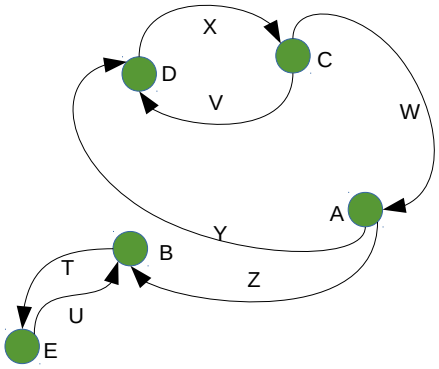
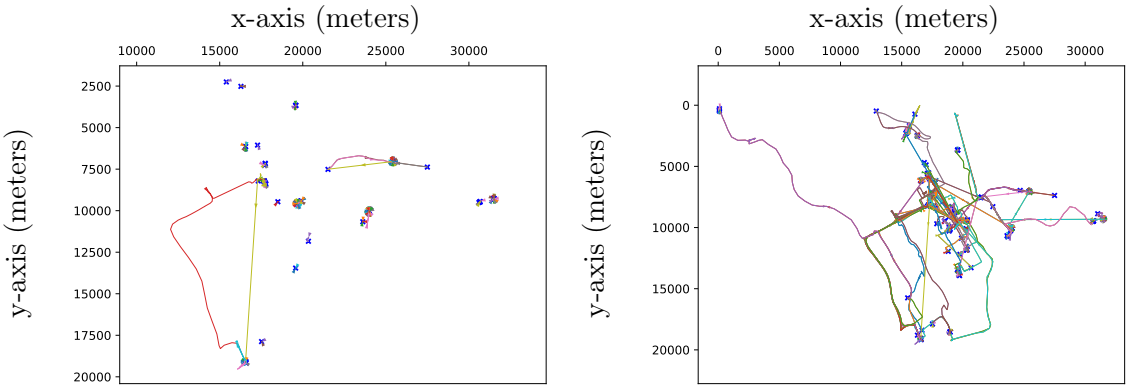
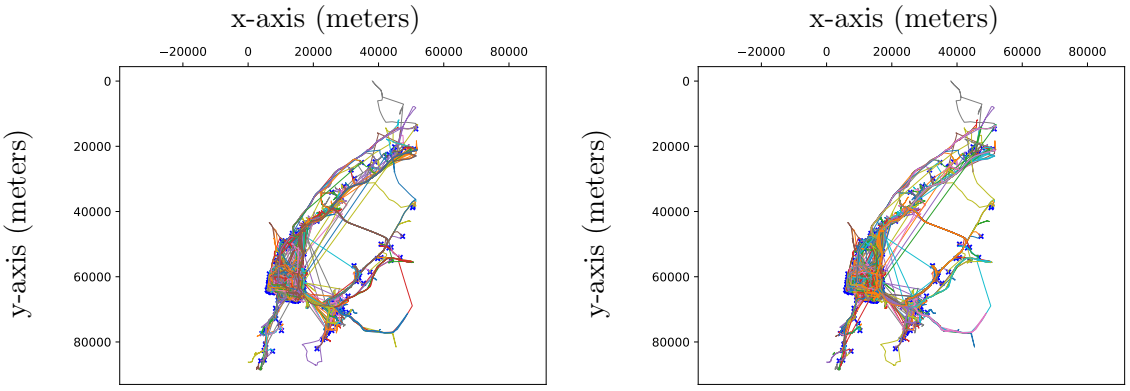


Figure 3.5: Representation of a TRAILS graph with POIs [A,B,C,D,E], with and links [T,U,V,W,X,Y,Z].



A. Without no-return links B. With no-return links

Figure 3.6: TRAILS graph with 293 links, and 137 no-return links from traces of 39 pedestrians during 22.7 hours in New York [33]



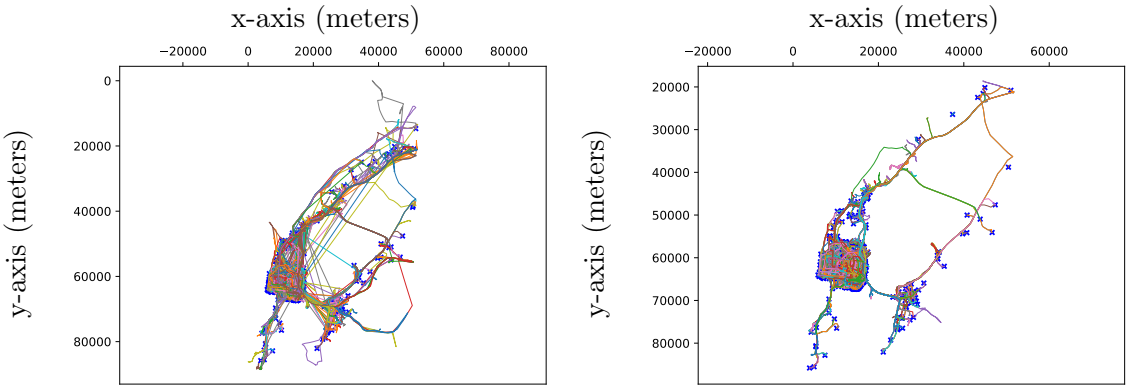
A. Without no-return links B. With no-return links

Figure 3.7: TRAILS graph with 7249 links, and 33 no-return links from traces of 22 taxis during 30 days in San Francisco [47]

3.3.2 Unrealistic links

As it can be seen in section A of Figure 3.8 there are consecutive trace-points that are very distant with each other, the reason is that the traces have been recorded with a highly variable sampling time (from 30 to 3000 seconds). If is taken into consideration that the users in the TRAILS graph of Figure 3.8 are taxis, it is observable in plot A the straight line between two very distant consecutive trace-points is far for representing a real movement. The TRAILS generator solves this problem by removing unrealistic links as shown in plot B of Figure 3.8.

An unrealistic link is defined as a link that has a distance between two consecutive trace-points (tp[i] and tp[i+1]) larger than a threshold (MinU). In the example of Figure 3.8, the Minimum Distance for an unrealistic path (MinU) is equal to 2Km.



A. With unrealistic links B. Without unrealistic links

Figure 3.8: TRAILS graph from traces of 22 taxis during 30 days in San Francisco [47]

CHAPTER 4

TRAILS computation time optimization

The computation time for generating TRAILS graphs depends on the size of the original traces. With the implementation of the following algorithms the computation time is reduced significantly without changing the final results.

This chapter describes three algorithms designed to reduce the computation time of the TRAILS generator. The first algorithm remove redundant trace-points. The second eliminate unnecessary mathematical operations. The last algorithm reduce the number comparisons between point's coordinates.

4.1 Remove redundant trace-points

In traces graphs there are usually trace-points that do not provide new information, those trace-points are called redundant trace-points. A redundant trace-point is defined as a trace point (tp[i]) that has the same spatial position as its previous and next trace-points (tp[i-1],tp[i+1]). The computation time of the TRAILS generator depends on the total number of trace-points. Therefore, redundant trace-points are detected and removed from the traces as the example shown in 4.1, 4.2.

Trace-point	tp[0]	tp[1]	**tp[2]**	tp[3]	tp[4]
X (m)	5.5	4.3	**4.3**	4.3	3.8
Y (m)	6.7	5.2	**5.2**	5.2	4.1
Time (s)	0	30	**60**	90	120

Table 4.1: Trace with redundant trace-points

Trace-point	tp[0]	tp[1]	tp[2]	tp[3]
X (m)	5.5	4.3	4.3	3.8
Y (m)	6.7	5.2	5.2	4.1
Time (s)	0	30	90	120

Table 4.2: Trace without redundant trace-points

4.2 Comparison of multiple distances algorithm (COMD)

The process used to generate a TRAILS graph frequently compares a constant range with the euclidean distance between two points. The use of a direct distance comparison leads to unnecessary computation time. Therefore, an algorithm called COMD has been designed for this purpose. COMD reduces the computation time by comparing the Chebyshev, Manhattan, and square-euclidean distances with equivalent ranges instead of computing the euclidean distance.

4.2.1 Equivalent Chebyshev ranges

COMD compares the Chebyshev distance (Cd) between two points (A,B) with the Minimum Chebyshev range (MinCR) and with the Maximum Chebyshev range (MaxCR). The Chebyshev distance between two points (A,B) in Cartesian coordinates is defined by equation 4.1. [50]

$$Cd = max(abs(A.x - B.x),abs(A.y - B.y))[50] \tag{4.1}$$

The MinCR of a threshold R is defined as the minimum Chebyshev distance between two points (A,B), where the euclidean distance of [A,B] is bigger than R.
On the other hand, the MaxCR of a threshold R is defined as the maximum Chebyshev distance between two points (A,B), where the euclidean distance of [A,B] is smaller than R.

To find out the values MinCR and MaxCR for a constant R, it is necessary to define the relation of Cd and R, as shown in equation 4.2 and figure 4.1.

$$Cd = max(R * abs(\cos(\theta)),R * abs(\sin(\theta))) \tag{4.2}$$

Based on the equation 4.2, the equivalent Chebyshev ranges are equal to the relations 4.3 and 4.4.

$$MinCR = R \tag{4.3}$$

$$MaxCR = R/\sqrt{2} \tag{4.4}$$

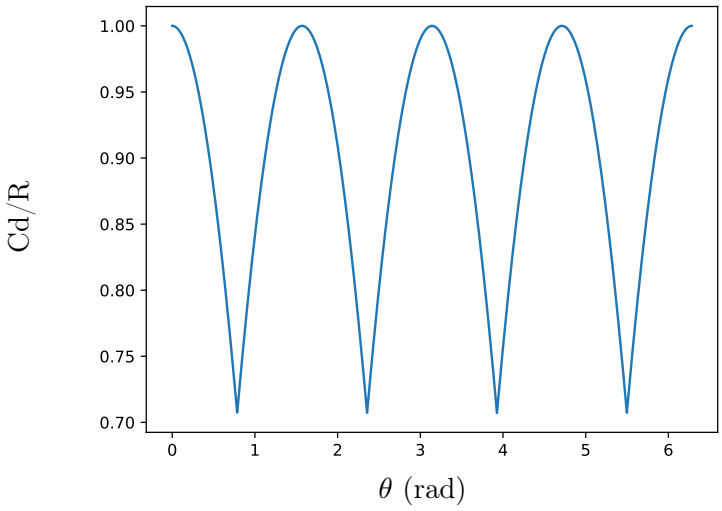


Figure 4.1: Relation between Chebyshev distance and Euclidean distance

4.2.2 Equivalent Manhattan ranges

COMD also compares the Manhattan distance (Md) between two points (A,B) with the Minimum Manhattan range (MinMR) and with Maximum Manhattan range (MaxMR).
The Manhattan distance between two points (A,B) in Cartesian coordinates is defined by equation 4.5. [50]

$$Md = abs(A.x - B.x) + abs(A.y - B.y)[50] \tag{4.5}$$

The MinMR of a threshold R is defined as the minimum Manhattan distance between two points (A,B), where the euclidean distance of [A,B] is bigger than R. On the other hand, the MaxMR of a threshold R is defined as the maximum Manhattan distance between two points (A,B), where the euclidean distance of [A,B] is smaller than R. To find out the values MinMR and MaxMR for a threshold R, it is essential to define the relation of Md and R, as shown in equation 4.6 and figure 4.2.

$$Md = R * abs(\cos(\theta)) + R * abs(\sin(\theta)) \tag{4.6}$$

Based on the equation 4.6, the equivalent Manhattan ranges are equal to the relations 4.7 and 4.8.

$$MinMR = R * \sqrt{2} \tag{4.7}$$

$$MaxMR = R \tag{4.8}$$

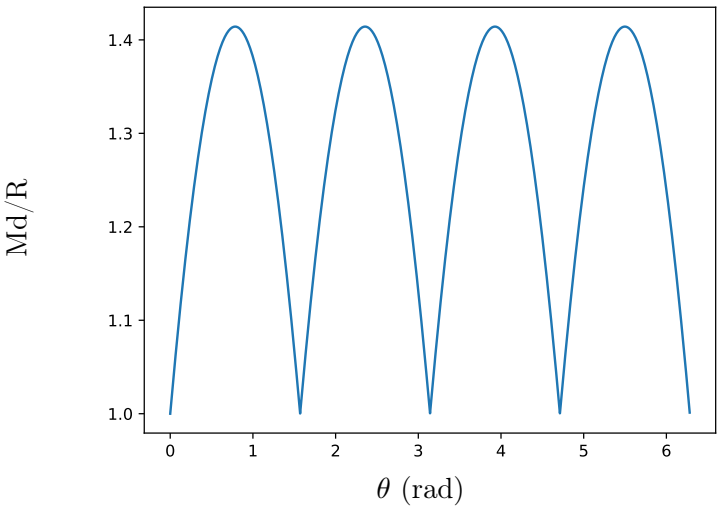


Figure 4.2: Relation between Manhattan distance and Euclidean distance

4.2.3 Algorithm description

COMD provides the same result than the comparison between a range R and the euclidean distance of two points (A,B) by following the next steps.

1. Compute the X distance (rx) between [A,B].
2. Compute the Y distance (ry) between [A,B].
3. Compute Cd, $Cd = max(rx,ry)$.
4. If Cd is bigger than MinCR (R), return False.
5. If Cd is smaller than MaxCR ($R/\sqrt{2}$), return True.
6. Compute Md, $Md = rx + ry$.
7. If Md is bigger than MinMR ($\sqrt{2} * R$), return False.
8. If Md is smaller than MaxMR (R), return True.
9. Compute the Square euclidean distance (Sd), $Sd = rx^2 + ry^2$.
10. If Sd is smaller than the square range (R^2), return True. On the other case, return False.

According to figure 4.3, if a point A is in the center of the blue circle (C) and the distance between A and C is R, then Sd would be computed only if the point B is inside the painted area. Consequently, COMD requires less computation time than an algorithm that only compares the euclidean distance of [A,B] with R.

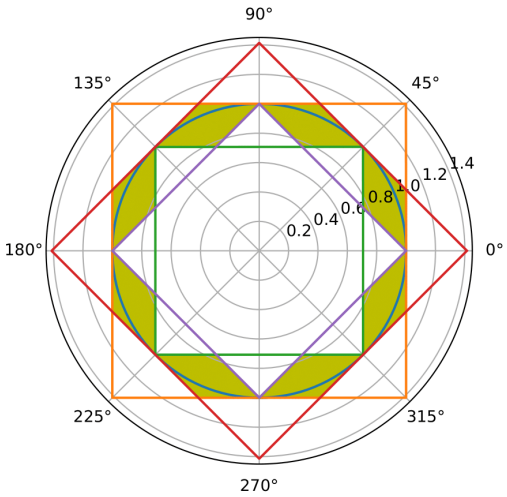


Figure 4.3: Area in which the square euclidean distance is computed

4.3 Neighbor Cell Algorithm

NCL defines a grid that covers all the area of a Traces graph, it assigns POIs to the corresponding cells, and it compares the distances (by using COMD) of a trace-point (tp[i]) with the POIs that are placed in the same cell (C) as tp[i], and with the POIs in the cells around C. The size of a cell is MxM, where M is the spatial range (SR) over two. In the example of 4.4, NCL compares the distances of Z with A,C,D,E instead of comparing the distances of Z with all the POIs. Therefore, NCL optimizes the computation time of the TRAILS extraction algorithm.

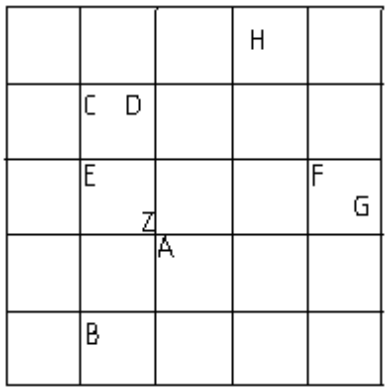


Figure 4.4: Grid of TRAILS graph with Z as trace-point and [A,B,C,D,E,F,G,H] as POIs

CHAPTER 5

Implementation

The TRAILS mobility model was implemented in two phases. The first phase is the TRAILS generator which consists in a python program with several functions. The first function imports and process a trace graph; the second function creates a plot or graphical representation of the traces; the third function extracts POIs and links from traces; the fourth function identifies no-return links; the fifth function creates a plot or graphical representation of TRAILS; and finally, the sixth function exports a TRAILS graph. The TRAILS generator program exports a TRAILS mobility graph as two CSV tables (one for POIs and one for links).

The second phase consists in two mobility simulators implemented in Omnet++. The first simulator entitled **TRACES** imports and simulates traces with the same format as the trace-set described in [33], and the second simulator designated as **TRAILS** loads a TRAILS graph, and includes different links based on its configuration parameters.

This chapter describes the format of TRAILS graphs, then it explains the structure and the configuration parameters of the different implementation phases. TRAILS implementation can be found in [51].

5.1 TRAILS Graph format

TRAILS graphs are composed by two Comma-separated values file format (CSV) files. The first file describes the parameters of POIs (POIs.csv). The second file portrays the parameters of every link (Links.csv). The CSV table delimiter is ESP.

5.1.1 POIs.csv

Each POI is described in two rows. The first row contains the position of the POI in Cartesian coordinates in meters. The second row shows the list of stay-times of the POI in seconds, as shown in tables 5.1, 5.2.

5.1.2 Links.csv

Each link is described in 4 rows. The first row contains the following elements.

1. **InitialPOI**: it is the identifier of the POI where the link begins. The POI identifier is the index divided by two of the POI's first row in POIs.csv.
2. **FinalPOI**: it is the identifier of the POI where the link ends.
3. **Unrealistic Flag**: if the link is unrealistic, then the flag is equal to 1. On the other case, the flag is equal to 0.
4. **ReturnI Flag**: if the link is not a no-return link, then the flag is equal to 1. On the other hand, the flag is equal to 0 (taking into consideration that unrealistic links are included).

5. **ReturnE Flag**: if the link is not a no-return link, then the flag is equal to 1. In the other case, the flag is equal to 0 (taking into consideration that unrealistic links are excluded).

The second row contains the time intervals (t) between consecutive trace-points in the trajectory of the link (tp[i-1].t-tp[i].t). The third and fourth rows contain the Cartesian coordinates of the trace-points (tp[i].x,tp[i].y) in the trajectory of the link, as it is shown in the tables 5.3, 5.4.

X coordinate	Y coordinate	
stay-time[0]	stay-time[1]	stay-time[2]

Table 5.1: POIs file format

23609.795	10654.93	
1560	900	390
23611.45	10667.6	
600	300	390
23623.1	10649.1	
210		

Table 5.2: Example of POIs.csv

InitialPOI	FinalPOI	Unrealistic	ReturnI	ReturnE
tp[i-1].t-tp[i].t	tp[i].t-tp[i+1].t	tp[i+1].t-tp[i+2].t		
tp[i].x	tp[i+1].x			
tp[i].y	tp[i+1].y			

Table 5.3: Links file format

29	55	0	1	1	
30	30	30	30	30	30
24012	24011.9	24010.5	24008.4	24015.8	
10038	10046.1	10042	10043.5	10037.1	
34	35	0	0	0	
30	30	30	30	30	
16297.5	16302	16308.1	16301.7		
2491.1	2484.9	2475.4	2469.8		

Table 5.4: Example of Links.csv

5.2 TRAILS generator

As described in the section above, during the first phase the TRAILS generator creates a TRAILS graph based on traces from real scenarios and configuration parameters such as the POI's maximum diameter (SR) and the minimum time interval spent in a POI by a mobile user (TR). TRAILS generator is a modular program developed in object oriented programming language entitled python 2.7. [52]
This section describes the design of the TRAILS generator program, it presents the modes of operation, and the configuration parameters.

5.2.1 Software design

The generator program is formed by eighteen python modules and three packages. The packages are located in a source directory alongside with six python files. The file structure is shown in figure 5.1.

The **Compare** package is composed by comd.py file which is the implementation of COMD. COMD is equivalent to the direct comparison of the euclidean distance of two points with a constant threshold, but it needs less computation, as it is explained in the section 4.2.

ImportTraces package contains all the algorithms used to import, process, and represent traces. The package is composed by several python files. The first file which is **users.py** contains functions and objects to process and represent traces; the second file **tracefilter.py** describes algorithms to detect and remove undesired trace-points. The algorithms are described in sections 3.1; the third, fourth, and fifth file are called **type0.py**, **type1.py**, and **type2.py**, and they portray the functions to import traces with the same format as the trace-sets described in [33], [47], and [53] accordingly.

The **TrailsGenerator** package contains all the algorithms used to extract features from traces and to create TRAILS mobility graphs. The package is composed by the different modules. The first module called **objects.py** contains the objects used to represent TRAILS graphs. The second file entitled **trails.py** describes the functions to generate and export TRAILS graphs. The third module, **smallestcircle.py** portrays the functions to group trace-points by their smallest enclosing circle. The module **extraction.py** which is the fourth file contains functions to extract POIs from traces. The fifth file, **connection.py** are functions to create links between POI and to find unrealistic links. Finally, the last module called **classification.py** describes the algorithms to find no-return links.

In **TrailsGenerator** was used a python library from Project Nayuki called "Smallest enclosing circle" [54] in order to group trace-points by their smallest enclosing circle. The library is an implementation of the algorithm proposed by Emo Welzl in his paper "Smallest enclosing disks (balls and ellipsoids)" [48]. Additionally, the package **TrailsGenerator** presents the implementation of the algorithms described in sections 3.2 and 3.3.

The **_operations.py** module is a file that is not contained in a package. The file **_operations.py** is designed to load configuration parameters and to execute the functions implemented in the packages **Compare**, **ImportTraces**, and **TrailsGenerator**.

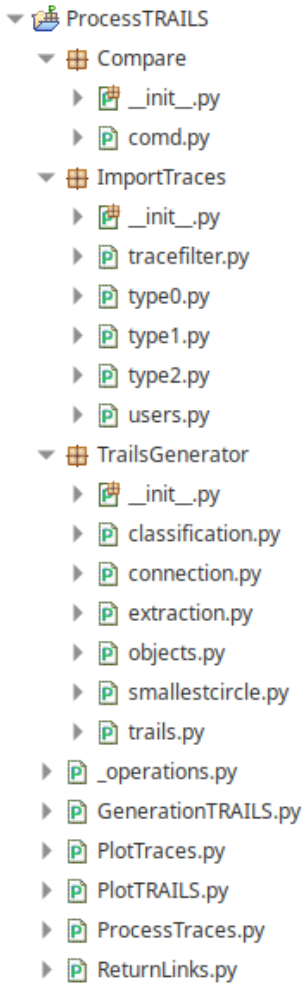


Figure 5.1: Source folder of the TRAILS generator

5.2.2 Modes of operation

In order to have a modular TRAILS generator, five operation modes were implemented and each mode was executed by a different python file.

The **ProcessTraces.py** file works as a mode of operation that imports recorded traces from the input folder. It is possible to import traces with the formats of the trace-sets [33], [47], [53]. The file process traces with the algorithms described in section 3.1. Also it exports the filtered traces in a new folder called Traces in the output directory. The exported traces have the same format used in the trace-set [33]. Finally it creates a file called **TracesGraphOuput.txt** in the output folder with the different parameters of the imported traces. **TracesGraphOuput.txt** contains the width of the area in meters, the length of the area in meters, the number of users, and the recording time in seconds as well as the time to process Traces also in seconds.

PlotTraces.py file imports traces from the input folder with the format of the trace-set [33], it plots the traces and save the figure as traces.pdf in the output directory, and it saves a file called TracesPlotOuput.txt in the output directory with the time in seconds to generate the plot.

The file **GenerationTRAILS.py** has different functions such as importing traces from the input folder with the format of trace-set [33], extracting POIs from traces by using the algorithm described in section 3.2, and generating links between POIs by using the algorithm described in section 3.3. The file also finds unrealistic links (as explained in section 3.3.2), it creates a folder called TRAILS in the output directory, it removes POIs that do not belong to any link, and it generates inside the TRAILS folder the files POIs.csv and Links.csv with the format described in section 5.1 (no link is exported as no-return link). Finally, GenerationTRAILS.py generates in the output directory a file called TRAILS-GraphOuput.txt with elements such as number of extracted POIs, number of unrealistic links, number of POIs with links, and time in seconds to generate the TRAILS graph.

ReturnLinks.py performs different operations. It imports a TRAILS graph (POIs.csv, Links.csv) from the input folder; it also finds no-return links as it is explained in section 3.3.1; it creates a folder called SmartTRAILS in the output directory; and it generates the files POIs.csv and Links.csv inside the folder with SmartTRAILS the format described in section 5.1. Additionally, ReturnLinks.py creates in the output directory a file called TRAILSLinksOuput.txt with the elements such as number of total links minus no-return links (taking into consideration that unrealistic links are included), number of no-return links (taking into consideration that unrealistic links are included), number of total links minus no-return links (taking into consideration that unrealistic links are excluded), number of no-return links (taking into consideration that unrealistic links are excluded), and time in seconds to generate the TRAILS graph.

The file entitled **PlotTRAILS.py** is composed by different algorithms. The first algorithm imports a TRAILS graph (POIs.csv, Links.csv) from the input folder. The second creates different plots for the TRAILS graph in the output folder. Finally, the last algorithm creates a file called **TRAILSPlotOuput.txt** in the output directory with the time in seconds to generate the plots. The following plots are generated in order to have a complete graphical representation of a TRAILS graph.

- **trails.pdf** Exclude no-return links and exclude unrealistic links.
- **trails_noReturn.pdf** Include no-return links and exclude unrealistic links.
- **trails_Unrealistic.pdf** Exclude no-return links and include unrealistic links.
- **trails_noReturn_Unrealistic.pdf** Include no-return links and include unrealistic links.

5.2.3 Configuration parameters

The configuration parameters of the TRAILS generator are described in _operations.py. There are fourteen parameters organized in four different objects.

1. **InOutFiles** Object with file's directories.
   (a) **inFolder** Parameter with the input directory.
   (b) **outFolder** Parameter with the output directory.
2. **TraceParameters** Object with parameters to process traces.
   (a) **traceType** Parameter to select the format of the imported traces. The values are 0, 1, and 2 are used to import traces with the format of the trace-set [33], [47], and [53] accordingly.
   (b) **MaxV** Velocity threshold (meters/second) to remove noise in traces by using the algorithm described in section 3.1.2.

   (c) **MaxD** Distance threshold (meters) to remove noise in traces by using the algorithm described in section 3.1.2.
   (d) **resolution** Set a new resolution (meters) for the coordinates of the traces.
3. **TrailsParameters** Object with parameters to generate TRAILS graphs.
   (a) **SR** Spatial range (meters) used to form POIs with the algorithm described in section 3.2.
   (b) **TR** Temporal range, minimum time for a valid POI. The parameter is used by the algorithm decribed in section 3.2.
   (c) **MinU** Distance threshold to detect unrealistic links by using the algorithm described in section 3.3.2.
4. **PlotParameters** Object with parameters to represent traces and TRAILS graphs in a graphical way by using the matplotlib library. [55]
   (a) **POISize** Size of the figure (dots) representing POIs in meters.
   (b) **arrowSize** Size of the arrows (dots) used to describe the direction of links.
   (c) **lineWith** Width of the line (dots) representing a link or a trace.
   (d) **dpi** Plot resolution in dots per inch.
   (e) **fSize** Plot Size (width,length) in inches.

5.3 Mobility model simulator

Two mobility model simulators were developed in the C++ 11 standard [56]. Both models are implemented as extensions of the mobility packet of the INET framework [57], and they are executed under the discrete event simulator Omnet++[31]. The mobility models inherit the LineSegmentsMobilityBase class of the INET framework in order to simulate movement [57]. The purpose of the simulators is to compare the results of Trace-based mobility models and TRAILS.

5.3.1 Traces simulator

The Traces simulator is composed by the traces module, and it has only one configuration parameter called **csvFolder**. The **csvFolder** represents the input directory that describes the folder where the traces are saved.
The traces module is formed by three files (traces.cpp, traces.h, traces.ned). The module is compiled inside the INET framework in $INET\_PATH/src/inet/mobility/traces$.
The Traces simulator is capable of simulating traces with the format of the trace-set described in [33].

According the Traces Simulator each mobile user follows the next steps.

1. Read its own index.
2. Load the trace that corresponds with user's index (for example load the 5.csv if the index is equal to 5).
3. Start at the position of the first trace-point.
4. Move to the next trace-point (tp) and arrive to tp at the time described in its timestamp.
5. Go to step 4

5.3.2 TRAILS simulator

The TRAILS simulator is composed by the two modules. The first module called **Coordinator** loads a TRAILS graph and it gives instructions to each simulated user depending

on the user's state. The second entitled **User** starts at a random POI, and then it follows the instructions received from the module **Coordinator**. Each module is formed by three files as shown in figure 5.2. The modules are compiled inside the INET framework in $INET\_PATH/src/inet/mobility/trails$.

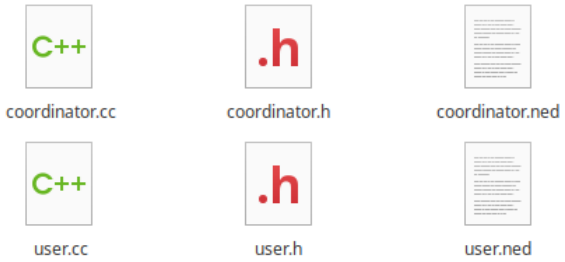


Figure 5.2: Source folder of the TRAILS simulator

5.3.2.1 Configuration parameters

The following parameters are used to configure the TRAILS simulator. The configuration parameters are defined in the **Coordinator** module (coordinator.ned).

1. **enable**: if this parameter is true, it enables the coordinator module.
2. **trailsFolder** Input folder where POIs.csv and Links.csv are saved.
3. **mode**: it defines how links are loaded based on the following values.
   - **0**: it does not load no-return links.
   - **1**: it includes no-return links, then it creates a new link for each loaded link, but in the opposite direction.
   - **2**: it includes no-return links, then it creates a new link for each no-return link, but in the opposite direction.
4. **includeUnrealistic**: if this parameter is true, it includes unrealistic links in the TRAILS graph.

5.3.2.2 TRAILS Simulator algorithm

In the TRAILS simulator each user asks instructions to the coordinator every time it arrives to a new POI, while the Coordinator builds the instructions based on the current POI of each user.

The **Coordinator** module imports TRAILS graphs in its initialization process by following the next steps. First, it loads the files **POIs.csv** and **Links.csv**. Then, the module creates a list of POIs in which each POI has a list of stay times and the coordinates of the POI's position. Later, **Coordinator** loads links based on the values of the *mode* and *includeUnrealistic* parameters (the meaning of those parameters is explained in section 5.3.2.1). Finally, links with the same initial POI are grouped as lists, and each list is saved as an attribute of the corresponding initial POI.

The **Coordinator** has two functions that are triggered by user's events. The first function called **getInitialPOI()** is triggered at the user's initialization. The function **getInitialPOI()** chooses a random POI from the list of POIs and give the pointer to the user. The second function entitled **getInstruction(*new_POI)** is triggered every time a user arrives to a new POI, The function **getInstruction(*new_POI)** reads the new POI and

it gives an instruction to the user. An **instruction** is an object that contains the time a user should stay in the new POI (stay-time), and a pointer to the link that the user should follow after the stay-time is over. The link and stay-time are chosen randomly from the new POI's attributes.

Each mobile user moves between POIs through different links by following the functions described in the **User** module. The algorithm of the **User** module is summarized in the following steps.

1. Get an initial POI from the coordinator (coordinator.getInitialPOI()).
2. Start at the position of the POI.
3. Get an instruction from the Coordinator by giving the pointer of the POI where the user is located.
4. Wait in the current POI a time interval (stay-time) that is specified in the instruction.
5. Follow the link specified in the instruction until the user arrives to the final POI of the link.
6. Go to step 3.

CHAPTER 6

Results

In order to evaluate and validate the TRAILS model and its implementation, this chapter presents five metrics used to evaluate the performance of TRAILS, then it describes the four scenarios or cases of study used to generate TRAILS graphs. Later on, it shows the computation time required to generate TRAILS graphs, after this it compares mobility graphs sizes and the simulation time between traces and TRAILS. Subsequently, it describes the relation of mobility metrics with the simulation time and the number of users. Finally, it explains the relation of the users' behavior between traces and TRAILS.

6.1 Scenarios

In order to analyze TRAILS mobility model we used four traces with different areas, recorded times, and number of users, as it is shown in table 6.1. For each trace we created a TRAILS graph with the following characteristics: the SR of a POI is 30 meters, and TR is 5 minutes; unrealistic links are included; finally, the graph includes no-return links, and for each link there is another one in the opposite direction. In other words, all links are bidirectional.

Name	Area Km^2	Time Hours	Users
San Francisco [47]	7145.486	575.425	536
Rome [53]	10238.67	720	315
New York [33]	618.699	22.658	39
Orlando [33]	276.592	14.283	41

Table 6.1: Characteristics of different scenarios

Table 6.2 shows the total number of POIs and the number of links that compose the TRAILS graphs of each scenario. It can be observed in the table 6.2 that POI and links tend to increase when the simulation time or the number of users is higher.

Scenario	Number of POIs	Number of links
San Francisco	40701	189765
Rome	14985	149033
New York	198	293
Orlando	882	1412

Table 6.2: TRAILS graphs characteristics

6.2 Mobility metrics

One of the purposes of using mobility models is to simulate the behavior of decentralized wireless network protocols before they are implemented in a real scenario. Nonetheless, the performance of a decentralized wireless network protocol depends highly on the interaction between users [5]. Therefore, if a mobility model such as TRAILS produce a similar interaction between users than the recorded traces we can assume that the performance of a decentralized wireless network protocol would be similar if it is simulated with TRAILS or if it is implemented in the same scenario where the traces were recorded. [58]

In order to compare TRAILS with real scenarios we need to measure the interaction between users. In consequence, we use the following mobility metrics. [58], [17], [3]

1. **Contact duration between nodes (CDBN)**: it is the time interval when two users are in contact. Being in contact means that two users are at a distance smaller than a threshold SR of each other. SR is equal to 30m for this case of study.
2. **Contact probability (CNPR)**: it is the estimation of the probability of two mobile nodes being at a distance smaller than SR. It is calculated by dividing the total time of the two nodes being at a distance smaller than SR by the total simulation time.
3. **Number of of contacts between nodes (NCBN)**: it is the total number of times that each user was in contact with another user.
4. **Number of of contacts between the same pair of nodes (NCBS)**: it is the total number of times that a pair of users were in contact.
5. **Contact time between nodes (CTBN)**: it is the time interval between the last contact and a new contact of a user.

To obtain samples of the mentioned metrics the recorded traces and the TRAILS graphs in Omnet++ [31] were simulated with an additional tool called **MobilityModelCheck** [59], as it is shown in figure 6.1.

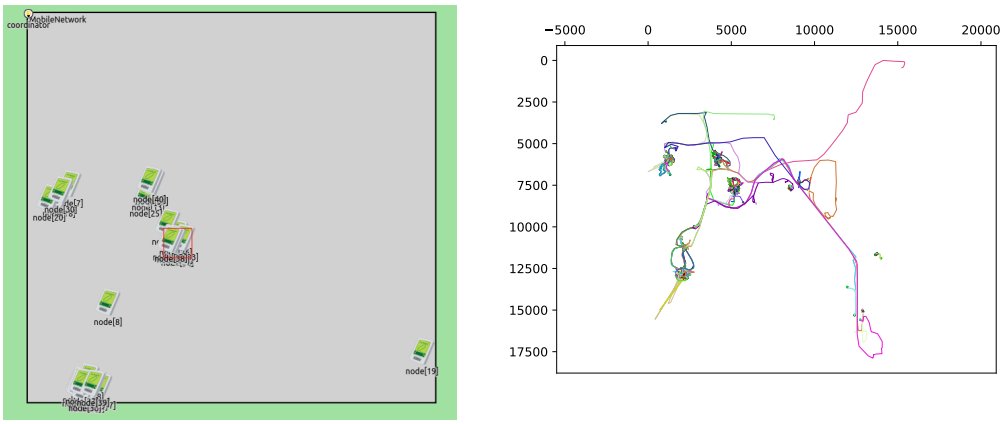


Figure 6.1: Orlando TRAILS Simulation (Left), Orlando TRAILS graph (Right)

6.3 Model Validation

A model validation is essential to define in which cases a model is useful and if a model is capable of representing certain parameters from a real scenario. This section evaluates five mobility metrics from TRAILS graphs of different scenarios and it compares them with the metrics of the original traces.

6.3.1 San Francisco scenario

We compute the distributions of the metrics of the San Francisco traces and of the corresponding TRAILS graphs as is shown in figures 6.2, 6.3, 6.4, 6.5, and 6.6. Additionally, in appendix A we show the sample distributions of TRAILS' and traces' metrics of the scenarios of Rome, New York, and Orlando.

We can observe in figures 6.4, and 6.5 that some metrics of TRAILS have different distributions in relation to traces. On the other hand, figures 6.2 and 6.3 present sample distributions of TRAILS' metrics that are visually similar to sample distributions of traces' metrics. However, the differences between the metrics of TRAILS and traces cannot be quantified from the distribution's plots. For that reason, we use statistical tests in order to evaluate the relation between distributions of metrics. Additionally, we observe in figures 6.2, 6.3, 6.4, 6.5, and 6.6 that the samples do not have a normal distribution. In consequence, we perform non-parametric statistical tests with the sample distributions of the metrics in order to avoid misleading results.

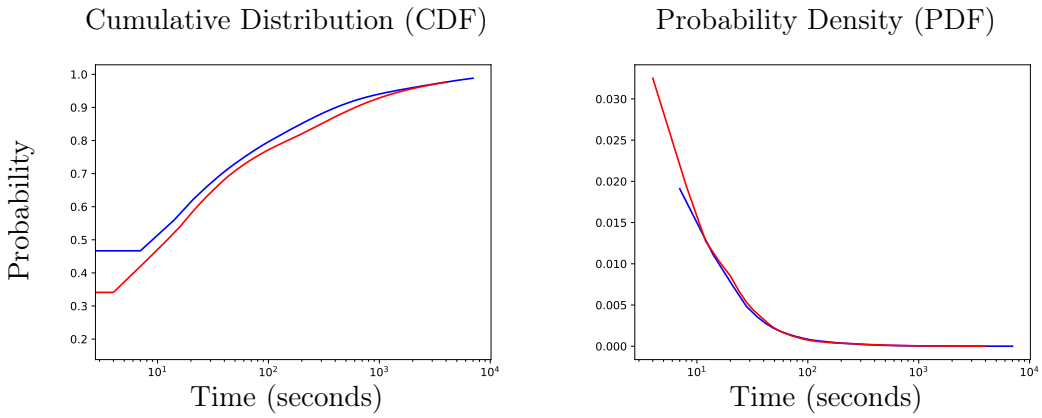


Figure 6.2: CDBN of TRAILS (red) and traces (blue) of 536 taxis in San Francisco

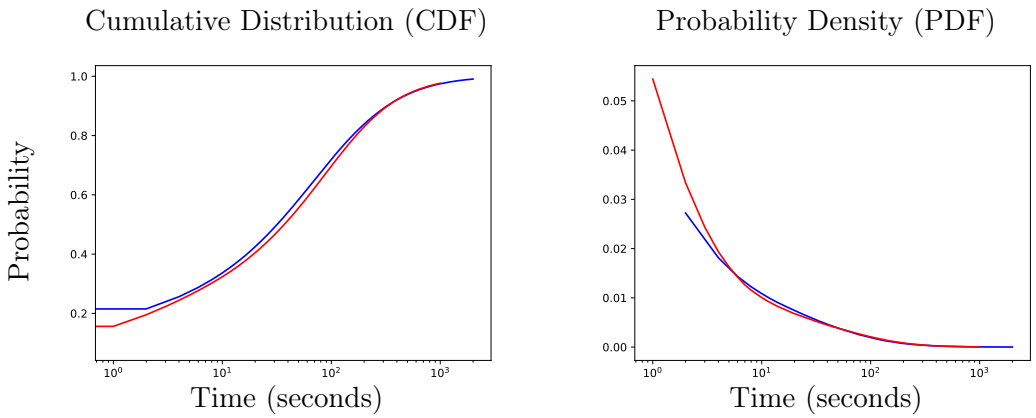


Figure 6.3: CTBN of TRAILS (red) and traces (blue) of 536 taxis in San Francisco

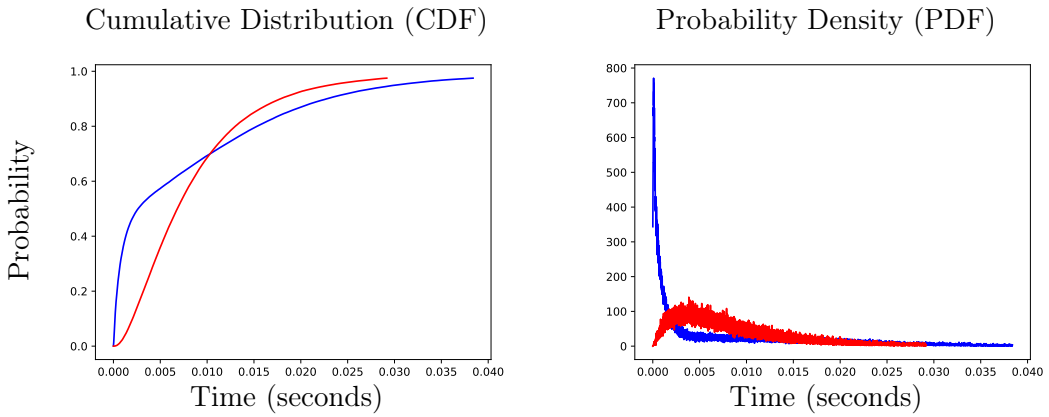


Figure 6.4: CNPR of TRAILS (red) and traces (blue) of 536 taxis in San Francisco

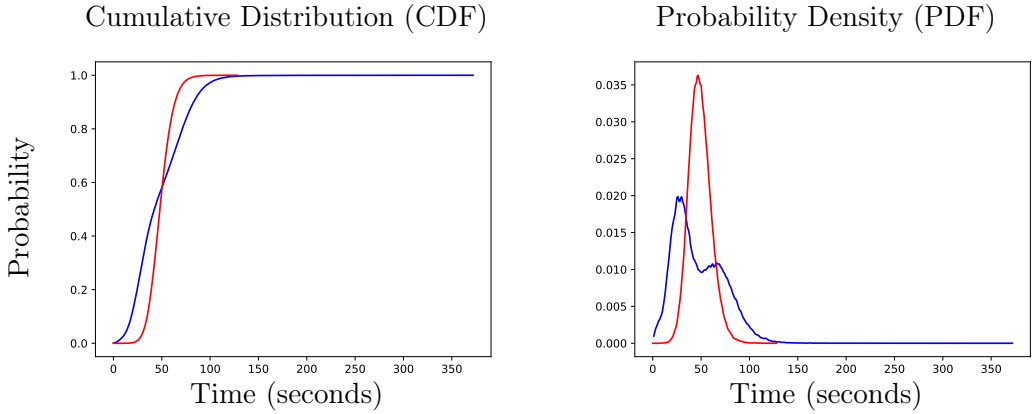


Figure 6.5: NCBS of TRAILS (red) and traces (blue) of 536 taxis in San Francisco

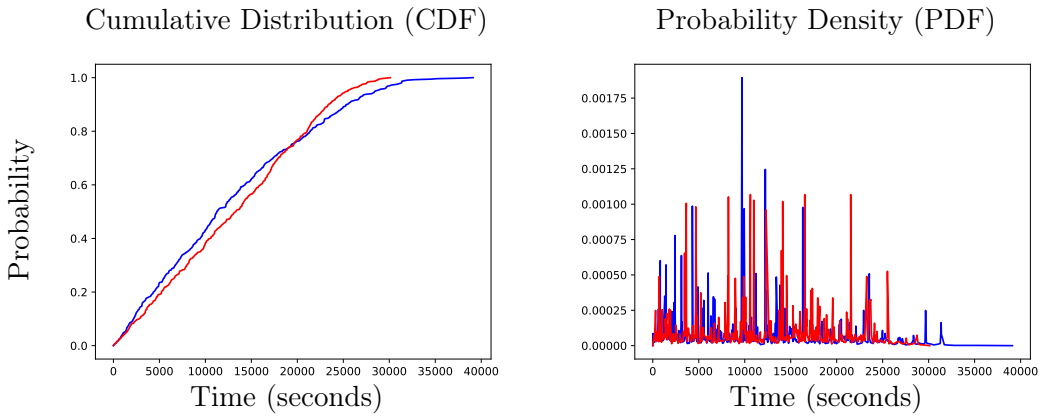


Figure 6.6: NCBN of TRAILS (red) and traces (blue) of 536 taxis in San Francisco

6.3.1.1 Distribution test

To measure how a sample of the metrics from a TRAILS graph is related with a sample from recorded traces, a statistical test was performed.

Our Null hypothesis establishes that both samples are totally different or unrelated. To accept the null hypothesis the Probability value (p-value) should be smaller than 0.05 (5%). Our alternative hypothesis is that both samples belong to populations with the same distributions. In other words, the behavior of certain metric is equivalent in the TRAILS model and in traces. To accept the alternative hypothesis the p-value should be higher than 0.95 (95%). If the p-value is between 0.05 and 0.95, we assume that there is an strong relation between the samples.

In order to compare two samples, the Mann–Whitney U test was used. Mann–Whitney U test is a non parametric test to calculate how likely it is that two samples come from populations with the same distributions.[60]

Metric	ρ-score	p-value
CDBN	11.106	$1.1715e^{-128}$
CNPR	124.379	0.0
NCBS	54.732	0.0
NCBN	1.273	0.202
CTBN	45.575	0.0

Table 6.3: Comparison between distributions of metrics of the San Francisco scenario

Table 6.3 shows the results of the statistical test. We can observe that the p-value for CDBN is significantly low. On the other hand, the CDBN distributions in figure 6.2 seem to be very similar. The reason is that the null hypothesis of the test establishes that a random selected value from one sample would have the same probability to be bigger or smaller than a random selected value from another sample [60], and according to the Cumulative distribution function (CDF) of CDBN in figure 6.2, the CDBN probability in TRAILS tend to be always smaller than the the CDBN probability in traces. The same case can be observed in the CTBN metric.

According to the test, the samples from traces and TRAILS are unrelated in all the metrics except of NCBN. The reason of this results is that the stay times and the links that a user is allowed to choose at each POI in the TRAILS model are always the same. For example, in a trails model it is possible that a user goes to a supermarket and stays for 20 minutes there in the middle of the night, even if that never happened in the recorded traces. In other words, we can infer that in the original traces the users behave in different ways depending the time of the day or the day of the week, and the TRAILS model is not considering those time dependent changes. To prove the previous point a time invariability test was performed for the metrics of the recorded traces.

6.3.1.2 Time invariability test

The null hypothesis establishes that the distribution of a sample does not change significantly over time. To accept the null hypothesis the p-value (probability value) should be greater than 0.05 (5%).

Before performing the test, a complete sample of 575.425 hours was taken and it was divided into samples (cycles) of eight hours each. Then, a Kruskal–Wallis test was performed with all the small samples. The Kruskal–Wallis test is a non parametric test to calculate how much likely is that more than two samples come from populations with equal distributions. [61]

Metric	H-score	p-value
CDBN	11221.041	0.0
CNPR	7847.123	0.0
NCBS	7847.855	0.0
NCBN	84.459	0.131
CTBN	10017.024	0.0

Table 6.4: Time invariability test results

Table 6.4 shows that distributions of all metrics in traces except of NCBN change drastically over time. Therefore, the TRAILS model can only represent the distribution of NCBN of the original traces in this case.

6.3.2 Other scenarios

The purpose of analyzing four different scenarios is to find results that represent the relation of TRAILS graphs with recorded traces from a general perspective, and to avoid biased conclusions driven by the characteristics of an specific scenario. Therefore, Tables 6.5, 6.6, and 6.7 describe the relation between the metrics of TRAILS and traces of the scenarios of Rome, New York, and Orlando respectively, based on the procedure explained in section 6.3.1.1.

It can be observed in table 6.5 (which represents the Rome scenario) that there is only one metric (NCBN) that is not completely different between TRAILS and traces. On the other hand, table 6.7 (which represents the Orlando scenario) presents evidence of similarities between TRAILS and traces in four metrics (CDBN, CNPR, NCBS, and NCBN). Taking into consideration the results of the comparison between TRAILS and traces in the four scenarios (Tables 6.3, 6.5, 6.6, and 6.7) we found that the relation of TRAILS and traces becomes more notorious when the record time is shorter, as it is shown in figure 6.7.

Based on the facts presented in section 6.3.1.2, the users in the real traces are changing their relation with each other frequently. In consequence, traces with a longer record time tend to present more changes in the social relation between mobile users. However TRAILS do not consider the dynamic changes in the association between users (as is explained in section 6.3.1.1). Therefore, a TRAILS graph can present more similarities with its original trace, if the record time is shorter, as is shown if figure 6.7.

Metric	ρ-score	p-value
CDBN	-68.646	$1.1715e^{-128}$
CNPR	57.471	0.0
NCBS	49.73	0.0
NCBN	-0.0752	0.94
CTBN	40.737	0.0

Table 6.5: Comparison between distributions of metrics of the Rome scenario

Metric	ρ-score	p-value
CDBN	2.329	0.0198
CNPR	-0.182	0.854
NCBS	-0.259	0.795
NCBN	2.145	0.0319
CTBN	-4.61	$4.0116e^{-06}$

Table 6.6: Comparison between distributions of metrics of the New York scenario

Metric	ρ-score	p-value
CDBN	0.462	0.643
CNPR	0.87	0.383
NCBS	0.906	0.364
NCBN	1.587	0.112
CTBN	2.444	0.0145

Table 6.7: Comparison between distributions of metrics of the Orlando scenario

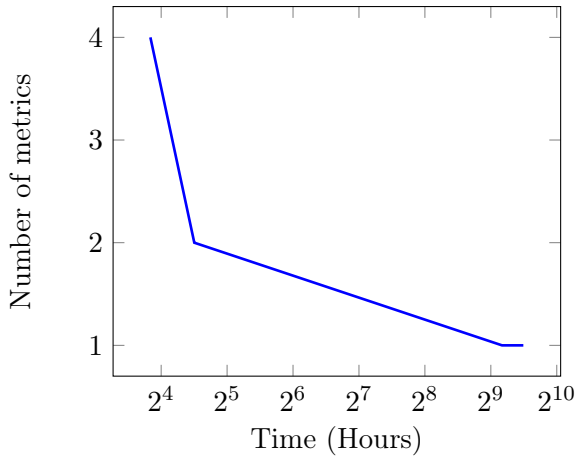


Figure 6.7: Relation between record time and number of TRAILS metrics that present evidence of similarity with traces

6.4 Model scalability

The same TRAILS graph can be simulated for different time intervals and with a different number of users. On the other hand, Trace-Based mobility models lack of that kind of flexibility. In consequence, TRAILS is considered a fully scalable mobility model. [18]

Figures 6.8, 6.9, 6.10, 6.11, and 6.12 present how the metrics described in section 6.2 behave when the simulation time or the number of users change in the TRAILS simulations of the San Francisco scenario. We chose the San Francisco scenario because its traces have a high number of users and a long record time.

According to figure 6.8, CDBN has a similar mean value for different number of users. When the simulation time increases, the mean of CDBN increases drastically and then it becomes approximately constant. A very similar behavior can be noticed in figure 6.9 with the CNPR. The reason of the previous statement is that the simulation is not able to register long contacts between nodes when the simulation time is not long enough.

Figure 6.10 shows that the mean of CTBN does not vary significantly when the simulation time increases but the mean of CTBN decreases drastically also with a logarithmic relation when the number of node increases . In consequence, the probability of a user finding another user in a shorter time interval increases when the number of users is higher.

In figure 6.11 can be observed that the mean of NCBN has linear relation with the simulation time and the number of users. On the other hand, the mean of NCBS has linear relation only with the simulation time, and the mean of NCBS seem to remain approximately constant when the number of users increases, as is shown in figure 6.12. In consequence, the NCBS is not influenced by the number of users, but NCBS has a linear influence with the simulation time. NCBN is affected linearly by the number of users and by the simulation time.

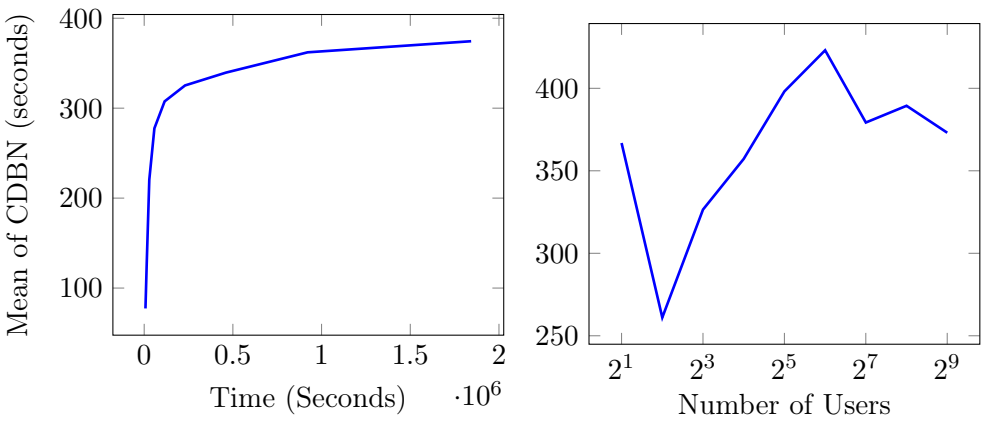


Figure 6.8: Relation of CDBN mean of TRAILS results of the San Francisco scenario

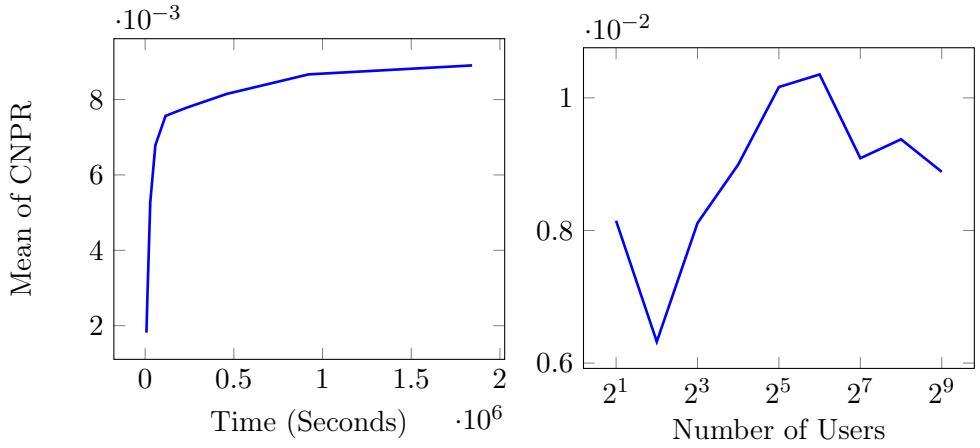


Figure 6.9: Relation of CNPR mean of TRAILS results of the San Francisco scenario

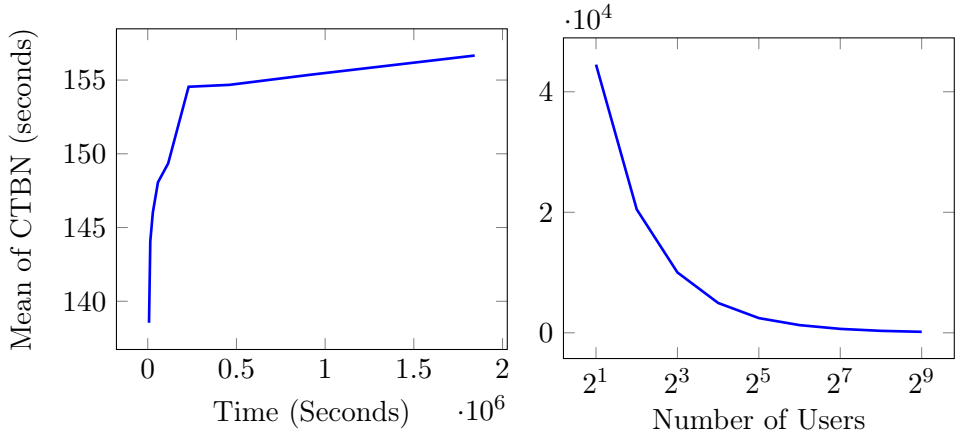


Figure 6.10: Relation of CTBN mean of TRAILS results of the San Francisco scenario

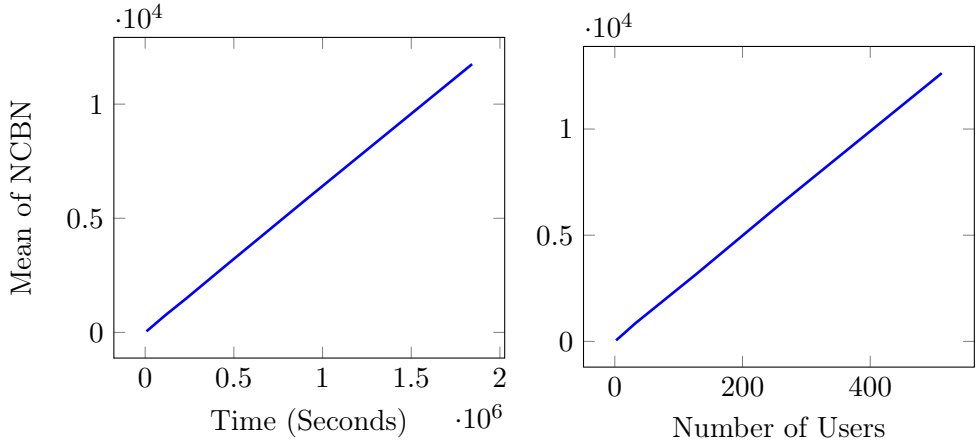


Figure 6.11: Relation of NCBN mean of TRAILS results of the San Francisco scenario

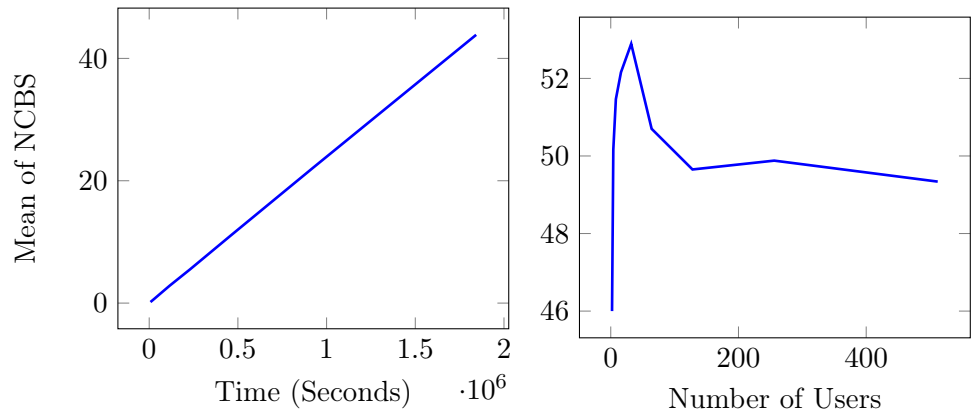


Figure 6.12: Relation of NCBS mean of TRAILS results of the San Francisco scenario

6.5 TRAILS performance

Different results related to data memory and computation time are presented in order to analyze the performance of TRAILS. The computation time of processing and simulating TRAILS and traces is presented in tables 6.8 and 6.10. Additionally, the file's size of the graphs of both mobility models is analyzed in table 6.9. The processing algorithms and the simulations were executed on a virtual machine with 48 GB of memory RAM mounted on a server with a processor of 8 cores and a clock frequency of 2.3 GHz.

Table 6.8 describes the computation time of the TRAILS processing algorithms for each case scenario. The first column is the time spent filtering and reformatting traces with the algorithms explained in section 3.1. The second column is the computation time of finding POIs and links from traces with the algorithms explained in sections 3.2 and 3.3, and the third column describes the execution time of classifying no-return links as is explained in section 3.3.1. It can be observed that the TRAILS graphs that have a high number of POIs require a considerable amount time to classify no-return links. The reason is that in the recursive breath-search algorithm implemented to classify links, the number of times that a function is recursively called depends on the number of POIs in the graph.

Scenario	Trace processing	TRAILS graph generation	Links classification
San Francisco	127.004	214.139	456719.475
Rome	525.709	517.314	71429.58
New York	0.229	1.052	0.0335
Orlando	0.988	1.934	0.892

Table 6.8: Time in seconds to generate TRAILS graphs

In table 6.9 is shown the data compression ratio between TRAILS graphs and the traces after being processed with the algorithms explained in section 3.1. The data compression ratio has a mean value of 1.766 and 95% confidence interval between 1.278 and 2.254. It is also shown in table 6.9 that processing traces not always lead to a new graph with a smaller size. The algorithm implemented to transform the coordinates in traces (explained in section 3.1.1) can return new coordinates in meters with a higher number of decimals than the prepossessed coordinates described in degrees. Therefore, it is possible to have processed traces with a higher data-size than the original traces, as it is the case for the San Francisco scenario.

Scenario	Original Traces	Processed traces	TRAILS graphs
San Francisco	379	474.2	357.5
Rome	1638.4	755.3	559.8
New York	3.2	0.431	0.202
Orlando	3.6	1.3	0.576

Table 6.9: Size (in Mega Bytes) of mobility graphs

Table 6.10 shows a relation between the time spent by simulating TRAILS graphs and traces. The time relation has a mean value of 0.719 with 95% confidence interval between 0.663 and 0.774. In other words, it is possible to save approximately 30% of time by simulating TRAILS instead of traces, if the same number of users and simulation time was used for both mobility models.

Scenario	Traces	TRAILS
San Francisco	82273.985	63318.721
Rome	35337.649	24466.268
New York	83.208	60.397
Orlando	56.683	40.563

Table 6.10: Time spent (in seconds) to simulate mobility graphs

CHAPTER 7

Discussion

This thesis presented an optimized implementation of a TRAILS graph generator and a TRAILS simulator. Additionally, it introduced mobility metrics to describe the interaction between users. Finally it evaluated the performance of TRAILS mobility model with respect to four different traces. Two of the four analyzed traces describe the movement of vehicles and other two traces describe the movement of pedestrians. This chapter summarizes the conclusions obtained in this research work and presents recommendations for future work to improve the current state of the TRAILS model.

7.1 Conclusions

The TRAILS mobility model is able to capture certain mobility characteristics of realistic scenarios such as temporal dependency, spatial dependency and geographic restrictions. In a TRAILS simulation, a user mimics temporal dependency of a real scenario by moving between POIs with the same speed as another user in a recorded trace. TRAILS does not capture spatial dependency between users in a direct manner as it is done in mobility models such as CMM with the interaction matrix [17], but TRAILS contains spatial dependency with POIs. Strictly speaking, POIs that are more frequently visited or have longer stay times in real scenarios would have more links and also longer stay times in a TRAILS graph. Therefore, if users stayed more time in certain POIs than others, this behavior would also be reflected in a TRAILS simulation. In conclusion, TRAILS mimics spatial dependency between users by capturing the relation between users and POIs, as it is shown in section 6.3. A user in a TRAILS simulation only moves through links between POIs, and links capture movement patterns in a real scenario. In consequence, TRAILS is capable of representing geographic restrictions of real scenarios.

The movement patterns of the users vary over time in real traces because the users have different routines at different hours or days. Therefore, the interaction between users also changes, as it is proved in section 6.3.1.2. TRAILS does not consider changes in the movement patterns of the users. Additionally, users of a trace with a short record time do not have enough time to change the interaction between each other. Therefore, a TRAILS simulation would present more similarities to a trace-set if the record time is shorter, as it is proved in section 6.3.1.1.

TRAILS graphs are capable of summarizing several trace-points in a POI. In consequence, the data size of TRAILS graphs are shorter than traces and a discrete event simulator requires more computation time if there are more events to attend. A simulation of TRAILS produces less events than a simulation of traces, because when a user stays in a POI, it does not produce extra events. On the other hand, a user in a simulation of traces may produce events even if the user is not moving or if it is moving inside a small area. Therefore, simulating TRAILS graphs requires less computation time than simulating traces, as it is shown in section 6.5.

In summary, TRAILS mobility model is able to capture certain mobility characteristics of realistic scenarios such as temporal dependency, spatial dependency and geographic restrictions. TRAILS simulation would present more similarities to a trace-set if the record time is shorter. Finally, TRAILS graphs are shorter than traces and simulating TRAILS graphs requires less computation time than simulating trace-based models.

7.2 Recommendations

To increase the similarities between traces and a full scalable model as TRAILS, It is suggested the design of a new mobility model based on TRAILS in which the stay times and the links that a user is allowed to choose are related to time-slots. For example, if in traces a supermarket is visited very often from 8:00 until 20:00, then in the TRAILS graph, the links that connect to the supermarket should be available only from 8:00 to 20:00 (simulation time). By changing the available stay times and links over time in a TRAILS simulation, the interaction between users would also change in a similar way as in real scenarios. In consequence, the spatial dependency of traces with log recorded time would be represented in a more accurate manner than the current state of TRAILS.

To classify links, the TRAILS generator applies the breath-first search algorithm with recursive functions. Recursive algorithms tend to consume more memory than iterative algorithms. For this reason, a way to optimize the current tree search problem is to replace the implemented breath-first search algorithm by the iterative deepening search. [62]

All of the implemented algorithms in the TRAILS graph generator and simulator use lists or vectors dynamically allocated, which leads to a higher memory consumption and a higher computation time. However, dynamic memory allocation allows the processor to save pieces of the same list in different blocks of the memory. On the other hand, an array with static allocation requires to be saved in the same memory block. In consequence, large arrays may produce execution errors because they cannot find a large empty block in the memory. In any case, if the algorithms are constrained to be used only on servers with a large memory RAM, it is possible to reduce the computation time by implementing the same algorithms with static allocated arrays. Another solution would be to find the functions that generate relative small vectors in the software, and to replace those vectors with static allocated arrays.

APPENDIX A

Sample Distributions

A.1 Rome scenario

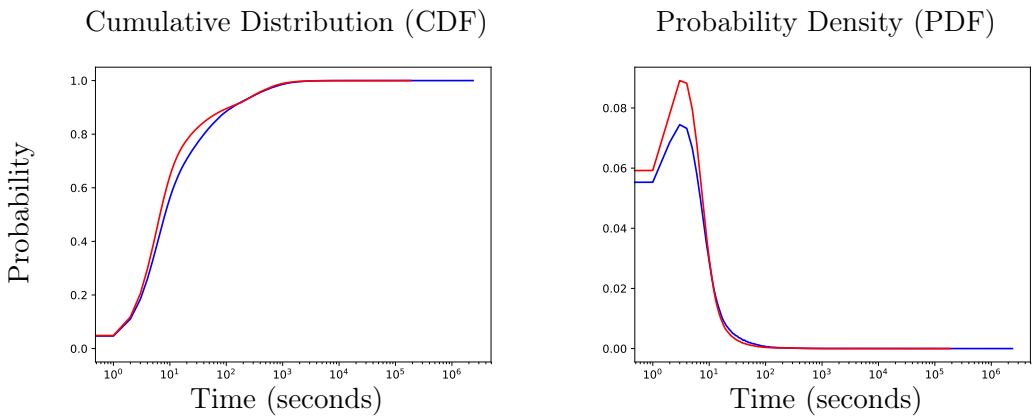


Figure A.1: CDBN of TRAILS (red) and traces (blue) of 315 taxis in Rome

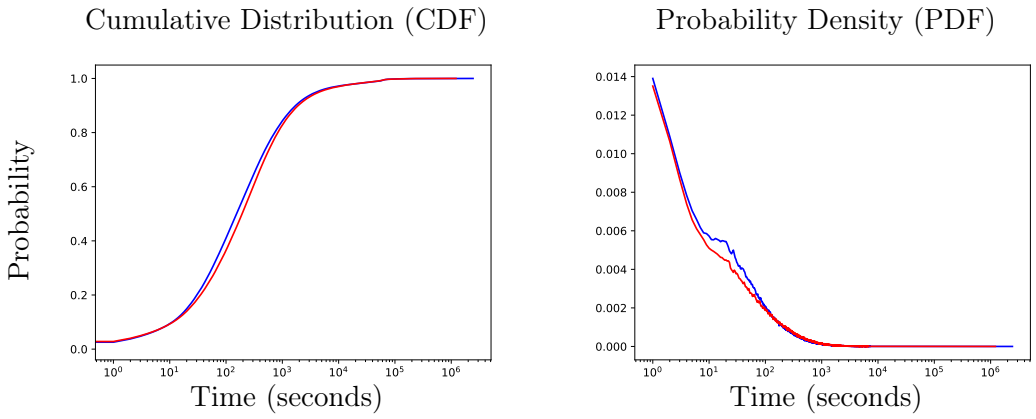


Figure A.2: CTBN of TRAILS (red) and traces (blue) of 315 taxis in Rome

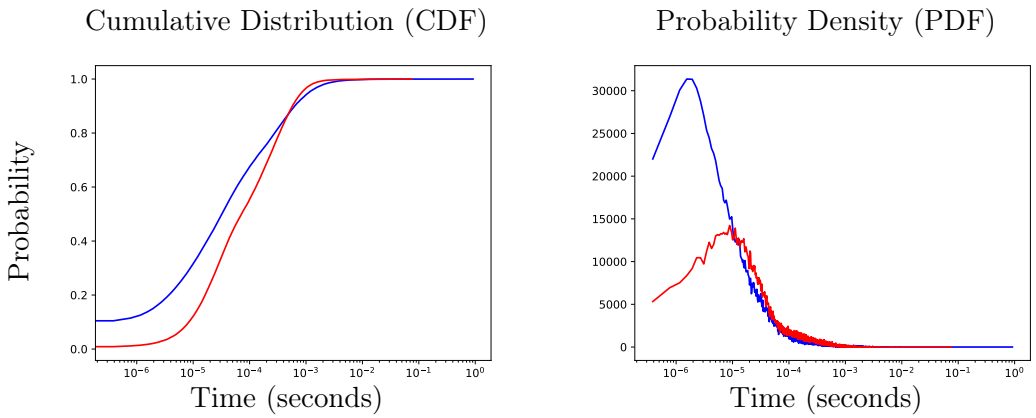


Figure A.3: CNPR of TRAILS (red) and traces (blue) of 315 taxis in Rome

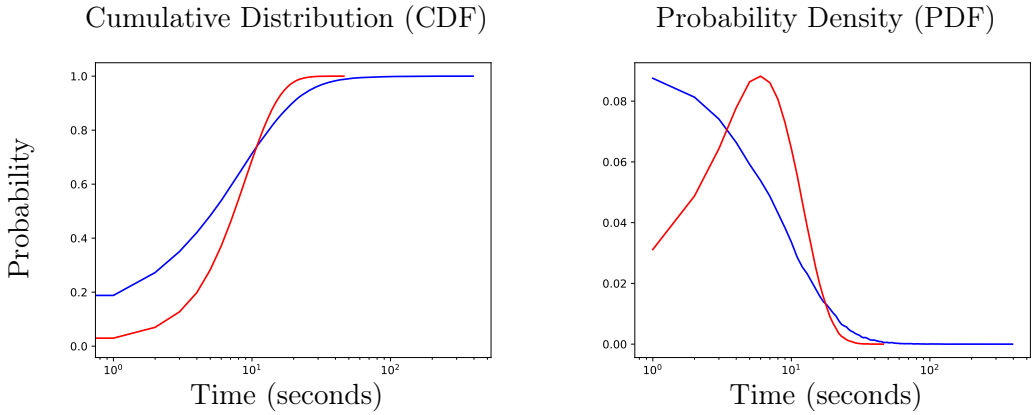


Figure A.4: NCBS of TRAILS (red) and traces (blue) of 315 taxis in Rome

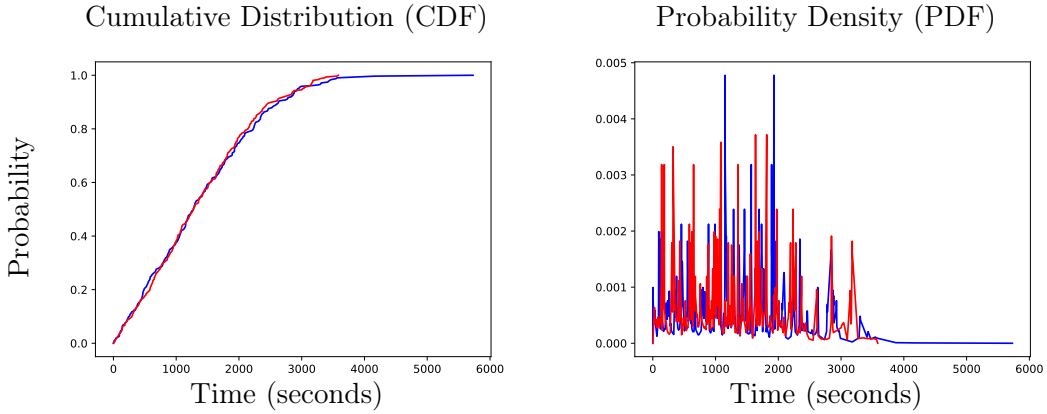


Figure A.5: NCBN of TRAILS (red) and traces (blue) of 315 taxis in Rome

A.2 New York scenario

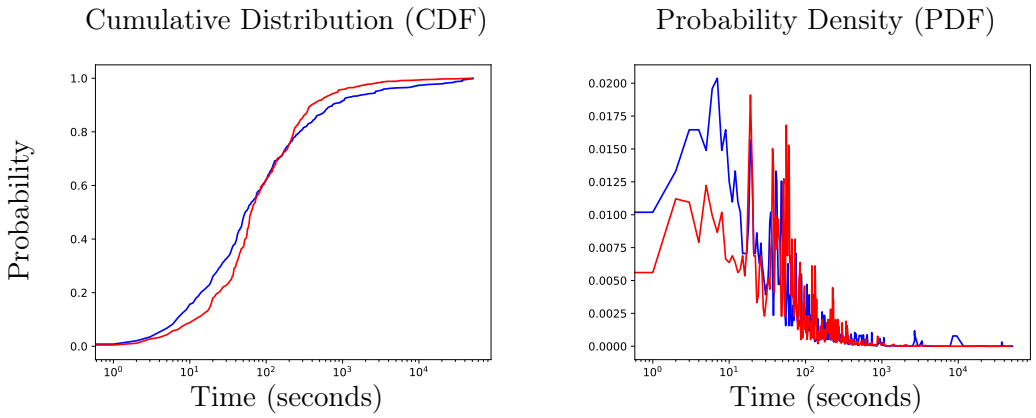


Figure A.6: CDBN of TRAILS (red) and traces (blue) of 39 pedestrians in New York

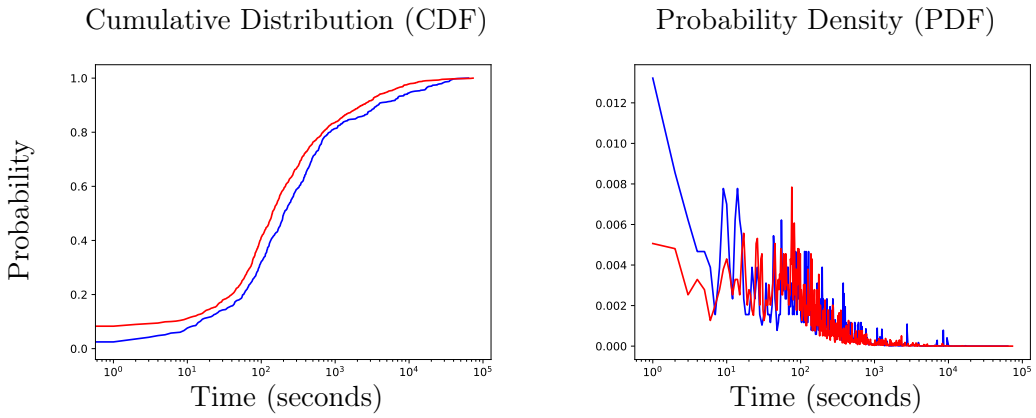


Figure A.7: CTBN of TRAILS (red) and traces (blue) of 39 pedestrians in New York

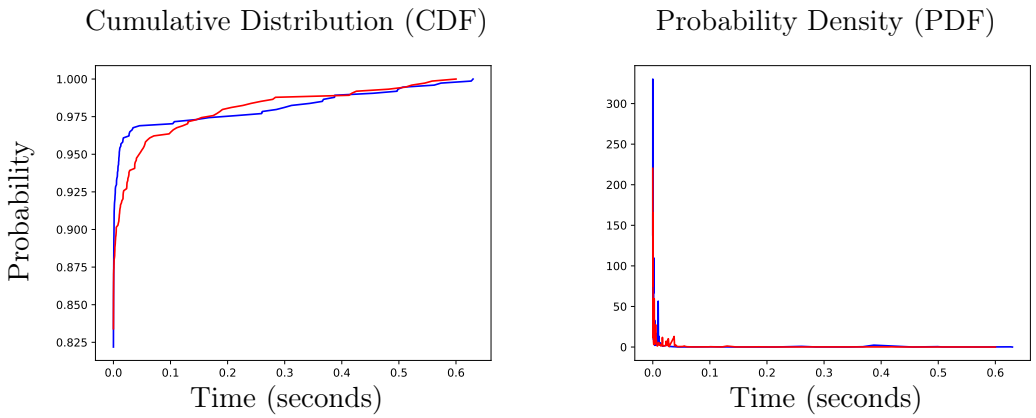


Figure A.8: CNPR of TRAILS (red) and traces (blue) of 39 pedestrians in New York

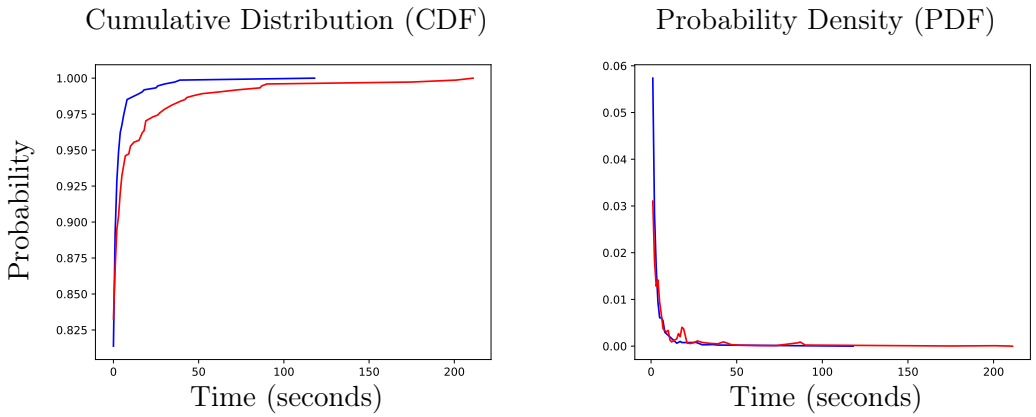


Figure A.9: NCBS of TRAILS (red) and traces (blue) of 39 pedestrians in New York

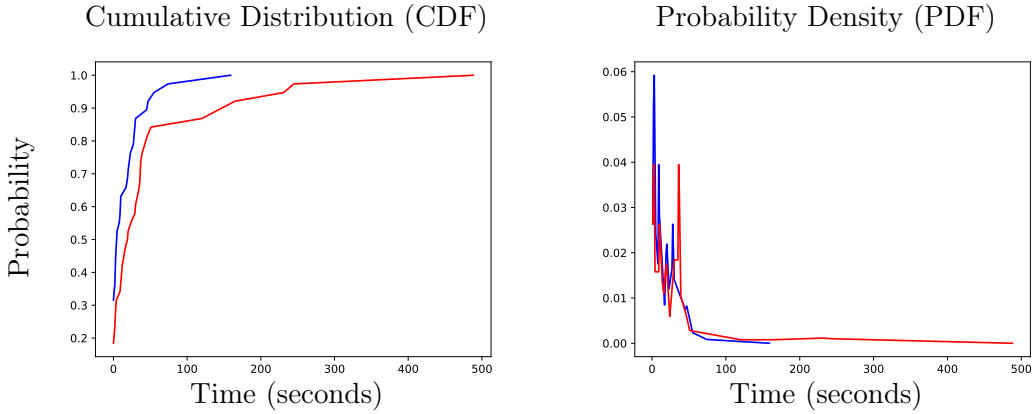


Figure A.10: NCBN of TRAILS (red) and traces (blue) of 39 pedestrians in New York

A.3 Orlando scenario

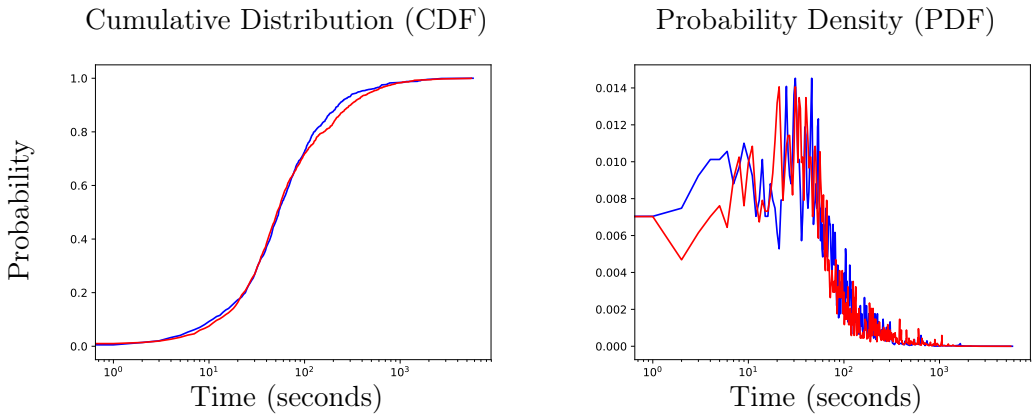


Figure A.11: CDBN of TRAILS (red) and traces (blue) of 41 pedestrians in Orlando

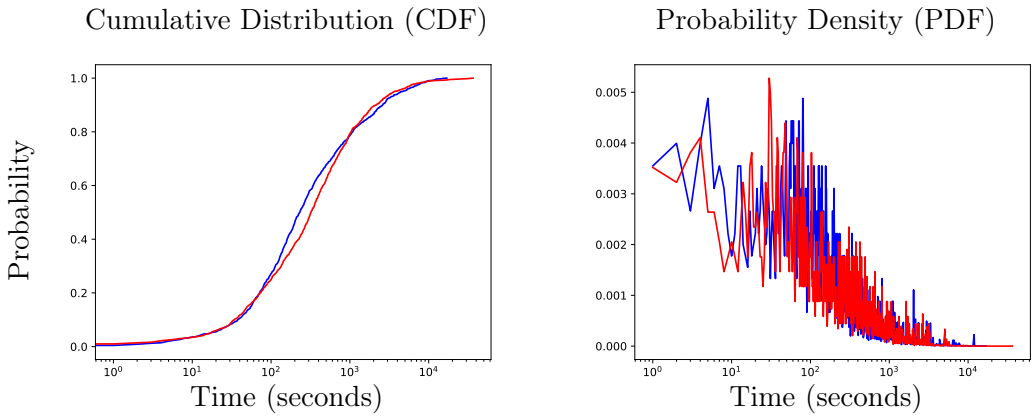


Figure A.12: CTBN of TRAILS (red) and traces (blue) of 41 pedestrians in Orlando

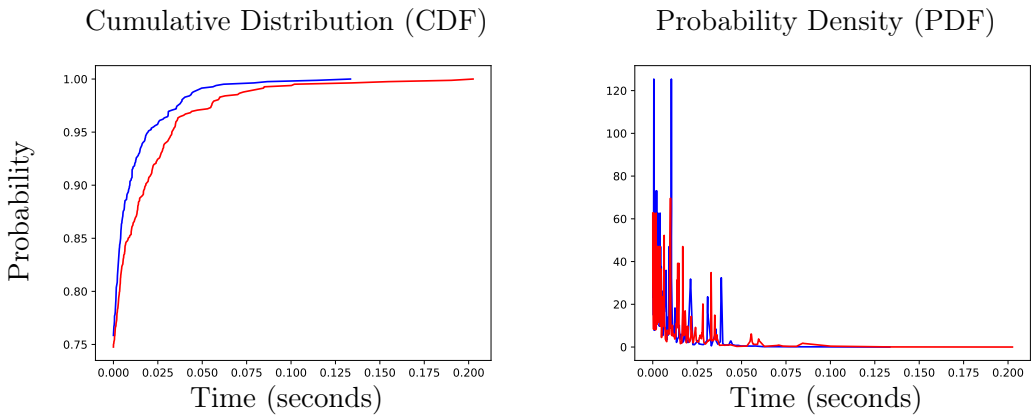


Figure A.13: CNPR of TRAILS (red) and traces (blue) of 41 pedestrians in Orlando

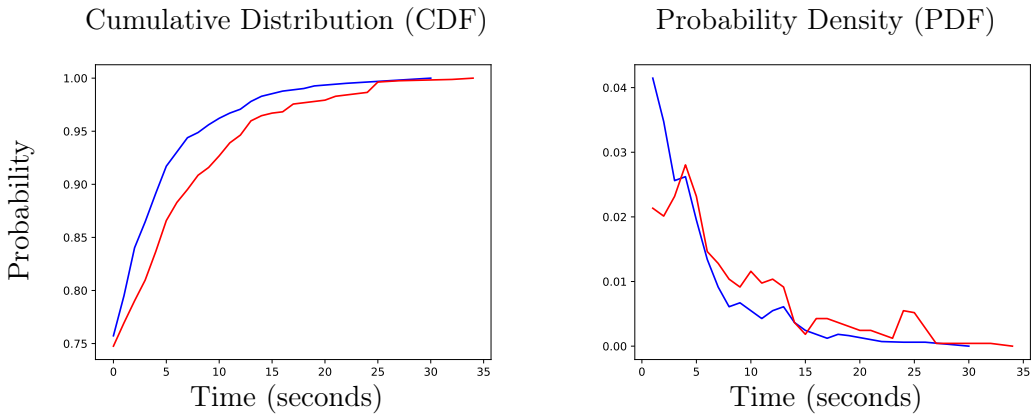


Figure A.14: NCBS of TRAILS (red) and traces (blue) of 41 pedestrians in Orlando

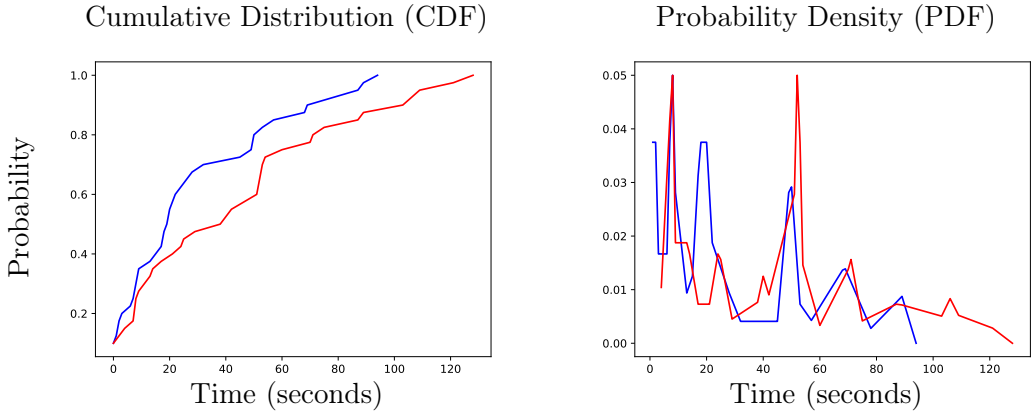


Figure A.15: NCBN of TRAILS (red) and traces (blue) of 41 pedestrians in Orlando

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

TRAILS	TRAce-based ProbabILiStic Mobility Model . . . 3
GNSS	Global Navigation Satellite System . . . 8
MaxD	Distance threshold to filter trace-points . . . 13
MaxV	Velocity threshold to filter trace-points . . . 14
POI	Point of interest . . . 11
STP	Set of consecutive trace-points . . . 15
SR	Spatial range . . . 11
TR	Temporal range . . . 11
SEC	Smallest enclosing circle algorithm . . . 16
COMD	Comparison of multiple distances algorithm . . . 15
NCL	Neighbor Cell Algorithm . . . 15
Cd	Chebyshev distance . . . 22
Md	Manhattan distance . . . 22
Sd	Square euclidean distance . . . 23
MinCR	Minimum Chebyshev range . . . 22
MaxCR	Maximum Chebyshev range . . . 22
MinMR	Minimum Manhattan range . . . 22
MaxMR	Maximum Manhattan range . . . 22
MinU	Minimum Distance for an unrealistic path . . . 20
CSV	Comma-separated values file format . . . 25
RSM	Related STP method . . . 15
ISM	Independent STP method . . . 16
Tp	List of trace-points . . . 15
Lp	List of POIs . . . 15
CDBN	Contact duration between nodes . . . 34
CNPR	Contact probability . . . 34
NCBN	Number of of contacts between nodes . . . 34
NCBS	Number of of contacts between the same pair of nodes . . . 34
CTBN	Contact time between nodes . . . 34
CDF	Cumulative distribution function . . . 37
p-value	Probability value . . . 37
CMM	Community-based Mobility Model . . . 5
MANET	Mobile Ad hoc NETworks . . . 3
RWP	Random WayPoint . . . 5
OPS	Opportunistic protocol simulator . . . 8
RFID	Radio Frequency Identification . . . 8
RSSI	Received signal strength indicator . . . 9
OppNet	Opportunistic Networks . . . 3

BIBLIOGRAPHY

[1] A. Bhoi, "Analysis & comparison of mobility models for ad-hoc network," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 5, pp. 1357–1362.

[2] K. Lee, S. Hong, S. Kim, I. Rhee, and S. Chong, "Slaw: A mobility model for human walks," in *IEEE INFOCOM 2009*. IEEE, 2009, pp. 855–863.

[3] V. Kuppusamy, L. Sarmiento, A. Udugama, and A. Förster, "Community-based mobility model and probabilistic orbit mobility model implementations in omnet++." in *OMNeT++*, 2018, pp. 23–34.

[4] C. Tuduce and T. Gross, "A mobility model based on wlan traces and its validation," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 1. IEEE, 2005, pp. 664–674.

[5] F. Bai and A. Helmy, "A survey of mobility models," *Wireless Adhoc Networks. University of Southern California, USA*, vol. 206, p. 147, 2004.

[6] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless communications and mobile computing*, vol. 2, no. 5, pp. 483–502, 2002.

[7] Q. Dong and W. Dargie, "A survey on mobility and mobility-aware mac protocols in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 88–100, 2012.

[8] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. Acm, 2002, pp. 88–97.

[9] K. Lorincz, D. J. Malan, T. R. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, "Sensor networks for emergency response: challenges and opportunities," *IEEE pervasive Computing*, no. 4, pp. 16–23, 2004.

[10] S. H. Lee, S. Lee, H. Song, and H. S. Lee, "Wireless sensor network design for tactical military applications: Remote large-scale environments," in *MILCOM 2009-2009 IEEE Military communications conference*. IEEE, 2009, pp. 1–7.

[11] A. A. Taleb, T. Alhmiedat, O. A.-h. Hassan, and N. M. Turab, "A survey of sink mobility models for wireless sensor networks," *Journal of Emerging Trends in Computing and Information Sciences*, vol. 4, no. 9, pp. 679–687, 2013.

[12] I. Rhee, K. Lee, S. Hong, S. Kim, and S. Chong, "Demystifying the levy-walk nature of human walks," *Techical Report, NCSU, http://netsrv. csc. ncsu. edu/export/Demystifying Levy Walk Patterns. pdf*, 2008.

[13] M. E. Newman and J. Park, "Why social networks are different from other types of networks," *Physical review E*, vol. 68, no. 3, p. 036122, 2003.

[14] A. Munjal, T. Camp, and N. Aschenbruck, "Changing trends in modeling mobility," *Journal of Electrical and Computer Engineering*, vol. 2012, 2012.

[15] G. Rodolakis, "Analytical models and performance evaluation in massive mobile ad hoc networks," Ph.D. dissertation, Université Paris, 2006.

[16] C. Bettstetter, H. Hartenstein, and X. Pérez-Costa, "Stochastic properties of the random waypoint mobility model," *Wireless Networks*, vol. 10, no. 5, pp. 555–567, 2004.

[17] M. Musolesi and C. Mascolo, "Designing mobility models based on social network theory," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 11, no. 3, pp. 59–70, 2007.

[18] A. Förster, A. Bin Muslim, and A. Udugama, "Trails-a trace-based probabilistic mobility model," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM, 2018, pp. 295–302.

[19] J. Dede, A. Förster, E. Hernández-Orallo, J. Herrera-Tapia, K. Kuladinithi, V. Kuppusamy, P. Manzoni, A. bin Muslim, A. Udugama, and Z. Vatandas, "Simulating opportunistic networks: Survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1547–1573, 2017.

[20] D. Djenouri, A. Derhab, and N. Badache, "Ad hoc networks routing protocols and mobility." *Int. Arab J. Inf. Technol.*, vol. 3, no. 2, pp. 126–133, 2006.

[21] C.-M. Huang, K.-c. Lan, and C.-Z. Tsai, "A survey of opportunistic networks," in *22nd International Conference on Advanced Information Networking and Applications-Workshops (aina workshops 2008)*. IEEE, 2008, pp. 1672–1677.

[22] B.-J. Kwak, N.-O. Song, and L. E. Miller, "A mobility measure for mobile ad hoc networks," *IEEE Communications Letters*, vol. 7, no. 8, pp. 379–381, 2003.

[23] T. Larsson and N. Hedman, "Routing protocols in wireless ad-hoc networks: a simulation study," 1998.

[24] V. Timcenko, M. Stojanovic, and S. B. Rakas, "Manet routing protocols vs. mobility models: performance analysis and comparison," in *Proceedings of the 9th WSEAS international conference on Applied informatics and communications*. World Scientific and Engineering Academy and Society (WSEAS), 2009, pp. 271–276.

[25] M. Särelä and M. Hietalahti, "Security topics and mobility management in hierarchical ad hoc networks: A literature survey," *Helsinki University of Technology Laboratory for Theoretical Computer Science April*, vol. 29, 2004.

[26] A. Vahdat, D. Becker *et al.*, "Epidemic routing for partially connected ad hoc networks," 2000.

[27] A. Balasubramanian, B. Levine, and A. Venkataramani, "Dtn routing as a resource allocation problem," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 373–384.

[28] A. Keränen, J. Ott, and T. Kärkkäinen, "The one simulator for dtn protocol evaluation," in *Proceedings of the 2nd international conference on simulation tools and techniques*. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2009, p. 55.

[29] N. Papanikos, D.-G. Akestoridis, and E. Papapetrou, "Adyton: A network simulator for opportunistic networks," 2015.

[30] A. Udugama, A. Förster, J. Dede, V. Kuppusamy, and A. B. Muslim, "Opportunistic networking protocol simulator for omnet++," *arXiv preprint arXiv:1709.02210*, 2017.

[31] A. Varga, "Omnet++ discrete event simulation system, version 5.4," 2019.

[32] M. Ali, T. Suleman, and Z. A. Uzmi, "Mmac: A mobility-adaptive, collision-free mac protocol for wireless sensor networks," in *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference, 2005.* IEEE, 2005, pp. 401–407.

[33] I. Rhee, M. Shin, S. Hong, K. Lee, S. Kim, and S. Chong, "CRAWDAD dataset ncsu/mobilitymodels (v. 2009-07-23)," Downloaded from https://crawdad.org/ncsu/mobilitymodels/20090723, Jul. 2009.

[34] P. D. Groves, *Principles of GNSS, inertial, and multisensor integrated navigation systems.* Artech house, 2013.

[35] R. Marin-Perianu, M. Marin-Perianu, P. Havinga, and H. Scholten, "Movement-based group awareness with wireless sensor networks," in *International Conference on Pervasive Computing.* Springer, 2007, pp. 298–315.

[36] R. B. Langley, "Nmea 0183: A gps receiver interface standard," *GPS world*, vol. 6, no. 7, pp. 54–57, 1995.

[37] M. L. Ng, K. S. Leong, D. M. Hall, and P. H. Cole, "A small passive uhf rfid tag for livestock identification," in *2005 IEEE International Symposium on Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications*, vol. 1. IEEE, 2005, pp. 67–70.

[38] E. Elnahrawy, X. Li, and R. P. Martin, "The limits of localization using signal strength: A comparative study," in *2004 First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004.* IEEE, 2004, pp. 406–414.

[39] S. Čapkun, M. Hamdi, and J.-P. Hubaux, "Gps-free positioning in mobile ad hoc networks," *Cluster Computing*, vol. 5, no. 2, pp. 157–167, 2002.

[40] X. Huang, M. Barralet, and D. Sharma, "Behaviors of antenna polarization for rssi location identification," in *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, vol. 1. IEEE, 2009, pp. 151–154.

[41] Y. Xu, Y. Ouyang, Z. Le, J. Ford, and F. Makedon, "Mobile anchor-free localization for wireless sensor networks," in *International Conference on Distributed Computing in Sensor Systems.* Springer, 2007, pp. 96–109.

[42] W. Burgard, D. Fox, D. Hennig, and T. Schmidt, "Estimating the absolute position of a mobile robot using position probability grids," in *Proceedings of the national conference on artificial intelligence*, 1996, pp. 896–901.

[43] D. Kotz and T. Henderson, "Crawdad: A community resource for archiving wireless data at dartmouth," *IEEE Pervasive Computing*, vol. 4, no. 4, pp. 12–14, 2005.

[44] M. Karaliopoulos, M. Papadopouli, and E. Raftopoulos, "Measurement-based modeling of large wireless local area networks: the unc-forth wireless measurements project experience," 2006.

[45] A. Helmy, F. Bai, and W. Hsu, "Mobilib: Community-wide library of mobility and wireless networks measurements," *Retrieved from MobiLib. Available online: http://www. cise. ufl. edu/˜ helmy/MobiLib. htm (accessed on 14 July 2016)*, 2008.

[46] C. C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.

[47] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD dataset epfl/mobility (v. 2009-02-24)," Downloaded from https://crawdad.org/epfl/mobility/20090224, Feb. 2009.

[48] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in *New results and new trends in computer science*. Springer, 1991, pp. 359–370.

[49] C. Y. Lee, "An algorithm for path connections and its applications," *IRE transactions on electronic computers*, no. 3, pp. 346–365, 1961.

[50] J. Todd, "Induced norms," in *Basic Numerical Mathematics*. Springer, 1977, pp. 19–28.

[51] L. Sarmiento, "Trails," https://github.com/Leo1690/TRAILS, 2019.

[52] P. S. Foundation, "Python language reference, version 2.7," 2019.

[53] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "CRAWDAD dataset roma/taxi (v. 2014-07-17)," Downloaded from https://crawdad.org/roma/taxi/20140717/taxicabs, Jul. 2014, traceset: taxicabs.

[54] P. Nayuki, *Smallest enclosing circle*, 2018. [Online]. Available: https://www.nayuki.io/page/smallest-enclosing-circle

[55] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[56] ISO, *ISO/IEC 14882:2011 Information technology — Programming languages — C++*, 3rd ed., Sep. 2011. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372

[57] A. Varga, "Inet framework for the omnet++ discrete event simulator, version 4.1," 2019.

[58] T. Hossmann, T. Spyropoulos, and F. Legendre, "Putting contacts into context: Mobility modeling beyond inter-contact times," in *Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing*. ACM, 2011, p. 18.

[59] V. P. J. D. Asanga Udugama, Anna Förster, "Mobilitymodelcheck," https://github.com/ComNets-Bremen/MobilityModelCheck, 2019.

[60] P. E. McKnight and J. Najab, "Mann-whitney u test," *The Corsini encyclopedia of psychology*, pp. 1–1, 2010.

[61] P. E. McKight and J. Najab, "Kruskal-wallis test," *The corsini encyclopedia of psychology*, pp. 1–1, 2010.

[62] R. E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial intelligence*, vol. 27, no. 1, pp. 97–109, 1985.